SIEMENS

SIMATIC

S7
S7-200 SMART

System Manual

Preface

Product overview 1

Getting started 2

Installation 3

PLC concepts 4

Programming concepts 5

PLC device configuration 6

Program instructions 7

Communication 8

Libraries 9

Debugging and troubleshooting 10

PID loops and tuning 11

Open loop motion control 12

Technical specifications A

Calculating a power budget B

Error codes C

Special memory (SM) and system symbol names D

References E

Ordering information F

V2.5, 01/2020

A5E03822230-AI

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E03822230-AI
℗ 12/2019 Subject to change

Copyright © Siemens AG 2020.
All rights reserved

Preface

Purpose of the manual

The S7-200 SMART series is a line of micro-programmable logic controllers (Micro PLCs) that can control a variety of automation applications. Compact design, low cost, and a powerful instruction set make the S7-200 SMART a perfect solution for controlling small applications. The wide variety of S7-200 SMART models and the Windows-based programming tool give you the flexibility you need to solve your automation problems.

This manual provides information about installing and programming the S7-200 SMART CPUs and is designed for engineers, programmers, installers, and electricians who have a general knowledge of programmable logic controllers.

Required basic knowledge

To understand this manual, it is necessary to have a general knowledge of automation and programmable logic controllers.

Scope of the manual

This manual describes the following products:

- STEP 7-Micro/WIN SMART V2.5
- S7-200 SMART CPU firmware release V2.5

For a complete list of the S7-200 SMART products and article numbers described in this manual, see Technical Specifications (Page 751).

Certification, CE label and other standards

Refer to the technical specifications for more information.

Service and support

In addition to our documentation, we offer our technical expertise on the Internet on the customer support web site (http://www.siemens.com/automation/).

Contact your Siemens distributor or sales office for assistance in answering any technical questions, for training, or for ordering S7 products. Because your sales representatives are technically trained and have the most specific knowledge about your operations, process and industry, as well as about the individual Siemens products that you are using, they can provide the fastest and most efficient answers to any problems you might encounter.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial security measures that may be implemented, please visit (http://www.siemens.com/industrialsecurity).

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customers' exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed visit (http://www.siemens.com/industrialsecurity).

Table of contents

Preface **3**

1 Product overview **19**

1.1 S7-200 SMART CPU 19

1.2 New features 22

1.3 S7-200 SMART expansion modules 23

1.4 HMI devices for S7-200 SMART 24

1.5 Communications options 25

1.6 Programming software 26

2 Getting started **27**

2.1 Connecting to the CPU 27
2.1.1 Configuring the CPU for communication 28
2.1.1.1 Overview 28
2.1.1.2 Establishing the Ethernet hardware communication connection 29
2.1.1.3 Setting up Ethernet communication with the CPU 30
2.1.1.4 Establishing the RS485 hardware communication connection 32
2.1.1.5 Setting up RS485 communication with the CPU 32

2.2 Creating the sample program 35
2.2.1 Network 1: Starting the timer 36
2.2.2 Network 2: Turning the output on 37
2.2.3 Network 3: Resetting the timer 38
2.2.4 Setting the CPU type and version for your project 39
2.2.5 Saving the sample project 39

2.3 Downloading the sample program 40

2.4 Changing the operating mode of the CPU 41

3 Installation **43**

3.1 Guidelines for installing S7-200 SMART devices 43

3.2 Power budget 45

3.3 Installation and removal procedures 46
3.3.1 Mounting dimensions for the S7-200 SMART devices 46
3.3.2 Installing and removing the CPU 47
3.3.3 Installing and removing a signal board or battery board 50
3.3.4 Removing and reinstalling the terminal block connector 52
3.3.5 Installing and removing an expansion module 53
3.3.6 Installing and removing the expansion cable 55

3.4 Wiring guidelines 56

4 PLC concepts **63**

4.1 Execution of the control logic 63

4.1.1 Reading the inputs and writing to the outputs ... 64
4.1.2 Immediately reading or writing the I/O ... 65
4.1.3 Executing the user program ... 65

4.2 Accessing data ... 67
4.2.1 Accessing memory areas ... 68
4.2.2 Format for Real numbers ... 74
4.2.3 Format for strings ... 75
4.2.4 Assigning a constant value for instructions ... 75
4.2.5 Addressing the local and expansion I/O ... 76
4.2.6 Using pointers for indirect addressing ... 76
4.2.7 Pointer examples ... 79

4.3 Saving and restoring data ... 80
4.3.1 Downloading project components ... 80
4.3.2 Uploading project components ... 83
4.3.3 Types of storage ... 84
4.3.4 Using a memory card ... 85
4.3.5 Inserting a memory card in a standard CPU ... 87
4.3.6 Transferring your program with a memory card ... 87
4.3.7 Restoring data after power on ... 91

4.4 Changing the operating mode of the CPU ... 92

4.5 Status LEDs ... 92

5 Programming concepts ... 95

5.1 Guidelines for designing a PLC system ... 95

5.2 Elements of the user program ... 96

5.3 Creating your user program ... 98
5.3.1 STEP 7-Micro/WIN SMART compatibility ... 98
5.3.2 Earlier versions of STEP 7-Micro/WIN projects ... 99
5.3.3 Using STEP 7-Micro/WIN SMART user interface ... 101
5.3.4 Using STEP 7-Micro/WIN SMART to create your programs ... 102
5.3.5 Using wizards to help you create your control program ... 103
5.3.6 Features of the LAD editor ... 104
5.3.7 Features of the FBD editor ... 105
5.3.8 Features of the STL editor ... 105

5.4 Data block (DB) editor ... 106

5.5 Symbol table ... 108

5.6 Variable table ... 112

5.7 PLC error reaction ... 116
5.7.1 System Information ... 118
5.7.1.1 System ... 118
5.7.1.2 CPU ... 119
5.7.1.3 PROFINET device ... 121
5.7.2 Event Log ... 122
5.7.3 PROFINET Alarm ... 122
5.7.4 Scan Rates ... 123
5.7.5 Non-fatal errors and I/O errors ... 123
5.7.6 Fatal errors ... 124

5.8 Program edit in RUN mode 125
5.9 Features for debugging your program 128
6 PLC device configuration 129
6.1 Configuring the operation of the PLC system 129
6.1.1 System block 129
6.1.2 Configuring communication 131
6.1.3 Configuring the digital inputs 133
6.1.4 Configuring the digital outputs 136
6.1.5 Configuring the retentive ranges 137
6.1.6 Configuring system security 138
6.1.7 Configuring the startup options 142
6.1.8 Configuring the analog inputs 143
6.1.9 Reference to the analog inputs technical specifications 145
6.1.10 Configuring the analog outputs 146
6.1.11 Reference to the analog outputs technical specifications 147
6.1.12 Configuring the RTD analog inputs 147
6.1.13 Configuring the TC analog inputs 152
6.1.14 Configuring the RS485/RS232 CM01 communications signal board 155
6.1.15 Configuring the BA01 battery signal board 156
6.1.16 Clearing PLC memory 158
6.1.17 Creating a reset-to-factory-defaults memory card 160
6.2 High-speed I/O 161
7 Program instructions 163
7.1 Bit logic 163
7.1.1 Standard inputs 163
7.1.2 Immediate inputs 165
7.1.3 Logic stack overview 166
7.1.4 STL logic stack instructions 168
7.1.5 NOT 170
7.1.6 Positive and negative transition detectors 171
7.1.7 Coils: output and output immediate instructions 172
7.1.8 Set, reset, set immediate, and reset immediate functions 173
7.1.9 Set and reset dominant bistable 174
7.1.10 NOP (No operation) instruction 175
7.1.11 Bit logic input examples 176
7.1.12 Bit logic output examples 177
7.2 Clock 178
7.2.1 Read and set real-time clock 178
7.2.2 Read and set real-time clock extended 181
7.3 Communication 184
7.3.1 GET and PUT (Ethernet) 184
7.3.2 Transmit and receive (Freeport on RS485/RS232) 191
7.3.3 Get port address and set port address (PPI protocol on RS485/RS232) 204
7.3.4 Get IP address and set IP address (Ethernet) 205
7.3.5 Open user communication 208
7.3.5.1 OUC instructions 208
7.3.5.2 OUC instruction error codes 219
7.4 Compare 220

7.4.1 Compare number values ... 220
7.4.2 Compare character strings ... 224

7.5 Convert ... 226
7.5.1 Standard conversion instructions ... 226
7.5.2 ASCII character array conversion ... 229
7.5.3 Number value to ASCII string conversion ... 235
7.5.4 ASCII sub-string to number value conversion ... 239
7.5.5 Encode and decode ... 242

7.6 Counters ... 244
7.6.1 Counter instructions ... 244
7.6.2 High-speed counter instructions ... 248
7.6.3 High-speed counter summary ... 251
7.6.4 Noise reduction for high-speed inputs ... 252
7.6.5 High-speed counter programming ... 254
7.6.6 Example initialization sequences for high-speed counters ... 265

7.7 Pulse output ... 273
7.7.1 Pulse output instruction (PLS) ... 273
7.7.2 Pulse train output (PTO) ... 275
7.7.3 Pulse width modulation (PWM) ... 277
7.7.4 Using SM locations to configure and control the PTO/PWM operation ... 278
7.7.5 Calculating the profile table values ... 281

7.8 Math ... 284
7.8.1 Add, subtract, multiply, and divide ... 284
7.8.2 Multiply integer to double integer and divide integer with remainder ... 288
7.8.3 Trigonometry, natural logarithm/exponential, and square root ... 290
7.8.4 Increment and decrement ... 292

7.9 PID ... 293
7.9.1 Using the PID wizard ... 295
7.9.2 PID algorithm ... 300
7.9.3 Converting and normalizing the loop inputs ... 303
7.9.4 Converting the loop output to a scaled integer value ... 304
7.9.5 Forward- or reverse-acting loops ... 304

7.10 Interrupt ... 307
7.10.1 Interrupt instructions ... 307
7.10.2 Interrupt routine overview and CPU model event support ... 309
7.10.3 Interrupt programming guidelines ... 310
7.10.4 Types of interrupt events that the S7-200 SMART CPU supports ... 312
7.10.5 Interrupt priority, queuing, and example program ... 313

7.11 Logical operations ... 319
7.11.1 Invert ... 319
7.11.2 AND, OR, and exclusive OR ... 320

7.12 Move ... 322
7.12.1 Move byte, word, double word, or real ... 322
7.12.2 Block move ... 323
7.12.3 Swap bytes ... 324
7.12.4 Move byte immediate (read and write) ... 325

7.13 Program control ... 326
7.13.1 FOR-NEXT loop ... 326

7.13.2 JMP (jump to label) 327
7.13.3 SCR (sequence control relay) 329
7.13.4 END, STOP, and WDR (watchdog timer reset) 337
7.13.5 GET_ERROR (Get non-fatal error code) 338

7.14 Shift and rotate 339
7.14.1 Shift and rotate 339
7.14.2 Shift register bit 342

7.15 String 344
7.15.1 String (Get length, copy, and concatenate) 344
7.15.2 Copy substring from string 346
7.15.3 Find string and first character within string 347

7.16 Table 350
7.16.1 Add to table 350
7.16.2 First-in-first-out and last-in-first-out 351
7.16.3 Memory fill 353
7.16.4 Table find 354

7.17 Timer 358
7.17.1 Timer instructions 358
7.17.2 Timer programming tips and examples 360
7.17.3 Interval timers 367

7.18 Subroutine 368
7.18.1 CALL (subroutine) and RET (conditional return) 368

7.19 PROFINET 374
7.19.1 Features of the programming instruction "PROFINET" 374
7.19.2 Read and Write data record 374
7.19.2.1 Input and output interface of RDREC and WRREC instruction 374
7.19.2.2 System-defined error code of the instructions RDREC and WRREC 377
7.19.3 Read and Write multiple bytes between physical PROFINET and memory address 378
7.19.3.1 Input and output interface of BLKMOV_BIR and BLKMOV_BIW 378
7.19.3.2 Error code of the instructions BLKMOV_BIR and BLKMOV_BIW 379

8 Communication 381

8.1 CPU communication connections 382

8.2 CPU communication ports 383

8.3 HMIs and communication drivers 384

8.4 Ethernet 385
8.4.1 Overview 385
8.4.2 Local/partner connection 386
8.4.3 Sample Ethernet network configurations 386
8.4.4 Assigning Internet Protocol (IP) addresses 387
8.4.4.1 Assigning IP addresses to programming and network devices 387
8.4.4.2 Configuring or changing an IP address for a CPU or device in your project 389
8.4.4.3 Searching for CPUs and devices on your Ethernet network 397
8.4.5 Locating the Ethernet (MAC) address on the CPU 398
8.4.6 HMI-to-CPU communication 399
8.4.7 Open user communication 401
8.4.7.1 Protocols 401
8.4.7.2 Connections 402

8.4.7.3 Ports and TSAPs ... 402
8.4.8 PROFINET ... 404
8.4.8.1 Introduction ... 404
8.4.8.2 I-Device ... 405
8.4.8.3 Device database file in XML: GSDML ... 407
8.4.8.4 GSDML Management ... 408
8.4.8.5 PROFINET device naming ... 410
8.4.8.6 LED status indicators for PROFINET network ... 413
8.4.8.7 Configuring CPU as a controller ... 414
8.4.8.8 Configuring CPU as an I-Device without lower-level PROFINET IO system ... 421
8.4.8.9 Configuring an I-Device project with lower-level PROFINET IO system ... 424
8.4.8.10 Example: Configuring a PROFINET network ... 426

8.5 PROFIBUS ... 442
8.5.1 EM DP01 PROFIBUS DP module ... 444
8.5.1.1 Distributed Peripheral (DP) standard communications ... 444
8.5.1.2 Using the EM DP01 to connect an S7-200 SMART as a DP device ... 444
8.5.1.3 Configuring the EM DP01 ... 446
8.5.1.4 Data consistency ... 447
8.5.1.5 Supported configurations ... 448
8.5.1.6 Installing the EM DP01 GSD file ... 448
8.5.1.7 Configuring the EM DP01 I/O ... 450
8.5.1.8 Example of V memory and I/O address area ... 453
8.5.1.9 User program considerations ... 454
8.5.1.10 LED status indicators for the EM DP01 PROFIBUS DP ... 456
8.5.1.11 Using HMIs and S7-CPUs with the EM DP01 ... 457
8.5.1.12 Device database file: GSD ... 458
8.5.1.13 PROFIBUS DP communications to a CPU example program ... 462
8.5.1.14 Reference to the EM DP01 PROFIBUS DP module technical specifications ... 464

8.6 RS485 ... 464
8.6.1 PPI protocol ... 465
8.6.2 Baud rate and network address ... 466
8.6.2.1 Definition of baud rate and network address ... 466
8.6.2.2 Setting the baud rate and network address for the S7-200 SMART CPU ... 466
8.6.3 Sample RS485 network configurations ... 468
8.6.3.1 Single-master PPI networks ... 468
8.6.3.2 Multi-master and multi-slave PPI networks ... 469
8.6.4 Assigning RS485 addresses ... 470
8.6.4.1 Configuring or changing an RS485 address for a CPU or device in your project ... 470
8.6.4.2 Searching for CPUs and devices on your RS485 network ... 474
8.6.5 Building your network ... 475
8.6.5.1 General guidelines ... 475
8.6.5.2 Determining the distances, transmission rates, and cable lengths for your network ... 475
8.6.5.3 Repeaters on the network ... 476
8.6.5.4 Specifications for RS485 cable ... 477
8.6.5.5 Connector pin assignments ... 477
8.6.5.6 Biasing and terminating the network cable ... 478
8.6.5.7 Biasing and terminating the CM01 signal board ... 479
8.6.5.8 Using HMI devices on your RS485 network ... 480
8.6.6 Freeport mode ... 481
8.6.6.1 Creating user-defined protocols with Freeport mode ... 481
8.6.6.2 Using the RS232/PPI Multi-Master cable and Freeport mode with RS232 devices ... 483

8.7 RS232 485

9 Libraries 487

9.1 Library types (Siemens and user-defined) 487

9.2 Overview of Modbus communication 489
9.2.1 Modbus addressing 489
9.2.2 Modbus read and write functions 492

9.3 Modbus RTU library 492
9.3.1 Modbus communication overview 492
9.3.1.1 Modbus RTU library features 492
9.3.1.2 Requirements for using Modbus instructions 493
9.3.1.3 Initialization and execution time for Modbus protocol 494
9.3.2 Modbus RTU master 495
9.3.2.1 Using the Modbus RTU master instructions 495
9.3.2.2 MBUS_CTRL / MB_CTRL2 instruction (initialize master) 496
9.3.2.3 MBUS_MSG / MB_MSG2 instruction 498
9.3.2.4 Modbus RTU master execution error codes 501
9.3.3 Modbus RTU slave 502
9.3.3.1 Using the Modbus RTU slave instructions 502
9.3.3.2 MBUS_INIT instruction (initialize slave) 504
9.3.3.3 MBUS_SLAVE instruction 505
9.3.3.4 Modbus RTU slave execution error codes 506
9.3.4 Modbus RTU master example program 507
9.3.5 Modbus RTU advanced user information 509

9.4 Modbus TCP library 511
9.4.1 Modbus TCP library features 511
9.4.2 Modbus TCP client 513
9.4.2.1 MBUS_CLIENT instruction 513
9.4.2.2 Modbus TCP client execution error codes 516
9.4.3 Modbus TCP server 517
9.4.3.1 MBUS_SERVER instruction 517
9.4.3.2 Modbus TCP server execution error codes 519
9.4.4 Example: Modbus TCP application 520
9.4.5 Modbus TCP advanced user information 525
9.4.6 Modbus TCP general exception codes 528
9.4.7 Modbus TCP general communication exception codes 528

9.5 Open user communication library 529
9.5.1 Parameters common to the OUC library instructions 530
9.5.2 Open user communication library instructions 532
9.5.2.1 TCP_CONNECT instruction 532
9.5.2.2 ISO_CONNECT instruction 535
9.5.2.3 UDP_CONNECT instruction 538
9.5.2.4 TCP_SEND instruction 540
9.5.2.5 TCP_RECV instruction 542
9.5.2.6 UDP_SEND instruction 545
9.5.2.7 UDP_RECV instruction 548
9.5.2.8 DISCONNECT instruction 550
9.5.3 Open user communication library instruction error codes 552
9.5.4 Open user communication library example 553
9.5.4.1 Active partner (client) 554

9.5.4.2 CheckErrors subroutine ... 562
9.5.4.3 Active partner symbol table ... 563
9.5.4.4 Passive partner (server) ... 564
9.5.4.5 CheckErrors subroutine ... 570
9.5.4.6 Passive partner symbol table ... 570

9.6 PN Read Write Record library ... 571
9.6.1 PN Read Write Record features ... 571
9.6.2 Input and output interface of PN Read Write Record library ... 572
9.6.3 Definition of parameters for input signal "STATUS" ... 573
9.6.4 System_defined error code of the library PN Read Write Record ... 574

9.7 USS library ... 574
9.7.1 USS communication overview ... 574
9.7.1.1 USS protocol overview ... 574
9.7.1.2 Requirements for using the USS protocol ... 575
9.7.1.3 Calculating the time required for communicating with the drive ... 576
9.7.2 USS program instructions ... 577
9.7.2.1 Using the USS protocol instructions ... 577
9.7.2.2 USS_INIT instruction ... 578
9.7.2.3 USS_CTRL instruction ... 580
9.7.2.4 USS_RPM_x instruction ... 583
9.7.2.5 USS_WPM_x instruction ... 585
9.7.2.6 USS protocol execution error codes ... 588
9.7.2.7 USS protocol example program ... 588

9.8 SINAMICS Library ... 591
9.8.1 SINA_POS instruction ... 592
9.8.1.1 Prerequisite of using the SINA_POS instruction ... 592
9.8.1.2 Input and output interface of SINA_POS instruction ... 595
9.8.1.3 Mode selection of SINAMICS with the SINA_POS instruction ... 601
9.8.1.4 Relative positioning ... 602
9.8.1.5 Absolute positioning ... 605
9.8.1.6 Setup mode ... 609
9.8.1.7 Referencing (active referencing) ... 613
9.8.1.8 Referencing (set reference point) ... 616
9.8.1.9 Traversing blocks ... 618
9.8.1.10 Jog ... 621
9.8.1.11 Incremental jog ... 624
9.8.2 SINA_SPEED instruction ... 628
9.8.2.1 Prerequisite of using the SINA_SPEED instruction ... 628
9.8.2.2 Input and output interface of SINA_SPEED instruction ... 631
9.8.2.3 Definition of "ConfigAxis" parameters ... 633
9.8.2.4 Example of SINA_SPEED instruction ... 633
9.8.3 SINA_PARA_S instruction ... 635
9.8.3.1 Prerequisite of using the SINA_PARA_S instruction ... 635
9.8.3.2 Input and output interface of SINA_PARA_S instruction ... 637
9.8.3.3 Example of the SINA_PARA_S instruction ... 642

9.9 Creating a user-defined library of instructions ... 644

10 Debugging and troubleshooting ... 647

10.1 Debugging your program ... 647
10.1.1 Bookmark functions ... 647

10.1.2 Cross reference table 647
10.2 Displaying program status 649
10.2.1 Displaying status in the program editor 649
10.2.2 Configuring the STL status options 652
10.3 Using a status chart to monitor your program 652
10.4 Forcing specific values 654
10.5 Writing and forcing outputs in STOP mode 655
10.6 How to execute a limited number of scans 656
10.7 Hardware troubleshooting guide 658
11 PID loops and tuning 661
11.1 PID loop definition table 662
11.2 Prerequisites 665
11.3 Auto-hysteresis and auto-deviation 666
11.4 Auto-tune sequence 666
11.5 Exception conditions 668
11.6 Notes concerning PV out-of-range (result code 3) 669
11.7 PID Tune control panel 669
12 Open loop motion control 673
12.1 Using the PWM output 674
12.1.1 Configuring the PWM output 674
12.1.2 PWMx_RUN subroutine 675
12.2 Using motion control 676
12.2.1 Maximum and start/stop speeds 676
12.2.2 Entering the acceleration and deceleration times 677
12.2.3 Configuring the motion profiles 678
12.3 Features of motion control 680
12.4 Programming an Axis of Motion 682
12.5 Configuring an Axis of Motion 683
12.6 Subroutines created by the Motion wizard for the Axis of Motion 695
12.6.1 Guidelines for using the Motion subroutines 696
12.6.2 AXISx_CTRL subroutine 697
12.6.3 AXISx_MAN subroutine 698
12.6.4 AXISx_GOTO subroutine 700
12.6.5 AXISx_RUN subroutine 701
12.6.6 AXISx_RSEEK subroutine 702
12.6.7 AXISx_LDOFF subroutine 703
12.6.8 AXISx_LDPOS subroutine 704
12.6.9 AXISx_SRATE subroutine 705
12.6.10 AXISx_DIS subroutine 706
12.6.11 AXISx_CFG subroutine 706
12.6.12 AXISx_CACHE subroutine 707
12.6.13 AXISx_RDPOS subroutine 708

12.6.14 AXISx_ABSPOS subroutine ... 709
12.7 Using the AXISx_ABSPOS subroutine to read the absolute position from a SINAMICS servo drive ... 710
12.7.1 AXISx_ABSPOS and AXISx_LDPOS subroutines usage examples ... 711
12.7.2 Interconnections ... 712
12.7.3 Commissioning ... 713
12.7.3.1 Control mode ... 713
12.7.3.2 Setpoint pulse input channel ... 713
12.7.3.3 Setpoint pulse train input format ... 713
12.7.3.4 Common engineering units basis ... 713
12.7.4 Important facts to know ... 716
12.8 Axis of Motion example programs ... 717
12.8.1 Axis of Motion simple relative move (cut-to-length application) example ... 717
12.8.2 Axis of Motion AXISx_CTRL, AXISx_RUN, AXISx_SEEK, and AXISx_MAN example ... 719
12.9 Monitoring the Axis of Motion ... 723
12.9.1 Displaying and controlling the operation of the Axis of Motion ... 724
12.9.2 Displaying and modifying the configuration of the Axis of Motion ... 730
12.9.3 Displaying the profile configuration for the Axis of Motion ... 730
12.9.4 Error codes for the Axis of Motion (WORD at SMW620, SMW670, or SMW720) ... 731
12.9.5 Error codes for the Motion instruction (seven LS bits of SMB634, SMB684, or SMB734) ... 732
12.10 Advanced topics ... 734
12.10.1 Understanding the configuration/profile table for the Axis of Motion ... 734
12.10.2 Special memory (SM) locations for the Axis of Motion ... 742
12.11 Understanding the RP Seek modes of the Axis of Motion ... 745
12.11.1 Selecting the work zone location to eliminate backlash ... 750
A Technical specifications ... 751
A.1 General specifications ... 751
A.1.1 General technical specifications ... 751
A.2 S7-200 SMART CPUs ... 756
A.2.1 CPU ST20, CPU SR20, and CPU CR20s ... 756
A.2.1.1 General specifications and features ... 756
A.2.1.2 Digital inputs and outputs ... 761
A.2.1.3 Wiring diagrams ... 764
A.2.2 CPU ST30, CPU SR30, and CPU CR30s ... 767
A.2.2.1 General specifications and features ... 767
A.2.2.2 Digital inputs and outputs ... 772
A.2.2.3 Wiring diagrams ... 775
A.2.3 CPU ST40, CPU SR40, CPU CR40s, and CPU CR40 ... 778
A.2.3.1 General specifications and features ... 778
A.2.3.2 Digital inputs and outputs ... 783
A.2.3.3 Wiring diagrams ... 786
A.2.4 CPU ST60, CPU SR60, CPU CR60s, and CPU CR60 ... 789
A.2.4.1 General specifications and features ... 789
A.2.4.2 Digital inputs and outputs ... 794
A.2.4.3 Wiring diagrams ... 797
A.2.5 Wiring diagrams for sink and source input, and relay output ... 800
A.3 Digital inputs and outputs expansion modules (EMs) ... 801

A.3.1 EM DE08 and EM DE16 digital input specifications 801
A.3.2 EM DT08, EM DR08, EM QR16, and EM QT16 digital output specifications 803
A.3.3 EM DT16, EM DR16, EM DT32, and EM DR32 digital input/output specifications 807

A.4 Analog inputs and outputs expansion modules (EMs) 813
A.4.1 EM AE04 and EM AE08 analog input specifications 813
A.4.2 EM AQ02 and EM AQ04 analog output module specifications 816
A.4.3 EM AM03 and EM AM06 analog input/output module specifications 818
A.4.4 Step response of the analog inputs 822
A.4.5 Sample time and update times for the analog inputs 822
A.4.6 Measurement ranges of the analog inputs for voltage and current (SB and EM) 823
A.4.7 Measurement ranges of the analog outputs for voltage and current (SB and EM) 824

A.5 Thermocouple and RTD expansion modules (EMs) 825
A.5.1 Thermocouple expansion modules (EMs) 825
A.5.1.1 EM AT04 thermocouple specifications 825
A.5.2 RTD expansion modules (EMs) 830

A.6 Digital signal boards 835
A.6.1 SB DT04 digital input/output specifications 835

A.7 Analog signal boards 838
A.7.1 SB AE01 analog input specifications 838
A.7.2 SB AQ01 analog output specifications 840

A.8 RS485/RS232 signal boards 842
A.8.1 SB RS485/RS232 specifications 842

A.9 Battery board signal boards (SBs) 844
A.9.1 SB BA01 Battery board 844

A.10 EM DP01 PROFIBUS DP module 845
A.10.1 S7-200 SMART CPUs that support the EM DP01 PROFIBUS DP module 847
A.10.2 Connector pin assignments for EM DP01 847
A.10.3 EM DP01 PROFIBUS DP module wiring diagram 848

A.11 S7-200 SMART cables 849
A.11.1 S7-200 SMART I/O expansion cable 849
A.11.2 RS-232/PPI Multi-Master Cable and USB/PPI Multi-Master Cable 850
A.11.2.1 Overview 850
A.11.2.2 RS-232/PPI Multi-Master Cable 851
A.11.2.3 USB/PPI Multi-Master Cable 853

B Calculating a power budget 855

B.1 Power budget 855

B.2 Calculating a sample power requirement 857

B.3 Calculating your power requirement 858

C Error codes 859

C.1 Timestamp mismatch 859

C.2 PLC non-fatal error codes 860

C.3 PLC non-fatal error SM flags 862

C.4 PLC fatal error codes 863

D Special memory (SM) and system symbol names 865

D.1 SM (Special Memory) overview 865

D.2 SMB0: System status 867

D.3 SMB1: Instruction execution status 868

D.4 SMB2: Freeport receive character 869

D.5 SMB3: Freeport character error 870

D.6 SMB4: Interrupt queue overflow, run-time program error, interrupts enabled, freeport transmitter idle, and value forced 870

D.7 SMB5: I/O error status 871

D.8 SMB6-SMB7: CPU ID, error status, and digital I/O points 871

D.9 SMB8-SMB19: I/O module ID and errors 872

D.10 SMW22-SMW26: Scan times 873

D.11 SMB28-SMB29: Signal board ID and errors 873

D.12 SMB30: (Port 0) and SMB130: (Port 1) 873

D.13 SMB34-SMB35: Time intervals for timed interrupts 874

D.14 SMB36-SMB45 (HSC0), SMB46-SMB55 (HSC1), SMB56-SM65 (HSC2), SMB136-SMB145 (HSC3), SMB146-SMB155 (HSC4), SMB156-SMB165 (HSC5): high-speed counters 875

D.15 SMB66-SMB85 (PTO0/PWM0, PTO1/PWM1), SMB166-SMB169 (PTO0), SMB176-SMB179 (PTO1), and SMB566-SMB579 (PTO2/PWM2): high-speed outputs 880

D.16 SMB86-SMB94 and SMB186-SMB194: Receive message control 883

D.17 SMW98: Expansion I/O bus communication errors 885

D.18 SMW100-SMW114 System alarms 886

D.19 SMB130: Freeport control for port 1 (See SMB30) 887

D.20 SMB146-SMB155 (HSC4) and SMB156-SMB165 (HSC5) 887

D.21 SMB186-SMB194: Receive message control (See SMB86-SMB94) 887

D.22 SMB480-SMB515: Data log status 887

D.23 SMB600-SMB749: Axis (0, 1, and 2) open loop motion control 888

D.24 SMB650-SMB699: Axis 1 open loop motion control (See SMB600-SMB740) 889

D.25 SMB700-SMB749: Axis 2 open loop motion control (See SMB600-SMB740) 889

D.26 SMB1000-SMB1049: CPU hardware/firmware ID 889

D.27 SMB1050-SMB1099: SB (signal board) hardware/firmware ID 890

D.28 SMB1100-SMB1399: EM (expansion module) hardware/firmware ID 890

D.29 SMB1400-SMB1699: EM (expansion module) module-specific data 893

D.30 SMB1800-SMB1999: PROFINET device status 893

E References 895

E.1 Often-used special memory bits 895
E.2 Interrupt events in priority order 896
E.3 High-speed counter summary 897
E.4 STL instructions 898
E.5 Memory ranges and features 904
F Ordering information 909
F.1 CPU modules 909
F.2 Expansion modules (EMs) and signal boards (SBs) 909
F.3 Programming software 910
F.4 Communication 910
F.5 Spare parts and other hardware 911
F.6 Human Machine Interface devices 913
Index 915

Product overview

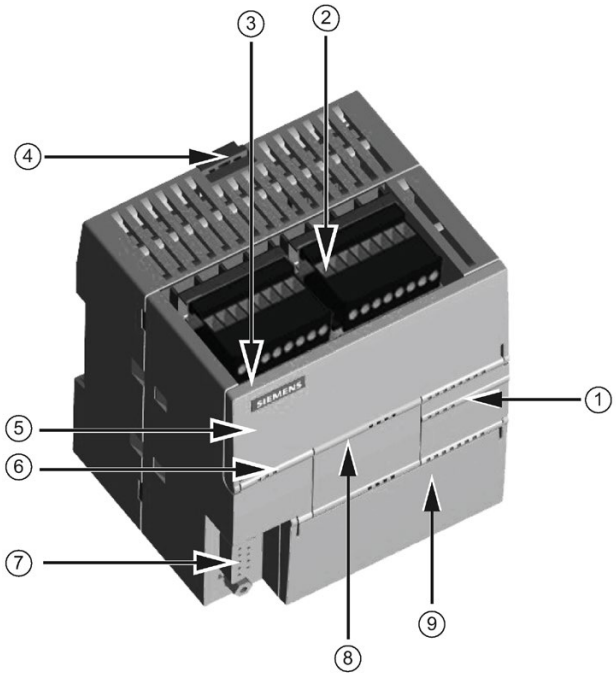
1

The S7-200 SMART series of micro-programmable logic controllers (Micro PLCs) can control a wide variety of devices to support your automation needs.

The CPU monitors inputs and changes outputs as controlled by the user program, which can include Boolean logic, counting, timing, complex math operations, and communications with other intelligent devices. The compact design, flexible configuration, and powerful instruction set combine to make the S7-200 SMART a perfect solution for controlling a wide variety of applications.

1.1 S7-200 SMART CPU

The CPU combines a microprocessor, an integrated power supply, input circuits, and output circuits in a compact housing to create a powerful Micro PLC. After you have downloaded your program, the CPU contains the logic required to monitor and control the input and output devices in your application.



① LEDs for the I/O
② Terminal connectors
③ Ethernet communication port
④ Clip for installation on a standard (DIN) rail
⑤ Ethernet status LEDs (under door): LINK, Rx/Tx
⑥ Status LEDs: RUN, STOP and ERROR
⑦ RS485 Communication port
⑧ Optional signal board (Standard models only)
⑨ Memory card reader (under door) (Standard models only)

The CPU provides different models with a diversity of features and capabilities that help you create effective solutions for your varied applications. The different models of CPUs are shown below. For detailed information about a specific CPU, see the technical specifications (Page 756).

The S7-200 SMART CPU family includes fourteen CPU models, separated into two lines: the Compact Line and the Standard Line. The first letter of the CPU designator indicates a line, either Compact (C) or Standard (S). The second letter of the designator indicates AC power supply / relay outputs (R) or DC power supply / DC transistor (T). The number in the designator indicates the total onboard digital I/O count. The new compact models are designated by a lower case "s" character (serial port only) following the I/O count.

Note

CPU CRs and CPU CR

S7-200 SMART CPU firmware release V2.4 and V2.5 do not apply to the CPU CRs and CPU CR models.

Table 1- 1 S7-200 SMART CPUs

	SR20	ST20	CR20s	SR30	ST30	CR30s	SR40	ST40	CR40s	CR40	SR60	ST60	CR60s	CR60
Compact serial, non-expandable			X			X			X	X			X	X
Standard, expandable	X	X		X	X		X	X			X	X		
Relay output	X		X	X		X	X		X	X	X		X	X
Transistor output (DC)		X			X			X				X		
I/O points (built-in)	20	20	20	30	30	30	40	40	40	40	60	60	60	60

Table 1- 2 Compact serial, non-expandable CPUs

Features		CPU CR20s	CPU CR30s	CPU CR40s, CPU CR40	CPU CR60s, CPU CR60
Dimensions: W x H x D (mm)		90 x 100 x 81	110 x 100 x 81	125 x 100 x 81	175 x 100 x 81
User memory	Program	12 Kbytes	12 Kbytes	12 Kbytes	12 Kbytes
	User data	8 Kbytes	8 Kbytes	8 Kbytes	8 Kbytes
	Retentive	2 Kbytes max.[1]	2 Kbytes max.[1]	2 Kbytes max.[1]	2 Kbytes max.[1]
On-board digital I/O	• Inputs • Outputs	• 12 DI • 8 DQ Relay	• 18 DI • 12 DQ Relay	• 24 DI • 16 DQ Relay	36 DI 24 DQ Relay
Expansion modules		None	None	None	None
Signal board		None	None	None	None
High-speed counters (4 total)	Single phase	4 at 100 kHz	4 at 100 kHz	4 at 100 kHz	4 at 100 kHz
	A/B phase	2 at 50 kHz	2 at 50 kHz	2 at 50 kHz	2 at 50 kHz

Features	CPU CR20s	CPU CR30s	CPU CR40s, CPU CR40	CPU CR60s, CPU CR60
PID loops	8	8	8	8
Real-time clock with 7-day back-up	No	No	No	No

[1] You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive timers) to be retentive, up to the specified maximum amount.

Table 1- 3 Standard expandable CPUs

Features		CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU SR60, CPU ST60
Dimensions: W x H x D (mm)		90 x 100 x 81	110 x 100 x 81	125 x 100 x 81	175 x 100 x 81
User memory	Program	12 Kbytes	18 Kbytes	24 Kbytes	30 Kbytes
	User data	8 Kbytes	12 Kbytes	16 Kbytes	20 Kbytes
	Retentive	10 Kbytes max.[1]	10 Kbytes max.[1]	10 Kbytes max.[1]	10 Kbytes max.[1]
On-board digital I/O	• Inputs • Outputs	• 12 DI • 8 DQ	• 18 DI • 12 DQ	• 24 DI • 16 DQ	• 36 DI • 24 DQ
Expansion modules		6 max.	6 max.	6 max.	6 max.
Signal board		1	1	1	1
High-speed counters (6 total)	Single phase	4 at 200 kHz 2 at 30 kHz	5 at 200 kHz 1 at 30 kHz	4 at 200 kHz 2 at 30 kHz	4 at 200 kHz 2 at 30 kHz
	A/B phase	2 at 100 kHz 2 at 20 kHz	3 at 100 kHz 1 at 20 kHz	2 at 100 kHz 2 at 20 kHz	2 at 100 kHz 2 at 20 kHz
Pulse outputs [2]		2 at 100 kHz	3 at 100 kHz	3 at 100 kHz	3 at 100 kHz
PID loops		8	8	8	8
Real-time clock with 7-day back-up		Yes	Yes	Yes	Yes

[1] You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive timers) to be retentive, up to the specified maximum amount.

[2] The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Refer to the technical specifications (Page 751) for the power requirements of the CPU and the expansion modules. Use the worksheets in Appendix B, Calculating a power budget (Page 858) to calculate your power budget (Page 855).

1.2 New features

Only the following CPU models with firmware V2.5 or later versions support the new features described in this chapter:

Table 1- 4 CPU models affected by firmware V2.5

CPU model	Article number
CPU SR20, AC/DC/Relay	6ES7288-1SR20-0AA0
CPU ST20, DC/DC/DC	6ES7288-1ST20-0AA0
CPU SR30, AC/DC/Relay	6ES7288-1SR30-0AA0
CPU ST30, DC/DC/DC	6ES7288-1ST30-0AA0
CPU SR40, AC/DC/Relay	6ES7288-1SR40-0AA0
CPU ST40, DC/DC/DC	6ES7288-1ST40-0AA0
CPU SR60, AC/DC/Relay	6ES7288-1SR60-0AA0
CPU ST60, DC/DC/DC	6ES7288-1ST60-0AA0

STEP 7-Micro/WIN SMART V2.5 release provides the following new features:

I-Device

STEP 7-Micro/WIN SMART V2.5 and the S7-200 SMART V2.5 CPU firmware adds I-Device (Page 405)functions for PROFINET communication.

I-Device configuration

PROFINET wizard provides the function to select CPU role as I-Device (Page 421).

GSDML file export

PROFINET wizard provides the function to export GSDML file. (Page 421)

LED status for PROFINET I-Device

The LED status indicators (Page 413) display information for PROFINET I-Device.

I-Device Diagnostic

Diagnostic functions (Page 119) are available for PROFINET I-Device.

Status Chart

The Status Chart (Page 652) function is available on PROFINET I-Device.

Catalog configuration

Catalog configuration (Page 435) is a new method to configure CPU as a controller for PROFINET communication.

1.3 S7-200 SMART expansion modules

To better solve your application requirements, the S7-200 SMART family includes a wide variety of expansion modules, signal boards, and a communications module. You can use these expansion modules with the standard CPU models (SR20, ST20, SR30, ST30, SR40, ST40, SR60 or ST60) to add additional functionality to the CPU. The following table provides a list of the expansion modules that are currently available. For detailed information about a specific module, see the technical specifications (Page 751).

Table 1- 5 Expansion modules and signal boards

Type	Input only	Output only	Combination In/Out	Other
Digital expansion module	• 8 x DC In • 16 x DC In	• 8 x DC Out • 8 x Relay Out • 16 x Relay Out • 16 x DC Out	• 8 x DC In / 8 x DC Out • 8 x DC In / 8 x Relay Out • 16 x DC In / 16 x DC Out • 16 x DC In / 16 x Relay Out	
Analog expansion modules	• 4 x Analog In • 8 x Analog In • 2 x RTD In • 4 x RTD In • 4 x TC In	• 2 x Analog Out • 4 x Analog Out	• 4 x Analog In / 2 x Analog Out • 2 x Analog In / 1 x Analog Out	
Signal boards	• 1 x Analog In	• 1 x Analog Out	• 2 x DC In x 2 x DC Out	• RS485/RS232 • Battery Board

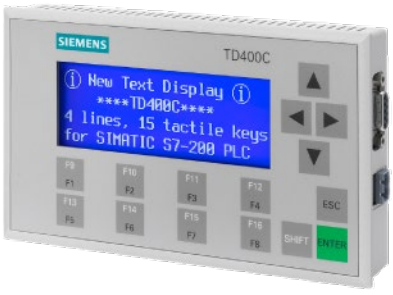
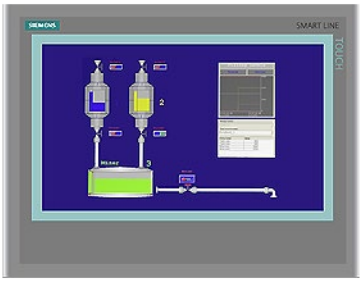
Table 1- 6 Communication expansion modules

Module	Type	Description
Communication expansion module (EM)	PROFIBUS DP SMART module	EM DP01 PROFIBUS DP

1.4 HMI devices for S7-200 SMART

The S7-200 SMART supports Comfort HMIs, SMART HMIs, Basic HMIs and Micro HMIs. The TD400C and the SMART LINE Touch Panel are shown below. Refer to "HMIs and communication drivers" (Page 384) for a list of supported devices.

Table 1- 7 HMI devices

	**Text Display unit:** The TD400C is an RS485-only display device that can be connected to the CPU. Using the Text Display wizard, you can easily program your CPU to display text messages and other data pertaining to your application. The TD400C device provides a low-cost interface to your application by allowing you to view, monitor, and change the process variables pertaining to your application.
	**SMART HMIs:** The SMART LINE Touch Panel provides operating and monitoring functions for small-scale machines and plants. Short configuration and commissioning times, their configuration in WinCC flexible (ASIA version), and a double-port Ethernet/RS485 interface form the highlights of these HMIs.

The Text Display wizard in STEP 7-Micro/WIN SMART helps you configure Text Display messages quickly and easily for the TD400C. To start the Text Display wizard, select the "Text Display" command from the "Tools" menu.

The SIMATIC Text Display (TD) User Manual can be downloaded from the Siemens customer support web site (http://www.siemens.com/automation/).

1.5 Communications options

The S7-200 SMART offers several types of communication between CPUs, programming devices, and HMIs:

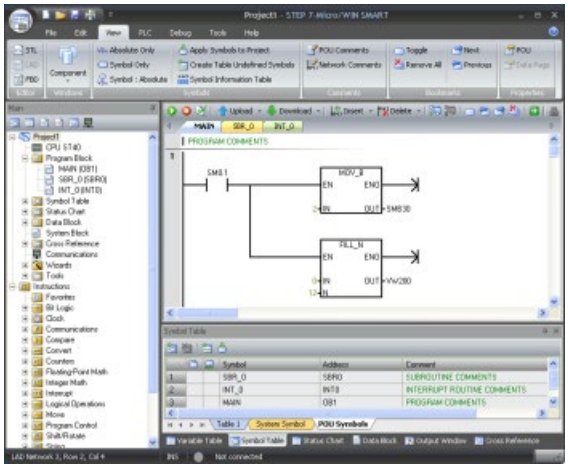
- Ethernet:
  - Exchange of data from the programming device to the CPU
  - Exchange of data between HMIs and the CPU
  - S7 peer-to-peer communication with other S7-200 SMART CPUs
  - Open User Communication (OUC) with other Ethernet-capable devices
  - PROFINET communication with PROFINET devices

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

- PROFIBUS:
  - High speed communications for distributed I/O (up to 12 Mbps)
  - One bus master connects to many I/O devices (supports 126 addressable devices).
  - Exchange of data between the master and I/O devices
  - EM DP01 module is a PROFIBUS I/O device.
- RS485:
  - Provides a STEP 7-Micro/WIN SMART connection for programming when using a USB-PPI cable
  - Supports a total of 126 addressable devices (32 devices per network segment)
  - Supports PPI (point-to-point interface) protocol
  - Exchange of data between HMIs and the CPU
  - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)
- RS232:
  - Supports a point-to-point connection to one device
  - Supports PPI protocol
  - Exchange of data between HMIs and the CPU
  - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)

1.6 Programming software



STEP7-Micro/WIN SMART provides a user-friendly environment to develop, edit, and monitor the logic needed to control your application.

At the top is a quick access toolbar for frequent tasks, followed by menus for all common functions. At the left is the project tree and navigation bar for easy access to components and instructions. The program editor and other components that you open occupy the remainder of the user interface.

STEP7-Micro/WIN SMART provides three program editors (LAD, FBD, and STL) for convenience and efficiency in developing the control program for your application.

To help you find the information you need, STEP7-Micro/WIN SMART provides an extensive online help system.

Computer requirements

STEP 7-Micro/WIN SMART runs on a personal computer. Your computer should meet the following minimum requirements:

- Operating system: Windows 7 or Windows 10 (both 32 bit and 64 bit versions)
- At least 350M bytes of free hard disk space
- Mouse (recommended)

Installing STEP 7-Micro/WIN SMART

Insert the STEP 7-Micro/WIN SMART CD into the CD-ROM drive of your computer or contact your Siemens distributor or sales office to download STEP7-Micro/WIN SMART from the customer support web site (Page 3). Installation starts automatically and prompts you through the installation process. Refer to the Readme file for more information about installing STEP 7-Micro/WIN SMART.

Note

To install STEP 7-Micro/WIN SMART on a Windows 7 or Windows 10 operating system, you must log in with Administrator privileges.

Getting started

2

STEP 7-Micro/WIN SMART makes it easy for you to program your CPU. In just a few short steps using a simple example, you can learn how to create a user program that you can download and run on your CPU.

All you need for this example is an Ethernet or USB-PPI communication cable, a CPU, and a programming device running the STEP 7-Micro/WIN SMART programming software.

2.1 Connecting to the CPU

Connecting your CPU is easy. For this example, you only need to connect power to your CPU and then connect the Ethernet or USB-PPI communication cable between your programming device and the CPU.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and no functions related to the use of Ethernet communications.

Connecting power to the CPU

WARNING

Ensure power is off prior to installing, wiring or removing devices

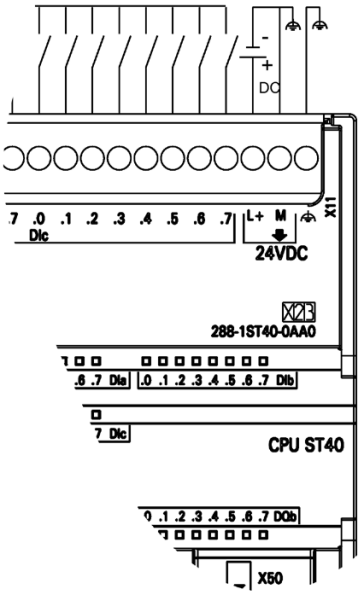
Before you install or remove any electrical device, ensure that the power to that equipment has been turned off.

Attempts to install or connect the wiring for the CPU or related equipment with power applied could cause electric shock or faulty operation of equipment. Failure to disable all power to the CPU and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.

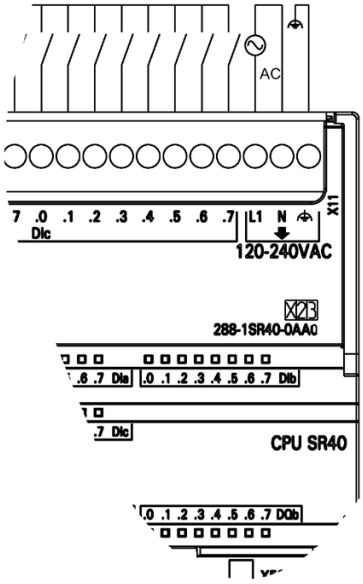
Always follow appropriate safety precautions and ensure that power to the CPU is disabled before attempting to install or remove the CPU or related equipment.

Connect the CPU to a power source. The following figure shows the wiring connections for either a DC or an AC model of the CPU.

DC installation



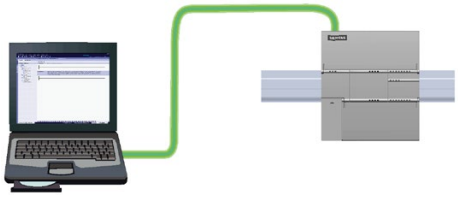
AC installation



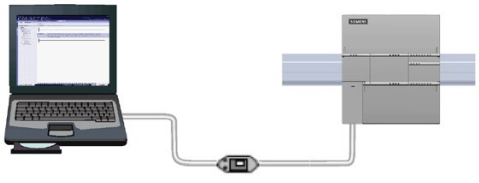
2.1.1 Configuring the CPU for communication

2.1.1.1 Overview

A CPU can communicate with a STEP 7-Micro/WIN SMART programming device on two types of communications networks:



A CPU can communicate with a STEP 7-Micro/WIN SMART programming device on an Ethernet network.



A CPU can communicate with a STEP 7-Micro/WIN SMART programming device on an RS485 network.

Consider the following when setting up Ethernet communications between a CPU and a programming device:

- Configuration/Setup: No hardware configuration is required for a single CPU. If you want multiple CPU's on the same network, then you must change the default IP addresses to new, unique IP addresses.
- No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

Note

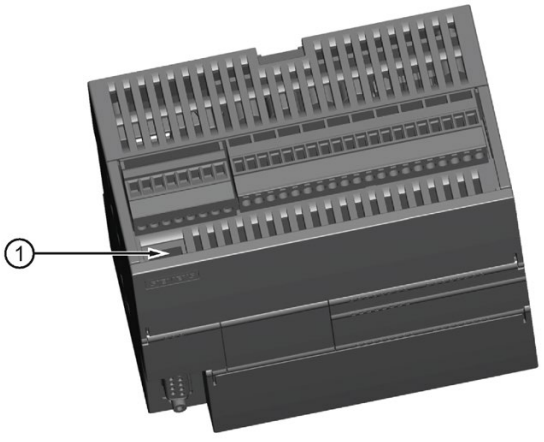
The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

2.1.1.2 Establishing the Ethernet hardware communication connection

The Ethernet interfaces establish the physical connections between a programming device and a CPU. Since Auto-Cross-Over functionality is built into the CPU, either a standard or crossover Ethernet cable can be used for the interface. An Ethernet switch is not required to connect a programming device directly to a CPU.

Follow the steps below to create the hardware connection between a programming device and a CPU:

1. Install the CPU.
2. Remove the RJ45 connection cover from the Ethernet port. Retain the cover for reuse.
3. Plug the Ethernet cable into the Ethernet port on the top left of the CPU as shown below.
4. Connect the Ethernet cable to the programming device.



① PROFINET (LAN) port

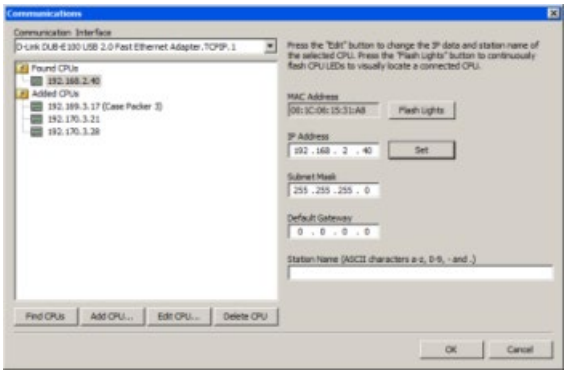
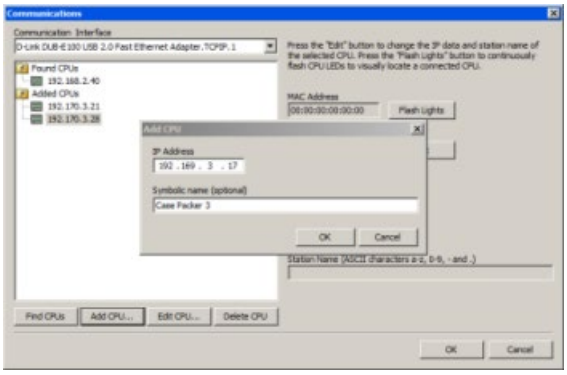
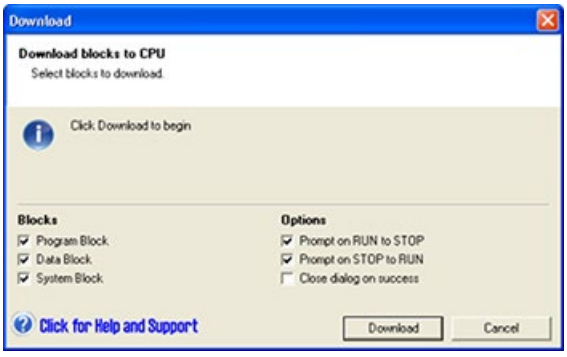
2.1.1.3 Setting up Ethernet communication with the CPU

From STEP 7-Micro/WIN SMART, use one of the following methods to display the Ethernet "Communications" dialog for configuring communication to the CPU.

- From the project tree, double-click the "Communications" node.
- Click the "Communications" button from the navigation bar.
- Select "Communications" from the "Component" drop-down list in the Windows area of the "View" menu ribbon strip.

The "Communications" dialog provides two methods of selecting the CPU to be accessed:

- Click the "Find CPUs" button to have STEP 7-Micro/WIN SMART search your local network for CPUs. The IP address of each CPU found on the network is listed under "Found CPUs".
- Click the "Add CPU" button to manually enter the access information (IP address and so forth) for a CPU that you wish to access. The IP address for each CPU, manually added with this method, is listed under "Added CPUs" and is retained.

	For "Found CPUs" (CPUs located on your local network), use the "Communications" dialog to connect with your CPU: • Select TCP/IP for your Communication Interface. • Click the "Find CPUs" button to display all operational CPUs ("Found CPUs") on the local Ethernet network. All CPUs have a default IP address. See the Note below. • Highlight a CPU, and then click "OK".
	For "Added CPUs" (CPUs on the local or remote networks), use the "Communications" dialog to connect with your CPU: • Select TCP/IP for your Communication Interface. • Click the "Add CPU" button to do one of the following: – Enter the IP address of a CPU that is accessible from the programming device, but is not on the local network. – Enter the IP address of a CPU directly that is on the local network. All CPUs have a default IP address. See the Note below. • Highlight a CPU, and then click "OK".
	After you have established communication with the CPU, you are ready to create and download the example program. To download all project components, click the Download button from the Transfer area of the File or PLC menu ribbon strip, or alternatively press the shortcut key combination CTRL+D. If STEP 7-Micro/WIN SMART does not find your CPU, check the settings for the communications parameters and repeat these steps.

Note

The CPU list will show all of the CPUs regardless of Ethernet network class and subnet.

To make a connection to your CPU, your Communication Interface (for Ethernet, a network interface card (NIC)) and the CPU must be on the same class of network and on the same subnet. You can either set up your network interface card to match the default IP address of the CPU, or you can change the IP address of the CPU to match the network class and subnet of your network interface card.

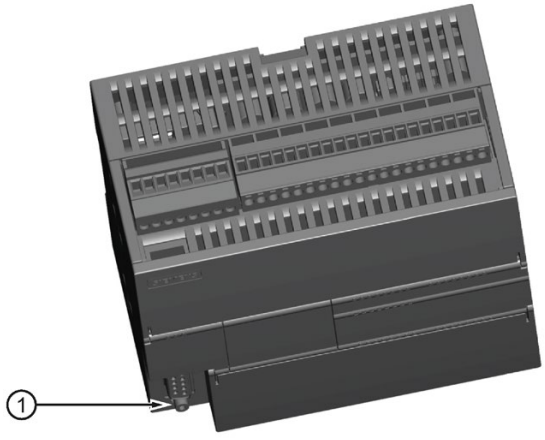
See the "Configuring or changing an IP address for a CPU or device in your project" (Page 389) for information about how to accomplish this.

2.1.1.4 Establishing the RS485 hardware communication connection

The RS485 interfaces establish the physical connections between a programming device and a CPU.

Follow the steps below to create the hardware connection between a programming device and a CPU:

1. Install the CPU.
2. Plug the USB/PPI cable into the RS485 port on the bottom left of the CPU as shown below.
3. Connect the USB/PPI cable to the programming device.



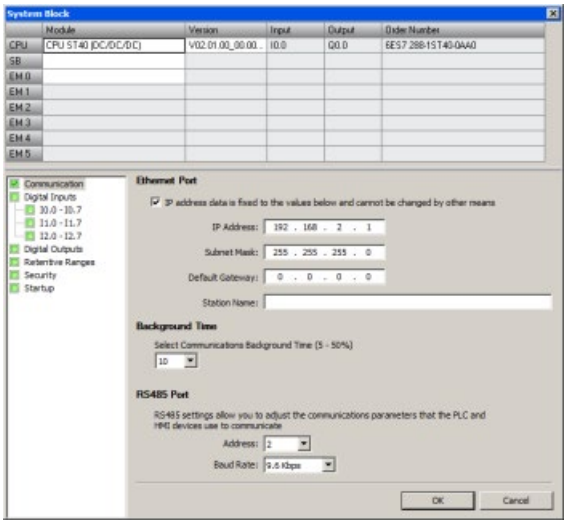
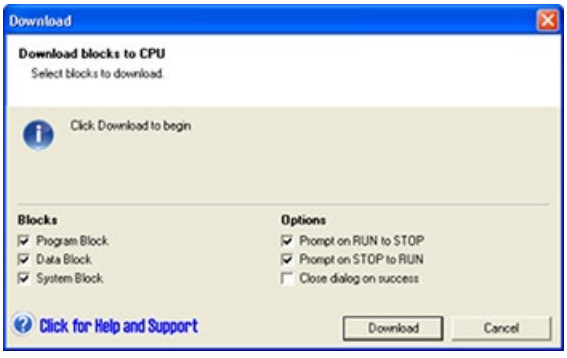
① RS485 port

2.1.1.5 Setting up RS485 communication with the CPU

RS485 network information configuration or changes done in the system block are part of the project and do not become active until you download your project to the CPU.

To access this dialog, perform one of the following:

- In the "Navigation" bar, click the "System Block" button.
- In the Project tree, select the "System Block" node, then press Enter; or double-click the "System Block" node.

	Enter or change the following access information: • RS485 port address • RS485 port baud rate
	After you have established communication with the CPU, you are ready to create and download the example program. To download all project components, click the Download button from the Transfer area of the File or PLC menu ribbon strip, or alternatively press the shortcut key combination CTRL+D. If STEP 7-Micro/WIN SMART does not find your CPU, check the settings for the communications parameters and repeat these steps.

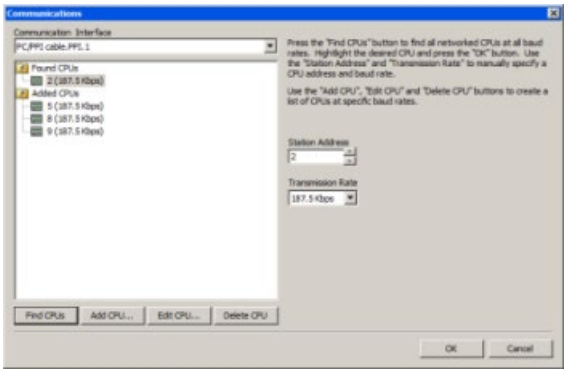
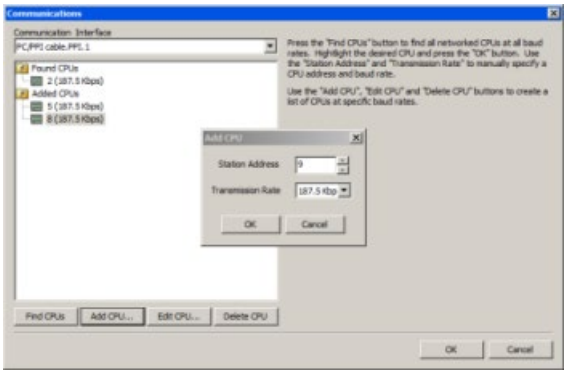
All CPUs and devices that have valid RS485 port addresses are displayed in the "Communications" dialog.

In STEP 7-Micro/WIN SMART, you can access CPUs in one of two ways:

- From the project tree, double-click the "Communications" node.
- Click the "Communications" button from the navigation bar.
- Select "Communications" from the "Component" drop-down list in the Windows area of the "View" menu ribbon strip.

The "Communications" dialog provides two methods of selecting the CPU to be accessed:

- Click the "Find CPUs" button to have STEP 7-Micro/WIN SMART search your local network for CPUs. The RS485 network address of each CPU found on the network is listed under "Found CPUs".
- Click the "Add CPU" button to manually enter the access information (RS485 network address and baud rate) for a CPU that you wish to access. The RS485 network address for each CPU, manually added with this method, is listed under "Added CPUs" and is retained.

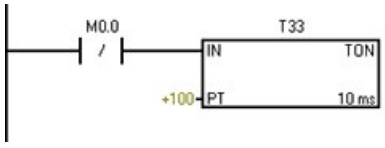
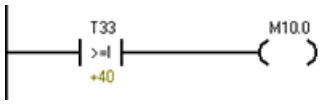
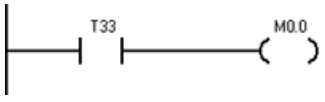
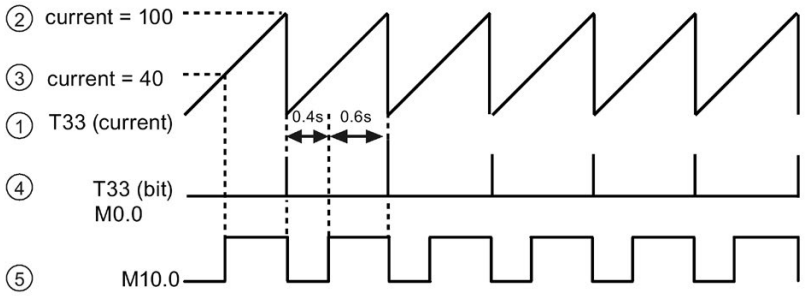
	For "Found CPUs" (CPUs located on the RS485 network), use the "Communications" dialog to connect with your CPU: • Select "PC/PPI cable.PPI.1" for your Communication Interface. • Click the "Find CPUs" button to display all operational CPUs ("Found CPUs") on the RS485 network. All CPUs default their RS485 network settings to address 2 and 9.6 Kbps. • Highlight a CPU, and then click "OK". Note: You can open multiple copies of STEP 7-Micro/WIN SMART on a computer. Be aware that when you open a second copy of STEP 7-Micro/WIN SMART or use the "Find CPUs" button in either copy, the communication connection to the CPU in your first/other copy of STEP 7-Micro/WIN SMART might be disconnected.
	For "Added CPUs" (CPUs on the RS485 network), use the "Communications" dialog to connect with your CPU: • Select "PC/PPI cable.PPI.1" for your Communication Interface. • Click the "Add CPU" button. • Enter the RS485 network address and baud rate of a CPU that you wish to access directly on the RS485 network. You can add multiple CPUs on the RS485 network. As always, STEP 7-Micro/WIN SMART communicates with one CPU at a time. All CPUs default their RS485 network settings to address 2 and 9.6 Kbps. • Highlight a CPU, and then click "OK".

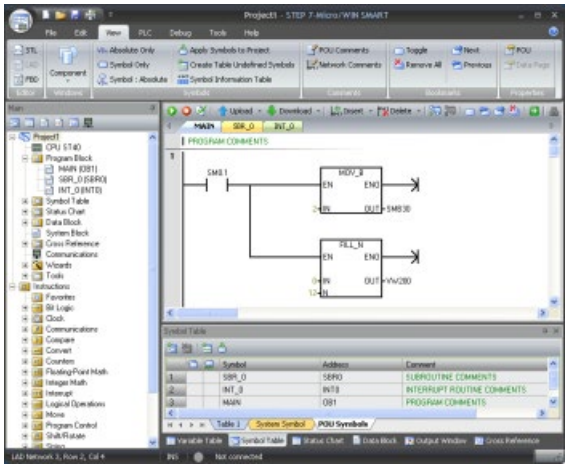
2.2 Creating the sample program

Entering this example of a control program will help you understand how easy it is to use STEP 7-Micro/WIN SMART. This program uses six instructions in three networks to create a very simple, self-starting timer that resets itself.

For this example, you use the Ladder (LAD) editor to enter the instructions for the program. The following example shows the complete program in both LAD and Statement List (STL). The description column explains the logic for each network. The timing diagram shows the operation of the program. There are no network comments in the STL program.

Table 2- 1 Sample program for getting started with STEP 7-Micro/WIN SMART

LAD/FBD	STL	Description
	**Network 1** **LDN M0.0** **TON T33, +100**	10 ms timer T33 times out after (100 x 10 ms = 1 s) M0.0 pulse is too fast to monitor with Status view.
	**Network 2** **LDW>= T33, +40** **= M10.0**	Comparison becomes true at a rate that is visible with Status view. Turn on M10.0 after (40 x 10 ms = 0.4 s) for a 40% OFF / 60% ON waveform.
	**Network 3** **LD T33** **= M0.0**	T33 (bit) pulse is too fast to monitor with Status view. Reset the timer through M0.0 after the (100 x 10 ms = 1 s) period.
		**Timing diagram:** • ① T33 (current) • ② Current = 100 • ③ Current = 40 • ④ T33 (bit) and M0.0 • ⑤ M10.0



Notice the project tree and the program editor. You use the project tree to insert instructions into the networks of the program editor by dragging and dropping the instructions from the "Instructions" portion of the Project tree to the networks.

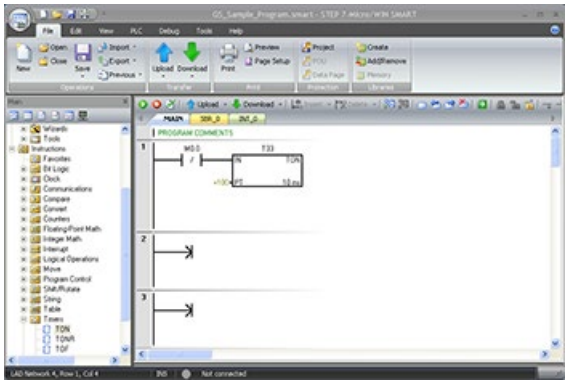
The Program Block folder in the project tree contains all of the blocks of your program.

The program editor toolbar icons provide shortcuts to PLC commands and programming operation.

After you enter and save the program, you can download the program to the CPU.

2.2.1 Network 1: Starting the timer

Network 1: Starting the timer



When M0.0 is off (0), this contact turns on and provides power flow to start the timer.

To enter the contact for M0.0:

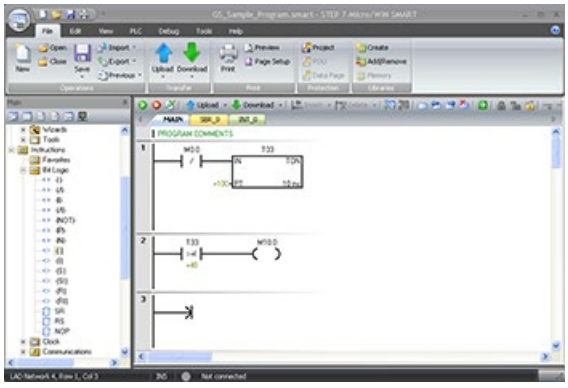
1. Either double-click the "Bit Logic" icon or click the plus sign (+) to display the bit logic instructions.
2. Select the "Normally Closed" contact.
3. Hold down the left mouse button and drag the contact onto the first network.
4. Enter the following address for the contact: M0.0
5. Press the Return key to enter the address for the contact.

To enter the timer instruction for T33:

1. Double-click the "Timers" icon to display the timer instructions.
2. Select the "TON" (on-delay timer) instruction.
3. Hold down the left mouse button and drag the timer onto the first network.
4. Enter the following timer number for the timer: T33
5. Press the Return key to enter the timer number and to move the focus to the preset time (PT) parameter.
6. Enter the following value for the preset time: +100.
7. Press the Return key to enter the value.

2.2.2 Network 2: Turning the output on

Network 2: Turning the output on



When the timer value for T33 is greater than or equal to 40 (40 times 10 milliseconds, or 0.4 seconds), the contact provides power flow to turn on output M10.0 of the CPU.

To enter the Compare instruction:

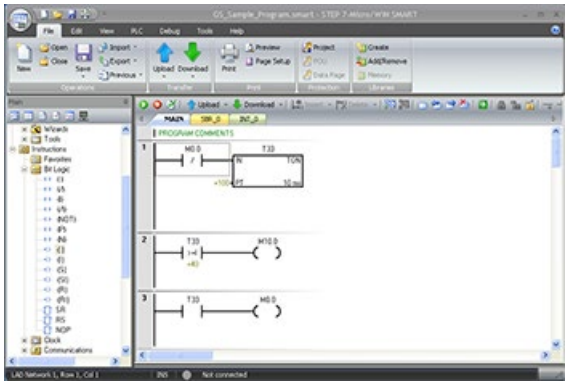
1. Double-click the Compare icon to display the compare instructions. Select the ">=I" instruction (greater-than-or-equal-to-integer).
2. Hold down the left mouse button and drag the compare instruction onto the second network.
3. Click "???" above the contact and enter the address for the timer value: T33
4. Press the Return key to enter the timer number and to move the focus to the other value to be compared with the timer value.
5. Enter the following value to be compared with the timer value: +40
6. Press the Return key to enter the value.

To enter the instruction for turning on output M10.0:

1. Double-click the Bit Logic icon to display the bit logic instructions and select the output coil.
2. Hold down the left mouse button and drag the coil onto the second network.
3. Click "???" above the coil and enter the following address: M10.0
4. Press the Return key to enter the address for the coil.

2.2.3 Network 3: Resetting the timer

Network 3: Resetting the timer



When the timer reaches the preset value (100) and turns the timer bit on, the contact for T33 turns on. Power flow from this contact turns on the M0.0 memory location. Because the timer is enabled by a Normally Closed contact for M0.0, changing the state of M0.0 from off (0) to on (1) resets the timer.

To enter the contact for the timer bit of T33:

1. Select the "Normally Open" contact from the bit logic instructions.
2. Hold down the left mouse button and drag the contact onto the third network.
3. Click "???" above the contact and enter the address of the timer bit: T33
4. Press the Return key to enter the address for the contact.

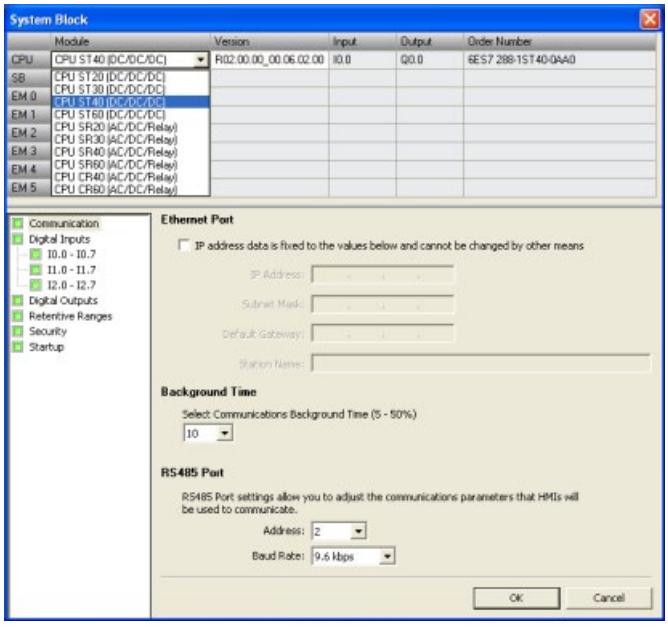
To enter the coil for turning on M0.0:

1. Select the output coil from the bit logic instructions.
2. Hold down the left mouse button and drag the output coil onto the third network.
3. Click "???" above the coil and enter the following address: M0.0
4. Press the Return key to enter the address for the coil.

2.2.4 Setting the CPU type and version for your project

Configure your project for the CPU and version matching your physical CPU. If the project is not configured for the correct CPU and CPU version, then the download could fail or the program may not run.

To select your CPU, click the "CPU" field under the "Module" column to display the dropdown list button, and select your CPU from the dropdown list. Using the same procedure, select your CPU version in the "Version" column.



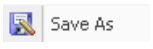
2.2.5 Saving the sample project

Saving the sample project

After entering the three networks of instructions, you have finished entering the program. When you save the program, you create a project that includes the CPU type and other parameters. To save the project in a file name and location that you specify:

1. Click the down arrow under the Save button from the Operations area of the File menu ribbon strip to display the Save As button.

2. Click the Save As button and provide a filename for saving your project.



3. Enter a name for the project in the "Save As" dialog.

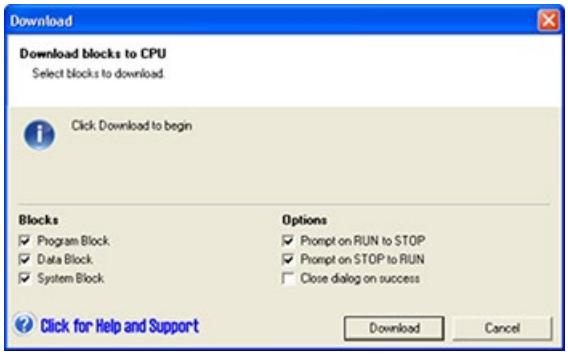
4. Browse to a location where you want to save your project.
5. Click "Save" to save the project.

After saving the project, you can download the program to the CPU.

2.3 Downloading the sample program

First, ensure that your network hardware and PLC connector cable for either Ethernet (Page 29) (standard CPUs only) or RS485 (Page 32) communications is working, and that PLC communication is operating properly.

To download all project components, click the "Download" button from the "Transfer" area of the File or PLC menu ribbon strip, or alternatively press the shortcut key combination "CTRL+D".



Click the Download dialog "Download" button.

STEP 7-Micro/WIN SMART copies the complete program or program components that you selected to the CPU.

If your CPU is in RUN mode, a dialog prompts you to place the CPU in STOP mode. Clicking "Yes" sets the CPU to STOP mode.

Note

Each project is associated with a CPU type. If the project type does not match the CPU to which you are connected, STEP 7-Micro/WIN SMART indicates a mismatch and prompts you to take an action.

See also

Hardware troubleshooting guide (Page 658)

PLC fatal error codes (Page 863)

Changing the operating mode of the CPU (Page 41)

2.4 Changing the operating mode of the CPU

The CPU has two modes of operation: STOP mode and RUN mode. The status LEDs on the front of the CPU indicates the current mode of operation. In STOP mode, the CPU is not executing the program, and you can download program blocks. In RUN mode, the CPU is executing the program; however, you can download program blocks.

Placing the CPU in RUN mode

1. Click the "RUN" button on either the PLC menu ribbon strip or on the program editor toolbar:
2. When prompted, click "OK" to change the operating mode of the CPU.

You can monitor the program in STEP 7-Micro/WIN SMART by clicking the "Program Status" button from the "Debug" menu ribbon strip, or from the program editor toolbar.
STEP 7-Micro/WIN SMART displays the values for the instructions.

Placing the CPU in STOP mode

To stop the program, click the "STOP" button and acknowledge the prompt to place the CPU in STOP mode. You can also place a STOP instruction (Page 337) in your program logic to put the CPU in STOP mode.

Installation

3

3.1 Guidelines for installing S7-200 SMART devices

The S7-200 SMART equipment is designed to be easy to install. You can install the S7-200 SMART either on a panel or on a standard DIN rail, and you can orient the S7-200 SMART either horizontally or vertically. The small size of the S7-200 SMART allows you to make efficient use of space.

⚠ WARNING

Safety requirements for installing S7-200 SMART PLCs

S7-200 SMART PLCs are Open Type Controllers. You must install the PLC in a housing, cabinet, or electric control room. Limit entry to the housing, cabinet, or electric control room to authorized personnel.

Failure to follow these installation requirements could result in death or serious injury to personnel, and/or damage to equipment.

Always follow these requirements when installing the PLC.

Separate the devices from heat, high voltage, and electrical noise

As a general rule for laying out the devices of your system, always separate the devices that generate high voltage and high electrical noise from the low-voltage, logic-type devices such as the PLC.

When configuring the layout of the PLC inside your panel, consider the heat-generating devices and locate the electronic-type devices in the cooler areas of your cabinet. Reducing the exposure to a high-temperature environment will extend the operating life of any electronic device.

Consider also the routing of the wiring for the devices in the panel. Avoid placing low-voltage signal wires and communications cables in the same tray with AC power wiring and high-energy, rapidly-switched DC wiring.

Provide adequate clearance for cooling and wiring

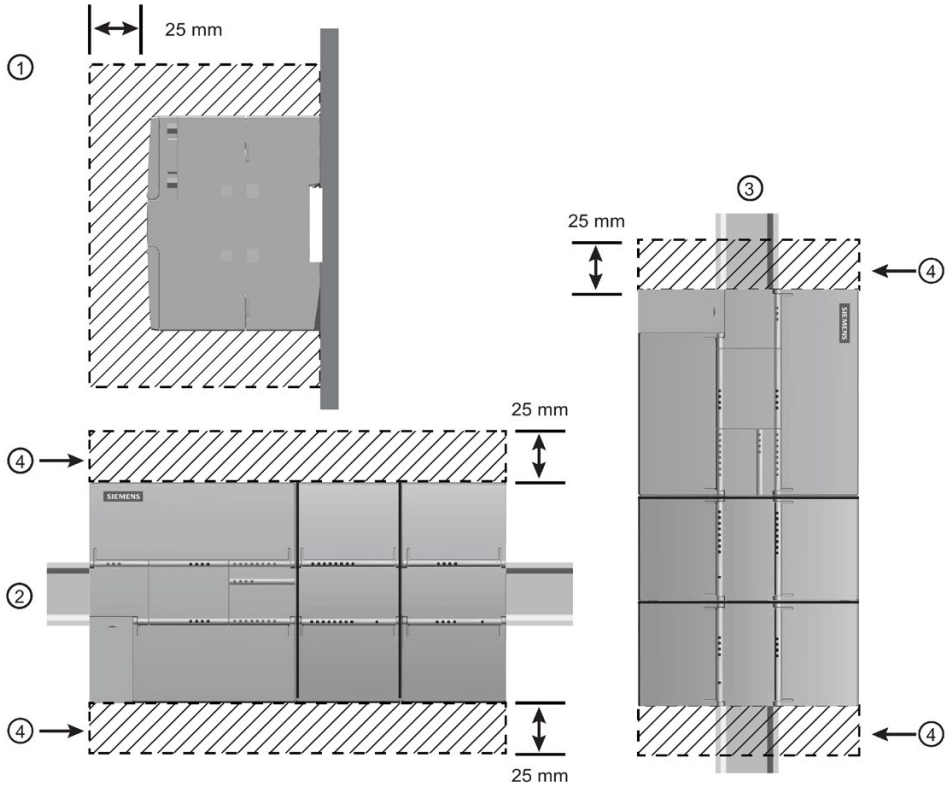
S7-200 SMART devices are designed for natural convection cooling. For proper cooling, you must provide a clearance of at least 25 mm above and below the devices. Also, allow at least 25 mm of depth between the front of the modules and the inside of the enclosure.

CAUTION

Temperature considerations

Vertical mounting reduces the maximum allowable ambient temperature by 10 degrees C. Operating outside the maximum temperature range could result in erratic process operation and could result in minor personal injury.

If your installation includes expansion modules, mount the CPU below them as shown in the following figure. Follow the prescribed guidelines for mounting modules to ensure proper cooling.



①	Side view	③	Vertical installation
②	Horizontal installation	④	Clearance area

When planning your layout for the PLC, allow enough clearance for the wiring and communications cable connections.

3.2 Power budget

Your CPU has an internal power supply that provides power for the CPU, the expansion modules, signal boards, and other 24 V DC user power requirements. Use the following information as a guide for determining how much power (or current) the CPU can provide for your configuration. The new compact CPUs (CRs) do not support expansion modules or signal boards.

Refer to the technical specifications for your particular CPU to determine the 24 V DC sensor supply power budget, the 5 V DC logic budget supplied by your CPU and the 5 V DC power requirements of the expansion modules and signal boards. Refer to the Calculating a power budget (Page 855) to determine how much power (or current) the CPU can provide for your configuration.

The standard CPU provides the 5 V DC logic power needed for any expansion in your system. Pay careful attention to your system configuration to ensure that the CPU can supply the 5 V DC power required by your selected expansion modules. If your configuration requires more power than the CPU can supply, you must remove a module.

Note

If the CPU power budget is exceeded, you may not be able to connect the maximum number of modules allowed for your CPU.

The standard CPU also provides a 24 V DC sensor supply that can supply 24 V DC for input points, for relay coil power on the expansion modules, or for other requirements. If your power requirements exceed the budget of the sensor supply, then you must add an external 24 V DC power supply to your system. You must manually connect the 24 V DC supply to the input points or relay coils.

If you require an external 24 V DC power supply, ensure that the power supply is not connected in parallel with the sensor supply of the CPU. For improved electrical noise protection, it is recommended that the commons (M) of the different power supplies be connected.

⚠ WARNING

Connecting power supplies safely

Connecting an external 24 V DC power supply in parallel with the 24 V DC sensor supply of the CPU can result in a conflict between the two supplies as each seeks to establish its own preferred output voltage level.

The result of this conflict can be shortened lifetime or immediate failure of one or both power supplies, with consequent unpredictable operation of the PLC system. Unpredictable operation could result in death or serious injury to personnel, and/or damage to equipment.

The DC sensor supply of the CPU and any external power supply should provide power to different points. A single connection of the commons is allowed.

Some of the 24 V DC power input ports in the S7-200 SMART system are interconnected, with a common logic circuit connecting multiple M terminals. For example, the following circuits are interconnected when designated as "not isolated" in the data sheets: the 24 V DC power supply of the CPU, the power input for the relay coil of an EM, or the power supply

for a non-isolated analog input. All non-isolated M terminals must connect to the same external reference potential.

WARNING

Avoiding unwanted current flow

Connecting non-isolated M terminals to different reference potentials will cause unintended current flows that may cause damage or unpredictable operation in the PLC and any connected equipment.

Failure to comply with these guidelines could cause damage or unpredictable operation which could result in death or severe personal injury and/or property damage.

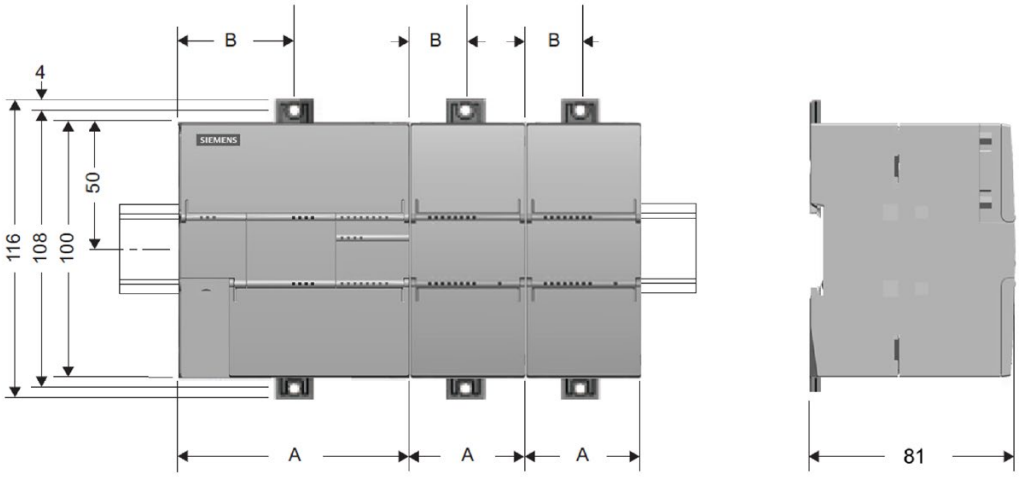
Always ensure that all non-isolated M terminals in an S7-200 SMART system are connected to the same reference potential.

Refer to the technical specifications for your particular CPU to determine the 24 V DC sensor supply power budget, the 5 V DC logic budget supplied by your CPU and the 5 V DC power requirements of the expansion modules and signal boards.

3.3 Installation and removal procedures

3.3.1 Mounting dimensions for the S7-200 SMART devices

The CPU and expansion modules include mounting holes to facilitate installation on panels.



S7-200 SMART module	Width A (mm)	Width B (mm)
CPU SR20, CPU ST20, and CPU CR20s	90	45
CPU SR30, CPU ST30, and CPU CR30s	110	55
CPU SR40, CPU ST40, and CPU CR40s	125	62.5

S7-200 SMART module		Width A (mm)	Width B (mm)
CPU SR60, CPU ST60, and CPU CR60s[1]		175	37.5[1]
Expansion modules:	EM 4AI, EM 8AI, EM 2AQ, EM 4AQ, EM 8DI, EM 16DI, EM 8DQ, and EM 8DQ RLY, EM 16DQ RLY, and EM 16DQ Transistor	45	22.5
	EM 8DI/8DQ and EM 8DI/8DQ RLY	45	22.5
	EM 16DI/16DQ and EM 16DI/16DQRLY	70	35
	EM 2AI/1AQ and EM 4AI/2AQ	45	22.5
	EM 2RTD, EM 4RTD	45	22.5
	EM 4TC	45	22.5
	EM DP01	70	35

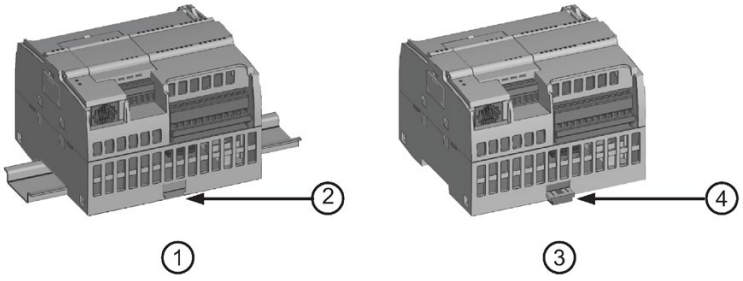
[1]The CPU xx60 models have two sets of mounting holes. The width "B" dimension is measured from the center of each mounting hole to the corresponding edge of the housing.

Note

The compact serial CPUs (CPU SR20s, CPU SR30s, CPU SR40s, and CPU SR60s) do not support expansion modules or signal boards.

3.3.2 Installing and removing the CPU

The CPU can be easily installed on a standard DIN rail or on a panel. DIN rail clips are provided to secure the device on the DIN rail. The clips also snap into an extended position to provide a screw mounting position for panel-mounting the unit.



① DIN rail installation
② DIN rail clip in latched position
③ Panel installation
④ Clip in extended position

Figure 3-1 Installation on a DIN rail or on a panel

Before you install or remove any electrical device, ensure that the power to that equipment has been turned off. Also, ensure that the power to any related equipment has been turned off.

WARNING

Remove power to PLC before installing or removing equipment

Attempts to install or remove the PLC or related equipment with the power applied could cause electric shock or faulty operation of equipment.

Failure to disable all power to the PLC and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.

Always follow appropriate safety precautions and ensure that power to the PLC is disabled before attempting to install or remove the CPU or related equipment.

Always ensure that whenever you replace or install a device, you use the correct module or equivalent device.

WARNING

Module replacement

If you install an incorrect module, the program in the CPU could function unpredictably.

Failure to replace a device with the same model, orientation, or order could result in death or serious injury to personnel, and/or damage to equipment.

Replace the device with the same model, and be sure to orient and position it correctly.

Note

Install expansion modules separately after the CPU has been installed. The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

Consider the following when installing the units on the DIN rail or on a panel:

- For DIN rail mounting, make sure the upper DIN rail clip is in the latched (inner) position and that the lower DIN rail clip is in the extended position for the CPU.
- After installing the devices on the DIN rail, move the lower DIN rail clips to the latched position to lock the devices on the DIN rail.
- For panel mounting, make sure the DIN rail clips are pushed to the extended position.

To install the CPU on a panel, follow these steps:

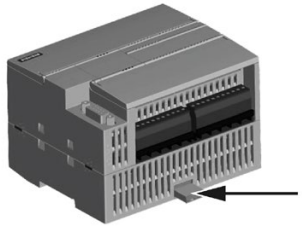
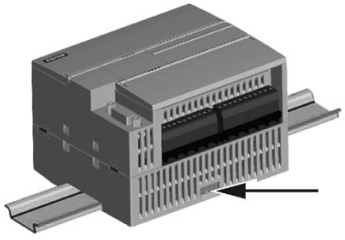
1. Locate, drill, and tap the mounting holes (M4 or American Standard number 8), using the dimensions in the table, Mounting dimensions (mm) (Page 46).
2. Ensure that the CPU and S7-200 SMART equipment are disconnected from electrical power.

3. Secure the module(s) to the panel, using a Pan Head M4 screw with spring and flat washer. Do not use a flat head screw.
4. If you are using an expansion module, put it next to the CPU and slide together until the connectors join securely.

Note

The type of screw will be determined by the material upon which it is mounted. You should apply appropriate torque until the spring washer becomes flat. Avoid applying excessive torque to the mounting screws. Do not use a flat head screw.

Table 3- 1 Installing a CPU on a DIN rail

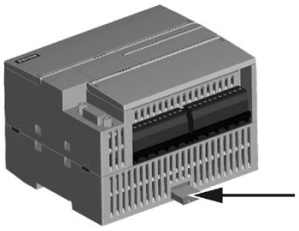
Task	Procedure
 	Follow the steps below to install a CPU on a DIN rail. 1. Secure the rail to the mounting panel every 75 mm. 2. Snap open the DIN clip (located on the bottom of the module) and hook the back of the module onto the DIN rail. 3. Rotate the module down to the DIN rail and snap the clip closed. Carefully check that the clip has fastened the module securely onto the rail. To avoid damage to the module, press on the tab of the mounting hole instead of pressing directly on the front of the module.

Note

Using DIN rail stops could be helpful if your CPU is in an environment with high vibration potential or if the CPU has been installed vertically. Use an end bracket (8WA1 808 or 8WA1 805) on the DIN rail to ensure that the modules remain connected.

If your system is in a high-vibration environment, then panel-mounting the CPU will provide a greater level of vibration protection.

Table 3- 2 Removing a CPU from a DIN rail

Task	Procedure
	Follow the steps below to remove a CPU from a DIN rail. 1. Remove power from the CPU and any attached I/O modules. 2. Disconnect all the wiring and cabling that is attached to the CPU. The CPU and most expansion modules have removable connectors to make this job easier. 3. Unscrew the mounting screws or snap open the DIN clip. 4. If you have expansion modules connected, slide the CPU to the left to disengage it from the expansion module connector. Note: unscrewing or unsnapping the DIN clips of the expansion modules can make it easier to disengage the CPU. 5. Remove the CPU.

3.3.3 Installing and removing a signal board or battery board

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules, signal boards or battery boards.

Table 3- 3 Installing a signal board on a CPU

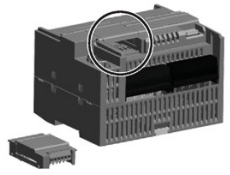
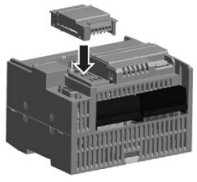
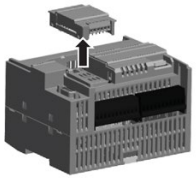
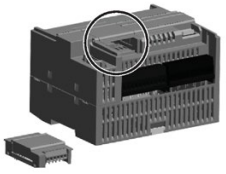
Task	Procedure
	Follow the steps below to install a signal board or battery board 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Remove the top and bottom terminal block covers from the CPU. 3. Place a screwdriver into the slot on top of the CPU at the rear of the cover. 4. Gently pry the cover up and remove it from the CPU. 5. Place the signal board or battery board straight down into its mounting position in the top of the CPU. 6. Firmly press the module into position until it snaps into place. 7. Replace the terminal block covers.
	

Table 3- 4 Removing a signal board or battery board on a CPU

Task	Procedure
	Follow the steps below to remove a signal board or battery board 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Remove the top and bottom terminal block covers from the CPU. 3. Place a screwdriver into the slot on top of the module. 4. Gently pry the module up to disengage it from the CPU. 5. Remove the module straight up from its mounting position in the top of the CPU. 6. Replace the cover onto the CPU. 7. Replace the terminal block covers.
	

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

Installing or replacing the battery in the SB BA01 battery board

The SB BA01 battery board requires battery type CR1025. The battery is not included with the SB BA01 and must be purchased.

To install the battery, follow these steps:

1. In the SB BA01, install the new battery with the positive side of the battery on top, and the negative side next to the printed wiring board.
2. The SB BA01 is now ready to be installed in the CPU. Follow the installation directions above.

To replace the battery, follow these steps:

1. Remove the SB BA01 from the CPU following the removal directions above.
2. Carefully remove the old battery using a small screwdriver. Push the battery out from under the clip.
3. Install a new CR1025 replacement battery with the positive side of the battery on top and the negative side next to the printed wiring board.
4. Re-install the SB BA01 battery board following the installation directions above.

3.3.4 Removing and reinstalling the terminal block connector

The S7-200 SMART modules have removable connectors to make connecting the wiring easy.

Table 3- 5 Removing the connector

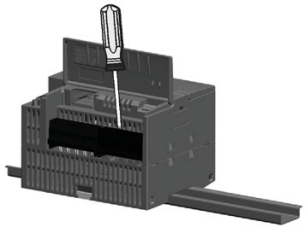
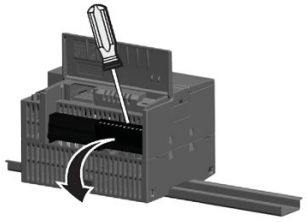
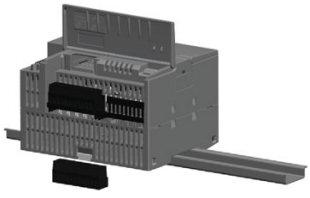
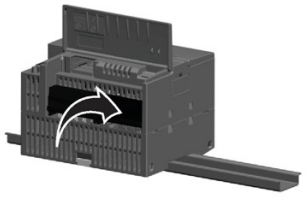
Task	Procedure
	Prepare the system for terminal block removal by removing the power from the CPU and opening the cover above the connector. 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Inspect the top of the connector and locate the slot for the tip of the screwdriver. 3. Insert a small screwdriver into the slot. 4. Gently pry the top of the connector away from the CPU. The connector will release with a snap. 5. Grasp the connector and remove it from the CPU.
	

Table 3- 6 Installing the connector

Task	Procedure
	Prepare the components for terminal block installation by removing power from the CPU and opening the cover above the connector. 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Align the connector with the pins on the unit. 3. Align the wiring edge of the connector inside the rim of the connector base. 4. Press firmly down and rotate the connector until it snaps into place. Check carefully to ensure that the connector is properly aligned and fully engaged.
	

3.3.5 Installing and removing an expansion module

Install expansion modules separately after the CPU has been installed. The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

Table 3- 7 Installing an expansion module

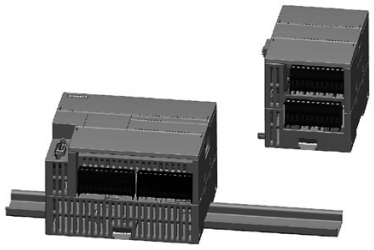
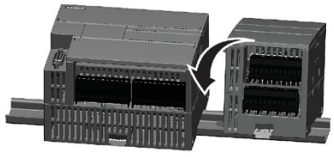
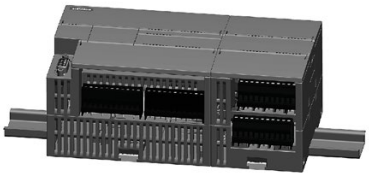
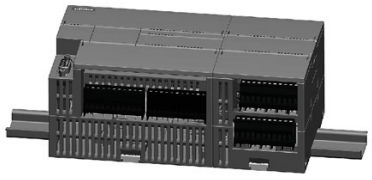
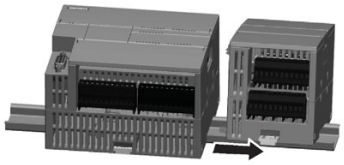
Task	Procedure
	Follow the steps below to install an expansion module: 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Remove the cover for the I/O bus connector from the right side of the CPU. 3. Insert a screwdriver into the slot above the cover. 4. Gently pry the cover out at its top and remove the cover. Retain the cover for reuse.
	Connect the expansion module to the CPU. 1. Pull out the bottom DIN rail clip to allow the expansion module to fit over the rail. 2. Position the expansion module to the right of the CPU. 3. Hook the expansion module over the top of the DIN rail. 4. Slide the expansion module to the left until the I/O connector fully engages the connector on the right of the CPU and push the bottom clip in to latch the expansion module onto the rail.
	

Table 3- 8 Removing an expansion module

Task	Procedure
	Follow the steps below to remove an expansion module: 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Remove the I/O connectors and wiring from the expansion module. Loosen the DIN rail clips of all the S7-200 SMART devices. 3. Physically slide the expansion module to the right.
	

3.3.6 Installing and removing the expansion cable

The S7-200 SMART expansion cable provides additional flexibility in configuring the layout of your S7-200 SMART system. Only one expansion cable is allowed per CPU system. You install the expansion cable either between the CPU and the first EM, or between any two EMs.

Table 3- 9 Installing and removing the male connector of the expansion cable

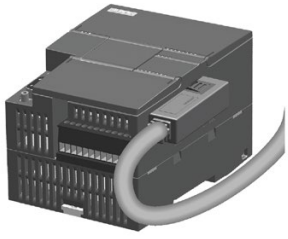
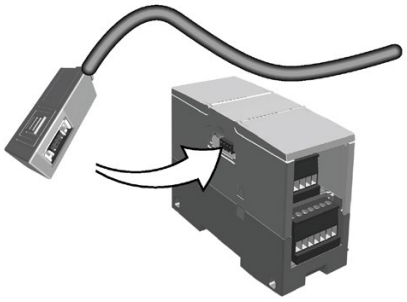
Task	Procedure
	To install the male connector: 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Push the male connector into the bus connector on the right side of the expansion module or CPU. 3. The male connector is locked in place when it is fully seeded. To remove the male connector: 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Use your thumb to press down the latch on the top of the male connector to release it from the expansion module or CPU. 3. Remove the male connector from the expansion module or CPU by pulling it straight out.
	

Table 3- 10 Installing and removing the female connector of the expansion cable

Task	Procedure
	To install the female connector: 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Push the female connector into the bus connector on the left side of the expansion module. 3. The female connector is locked in place when it is fully seeded. To remove the female connector: 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Use your thumb to press down the latch on the top of the female connector to release it from the expansion module. 3. Remove the female connector from the expansion module by pulling it straight out.

Note

Installing the expansion cable in a vibration environment

If the expansion cable is connected to modules that move or are not firmly fixed, the connection on the cable ends can gradually become loose.

Use a cable tie to fix the cable ends on the DIN-rail (or other place) to provide extra strain relief.

Avoid using excessive force when you pull the cable during installation. Ensure the cable-module connection is in the correct position once installation is complete.

3.4 Wiring guidelines

Proper grounding and wiring of all electrical equipment is important to help ensure the optimum operation of your system and to provide additional electrical noise protection for your application and the PLC. Refer to the technical specifications (Page 751) for the wiring diagrams.

Prerequisites

Before you ground or install wiring to any electrical device, ensure that the power to that equipment has been turned off. Also, ensure that the power to any related equipment has been turned off.

Ensure that you follow all applicable electrical codes when wiring the PLC and related equipment. Install and operate all equipment according to all applicable national and local standards. Contact your local authorities to determine which codes and standards apply to your specific case.

⚠ WARNING

Attempts to install or wire the PLC or related equipment with power applied could cause electric shock or faulty operation of equipment. Failure to disable all power to the PLC and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.

Always follow appropriate safety precautions and ensure that power to the PLC is disabled before attempting to install or remove the PLC or related equipment.

Always take safety into consideration as you design the grounding and wiring of your PLC system. Electronic control devices, such as the PLC, can fail and can cause unexpected operation of the equipment that is being controlled or monitored. For this reason, you should implement safeguards that are independent of the PLC to protect against possible personal injury or equipment damage.

⚠ WARNING

Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment.

Use an emergency stop function, electromechanical overrides, or other redundant safeguards that are independent of the PLC.

Isolation guidelines

The AC power supply boundaries and I/O boundaries to AC circuits have been designed and approved to provide safe separation between AC line voltages and low voltage circuits. These boundaries include double or reinforced insulation, or basic plus supplementary insulation, according to various standards. Components which cross these boundaries such as optical couplers, capacitors, transformers, and relays have been approved as providing safe separation. Only circuits rated for AC line voltage include safety isolation to other circuits. Isolation boundaries between 24 V DC circuits are functional only, and you should not depend on these boundaries for safety.

The sensor supply output, communications circuits, and internal logic circuits of an S7-200 SMART with included AC power supply are sourced as SELV (safety extra-low voltage) according to EN 61131-2.

To maintain the safe character of the S7-200 SMART low voltage circuits, external connections to communications ports, analog circuits, and all 24 V DC nominal power supply and I/O circuits must be powered from approved sources that meet the requirements of SELV, PELV, Class 2, Limited Voltage, or Limited Power according to various standards.

WARNING

Safe use of power converters

Use of non-isolated or single insulation supplies to supply low voltage circuits from an AC line can result in hazardous voltages appearing on circuits that are expected to be touch safe, such as communications circuits and low voltage sensor wiring.

Such unexpected high voltages could result in death or serious injury to personnel, and/or damage to equipment.

Use only high-voltage-to-low-voltage power converters that are approved as sources of touch-safe, limited-voltage circuits.

Grounding guidelines

The best way to ground your application is to ensure that all the common and ground connections of your PLC and related equipment are grounded to a single point. This single point should be connected directly to the earth ground for your system.

All ground wires should be as short as possible and should use a large wire size, such as 2 mm^2 (14 AWG).

When locating grounds, remember to consider safety grounding requirements and the proper operation of protective interrupting devices.

Wiring guidelines

When designing the wiring for your S7-200 SMART CPU, provide a single disconnect switch that simultaneously removes power from the CPU power supply, from all input circuits, and from all output circuits. Provide over-current protection, such as a fuse or circuit breaker, to limit fault currents on supply wiring. Consider providing additional protection by placing a fuse or other current limit in each output circuit.

Install appropriate surge suppression devices for any wiring that could be subject to lightning surges.

Avoid placing low-voltage signal wires and communications cables in the same wire tray with AC wires and high-energy, rapidly switched DC wires. Always route wires in pairs, with the neutral or common wire paired with the hot or signal-carrying wire.

Use the shortest wire possible and ensure that the wire is sized properly to carry the required current.

Use wire and cable with a temperature rating 30 °C higher than the ambient temperature around the S7-200 SMART CPU (for example, a minimum of 85 °C-rated conductors for 55 °C ambient temperature). You should determine other wiring type and material requirements from the specific electrical circuit ratings and your installation environment.

Use shielded wires for optimum protection against electrical noise. Typically, grounding the shield at the S7-200 SMART CPU gives the best results. You should ground communication cable shields to S7-200 SMART CPU communication connector shells using connectors that engage the cable shield, or by bonding the communication cable shields to a separate ground. You should ground other cable shields using clamps or copper tape around the shield to provide a high surface area connection to the grounding point.

When wiring input circuits that are powered by an external power supply, include an overcurrent protection device in that circuit. External protection is not necessary for circuits that are powered by the 24 V DC sensor supply from the S7-200 SMART CPU because the sensor supply is already current-limited.

All S7-200 SMART CPU modules have removable connectors for user wiring. To prevent loose connections, ensure that the connector is seated securely and that the wire is installed securely into the connector.

To help prevent unwanted current flows in your installation, the S7-200 SMART CPU provides isolation boundaries at certain points. When you plan the wiring for your system, you should consider these isolation boundaries. Refer to the technical specifications (Page 751) for the amount of isolation provided and the location of the isolation boundaries. Circuits rated for AC line voltage include safety isolation to other circuits. Isolation boundaries between 24 V DC circuits are functional only, and you should not depend on these boundaries for safety.

A summary of wiring rules for the S7-200 SMART CPUs, EMs, and SBs is shown below:

Table 3- 11 Wiring rules for S7-200 SMART CPUs, EMs, and SBs

Wiring rules for...	CPU and EM connector	SB connector
Connectible conductor cross-sections for standard wires	2 mm^2 to 0.3 mm^2 (14 AWG to 22 AWG)	1.3 mm^2 to 0.3 mm^2 (16 AWG to 22 AWG)
Number of wires per connection	1 or combination of 2 wires up to 2 mm^2 (total)	1 or combination of 2 wires up to 1.3 mm^2 (total)
Wire strip length	6.4 mm	6.3 to 7 mm
Tightening torque* (maximum)	0.56 N-m (5 inch-pounds)	0.33 N-m (3 inch-pounds)
Tool	2.5 to 3.0 mm flathead screwdriver	2.0 to 2.5 mm flathead screwdriver

* To avoid damaging the connector, be careful that you do not over-tighten the screws.

Note

Ferrules or end sleeves on stranded conductors reduce the risk of stray strands causing short circuits. Ferrules longer than the recommended strip length should include an insulating collar to prevent shorts due to side movement of conductors. Cross-sectional area limits for bare conductors also apply to ferrules.

See also

General technical specifications (Page 751)

Guidelines for lamp loads

Lamp loads are damaging to relay contacts because of the high turn-on surge current. This surge current will nominally be 10 to 15 times the steady state current for a tungsten lamp. A replaceable interposing relay or surge limiter is recommended for lamp loads that will be switched a large number of times during the lifetime of the application.

Guidelines for inductive loads

Use suppressor circuits with inductive loads to limit the voltage rise when a control output turns off. Suppressor circuits protect your outputs from premature failure caused by the high voltage transient that occurs when current flow through an inductive load is interrupted.

In addition, suppressor circuits limit the electrical noise generated when switching inductive loads. High frequency noise from poorly suppressed inductive loads can disrupt the operation of the PLC. Placing an external suppressor circuit so that it is electrically across the load and physically located near the load is the most effective way to reduce electrical noise.

S7-200 SMART DC outputs include internal suppressor circuits that are adequate for inductive loads in most applications. Since S7-200 SMART relay output contacts can be used to switch either a DC or an AC load, internal protection is not provided.

A good suppressor solution is to use contactors and other inductive loads for which the manufacturer provides suppressor circuits integrated in the load device, or as an optional accessory. However, some manufacturer provided suppressor circuits may be inadequate for your application. An additional suppressor circuit may be necessary for optimal noise reduction and contact life.

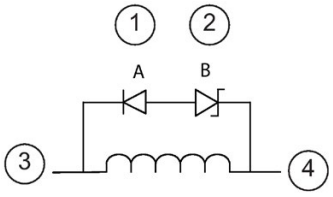
For AC loads, a metal oxide varistor (MOV) or other voltage clamping device may be used with a parallel RC circuit, but is not as effective when used alone. An MOV suppressor with no parallel RC circuit often results in significant high frequency noise up to the clamp voltage.

A well-controlled turn-off transient will have a ring frequency of no more than 10 kHz, with less than 1 kHz preferred. Peak voltage for AC lines should be within +/- 1200 V of ground. Negative peak voltage for DC loads using the PLC internal suppression will be ~40 V below the 24 V DC supply voltage. External suppression should limit the transient to within 36 V of the supply to unload the internal suppression.

Note

The effectiveness of a suppressor circuit depends on the application and must be verified for your particular usage. Ensure that all components are correctly rated and use an oscilloscope to observe the turn-off transient.

Typical suppressor circuit for DC or relay outputs that switch DC inductive loads



① 1N4001 diode or equivalent

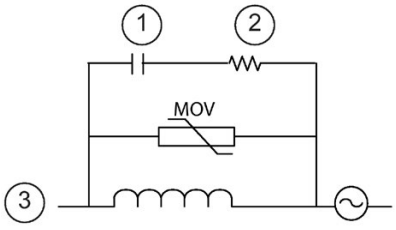
② 8.2 V Zener (DC outputs),
36 V Zener (Relay outputs)

③ Output point

④ M, 24 V reference

In most applications, the addition of a diode (A) across a DC inductive load is suitable, but if your application requires faster turn-off times, then the addition of a zener diode (B) is recommended. Be sure to size your zener diode properly for the amount of current in your output circuit.

Typical suppressor circuit for relay outputs that switch AC inductive loads



① See table for C value

② See table for R value

③ Output point

Ensure that the working voltage of the metal oxide varistor (MOV) is at least 20% greater than the nominal line voltage.

Choose pulse-rated, non-inductive resistors, and capacitors recommended for pulse applications (typically metal film). Verify the components meet average power, peak power, and peak voltage requirements.

If you design your own suppressor circuit, the following table suggests resistor and capacitor values for a range of AC loads. These values are based on calculations with ideal component parameters. I rms in the table refers to the steady-state current of the load when fully ON.

Table 3- 12 AC suppressor circuit resistor and capacitor values

Inductive load			Suppressor values		
I rms	230 V AC	120 V AC	Resistor		Capacitor
Amps	VA	VA	Ω	W (power rating)	nF
0.02	4.6	2.4	15000	0.1	15
0.05	11.5	6	5600	0.25	470
0.1	23	12	2700	0.5	100
0.2	46	24	1500	1	150
0.5	115	60	560	2.5	470

Inductive load			Suppressor values		
1	230	120	270	5	1000
2	460	240	150	10	1500

Conditions satisfied by the table values:

Maximum turn-off transition step < 500 V
Resistor peak voltage < 500 V
Capacitor peak voltage < 1250 V
Suppressor current < 8% of load current (50 Hz)
Suppressor current < 11% of load current (60 Hz)
Capacitor dV/dt < 2 V/µs
Capacitor pulse dissipation : $\int(dv/dt)^2dt < 10000\ V^2/\mu s$
Resonant frequency < 300 Hz
Resistor power for 2 Hz max switching frequency
Power factor of 0.3 assumed for typical inductive load

WARNING

Correct placement of external resistor/capacitor noise suppression circuit

When you use relay expansion modules to switch AC inductive loads, you must place the external resistor/capacitor noise suppression circuit across the AC load to prevent unexpected machine or process operation. Unexpected machine or process operation could result in death or severe personal injury.

Always be sure to follow these guidelines in placing the external resistor/capacitor noise suppression circuit.

PLC concepts

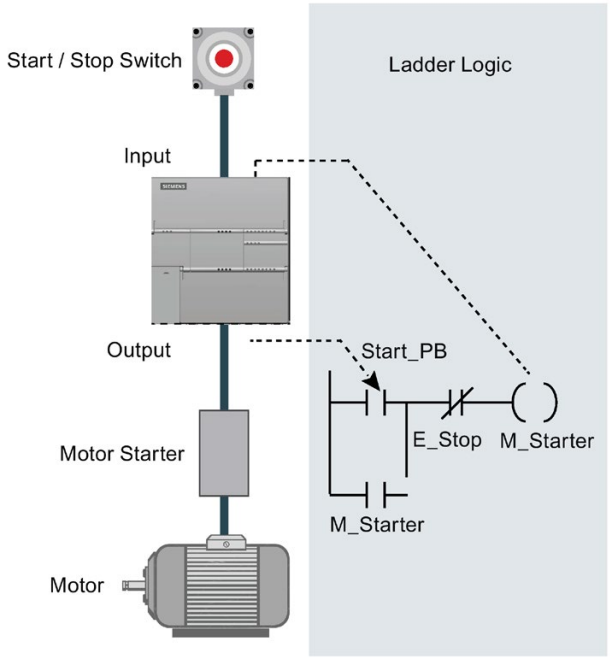
4

The basic function of the CPU is to monitor field inputs and, based on your control logic, turn on or off field output devices. This chapter explains the concepts used to execute your program, the various types of memory used, and how that memory is retained.

4.1 Execution of the control logic

The CPU continuously cycles through the control logic in your program, reading and writing data. The basic operation is very simple:

- The CPU reads the status of the inputs.
- The program that is stored in the CPU uses these inputs to evaluate the control logic.
- As the program runs, the CPU updates the data.
- The CPU writes the data to the outputs.

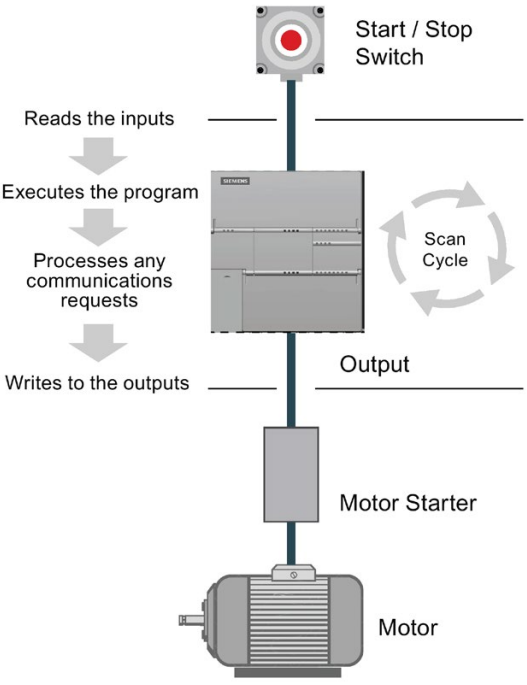


The figure shows a simple diagram of how an electrical relay diagram relates to the CPU. In this example, the state of the switch for starting the motor is combined with the states of other inputs. The calculations of these states then determine the state for the output that goes to the actuator which starts the motor.

Tasks in a scan cycle

The CPU executes a series of tasks repetitively. This cyclical execution of tasks is called the scan cycle. The execution of the user program is dependent upon whether the CPU is in STOP mode or in RUN mode. In RUN mode, your program is executed; in STOP mode, your program is not executed.

Table 4- 1 Tasks performed by the CPU in a scan cycle

Scan cycle	Description
	Reading the inputs: The CPU copies the state of the physical inputs to the process image input register. Executing the control logic in the program: The CPU executes the instructions of the program and stores the values in the various memory areas. Processing any communications requests: The CPU performs any tasks required for communications. Executing the CPU self-test diagnostics: The CPU ensures that the firmware, the program memory, and any expansion modules are working properly. Writing to the outputs: The values stored in the process image output register are written to the physical outputs.

4.1.1 Reading the inputs and writing to the outputs

Reading the inputs

Digital inputs: Each scan cycle begins by reading the current value of the digital inputs and then writing these values to the process image input register.

Analog inputs: The CPU does not read the analog input values as part of the normal scan cycle. Instead, an analog value is read immediately from the device when your program accesses the analog input.

Writing to the outputs

Digital outputs: At the end of every scan cycle, the CPU writes the values stored in the process-image output register to the digital outputs.

Analog outputs: The CPU does not write analog output values as part of the normal scan cycle. Instead, the analog outputs are written immediately when your program accesses the analog output.

4.1.2 Immediately reading or writing the I/O

The CPU instruction set provides instructions that immediately read from or write to the physical I/O. These immediate I/O instructions allow direct access to the actual input or output point, even though the image registers are normally used as either the source or the destination for I/O accesses. The corresponding process image input register location is not modified when you use an immediate instruction to access an input point. The corresponding process image output register location is updated simultaneously when you use an immediate instruction to access an output point.

Note

When you read an analog input, the value is read immediately. When you write a value to an analog output, the output is updated immediately.

It is usually advantageous to use the process image register rather than to directly access inputs or outputs during the execution of your program. There are three reasons for using the image registers:

- The sampling of all inputs at the start of the scan synchronizes and freezes the values of the inputs for the program execution phase of the scan cycle. The outputs are updated from the image register after the execution of the program is complete. This provides a stabilizing effect on the system.
- Your program can access the image register much more quickly than it can access I/O points, allowing faster execution of the program.
- I/O points are bit entities and must be accessed as bits or bytes, but you can access the image register as bits, bytes, words, or double words. Thus, the image registers provide additional flexibility.

4.1.3 Executing the user program

During the execution phase of the scan cycle, the CPU executes your main program, starting with the first instruction and proceeding to the last instruction. The immediate I/O instructions give you immediate access to inputs and outputs during the execution of either the main program or an interrupt routine.

If you use subroutines in your program, the subroutines are stored as part of the program. The subroutines are executed when they are called by the main program, by another subroutine, or by an interrupt routine. Subroutine nesting depth is 8 levels deep from the main and 4 levels deep from an interrupt routine.

If you use interrupts in your program, the interrupt routines that are associated with the interrupt events are stored as part of the program. The interrupt routines are not executed as part of the normal scan cycle, but are executed when the interrupt event occurs (which could be at any point in the scan cycle).

Local memory is reserved for each of 14 entities: the main program, eight subroutine nesting levels when initiated from the main program, one interrupt routine, and four subroutine nesting levels when initiated from an interrupt routine. Local memory has a local scope in that it is available only within its associated program entity, and cannot be accessed by the other program entities. For more information about Local memory, refer to Local Memory Area: L in this chapter.

The following figure depicts the flow of a typical scan including the Local memory usage and two interrupt events, one during the program-execution phase and another during the communications phase of the scan cycle. Subroutines are called by the next higher level, and are executed when called. Interrupt routines are not called; they are a result of an occurrence of the associated interrupt event.

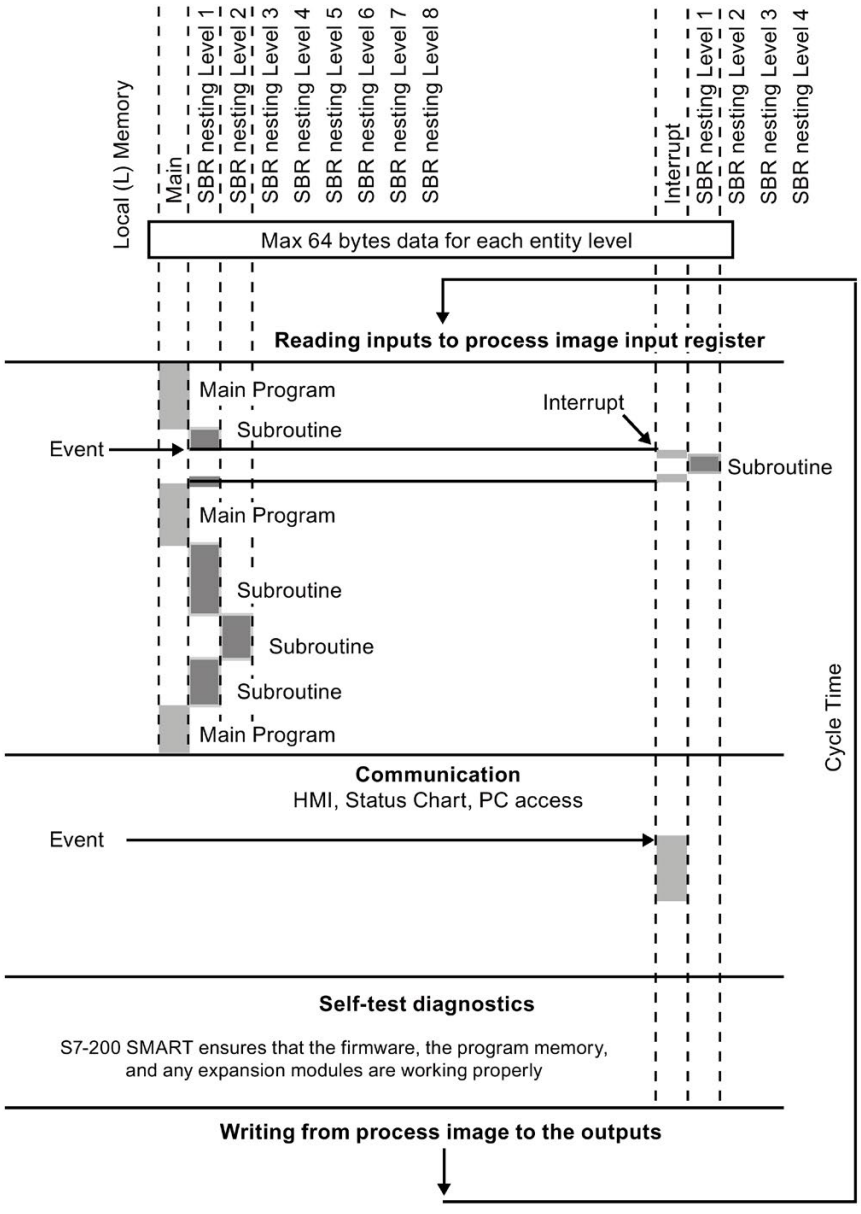
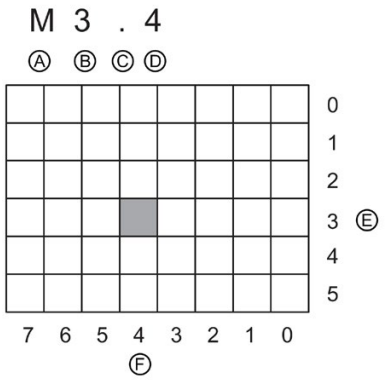


Figure 4-1 Typical scan flow

4.2 Accessing data

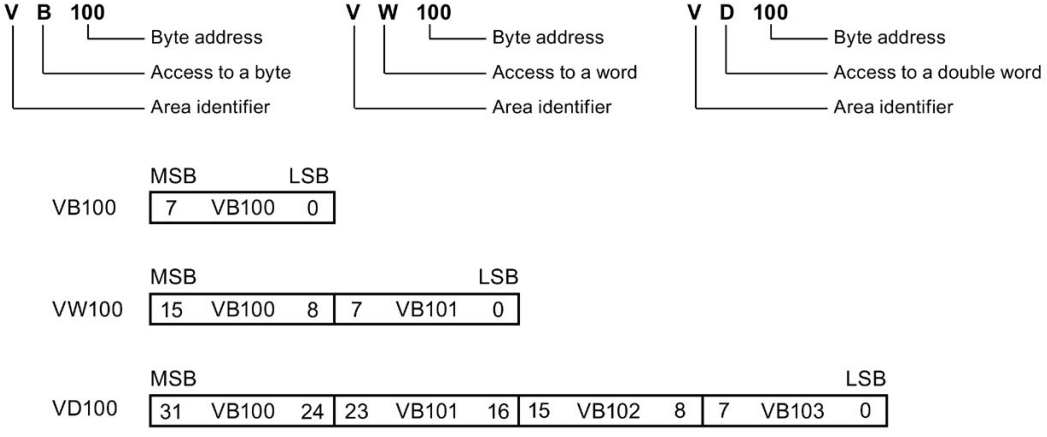
The CPU stores information in different memory locations that have unique addresses. You can explicitly identify the memory address that you want to access. This allows your program to have direct access to the information. To access a bit in a memory area, you specify the address, which includes the memory area identifier, the byte address, and the bit number (which is also called "byte.bit" addressing).

Table 4- 2 Bit addressing

Elements of a bit address		Description
	A	Memory area identifier
	B	Byte address: byte 3
	C	Separator ("byte.bit")
	D	Bit location of the byte (bit 4 of 8, bits numbered 7 to 0)
	E	Bytes of the memory area
	F	Bits of the selected byte

In this example, the memory area and byte address ("M3") designates byte 3 of M memory, with a period (".") to separate the bit address (bit 4).

You can access data in most memory areas (V, I, Q, M, S, L, and SM) as bytes, words, or double words by using the byte-address format. To access a byte, word, or double word of data in the memory, you must specify the address in a way similar to specifying the address for a bit. This includes an area identifier, data size designation, and the starting byte address of the byte, word, or double-word value, as shown in the following figure.



The following table shows the range of integer values that can be represented by the different sizes of data.

Table 4- 3 Decimal and hexadecimal ranges for the different sizes of data

Representation	Byte (B)	Word (W)	Double Word (D)
Unsigned Integer	0 to 255 16#00 to 16#FF	0 to 65,535 16#0000 to 16#FFFF	0 to 4,294,967,295 16#00000000 to 16#FFFFFFFF
Signed Integer	-128 to +127 16#80 to 16#7F	-32,768 to +32,767 16#8000 to 16#7FFF	-2,147,483,648 to +2,147,483,647 16#8000 0000 to 16#7FFF FFFF
Real (IEEE 32-bit Floating Point)	*Not applicable*	*Not applicable*	+1.175495E-38 to +3.402823E+38 (positive) -1.175495E-38 to -3.402823E+38 (negative)

Data in other memory areas (such as T, C, HC, and the accumulators) are accessed by using an address format that includes an area identifier and a device number.

4.2.1 Accessing memory areas

I (process-image input)

The CPU samples the physical input points at the beginning of each scan cycle and writes these values to the process image input register. You can access the process image input register in bits, bytes, words, or double words:

Table 4- 4 Absolute addressing for I memory

Bit:	I*[byte address].[bit address]*	I0.1
Byte, Word, or Double Word:	I*[size][starting byte address]*	IB4, IW7, ID20

Q (process-image output)

At the end of the scan cycle, the CPU copies the values stored in the process image output register to the physical output points. You can access the process image output register in bits, bytes, words, or double words:

Table 4- 5 Absolute addressing for Q memory

Bit:	Q*[byte address].[bit address]*	Q1.1
Byte, Word, or Double Word:	Q*[size][starting byte address]*	QB5, QW14, QD28

V (variable memory)

You can use V memory to store intermediate results of operations being performed by the control logic in your program. You can also use V memory to store other data pertaining to your process or task. You can access the V memory area in bits, bytes, words, or double words:

Table 4- 6 Absolute addressing for V memory

Bit:	V*[byte address].[bit address]*	V10.2
Byte, Word, or Double Word:	V*[size][starting byte address]*	VB16, VW100, VD2136

M (flag memory)

You can use the flag memory area (M memory) as internal control relays to store the intermediate status of an operation or other control information. You can access the flag memory area in bits, bytes, words, or double words:

Table 4- 7 Absolute addressing for M memory

Bit:	M*[byte address].[bit address]*	M26.7
Byte, Word, or Double Word:	M*[size][starting byte address]*	MB0, MW11, MD20

T (timer memory)

The CPU provides timers that count increments of time in resolutions (time-base increments) of 1 ms, 10 ms, or 100 ms. Two variables are associated with a timer:

- Current value: this 16-bit signed integer stores the amount of time counted by the timer.
- Timer bit: this bit is set or cleared as a result of comparing the current and the preset value. The preset value is entered as part of the timer instruction.

You access both of these variables by using the timer address (T + timer number). Access to either the timer bit or the current value is dependent on the instruction used: instructions with bit operands access the timer bit, while instructions with word operands access the current value. As shown in the following figure, the Normally Open Contact instruction accesses the timer bit, while the Move Word instruction accesses the current value of the timer.

Table 4- 8 Absolute addressing for T memory

Timer:	T*[timer number]*	T24

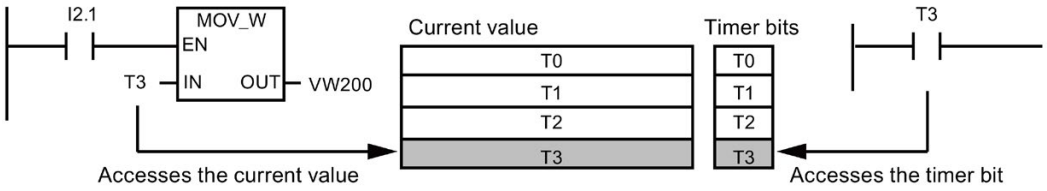


Figure 4-2 Accessing the timer bit or the current value of a timer

C (counter memory)

The CPU provides three types of counters that count each low-to-high transition event on the counter input(s): one type counts up only, one type counts down only, and one type counts both up and down. Two variables are associated with a counter:

- Current value: this 16-bit signed integer stores the accumulated count.
- Counter bit: this bit is set or cleared as a result of comparing the current and the preset value. The preset value is entered as part of the counter instruction.

You access both of these variables by using the counter address (C + counter number). Access to either the counter bit or the current value is dependent on the instruction used: instructions with bit operands access the counter bit, while instructions with word operands access the current value. As shown in the following figure, the Normally Open Contact instruction accesses the counter bit, while the Move Word instruction accesses the current value of the counter.

Table 4- 9 Absolute addressing of C memory

Counter	C*[counter number]*	C24

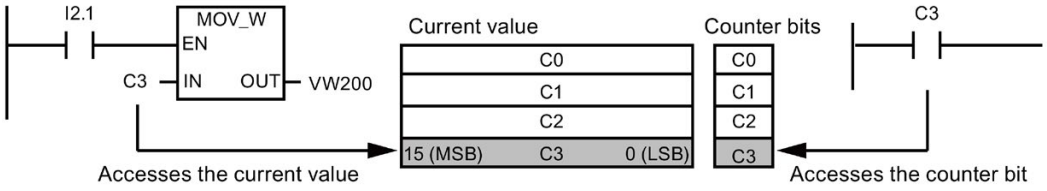


Figure 4-3 Accessing the counter bit or the current value of a counter

HC (high-speed counter)

The high-speed counters count high-speed events independent of the CPU scan. High-speed counters have a signed, 32-bit integer counting value (or current value). To access the count value for the high-speed counter, you specify the address of the high-speed counter, using the memory type (HC) and the counter number. The current value of the high-speed counter is a read-only value and can be addressed only as a double word (32 bits).

Table 4- 10 Absolute addressing of HC memory

High-speed counter	HC*[high-speed counter number]*	HC1

AC (accumulators)

The accumulators are read/write devices that can be used like memory. For example, you can use accumulators to pass parameters to and from subroutines and to store intermediate values used in a calculation. The CPU provides four 32-bit accumulators (AC0, AC1, AC2, and AC3). You can access the data in the accumulators as bytes, words, or double words.

The size of the data being accessed is determined by the instruction that is used to access the accumulator. As shown in the following figure, you use the least significant 8 or 16 bits of the value that is stored in the accumulator to access the accumulator as bytes or words. To access the accumulator as a double word, you use all 32 bits.

For information about how to use the accumulators within interrupt subroutines, refer to the Interrupt instructions (Page 307).

Table 4- 11 Absolute addressing of AC memory

Accumulator	AC*[accumulator number]*	AC0

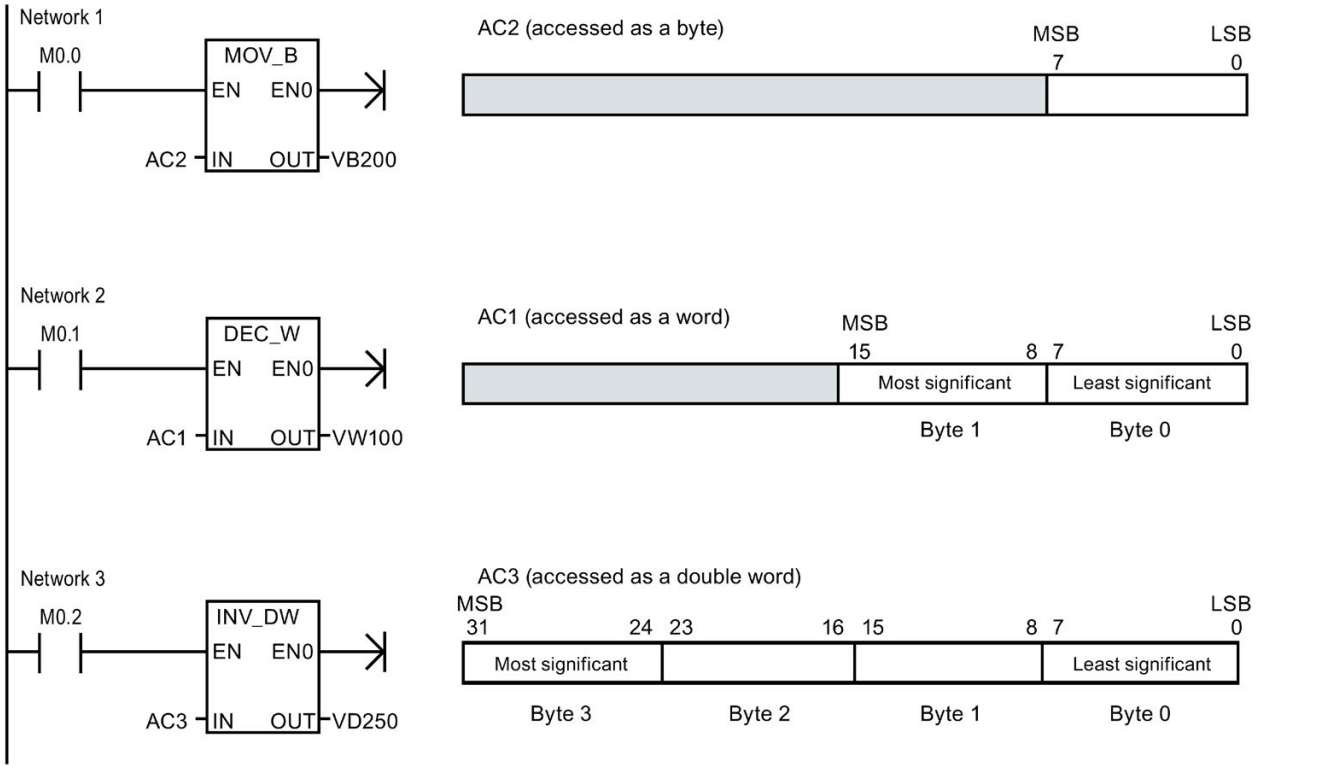


Figure 4-4 Accessing the accumulators

SM (special memory)

The SM bits provide a means for communicating information between the CPU and your user program. You can use these bits to select and control some of the special functions of the CPU, such as: a bit that turns on for the first scan cycle, a bit that toggles at a fixed rate, or a bit that shows the status of math or operational instructions. You can access the SM bits as bits, bytes, words, or double words:

Table 4- 12 Absolute addressing of SM memory

Bit:	SM*[byte address].[bit address]*	SM0.1
Byte, Word, or Double Word:	SM*[size][starting byte address]*	SMB86, SMW300, SMD1000

For more information, see the descriptions of the SM bits (Page 865).

L (local memory area)

The CPU provides 64 L memory bytes for each POU (program organizational unit) in a local memory stack. A POU's associated L memory addresses are accessible only by the currently executing POU (main, subroutine, or interrupt routine). When you use interrupt routines and subroutines, the L memory stack is used to preserve L memory values of a POU that temporarily suspends execution, so another POU can execute. The suspended POU can then resume execution with the L memory values that existed prior to giving execution control to another POU.

L memory stack maximum nesting limits:

- Eight subroutine nesting levels when initiated from the main program
- Four subroutine nesting levels when initiated from an interrupt routine

The nesting limits allow a 14 level execution stack in your program. For example, the main program (level 1) has eight nested subroutines (levels 2 to 9). During execution of the 9th level subroutine, an interrupt occurs (level 10). The interrupt routine contains four nested subroutines (levels 11 to 14).

L memory rules:

- You can use L memory for local scratchpad "TEMP" variables in all POU types (main, subroutine, and interrupt routines)
- Only subroutines can use L memory for "IN" IN_OUT", and "OUT" variable types that are passed to or from subroutines.
- If you are programming a subroutine in either LAD or FBD, only 60 bytes are allowed for TEMP, IN, IN_OUT, and OUT variables. STEP 7-Micro/WIN SMART uses the last four bytes of local memory

Local memory symbols, variable types, and data types are assigned in the Variable table that is available when the associated POU is opened in the program editor. Absolute L memory addresses are automatically assigned when a POU is successfully compiled.

In most cases, use L memory symbol name references in your program logic, because you cannot know all the absolute L memory addresses until after the complete POU is

successfully compiled. However, you can use absolute L memory addresses as shown in the following table.

Table 4- 13 Absolute addressing of L memory

Bit:	L*[byte address].[bit address]*	L0.0
Byte, Word, or Double Word:	L*[size] [starting byte address]*	LB33, LW5, LD20

Local memory and to global V memory use a similar address syntax, but V memory has a global scope while L memory has a local scope. Global scope means that the same memory address can be accessed from any POU. Local scope means that the L memory allocation is associated with a particular POU and cannot be accessed by another program unit.

The local scope of L memory also affects symbol usage, when a global symbol and a local symbol use the same name. If your program logic references that symbol name, the CPU ignores the global symbol and processes the address assigned to the local memory symbol.

Note

Local memory value assignments are not always preserved for successive executions of a POU

L memory addresses are reused for the next execution sequence, after the current nested sequence is completed. Depending on a POU's level in the execution stack and L memory assignments made since a POU's last execution, a POU's L memory assignments made in a previous execution may be overwritten with unexpected values.

Remember to reassign the correct values to L memory variables, in your program logic. Reinitialize all TEMP values before processing them and ensure that any output values (OUT and IN_OUT) are correct.

AI (analog input)

The CPU converts an analog value (such as temperature or voltage) into a word-length (16-bit) digital value. You access these values by the area identifier (AI), size of the data (W), and the starting byte address. Since analog inputs are words and always start on even-number bytes (such as 0, 2, or 4), you access them with even-number byte addresses (such as AIW0, AIW2, or AIW4). Analog input values are read-only values.

Table 4- 14 Absolute addressing of AI memory

Analog input	AIW*[starting byte address]*	AIW4

AQ (analog output)

The CPU converts a word-length (16-bit) digital value into a current or voltage, proportional to the digital value (such as for a current or voltage). You write these values by the area identifier (AQ), size of the data (W), and the starting byte address. Since analog outputs are words and always start on even-number bytes (such as 0, 2, or 4), you write them with even-number byte addresses (such as AQW0, AQW2, or AQW4). Analog output values are write-only values.

Table 4- 15 Absolute addressing of AQ memory

Analog output	AQW*[starting byte address]*	AQW4

S (sequence control relay)

S bits are associated with SCRs, which you can use to organize machine or steps into equivalent program segments. SCRs allow logical segmentation of the control program. You can access the S memory as bits, bytes, words, or double words.

Table 4- 16 Absolute addressing of S memory

Bit:	S*[byte address].[bit address]*	S3.1
Byte, Word, or Double Word:	S*[size][starting byte address]*	SB4, SW7, SD14

4.2.2 Format for Real numbers

Real (or floating-point) numbers are represented as 32-bit, single-precision numbers, whose format is described in the ANSI/IEEE 754-1985 standard. Real numbers are accessed in double-word lengths.

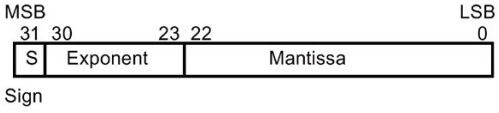


Figure 4-5 Format of a Real number

Note

Floating-point numbers are accurate up to 6 decimal places. Therefore, you can specify a maximum of 6 decimal places when entering a floating-point constant.

Calculations that involve a long series of values including very large and very small numbers can produce inaccurate results. This can occur if the numbers differ by 10 to the power of *x*, where $x > 6$. For example: 100 000 000 + 1 = 100 000 000

4.2.3 Format for strings

A string is a sequence of characters, with each character being stored as a byte. The first byte of the string defines the length of the string, which is the number of characters. The following figure shows the format for a string. A string can have a length of 0 to 254 characters, plus the length byte, so the maximum length for a string is 255 bytes. A string constant is limited to 126 bytes.

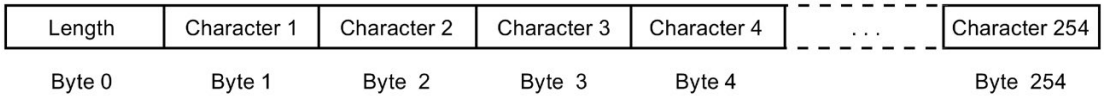


Figure 4-6 Format for strings

4.2.4 Assigning a constant value for instructions

You can use a constant value in many of the programming instructions. Constants can be bytes, words, or double words. The CPU stores all constants as binary numbers, which can then be represented in decimal, hexadecimal, ASCII, or real number (floating point) formats.

Table 4- 17 Representation of constant values

Representation	Format	Sample
Decimal	[decimal value]	20047
Hexadecimal	16#[hexadecimal value]	16#4E4F
Binary	2#[binary number]	2#1010_0101_1010_0101
ASCII	'[ASCII text]'	'ABCD'
Real	ANSI/IEEE 754-1985	+1.175495E-38 (positive) -1.175495E-38 (negative)
String	"[stringtext]"	"ABCDE"

Note

The CPU does not support "data typing" or data checking (such as specifying that the constant is stored as an integer, a signed integer, or a double integer). For example, an Add instruction can use the value in VW100 as a signed integer value, while an Exclusive Or instruction can use the same value in VW100 as an unsigned binary value.

4.2.5 Addressing the local and expansion I/O

The local I/O provided by the CPU provides a fixed set of I/O addresses. You can add I/O points by connecting expansion I/O modules to the right side of the CPU or by installing a signal board. The addresses of the points of the module are determined by the type of I/O and the position of the module in the chain. For example, an output module does not affect the addresses of the points on an input module, and vice versa. Likewise, analog modules do not affect the addressing of digital modules, and vice versa.

Note

Process image register space for digital I/O is always reserved in increments of eight bits (one byte). If a module does not provide a physical point for each bit of each reserved byte, these unused bits cannot be assigned to subsequent modules in the I/O chain. For input modules, the unused bits are set to zero with each input update cycle.

Analog I/O points are always allocated in increments of two points. If a module does not provide physical I/O for each of these points, these I/O points are lost and are not available for assignment to subsequent modules in the I/O chain.

The following table provides an example of the fixed mapping convention (established by STEP 7 Micro/WIN SMART and downloaded as part of the I/O configuration, in the system block).

Table 4- 18 CPU mapping convention

	CPU	Signal board	Expansion module 0	Expansion module 1	Expansion module 2	Expansion module 3	Expansion module 4	Expansion module 5
Starting address	I0.0 Q0.0	I7.0 Q7.0 AI12 AQ12	I8.0 Q8.0 AI16 AQ16	I12.0 Q12.0 AI32 AQ32	I16.0 Q16.0 AI48 AQ48	I20.0 Q20.0 AI64 AQ64	I24.0 Q24.0 AI80 AQ80	I28.0 Q28.0 AI96 AQ96

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

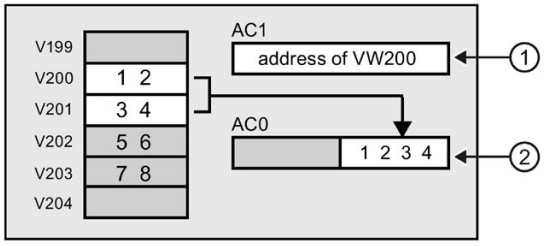
4.2.6 Using pointers for indirect addressing

Indirect addressing uses a pointer to access data in memory. Pointers are double word memory locations that contain the address of another memory location. You can only use V memory locations, L memory locations, or accumulator registers (AC1, AC2, AC3) as pointers. To create a pointer, you must use the Move Double Word instruction to move the address of the indirectly addressed memory location to the pointer location. Pointers can also be passed to a subroutine as a parameter.

An S7-200 SMART CPU allows pointers to access the following memory areas: I, Q, V, M, S, AI, AQ, SM, T (current value only), and C (current value only). You cannot use indirect addressing to access an individual bit or to access HC, L or accumulator memory areas.

To indirectly access the data in a memory address, you create a pointer to that location by entering an ampersand character (&) and the first byte of the memory location to be addressed. The input operand of the instruction must be preceded with an ampersand (&) to signify that the address of a memory location, instead of its contents, is to be moved into the location identified in the output operand of the instruction (the pointer).

Entering an asterisk (*) in front of an operand for an instruction specifies that the operand is a pointer. As shown in the following figure, entering *AC1 means that AC1 stores a pointer to the word-length value being referenced by the Move Word (MOVW) instruction. In this example, the values stored in both VB200 and VB201 are moved to accumulator AC0.



① MOVD &VB200, AC1

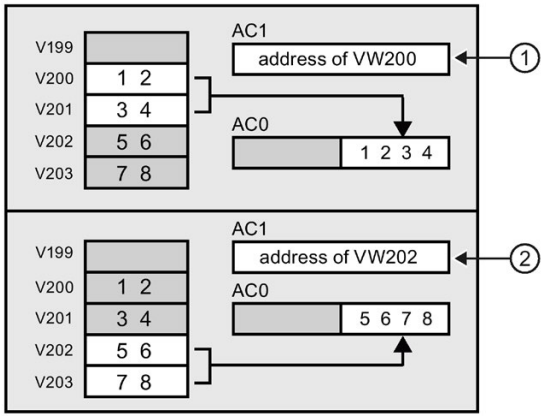
Creates the pointer by moving the address of VB200 (initial byte of VW200) to AC1

② MOVW *AC1, AC0

Moves the word value referenced by the pointer in AC1

Figure 4-7 Creating and using a pointer

As shown in the following figure, you can change the value of a pointer. Since pointers are 32-bit values, use double-word instructions to modify pointer values. Simple mathematical operations, such as adding or incrementing, can be used to modify pointer values.



① MOVD &VB200, AC1

Creates the pointer by moving the address of VB200 (initial byte of VW200) to AC1

MOVW *AC1, AC0

Moves the word value referenced by the pointer in AC1

② +D +2, AC1

Adds 2 to the accumulator to point to the next word location

MOVW *AC1, AC0

Moves the word value referenced by the pointer in AC1

Figure 4-8 Modifying a pointer

Note

When modifying the value of a pointer, remember to adjust for the size of the data that you are accessing: to access a byte, increment the pointer value by 1; to access a word or a current value for a timer or counter, add or increment the pointer value by 2; and to access a double word, add or increment the pointer value by 4.

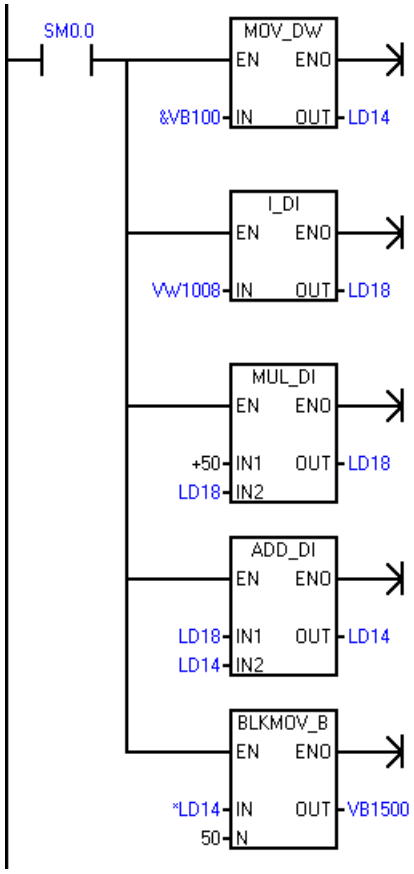
4.2.7 Pointer examples

Using a pointer to access data in a table

This example uses LD14 as a pointer to a recipe stored in a table of recipes that begins at VB100. In this example, VW1008 stores the index to a specific recipe in the table. If each recipe in the table is 50 bytes long, you multiply the index by 50 to obtain the offset for the starting address of a specific recipe. By adding the offset to the pointer, you can access the individual recipe from the table. In this example, the recipe is copied to the 50 bytes that start at VB1500.

Table 4- 19 Example: Using a pointer to access data in a table

LAD		STL

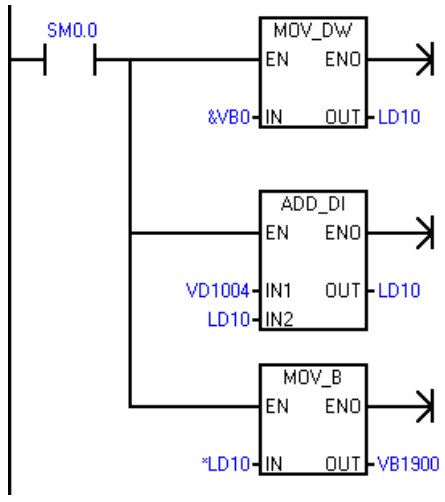
	To transfer a recipe from a table of recipes:
	• Each recipe is 50 bytes long.
	• The index parameter (VW1008) identifies the recipe to be loaded.
	
	Create a pointer to the starting address of the recipe table.
	
	Convert the index of the recipe to a double-word value.
	
	Multiply the offset to accommodate the size of each recipe.
	
	Add the adjusted offset to the pointer.
	
	Transfer the selected recipe to VB1500 through VB1549
		`Network 1`
		`LD SM0.0`
		`MOVD &VB100, LD14`
		
		`ITD VW1008, LD18`
		
		`*D +50, LD18`
		
		`+D LD18, LD14`
		
		`BMB *LD14, VB1500, 50`

Using an offset to access data

This example uses LD10 as a pointer to the address VB0. You then increment the pointer by an offset stored in VD1004. LD10 then points to another address in V memory (VB0 + offset). The value stored in the V memory address pointed to by LD10 is then copied to VB1900. By changing the value in VD1004, you can access any V memory location.

Table 4- 20 Example: Using an offset to read the value of any V memory location

LAD		STL

	Load the starting address of the V memory to a pointer.
	
	Add the offset value to the pointer.
	
	Copy the value from the V memory location (offset) to VB1900
		`Network 1`
		`LD SM0.0`
		`MOVD &VB0, LD10`
		
		`+D VD1004, LD10`
		
		`MOVB *LD10, VB1900`

4.3 Saving and restoring data

4.3.1 Downloading project components

Note

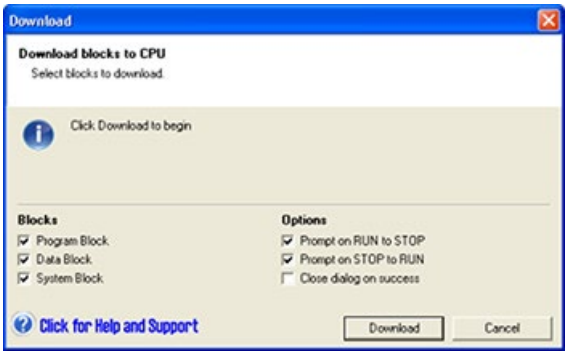
Downloading a program block, data block, or system block to the CPU completely overwrites any pre-existing contents of that block in the CPU. Be sure that you want to overwrite the block before performing a download.

To download project components from STEP 7-Micro/WIN SMART to the CPU, follow these steps:

1. Ensure that your Communication Interface and PLC connector cable for either Ethernet (Page 29) (standard CPUs only) or RS485 (Page 32) communications is working, and that PLC communication is operating properly.
2. Place the CPU in STOP mode (Page 41).

3. To download all project components, click the Download button from the Transfer area of the File or PLC menu ribbon strip, or alternatively press the shortcut key combination **CTRL+D**.

4. To download selected project components, click the down arrow under the Download button, and then select the specific project component you want to download (Program Block, Data Block, or System Block) from the drop-down list.
5. After clicking the Download button, if you see a Communications dialog, select the Communication Interface and the Ethernet IP address or RS485 network address for the PLC to which you want to download.
6. From the Download dialog, set the download options for the blocks, and whether you want to be prompted on CPU transitions from RUN to STOP mode (Page 41) and STOP to RUN mode (Page 41).
7. Optionally click the "Close dialog on success" check box if you want the dialog to automatically close after a successful download.
8. Click the Download button.



STEP 7-Micro/WIN SMART copies the complete program or program components that you selected to the CPU. The status icon indicates informational messages, or whether potential problems or errors occurred with the download. The status message provides specific results of the operation.

Note

You can download project components that you originally created for use in an S7-200 SMART CPU with firmware version V1.x to a CPU with firmware version V2.0 or later. However, you cannot download project components that you originally created for use in a CPU firmware version V2.0 or later to a CPU with firmware version V1.x, especially if the project components use functionality that firmware version V1.x did not support.

STEP 7-Micro/WIN SMART also supports program edit and download in RUN mode.

What happens when you download

STEP 7-Micro/WIN SMART and the CPU perform the following tasks in sequence on your project components when you download:

Step	Action	Related topics, additional description
1.	Project components in the program editors serve as input for the download operation, based on the download objects you selected. The program editors can include new program data that you've entered, a saved and opened .smart project, or an uploaded ASCII import file.	File open Range checking Project file I/O errors Program editor errors
2.	STEP 7-Micro/WIN SMART compile A compile or download command starts the compiler. If the compile passes, control passes to the next step; if not, the compile or download operation exits.	All STEP 7-Micro/WIN SMART compiler errors are listed in the Output Window. Double-click the error and the editor scrolls to the error location. A successful compile shows the resulting block size of the program and data block.
3.	Send blocks to CPU across communication network for PLC compile.	Communication Errors To download (Editor to PLC) or upload (PLC to Editor), PLC communication must be operating properly. Make sure your network hardware and PLC connector cable are working.
4.	PLC compile If PLC compile succeeds, control passes to the next step; if not, download exits with error(s).	The PLC Compiler verifies that the PLC hardware supports all program instructions, ranges, and structure. Click the PLC button from the Information area of the PLC menu to view the first compile error found.
5.	Program is in CPU permanent memory and ready to be executed in RUN mode.	Fatal Errors (Page 863) and non-fatal run-time errors (Page 860) are accessible from the Information area of the PLC menu.

If the download attempt produces compiler errors or download errors, correct the errors and reattempt the download.

See also

Uploading project components (Page 83)

See also

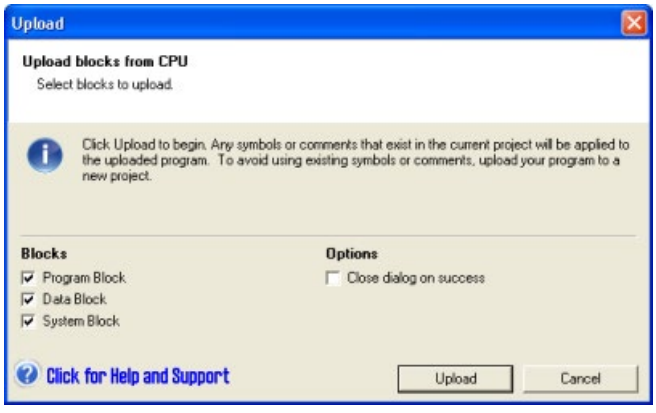
Hardware troubleshooting guide (Page 658)

4.3.2 Uploading project components

To upload project components from the PLC to a STEP 7-Micro/WIN SMART program editor, follow these steps:

1. Ensure that your network hardware and PLC connector cable (Ethernet (Page 29) or RS485 (Page 32)) are working, and that PLC communication is operating properly (Page 658).
2. To upload all project components, click the Upload button from the Transfer section of the File or PLC menu ribbon strip, or press the shortcut key combination **CTRL+U.**

3. To upload selected project components, click the down arrow under the Upload button, and then select the specific project component you want to upload (Program Block, Data Block, or System Block).
4. If you see a Communications dialog, select the Communication Interface and the Ethernet IP address or RS485 network address of the PLC from which you want to upload.
5. From the Upload dialog, you can change your selection for which blocks to upload if you choose.
6. Optionally click the "Close dialog on success" check box if you want the dialog to automatically close after a successful upload
7. Click the "Upload" button to start the upload.



STEP 7-Micro/WIN SMART copies the complete program or program components that you selected for uploading from the PLC to the currently open project. The status icon indicates informational messages, or whether potential problems or errors occurred with the upload. The status message provides specific results of the operation.

If the upload is successful, you can save the uploaded program, or make further changes. The PLC does not contain symbol or status chart information; hence, you cannot upload a symbol table or status chart.

Note

Uploading into a new project is a risk-free way to capture the program block, system block, and/or data block information. Since the project is empty, you cannot inadvertently destroy data. If you want to make use of material from a status chart or symbol table that are in another project, you can always open a second instance of STEP 7-Micro/WIN SMART and copy that information in from the other project file (Page 102).

Uploading into an existing project is useful if you want to overwrite all modifications that have been made to the program since it was downloaded (Page 80) to the PLC. Uploading into an existing project does, however, overwrite any additions or modifications you have made to the project. Use this option, only if you want to completely overwrite your STEP 7-Micro/WIN SMART project with the project stored in the PLC.

STEP 7-Micro/WIN SMART does not upload comments, but if you currently have a program with comments open in the program editor, the comments are retained. Take care if uploading over an existing project and use this method only if the projects are similar.

4.3.3 Types of storage

The CPU provides a variety of features to ensure that your user program and data are properly retained.

- Retentive memory: selectable areas of memory that remain unchanged over a power cycle. Retentive memory can be configured in the system data block. V, M, and current values to timers and counters are the only memory areas that can be configured to be retentive.
- Permanent memory: memory used to store the program block, data block, system block, forced values, as well as values configured to be retentive.
- Memory card: removable microSDHC card for standard CPUs that you can use for the following purposes:
  - To store the projects blocks as a program transfer card (Page 87)
  - To completely erase the PLC as a restore-to-factory-defaults card (Page 160)
  - To update the PLC and expansion module firmware as a firmware update card (Page 85)

4.3.4 Using a memory card

Using a memory card

The standard S7-200 SMART CPUs support the use of a microSDHC card for:

- User program transfer (Page 87)
- Reset CPU to factory default condition (Page 160)
- Firmware update of the CPU and attached expansion modules as supported

You can use any standard, commercial microSDHC card with a capacity in the range 4GB to 16GB.

The following CPU behaviors are common, regardless of the memory card usage:

1. Inserting a memory card into a CPU in RUN mode causes the CPU to automatically transition to STOP mode.
2. A CPU cannot advance to RUN mode if a memory card is inserted.
3. Memory card evaluation is performed only after a CPU power-up or warm restart. Therefore, program transfer and firmware update can only occur after a CPU power-up or warm restart.
4. The memory card can be used to store files and folders not related to program transfer and firmware update usage as long as their names do not conflict with the file and folder names used for program transfer and firmware update usage.

⚠ WARNING

Verify that the CPU is not actively running a process before installing the memory card.

Installing the memory card will cause the CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Before inserting the memory card, always ensure that the CPU is offline and in a safe state.

Program transfer card

You can use a memory card to transfer user program content into the CPU permanent memory, completely or partially replacing content already in the load memory.

To be used for program transfer purposes, the memory card is organized as follows:

Table 4- 21 Memory card used for program transfer card

At the root level of the card	
File: S7_JOB.S7S	A text file containing the word TO_ILM
Folder: SIMATIC.S7S	A folder containing user program files to be transferred to the CPU

Reset to factory defaults card

You can use a memory card to erase all retained data, putting the CPU back into a factory default condition.

To be used for reset to factory default (Page 160) purposes, the memory card is organized as follows:

Table 4- 22 Memory card used to reset to factory defaults

At the root of the card	
File: S7_JOB.S7S	A text file containing the word RESET_TO_FACTORY

Firmware update card

You can use a memory card to update the firmware in a CPU and any connected expansion modules.

The file and folder organization of a firmware update memory card is as follows:

Table 4- 23 Memory card used for firmware update purposes

At the root level of the card	
File: S7_JOB.S7S	A text file containing the word FWUPDATE
Folder: FWUPDATE.S7S	A folder containing update files (.upd) for each device to be updated

After power-up, if the CPU detects the presence of a memory card, it locates and opens the S7_JOB.SYS file on the card. If the CPU discovers the FWUPDATE string in that file, then the CPU enters a firmware update sequence.

The CPU examines each update file (.upd) in the FWUPDATE.S7S folder and if the order ID contained in the update file name matches the order ID (MLFB) of a connected device (CPU, expansion module or signal board), then the CPU updates the firmware of that device with the firmware content contained within the update file.

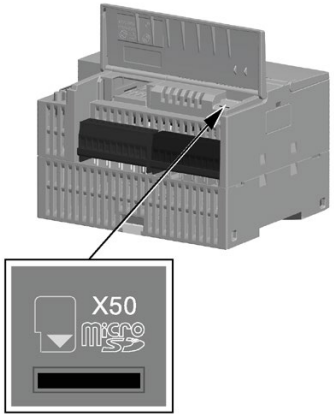
Note

Firmware update from STEP 7-Micro/WIN SMART

You can also perform a firmware update from STEP 7-Micro/WIN SMART using the RS485 port. Especially for CPU models that do not have a memory card, this method is valuable. Refer to the PLC menu section of the STEP 7-Micro/WIN SMART online help for instructions.

4.3.5 Inserting a memory card in a standard CPU

Table 4- 24 Inserting and removing a memory card in a standard CPU

Task	Procedure
	Follow the steps below to insert the microSDHC memory card into the CPU. 1. Open the bottom terminal block connector cover. 2. Insert the microSDHC memory card in the memory card slot (marked X50) located above the terminal block connectors. 3. Replace the terminal block connector cover after inserting the card to ensure that the card is secure. Follow the steps below to remove the microSDHC memory card from the CPU. 1. Open the bottom terminal block connector cover. 2. Grasp the microSDHC memory card from the CPU and pull it out of the card slot (marked Micro-SD X50). 3. Replace the bottom terminal block cover.

4.3.6 Transferring your program with a memory card

The standard S7-200 SMART CPU models support standard, commercial microSDHC cards with a capacity ranging from 4 GB to 16 GB using the FAT32 file system format. You can use a microSDHC card as a program transfer card for portable storage for your program and project data.

⚠ WARNING

Verify that the CPU is not running a process before inserting the memory card.

Inserting a memory card into a CPU in RUN mode causes the CPU to automatically transition to STOP mode.

Inserting a memory card into a running CPU can cause disruption to process operation, possibly resulting in death or severe personal injury.

Always ensure that the CPU is in STOP mode (Page 41) prior to inserting a memory card.

Creating a program transfer memory card on the PLC

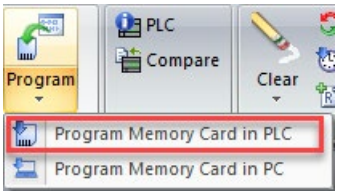
Note

STEP 7-Micro/WIN SMART first erases any SIMATIC content on the card prior to transferring the program to the card. Any other data on the card that you've stored using a card reader and Windows Explorer is left undisturbed.

Note also that you cannot change the CPU to RUN mode if a memory card is inserted.

To program the memory card as a program transfer card, follow these steps:

1. Ensure that your network hardware and PLC connector cable are working, the CPU is powered on and in STOP mode, and that PLC communication is operating properly (Page 30).
2. If not already inserted, insert a microSDHC memory card in the CPU. You can insert or remove the memory card while the CPU is powered on.
3. Download (Page 40) the program to the PLC, if not already downloaded.
4. Click the down arrow under the Program button in the PLC menu ribbon strip, and then select "Program Memory Card in PLC".



5. Select which of the following blocks (or all) to store on the memory card:
   - Program block
   - Data block
   - System Block (PLC configuration)

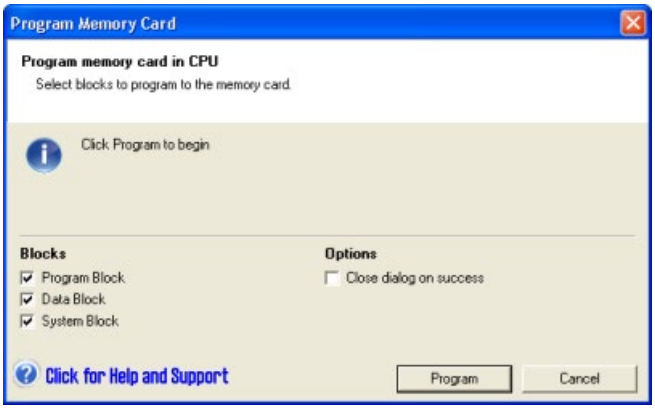
Note

Always download system blocks together with program blocks.

When the program has no password, you can download program blocks without system blocks.

You can download the data blocks separately.

6. Click the Program button.

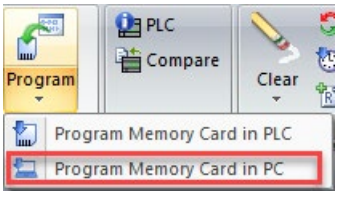


7. Enter the password (Page 138) if one is required for programming the memory card.

Creating a program transfer memory card on the PC

To save the program on the computer, follow the steps:

1. Click the down arrow under the Program button and then select "Program Memory Card in PC".



2. Select which of the following blocks (or all) to store on the memory card:
   - Program block
   - Data block
   - System Block (PLC configuration)

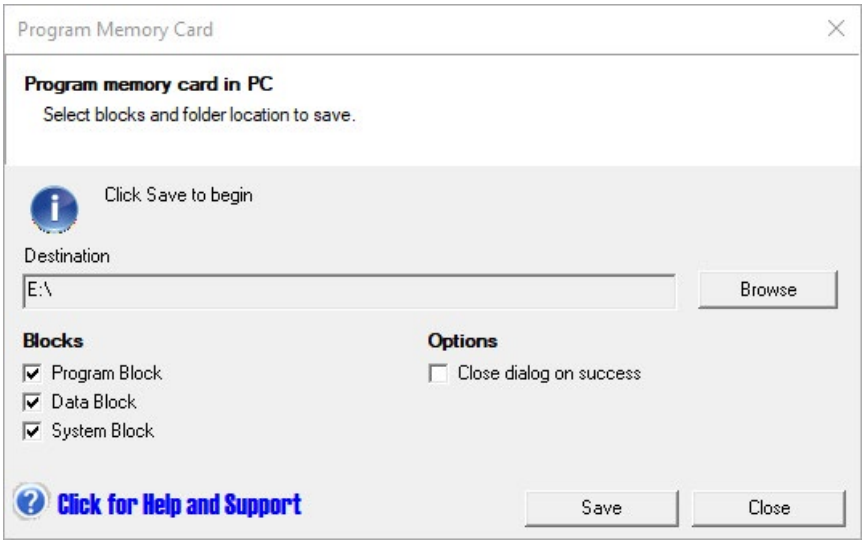
Note

Always download the system blocks with program blocks together.

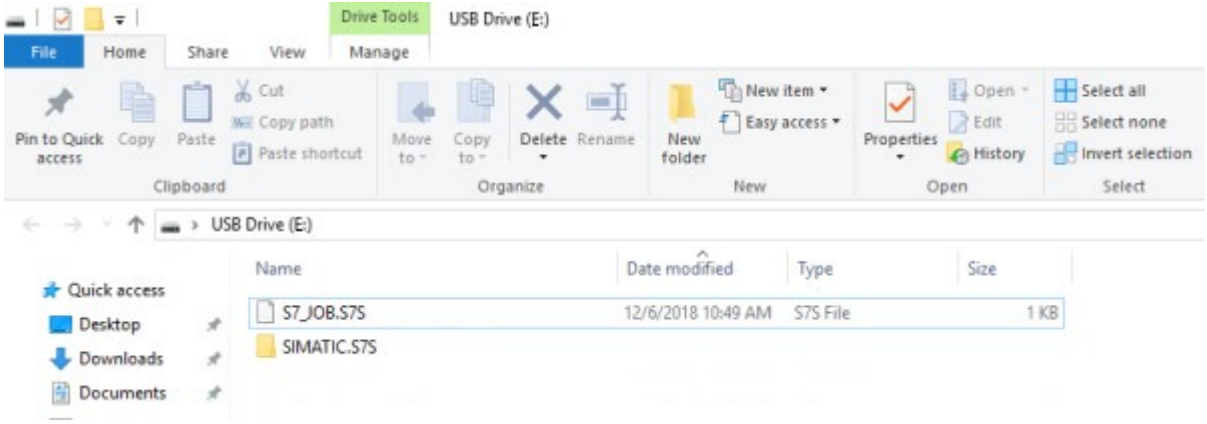
When the program has no password, you can download program blocks without system blocks.

You can download the data blocks separately.

3. Click "Browse" button and navigate to the root folder of the SD card.



4. Click the Save button and two files are generated in your SD card as follows:



Note

The PLC might fail to compile the program restored from an SD card by "Program Memory Card in PC". To ensure your program/configuration is valid, Siemens recommends that you connect to a PLC and download the program at least once.

Note

The memory card created in PC doesn't contain any force configuration in the program.

Restoring the program from a program transfer memory card

To copy the contents of the program transfer card to the PLC, you must cycle the power to the CPU with the program transfer card inserted. The CPU then performs the following tasks:

1. Clears RAM
2. Copies the user program, the system block (PLC configuration), and the data block from the memory card to CPU permanent memory.

While the copy operation is in progress, the STOP and RUN LEDs on the S7-200 SMART CPU alternately flash. When the S7-200 SMART CPU completes the copy operation, the LEDs stop flashing.

Note

Program transfer card compatibility

Restoring a program transfer card that you created on a different CPU model might fail due to the differences in model types. During the restore process, the CPU validates the following characteristics of the program content stored on the memory card:

- Size of program block
- Size of V memory specified in the data block
- Quantity of onboard digital I/O configured in the system block (Page 129)
- Each retentive range that is configured in the system block
- Expansion module and signal board configurations in the system block
- Axis of Motion configurations in the system block
- Forced memory locations

Note

In addition to using a memory card as a program transfer card, you can also create a reset-to-factory-defaults memory card (Page 160).

4.3.7 Restoring data after power on

The CPU performs the following actions after a power cycle:

- Restores the program block and the system block from permanent memory
- Restores the retentive memory assignments
- Restores the non-retentive portions of V memory from the contents of the data block in permanent memory
- Clears the non-retentive portions of other memory areas

4.4 Changing the operating mode of the CPU

The CPU has two modes of operation: STOP mode and RUN mode. The status LEDs on the front of the CPU indicates the current mode of operation. In STOP mode, the CPU is not executing the program, and you can download program blocks. In RUN mode, the CPU is executing the program; however, you can download program blocks.

Placing the CPU in RUN mode

1. Click the "RUN" button on either the PLC menu ribbon strip or on the program editor toolbar:
2. When prompted, click "OK" to change the operating mode of the CPU.

You can monitor the program in STEP 7-Micro/WIN SMART by clicking the "Program Status" button from the "Debug" menu ribbon strip, or from the program editor toolbar.
STEP 7-Micro/WIN SMART displays the values for the instructions.

Placing the CPU in STOP mode

To stop the program, click the "STOP" button and acknowledge the prompt to place the CPU in STOP mode. You can also place a STOP instruction (Page 337) in your program logic to put the CPU in STOP mode.

4.5 Status LEDs

The CPU and EM use LEDs to provide information about operational status.

CPU status LEDs

The CPU provides the following LED status indicators:

State	LED Behavior	Description
STOP	STOP: on RUN, ERROR: off	Applicable when the CPU is in STOP Mode
STOP with Force	RUN: off STOP: blink at 1 Hz rate ERROR: off	Applicable when the CPU is in STOP mode and a value is forced
STOP with PROFINET	STOP: on RUN: off ERROR: blink at 1 Hz rate	Applicable when the CPU is in STOP mode and any configured PROFINET device loses connection or gets alarms.
RUN	RUN: on STOP, ERROR: off	Applicable when the CPU is in RUN Mode
RUN with Force	RUN: on STOP: blink at 1 Hz rate ERROR: off	Applicable when the CPU is in RUN mode and a value is forced

State	LED Behavior	Description
RUN with PROFINET	STOP: off RUN: on ERROR: blink at 1 Hz rate	Applicable when the CPU is in RUN mode and any configured PROFINET device loses connection or gets alarms.
Busy	STOP, RUN: blink out of phase at 2 Hz rate ERROR: off	Applicable when after card evaluation during power-up or restart, a memory card operation is in progress or when a restart operation is in progress
Memory Card Inserted	STOP: blinks at 2 Hz rate RUN, ERROR: off	Applicable when someone inserts a memory card into a powered-on CPU
Memory Card OK	STOP: blinks at 2 Hz rate RUN, ERROR: off	Applicable when after card evaluation during power-up or restart, a memory card operation completes successfully
Memory Card Error	STOP, ERROR: blink in phase at 2 Hz rate RUN: off	Applicable when after card evaluation during power-up or restart, a memory card operation terminates with an error
Defective	STOP, ERROR: on RUN: off	Applicable when the CPU is in Defective Mode
Ping	STOP, RUN: blink out of phase at 2 Hz rate ERROR: blink in phase with RUN	Applicable when the CPU receives a Signal DCP control request (Flash LEDs)

EM status LEDs

The expansion modules (EM) provide the following LED status indicators:

Each digital EM provides a DIAG LED that indicates the status of the module:

- Green indicates that the module is operational
- Red indicates that the module is defective or non-operational

Each analog EM provides an I/O Channel LED for each of the analog inputs and outputs.

- Green indicates that the channel has been configured and is active
- Red indicates an error condition of the individual analog input or output

In addition, each analog EM provides a DIAG LED that indicates the status of the module:

- Green indicates that the module is operational
- Red indicates that the module is defective or non-operational

The EM DP01 has a different set of LEDs. See LED status indicators for the EM DP01 PROFIBUS DP (Page 456).

The EM detects the presence or absence of power to the module (field-side power, if required).

Table 4- 25 Status LEDs for a expansion module (EM)

Description	DIAG (Red / Green)	I/O Channel (Red / Green)
Field-side power is off *	Flashing red	Flashing red
Not configured or update in progress	Flashing green	Off
Module configured with no errors	On (green)	On (green)
Error condition	Flashing red	-
I/O error (with diagnostics enabled)	-	Flashing red
I/O error (with diagnostics disabled)	-	On (green)

* Status is only supported on analog signal modules.

Programming concepts

5

5.1 Guidelines for designing a PLC system

There are many methods for designing a PLC system. The following general guidelines can apply to many design projects. Of course, you must follow the directives of your own company's procedures and the accepted practices of your own training and location.

Partition your process or machine

Divide your process or machine into sections that have a level of independence from each other. These partitions determine the boundaries between controllers and influence the functional description specifications and the assignment of resources.

Create the functional specifications

Write the descriptions of operation for each section of the process or machine. Include the following topics: I/O points, functional description of the operation, states that must be achieved before allowing action for each actuator (such as solenoids, motors, and drives), description of the operator interface, and any interfaces with other sections of the process or machine.

Design the safety circuits

Identify equipment requiring hard-wired logic for safety. Control devices can fail in an unsafe manner, producing unexpected startup or change in the operation of machinery. Where unexpected or incorrect operation of the machinery could result in physical injury to people or significant property damage, consideration should be given to the use of electro-mechanical overrides which operate independently of the CPU to prevent unsafe operations.

The following tasks should be included in the design of safety circuits:

- Identify improper or unexpected operation of actuators that could be hazardous.
- Identify the conditions that would assure the operation is not hazardous, and determine how to detect these conditions independently of the CPU.
- Identify how the CPU and I/O affect the process when power is applied and removed, and when errors are detected. This information should only be used for designing for the normal and expected abnormal operation, and should not be relied on for safety purposes.
- Design manual or electro-mechanical safety overrides that block the hazardous operation independent of the CPU.
- Provide appropriate status information from the independent circuits to the CPU so that the program and any operator interfaces have necessary information.
- Identify any other safety-related requirements for safe operation of the process.

Specify the operator stations

Based on the requirements of the functional specifications, create drawings of the operator stations. Include the following items:

- Overview showing the location of each operator station in relation to the process or machine
- Mechanical layout of the devices, such as display, switches, and lights, for the operator station
- Electrical drawings with the associated I/O of the CPU or expansion module

Create the configuration drawings

Based on the requirements of the functional specification, create configuration drawings of the control equipment. Include the following items:

- Overview showing the location of each CPU in relation to the process or machine
- Mechanical layout of the CPU and expansion I/O modules (including cabinets and other equipment)
- Electrical drawings for each CPU and expansion I/O module (including the device model numbers, communications addresses, and I/O addresses)

Create a list of symbolic names (optional)

If you choose to use symbolic names for addressing, create a list of symbolic names for the absolute addresses. Include not only the physical I/O signals, but also the other elements to be used in your program.

5.2 Elements of the user program

A program organizational unit (POU) is composed of executable code and comments. The executable code consists of a main program and any subroutines or interrupt routines. The code is compiled and downloaded to the CPU. You can use the program organizational units (main program, subroutines, and interrupt routines) to structure your user program.

- The main body of the user program contains the instructions that control your application. The CPU executes these instructions sequentially, once per scan cycle.
- Subroutines are optional elements of your program which are executed only when called: by the main program, by an interrupt routine, or by another subroutine. Subroutines are useful in cases where you want to execute a function repeatedly. Rather than rewriting the logic for each place in the main program where you want the function to occur, you can write the logic once in a subroutine and call the subroutine as many times as needed during the main program. Subroutines provide several benefits:
  - Using subroutines reduces the overall size of your program.
  - Using subroutines decreases your scan time because you have moved the code out of the main program. The CPU evaluates the code in the main program every scan cycle, whether the code is executed or not, but the CPU evaluates the code in the

subroutine only when you call the subroutine, and does not evaluate the code during the scans in which the subroutine is not called.

– Using subroutines creates code that is portable. You can isolate the code for a function in a subroutine, and then copy that subroutine into other programs with little or no rework.

Note

Using V memory addresses can limit the portability of your subroutine, because it is possible for V memory address assignment from one program to conflict with an assignment in another program. Subroutines that use the local variable table (L memory) for all address assignments, by contrast, are highly portable because there is no concern about address conflicts between the subroutine and another part of the program when using local variables.

- Interrupt routines are optional elements of your program that react to specific interrupt events. You design an interrupt routine to handle a pre-defined interrupt event. Whenever the specified event occurs, the CPU executes the interrupt routine.

  The interrupt routines are not called by your main program. You associate an interrupt routine with an interrupt event, and the CPU executes the instructions in the interrupt routine only on each occurrence of the interrupt event.

  Note

  Because it is not possible to predict when the CPU might generate an interrupt, it is desirable to limit the number of variables that are used both by the interrupt routine and elsewhere in the program.

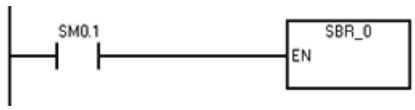
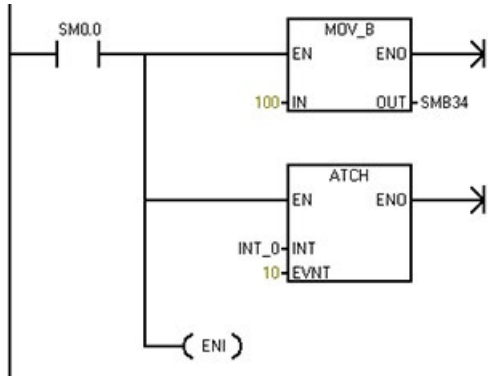
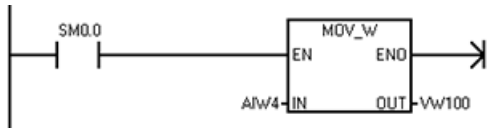
  Use the local variable table of the interrupt routine to ensure that your interrupt routine uses only the temporary memory and does not overwrite data used somewhere else in your program.

  There are a number of programming techniques you can use to ensure that data is correctly shared between your main program and the interrupt routines. Refer to the descriptions of the Interrupt instructions (Page 307).

- Other blocks contain information for the CPU. You can choose to download these blocks when you download your program:
  – System Block: The system block allows you to configure different hardware options for the CPU.
  – Data Block: The DB stores the initial values for different variables (V memory) used by your program.

The following example shows a program that includes a subroutine and an interrupt routine. This sample program uses a timed interrupt for reading the value of an analog input every 100 ms.

Table 5- 1 Sample program with a subroutine and an interrupt routine

Main		**Network 1** **LD SM0.1** **CALL SBR_0**	On first scan, call subroutine 0.
SBR 0		**Network 1** **LD SM0.0** **MOVB 100, SMB34** **ATCH INT_0, 10** **ENI**	Set the interval to 100 ms for the timed interrupt. Enable interrupt 0.
INT 0		**Network 1** **LD SM0.0** **MOVW AIW4,VW100**	Sample the value of analog input AI4.

5.3 Creating your user program

The STEP 7-Micro/WIN SMART user interface provides a convenient working space for creating your user project program. (STEP 7-Micro/WIN SMART projects are files with a .smart file name extension.) To open the user interface, double-click the STEP 7-Micro/WIN SMART icon, or select "STEP 7-MicroWIN SMART" from the "SIMATIC" element on the "Start" menu.

5.3.1 STEP 7-Micro/WIN SMART compatibility

Siemens recommends that you don't open the projects created by the current STEP 7-Micro/WIN with an earlier version of STEP-7 MicroWIN. For example, use STEP 7-Micro/WIN 2.3 open projects with PROFINET configuration created by STEP-7 MicroWIN 2.4 may make the program crash.

5.3.2 Earlier versions of STEP 7-Micro/WIN projects

To work with a project that was created in STEP 7-Micro/WIN Version 4.0 or greater, follow these steps:

- Click the Open button from the Operations area of the File menu ribbon strip and select the desired project.

- Correct program as needed.

You cannot open projects that were created with a version earlier than STEP 7-Micro/WIN Version 4.0. If you attempt to open such a project, STEP 7-Micro/WIN SMART informs you that you cannot open the project.

Note

Opening projects created with an older version

- Projects from earlier versions of STEP 7-Micro/WIN (.mwp files) might contain one or more logical constructs that STEP 7-Micro/WIN SMART does not support. STEP 7-Micro/WIN SMART omits the instructions that it does not support when opening the project. You must take care to examine your project and redesign sections where STEP 7-Micro/WIN SMART omitted program logic.
- STEP 7-Micro/WIN SMART ignores the system block of the old project and uses a default system block for the opened project.
- STEP 7-Micro/WIN SMART omits all wizard-generated program blocks of the old project.
- You cannot use Open to open a project that resides on a PLC. The project file must reside on your personal computer/ programming device.
- You can only open one project for each instance of STEP 7-Micro/WIN SMART. You must run two instances of STEP 7-Micro/WIN SMART to have two projects open at the same time. When two instances are open, you can copy and paste LAD/ FBD program elements and STL text from one project to the other.
- You can define a default project folder where you want STEP 7-Micro/WIN SMART to open and save new projects.

WARNING

Risks with STEP 7-Micro/WIN Version 4.0 or greater (.mwp files) with absolute special memory (SM) addressing

You can open a program (.mwp file) from an earlier version of STEP 7-Micro/WIN in STEP 7-Micro/WIN SMART. If that program uses symbolic special memory (SM) addressing, then insert the System Symbol table (Page 108) in your project. The symbols map correctly to the current SM addresses. If, however, the program uses absolute SM addressing, those absolute SM addresses might no longer exist.

Programs based on inconsistent definitions of SM addresses can result in unexpected machine or process operation. Unexpected machine or process operation can cause death or serious injury to personnel, and/or damage to equipment.

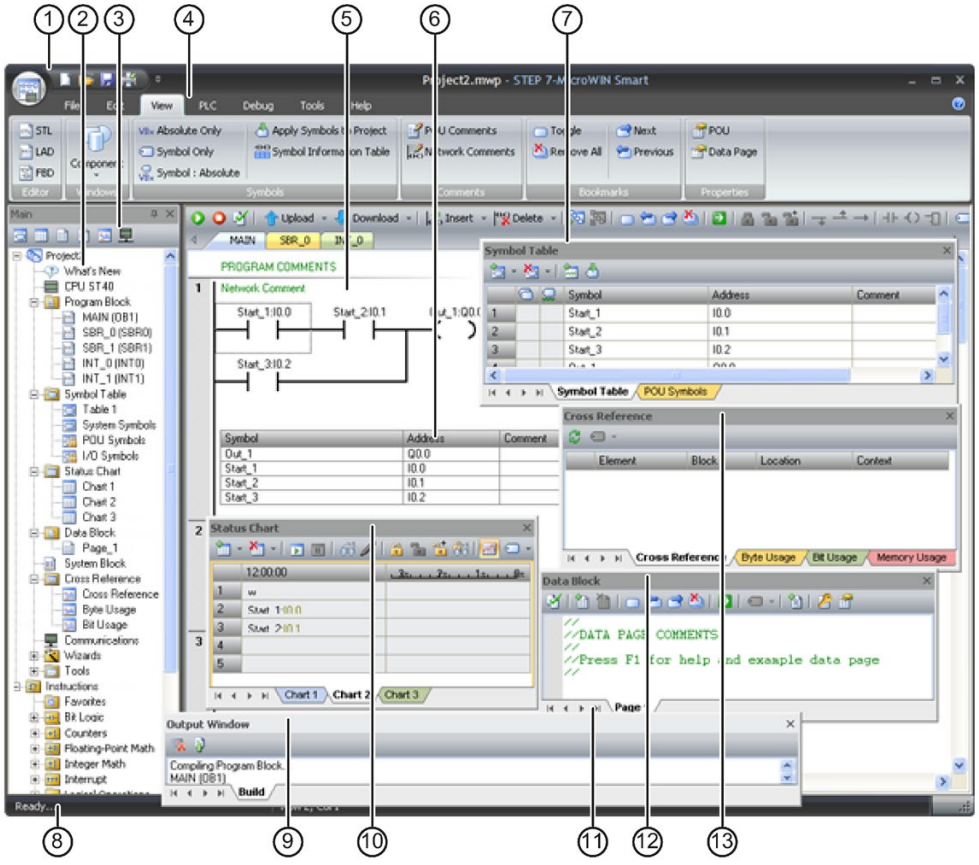
If you open an .mwp file in STEP 7-Micro/WIN SMART, delete the "S7-200 Symbols" table and insert the "System Symbols" table. The symbols in the former .mwp program map to the current SM address scheme. Convert any absolute SM addresses to use the corresponding symbol name.

See also

SM (Special Memory) overview (Page 865)

5.3.3 Using STEP 7-Micro/WIN SMART user interface

The STEP 7-Micro/WIN SMART user interface appears below. Note that each editing window can be docked or floated and arranged on the screen as you choose. You can display each window separately, as shown below, or you can combine windows such that each one is accessible from a separate tab:



① Quick access toolbar (Page 102)
② Project tree (Page 102)
③ Navigation bar (Page 102)
④ Menus (Page 102)
⑤ Program editor (Page 102)
⑥ Symbol information table (Page 108)
⑦ Symbol table (Page 108)
⑧ Status bar (Page 102)
⑨ Output window (Page 102)
⑩ Status chart (Page 652)
⑪ Variable table (Page 112)
⑫ Data block (Page 106)
⑬ Cross reference (Page 647)

5.3.4 Using STEP 7-Micro/WIN SMART to create your programs

Quick access toolbar

The quick access toolbar appears just above the menu tabs. The quick access file button provides quick and easy access to most of the functions of the File menu, and to recent documents. The other buttons on the quick access toolbar are for the file functions New, Open, Save, and Print.

Project tree

The project tree displays all of the project objects and the instructions for creating your control program. You can drag and drop individual instructions from the tree into your program, or you can double-click an instruction to insert it at the current location of the cursor in the program editor.

The Project tree organizes your project:

- Right-click the project to set a project password or set project options
- Right-click the Program Block folder to insert new subroutines and interrupt routines.
- Open the Program Block folder and right-click a POU to open the POU, edit its properties, password-protect it, or rename it.
- Right-click a Status Chart or Symbol Table folder to insert new charts or tables.
- Open the Status Chart or Symbol Table folder and right-click the icon in the instruction tree or double-click the appropriate POU tab to open it, rename it, or delete it.

Note

Increased security for project, POU, and data block (data page) passwords

STEP 7-Micro/WIN SMART V2.3 increased the security for passwords from previous versions. If you are working with a project that you created with a previous version of STEP 7-Micro/WIN SMART, re-enter your passwords to activate the increased security.

Navigation bar

The Navigation bar appears at the top of the project tree and provides quick access to objects on the project tree. Clicking a navigation bar button is equivalent to expanding the project tree and clicking the same selection. The navigation bar presents groups of icons for accessing different programming features of STEP 7-Micro/WIN SMART.

Menu ribbon strips

STEP 7-Micro/WIN SMART displays a menu ribbon strip for each menu. You can minimize the menu ribbon strip to save space by right-clicking in the menu ribbon strip area and selecting "Minimize the Ribbon".

Program editor

The program editor contains the program logic and a variable table where you can assign symbolic names for temporary program variables. Subroutines and interrupt routines appear as tabs at the top of the program editor window. Click the tabs to move between the subroutines, interrupts, and the main program.

STEP 7-Micro/WIN SMART provides three editors for creating your program:

- Ladder logic (LAD)
- Statement list (STL)
- Function block diagram (FBD)

With some restrictions, programs written in any of these program editors can be viewed and edited with the other program editors.

You can change the editor to LAD, FBD, or STL from the Editor section of the View menu ribbon strip. You can configure the default editor at startup from the Options button of the Settings area of the Tools menu ribbon strip.

Status bar

The status bar, which is located at the bottom of the main window, provides information on the editing mode or online status operations that you perform in STEP 7-Micro/WIN SMART.

Output window

The Output Window keeps a list of the most recently compiled POUs (Page 843) and any errors that occurred during the compilation. If you have the Program Editor window open as well as the Output Window, you can double-click an error message in the Output Window to scroll your program automatically to the network where the error is located.

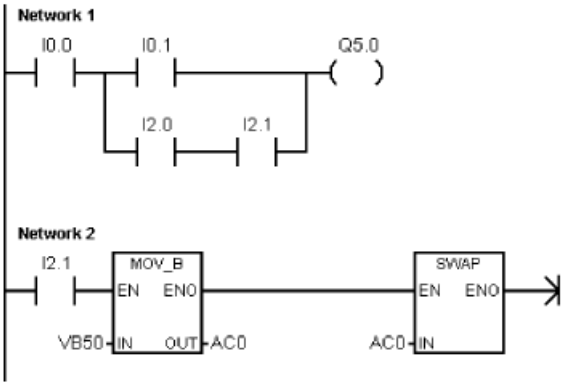
5.3.5 Using wizards to help you create your control program

STEP 7-Micro/WIN SMART provides the following wizards to make aspects of your programming easier and more automatic:

- High speed counter
- Motion
- PID
- PWM (Pulse Width Modulation)
- Text Display
- Get/Put
- Data Log (standard CPUs only)
- PROFINET

To start a wizard, select that wizard from the STEP 7-Micro/WIN SMART Tools menu ribbon strip or from the Wizards node in the project tree. You can press F1 when a wizard is displayed and get wizard details from the online Help system.

5.3.6 Features of the LAD editor



The LAD editor displays the program as a graphical representation similar to electrical wiring diagrams.

The LAD program emulates the flow of electric current from a power source through a series of logical input conditions that in turn enable logical output conditions.

A LAD program includes a left power rail that is energized. Contacts that are closed allow energy to flow through them to the next element, and contacts that are open block that energy flow. The logic is separated into networks. The program is executed one network at a time, from left to right and then top to bottom as dictated by the program.

The various instructions are represented by graphic symbols and include three basic forms:

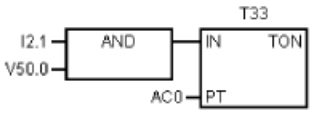
- Contacts represent logic input conditions such as switches, buttons, or internal conditions.
- Coils usually represent logic output results such as lamps, motor starters, interposing relays, or internal output conditions.
- Boxes represent additional instructions, such as timers, counters, or math instructions.

Consider these main points when you select the LAD editor:

- Ladder logic is easy for beginning programmers to use.
- Graphical representation is easy to understand and is popular around the world.
- You can always use the STL editor to display a program created with the SIMATIC LAD editor.

5.3.7 Features of the FBD editor

The FBD editor displays the program as a graphical representation that resembles common logic gate diagrams. There are no contacts and coils as found in the LAD editor, but there are equivalent instructions that appear as box instructions.



FBD does not use the concept of left and right power rails; therefore, the term "logic flow" is used to express the analogous concept of control flow through the FBD logic blocks.

The logic "1" path through FBD elements is called logic flow. The origin of a logic flow input and the destination of a logic flow output can be assigned directly to an operand.

The program logic is derived from the connections between these box instructions. That is, the output from one instruction (such as an AND box) can be used to enable another instruction (such as a timer) to create the necessary control logic. This connection concept allows you to solve a wide variety of logic problems.

Consider these main points when you select the FBD editor:

- The graphical logic gate style of representation is good for following program flow.
- You can always use the STL editor to display a program created with the SIMATIC FBD editor.

5.3.8 Features of the STL editor

The STL editor displays the program as a text-based language. The STL editor allows you to create control programs by entering the instruction mnemonics. The STL editor also allows you to create programs that you could not otherwise create with the LAD or FBD editors. This is because you are programming in the native language of the CPU, rather than in a graphical editor where some restrictions must be applied in order to draw the diagrams correctly. As shown in the following example, this text-based concept is very similar to assembly language programming.

Table 5- 2 Sample STL user program

LD	I0.0	// Read one input (I0.0).
A	I0.1	// AND with another input (Q1.0).
=	Q1.0	// Write the value to an output 1.

The CPU executes each instruction in the order dictated by the program, from top to bottom, and then restarts at the top.

STL uses a logic stack to resolve the control logic. You insert the STL instructions for handling the stack operations.

Consider these main points when you select the STL editor:

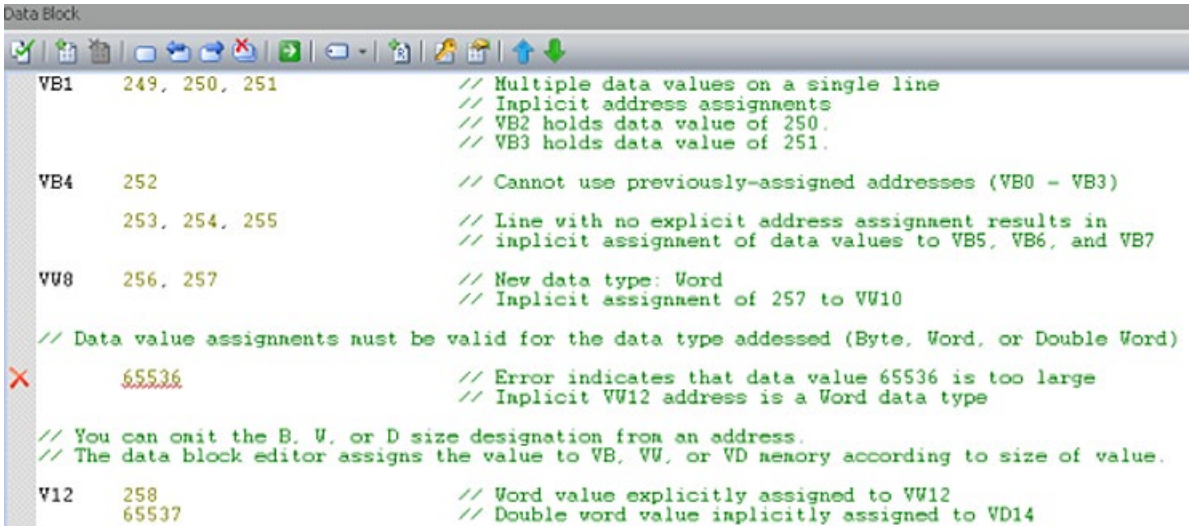
- STL is most appropriate for experienced programmers.
- STL sometimes allows you to solve problems that you cannot solve very easily with the LAD or FBD editor.
- While you can always use the STL editor to view or edit a program that was created with the LAD or FBD editors, the reverse is not always true. You cannot always use the LAD or FBD editors to display a program that was written with the STL editor.

5.4 Data block (DB) editor

The data block allows you to assign constants (Page 75) (either numeric values or character strings) to specific locations of V memory. You can make value assignments to byte (V or VB), word (VW), or double word (VD) addresses. You can also enter optional comments, preceded by // double forward slashes.

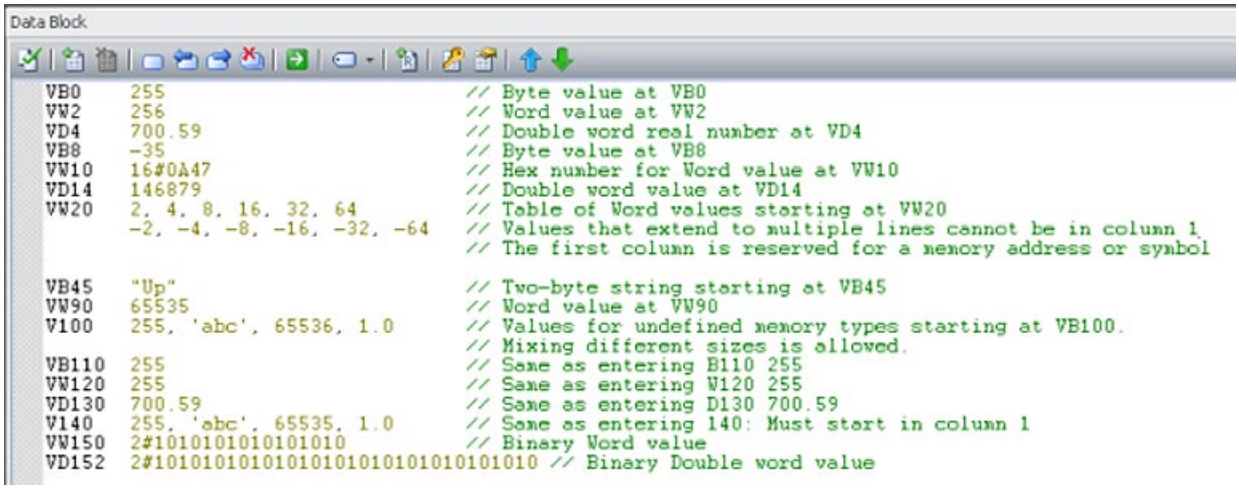
- The first line of the data block must have an explicit address assignment. You can use a memory address (absolute address) or a symbol name from the symbol table (Page 108) that you have previously assigned to an address (symbolic address).
- Subsequent lines can have explicit or implicit address assignments. An implicit address assignment is made by the editor when you type multiple data values after a single address assignment, or type a line that contains only data values. The editor assigns an appropriate amount of V memory based on your previous address allocations and the size (byte, word, or double word) of the data value(s).
- The data block editor is a free-form text editor; however, it expects an address or symbol name in the first position. If you are continuing an implicit data value entry, enter at least one space in the address position before entering the implicit data value assignment. After you finish typing a line and press the ENTER key, the data block editor formats the line (aligns columns of addresses, data, and comments; capitalizes V memory addresses) and redisplays it. The data block editor accepts uppercase or lowercase letters and allows commas, tabs, or spaces to serve as separators between addresses and data values.
- Pressing CTRL–ENTER after completing an assignment line auto-increments the address to the next available address.

Example: Data block page

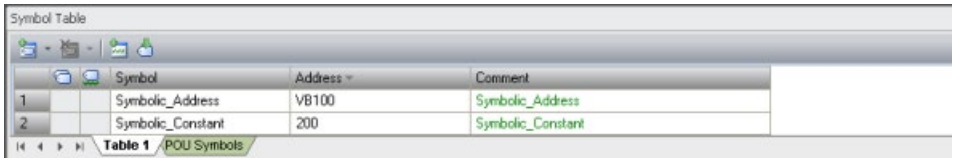


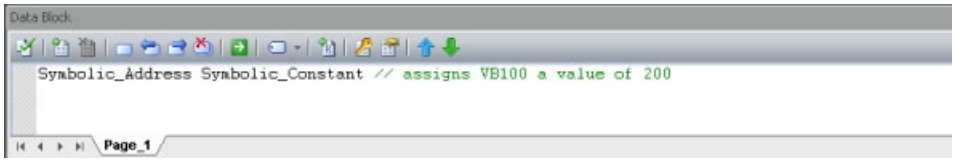
Note: Enter a space before the data values on the lines where you enter no explicit address.

Example: Direct address and number values



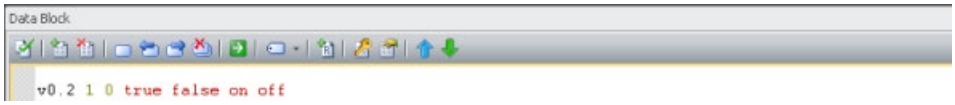
Example: Symbolic address and symbolic number assignment

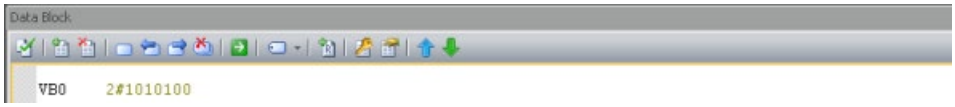




Example: Alternate binary entry methods and resultant binary assignment

You can enter values of 1 or 0 for binary assignments, or "true", "false", "on" or "off" (in either lower, upper, or mixed case). The data block editor interprets your entry and shows the resultant binary assignment.





5.5 Symbol table

A symbol is a symbolic name you assign to a memory address or a constant. You can create symbol names for the following memory types: I, Q, M, SM, AI, AQ, V, S, C, T, HC. Symbols defined in the symbol table are global in scope. You can use your defined symbols in all of the Program Organizational Units (Page 96) (POUs) of your program. If you make a variable name assignment in a variable table (Page 112), the variable is local in scope. It only applies to the POU where you defined it. This type of symbol is referred to as a "local variable" to differentiate it from symbols that are global in scope. You can define symbols either before or after you create your program logic.

⚠WARNING

Risks with STEP 7-Micro/WIN Version 4.0 or greater (.mwp files) with absolute special memory (SM) addressing

You can open a program (.mwp file) from an earlier version of STEP 7-Micro/WIN in STEP 7-Micro/WIN SMART. If that program uses symbolic special memory (SM) addressing, then insert the System Symbol table in your project. The symbols map correctly to the current SM addresses. If, however, the program uses absolute SM addressing, those absolute SM addresses might no longer exist.

Programs based on inconsistent definitions of SM addresses can result in unexpected machine or process operation. Unexpected machine or process operation can cause death or serious injury to personnel, and/or damage to equipment.

If you open an .mwp file in STEP 7-Micro/WIN SMART, delete the "S7-200 Symbols" table and insert the "System Symbols" table. The symbols in the former .mwp program map to the current SM address scheme. Convert any absolute SM addresses to use the corresponding symbol name.

Opening a symbol table

To open a symbol table from STEP 7-Micro/WIN SMART, use one of the following methods:

- Click the Symbol Table button on the navigation bar (Page 26).
- Select "Symbol Table" from the component drop down list in the Windows area of the View menu.
- Open the Symbol Table Folder in the project tree (Page 35); select a table name; then press Enter or double-click the table name.

System symbol table

You can also use symbols from the System symbol table in your project. The pre-defined table of system symbols provides access to commonly-used PLC special memory (Page 865) addresses.

If the System symbol table is missing from your project, follow these steps to insert it:

1. Right-click "Symbol Table" in the project tree
2. Select the "Insert > System Symbol Table" command from the shortcut menu.

Assigning symbols in the symbol table

To assign a symbol to an address or constant value, follow the procedure below:

1. Open the symbol table.
2. Type the symbol name (for example, Input1) in the Symbol column. The maximum number of characters in a symbol name is 23 single-byte characters.

   Note

   Until you assign an address or constant value to the symbol, it appears as an undefined symbol (green wavy underline). After you complete the Address column assignment, STEP 7-Micro/WIN SMART removes the green wavy underline.

   If you have selected to view both symbolic and absolute view of operands for your project, lengthy symbol names are truncated with a tilde (~) in the program editor. You can place your mouse pointer over the truncated name to see the entire name displayed in a tooltip.

3. Type the address or constant value (for example, VB0 or 123) in the Address column. Note that to assign a string constant to a symbol, you enclose the string constant in double quotation marks.
4. Optionally, type in a comment up to a maximum of 79 characters.

You can resize the width of the columns in the symbol table editor as needed.

Note

You can create multiple symbol tables; however, you cannot use the same symbol name more than once as a global symbol assignment.

By contrast, you can reuse symbol names in variable tables.

Syntax rules and indication of errors

STEP 7-Micro/WIN SMART indicates erroneous or incomplete symbol assignments with color and wavy underlining:

Symbol	Address
2name	I0.0
	VBB0
Begin	I0.2

Red text indicates invalid syntax.
A symbol cannot begin with a numeral.
VBB0 is an invalid address.
Begin is a reserved word and invalid as a symbol name.

Symbol	Address
Pump1	I0.0
Pump1	I0.0
SymConstant	1234
SymConstant	5678

A red wavy underline indicates invalid use.
Pump1 and SymConstant are duplicate symbol names.
I0.0 is a duplicate address.

Symbol	Address
Pump1	

A green wavy underline indicates an undefined symbol.
Pump1 has no address.

Observe the following syntax rules when defining symbols:

- Symbolic names can contain alphanumeric characters, underscores, and extended characters from ASCII 128 to ASCII 255. The first character cannot be a numeral.
- Use double quotation marks to enclose an ASCII constant string that you assign to a symbol name.
- Use single quotation marks to enclose an ASCII character constant in byte, word, or double word memory.
- Do not use keywords as symbol names.
- The maximum length for a symbol name is 23 characters.

Note

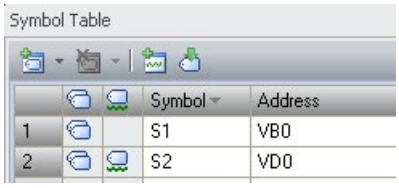
When you correct an erroneous symbol name or address, press the TAB key, ENTER key, or an ARROW key to complete the edited correction.

Indirect addressing

When referencing a symbol in the program editor, you can use indirect notation (& and *) with symbol names as with direct addresses. For more information about indirect addressing, see the topic on direct and indirect addressing.

Viewing overlapped and unused symbols

STEP 7-Micro/WIN SMART indicates overlapped symbols with the icon and unused symbols with the icon. In the symbol table below, symbols S1 and S2 overlap the VB0 memory address. Also, symbol S1 is not used in the project.



Inserting additional rows

Use one of the following methods to insert additional rows in the symbol table:

- Right-click a cell of the symbol table and select Insert>Row from the context menu. STEP 7-Micro/WIN SMART inserts a new row above the current location.
- From the Insert area of the Edit menu ribbon strip, select "Row". STEP 7-Micro/WIN SMART inserts a new row above the current location of the cursor in the symbol table.
- To insert a new row at the bottom of the Symbol Table, place the cursor in any cell of the last row and press the DOWN ARROW key.

Sorting a symbol table

You can sort a symbol table by the Symbol or by the Address column in either ascending or descending alphabetical order. In the Address column, numeric constants sort above string constants, which sort above addresses.

To sort a column, click either the Symbol or Address column header to sort by that value. To reverse the order of the sort, click the column again. STEP 7-Micro/WIN SMART displays an up or down arrow next to the column that is sorted to indicate the sort selection.

Note

You can print symbol tables from the Print area of the File menu ribbon strip.

You can view symbols on a network-by-network basis by displaying the symbol information table.

5.6 Variable table

A variable table allows you to define variables that are local to a specific POU. The following situations define when to use a local variable:

- You want to create portable subroutines that do not make references to absolute addresses or global symbols.
- You want to use interim variables (local variables declared as TEMP) to perform calculations in order to free up PLC memory.
- You want to define inputs and outputs for your subroutines.

If these descriptions do not fit your situation, you do not need to use local variables; you can make all of your symbolic values global by defining them in the symbol table (Page 108).

Understanding local variables

You can use the variable table of the program editor to assign variables that are unique to an individual subroutine or interrupt routine.

Local variables can be used as parameters that are passed in to a subroutine and can be used to increase the portability or reuse of a subroutine.

Each POU (Page 96) in your program has its own variable table, with 64 bytes of L memory (60 bytes if you are programming in LAD or FBD). These local variable tables allow you to define variables that are restricted in scope: a local variable is only valid inside the POU where it was created. By contrast, global symbols, which are valid in every POU, can only be defined in the symbol table. In cases where you use the same symbolic name (e.g., INPUT1) for a global symbol and a local variable, the local definition takes precedence inside the POU where the local variable has been defined and the global definition is used in the other POUs.

You assign a declaration type (TEMP, IN, IN_OUT, or OUT) and a data type, but not a memory address, when you make assignments in a Local Variable Table; the Program Editor automatically assigns memory locations in L memory for all local variables.

A variable table symbolic address assignment associates a symbol name with an L memory address, where the data value of concern is stored. The Local Variable Table does not support symbolic constants that assign a value directly to a symbol name (this is allowed in the Symbol\Global Variable tables).

Note

Local data values are not initialized to zero by the PLC. You must initialize the local variables that you use, in your program logic.

Declaration types for local variables

The type of local variable assignment you can make depends on the POU where you are making the assignment. The main program (OB1), interrupt routines, and subroutines can use temporary (TEMP) variables. Temporary variables are only available while the block is being executed and are then free to be overwritten when the block is completed.

Data values can be passed as parameters in and out of a subroutine as follows:

- If you want to pass a data value into a subroutine, then create a variable in the subroutine's variable table and specify its declaration type as IN.
- If you want to pass a data value established in the subroutine back to the calling routine, then create a variable in the subroutine's variable table and specify its declaration type as OUT.
- If you want to pass an initial data value into a subroutine, perform an operation that may modify the data value, and pass the modified result back to the calling routine, then create a variable in the subroutine's variable table and specify its declaration type as IN_OUT.

Declaration type	Description
IN	Input parameter provided by the calling POU.
OUT	Output parameter returned to the calling POU.
IN_OUT	Parameter whose value is supplied by the calling POU, modified by the subroutine, and then returned to the calling POU.
TEMP	Temporary variable that is saved temporarily in the local data stack. Once the POU has been executed completely, the value of the temporary variable is no longer available. Temporary variables do not keep their value between POU executions.

Data type checking for local variables

When you pass local variables as parameters for a subroutine, the data type that you have assigned in the Local Variable Table of that subroutine must match the data type of the value in the calling POU.

Example

You call SBR0 from OB1, using a global symbol called INPUT1 as an input parameter of the subroutine.

Inside the Local Variable Table of SBR0, you have defined a local variable called FIRST as an input parameter.

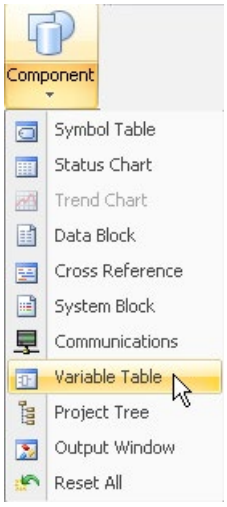
When OB1 calls SBR0, the value of INPUT1 is passed to FIRST.

The data types of INPUT1 and FIRST must match.

If INPUT1 is a REAL and FIRST is a REAL, the data types match. If INPUT1 is a REAL but FIRST is an INT, the data types do not match and the program cannot be compiled until this error is corrected.

Viewing the variable table

To view the variable table for the POU selected in the program editor, select "Variable table" from the Component drop-down list in the Windows area of the View menu.



Note

You can put the variable table on the quick access toolbar (Page 96) for easy access.

Making assignments in the variable table

Note

Make the assignments in the variable table **before** using local variables in your program. When you use symbolic names in your program, the program editor checks first the Local Variable Table of the appropriate POU, and then the symbol table. If the symbolic name is undefined in both places, the program editor treats it as an undefined global symbol which is indicated by a green wavy underline. The program editor does not automatically re-read the variable table and make corrections to your program logic. If you later make a data type assignment that defines that symbolic name (in the local variable table), you must manually insert a pound symbol (#) in front of the name, like this: #UndefinedLocalVar (in the program logic). For this reason, declaring the variables prior to usage minimizes the programming effort.

The maximum limit of input/output parameters for each subroutine call is 16. If you attempt to download a program that exceeds this limit, STEP 7-Micro/WIN SMART returns an error.

To make an assignment in a variable table, follow the procedure below.

1. Ensure that the correct POU is displayed in the Program Editor window by clicking, if necessary, on the tab of the desired POU. (Since every POU has its own variable table, you need to make sure that you are making assignments to the correct POU.)
2. Display the variable table if it is not already visible by selecting "Variable Table" from the Component drop-down list in the Windows area of the view menu.

3. Choose a row that has the right variable type for the kind of variable that you want to define, and type a name for the variable in the Symbol field. If you are making an assignment in OB1 or an interrupt routine, the variable table contains only TEMP variables. If you are making an assignment in a subroutine, the variable table contains IN, IN_OUT, OUT, and TEMP variables. Do not preface the name with a pound symbol in the variable table. Pound symbols are only used to precede local variables in the program code.

   Note

   Local variable names are permitted to contain a maximum of 23 alphanumeric characters and underscores. They are also permitted to contain extended characters (ASCII 128 to ASCII 255). The first character is restricted to alpha and extended characters only. It is illegal to use keywords as symbolic names, or to use names that begin with a number or contain characters that are not alphanumeric or in the extended character set.

   Local variable names are downloaded and stored in CPU memory. The use of longer variable names may reduce the memory available to store your program.

4. Click the mouse pointer in the Data Type field and use the list box to select an appropriate data type for the local variable.

   Note

   When you assign local variables as parameters for subroutines, you must ensure that the data type that you assign to the local variable does not conflict with the operand being used in the subroutine call.

5. Optionally provide a comment describing your local variable.

After you supply a value for the Symbol and Data Type fields, the Program Editor automatically assigns an L memory address to the local variable.

Entering additional variables

The variable table displays a fixed number of rows for local variables. To add more rows to the table, select a row in the table of the variable type that you want to add and click the Insert button in the variable table window. A new row is automatically generated above the row you selected, and is for the same variable type that you selected.

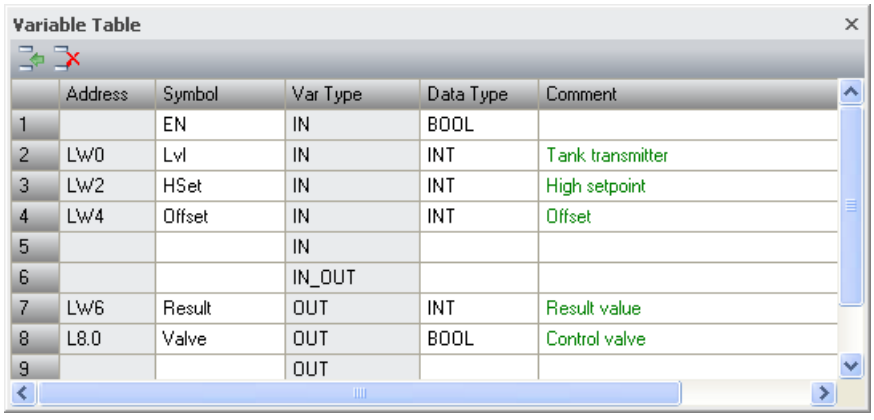
You can also add rows by right-clicking an existing row and selecting **Insert > Row** or **Insert > Row Below** from the context menu.

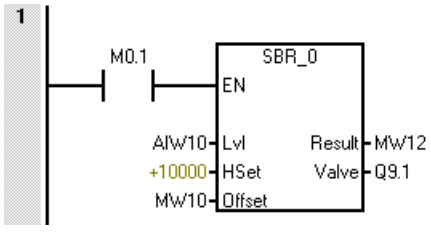
Deleting variables

To delete a local variable, select it in the variable table and click the Delete button . Alternatively you can delete a row by right-clicking it and selecting **Delete > Row** from the context menu.

Variable table example

The following example shows a typical variable table for SBR_0, and then a call to SBR_0 from another program block.



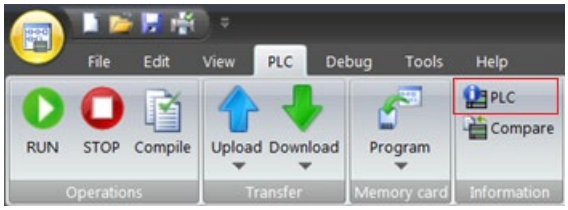


See also

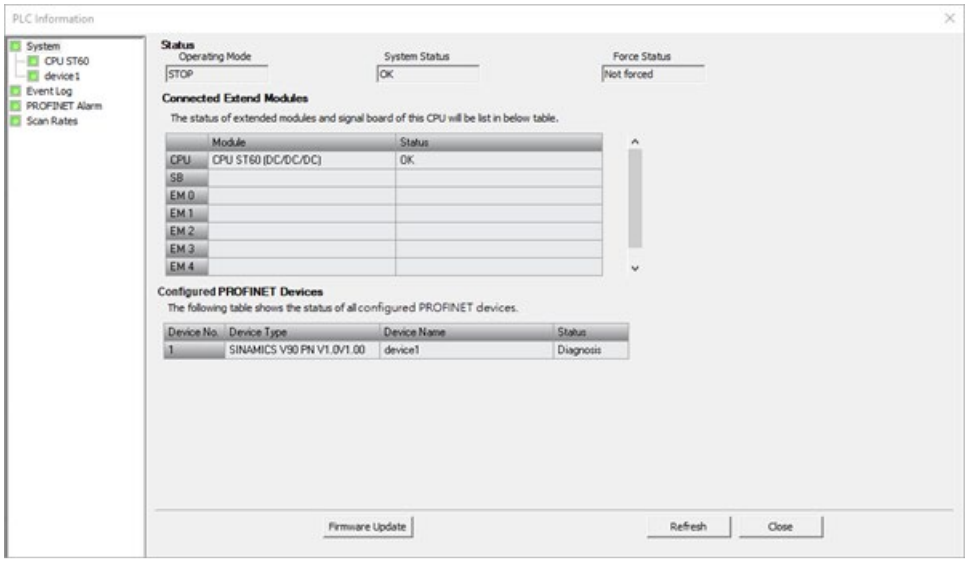
Programming software (Page 26)

5.7 PLC error reaction

Click the "PLC" button from the "Information" section of the "PLC" menu ribbon strip to see the current status.



The "PLC Information" dialog displays as follows:



The "PLC Information" dialog provides the following tree entries for status check:

- System:
  - Connected CPU: the name of connected CPU, for example, CPU ST 60.
  - Configured PROFINET device: the name of configured PROFINET device, for example, device 1.
- Event Log
- PROFINET Alarm
- Scan Rates

Note

The "Refresh" button is used for updating the PLC information. No matter where you click the "Refresh" button, all the PLC information is updated.

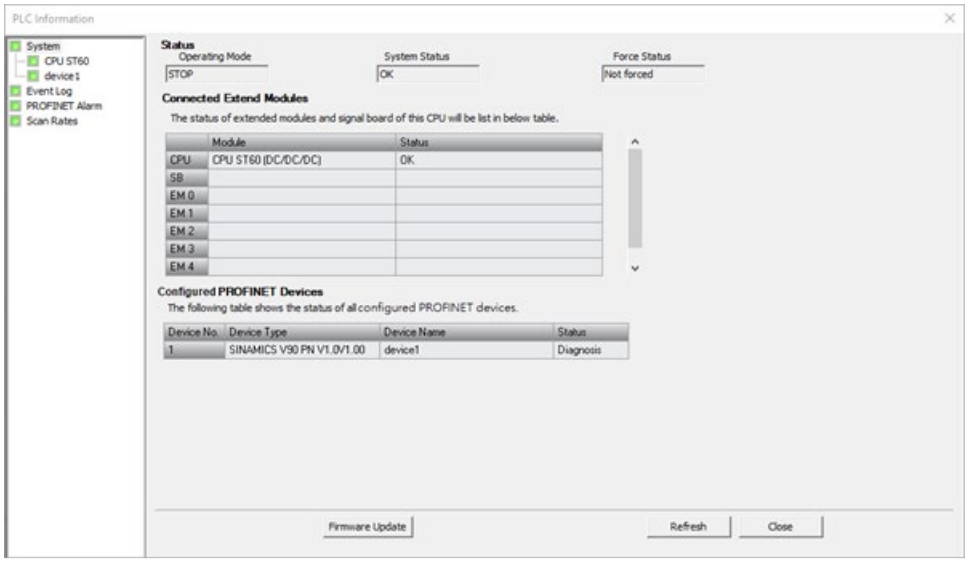
The "Firmware Update" button is used for updating the firmware.

Note the following information:

- The PLC provides SM bits for programmed reactions to errors. Refer to the listing of the SM bits (Page 865).
- The GET_ERROR (Get non-fatal error code) program instruction returns the PLC's current non-fatal error code and clears the non-fatal error information latched in the PLC. Refer to the GET_ERROR instruction (Page 338) for details.

5.7.1 System Information

5.7.1.1 System



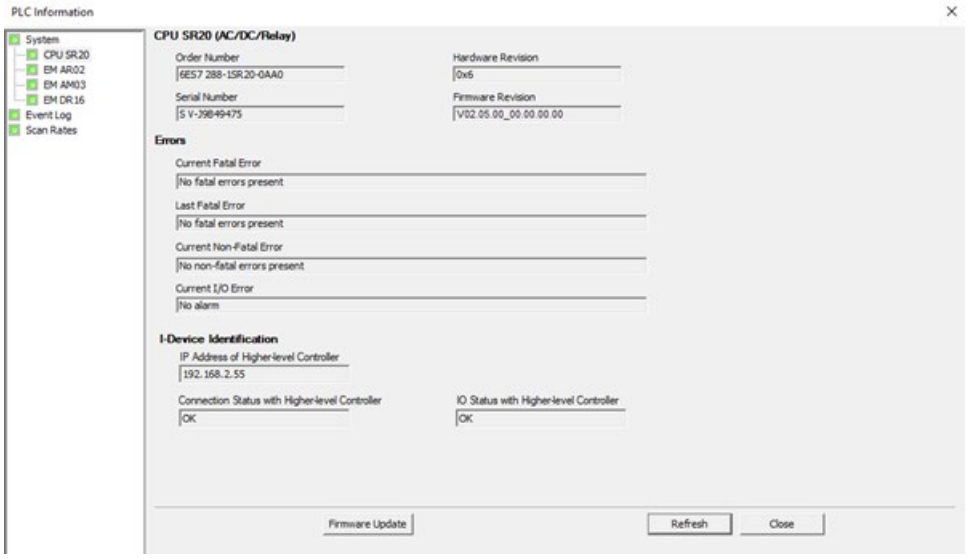
The "System" dialog displays the following information:

- Status: the status of system
  - Operating Mode: PLC operating modes (RUN or STOP)
  - System Status: the status of system (OK or Error)
  - Force Status: whether the variable is forced or not
- Connected Extend Modules: the status of extend modules and CPU signal board
- Configured PROFINET Devices: the status of PROFINET devices

  The status is as follows:
  - Not connected: The controller cannot connect with the device.
  - OK
  - Diagnosis: An alarm is reported.

5.7.1.2 CPU

The CPU dialog is as follows:



The CPU dialog displays the following information:

- Connected CPU: the CPU model

  The dialog lists the following CPU information:

  – Order Number
  – Hardware Revision
  – Serial Number
  – Firmware Revision

- Errors: the error information. To identify specific errors, refer to error codes (Page 863).
  – Current Fatal Error: the latest fatal error
  – Last Fatal Error: The "Last Fatal Error" field shows the previous fatal error code generated by the CPU. This value is retained after a power cycle. This location is cleared whenever all memory of the CPU is cleared.
  – Current Non-Fatal Error: the latest non-fatal error
  – Current I/O Error: the latest I/O error

- I-Device Identification: the I-Device information.

  Note

  If the CPU works as an I-Device, the CPU dialog displays the I-Device Identification information.

  – IP Address of Higher-level Controller: If I-Device is not connected with its higher-level controller, "--.--.--.--" is displayed here.
  – Connection Status with Higher-level Controller:

    Not connected: The I-Device is not connected with the higher-level controller.

    Diagnosis: The I-Device is connected with the higher-level controller, but the I-Device configuration doesn't match with the higher-level controller configuration.

    OK

  – IO Status with Higher-level Controller:

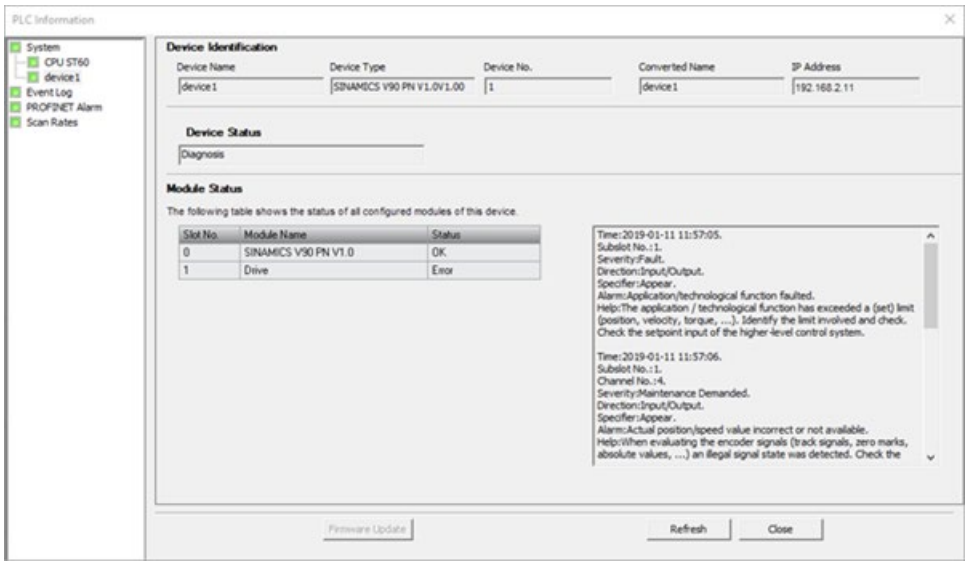
    Not connected: The I-Device is not connected with the higher-level controller.

    IO data error

    OK

5.7.1.3 PROFINET device

The PROFINET device dialog is as follows:



The PROFINET device dialog displays the following information of configured PROFINET devices:

- Device Identification
  - Device Name
  - Device Type
  - Device No.
  - Converted Name: The system converts the device name to the name in the format supported by the PROFINET protocol. For example, if you enter a Chinese name "设备 1", the corresponding converted name is "xn--1-b90bx17m". If you enter a English name, the converted name is the same as the name you enter.
  - IP Address
- Device Status

  The device status is classified as follows:
  - Not connected:The controller cannot connect with the device.
  - OK
  - Diagnosis: An alarm is reported.
- Module Status

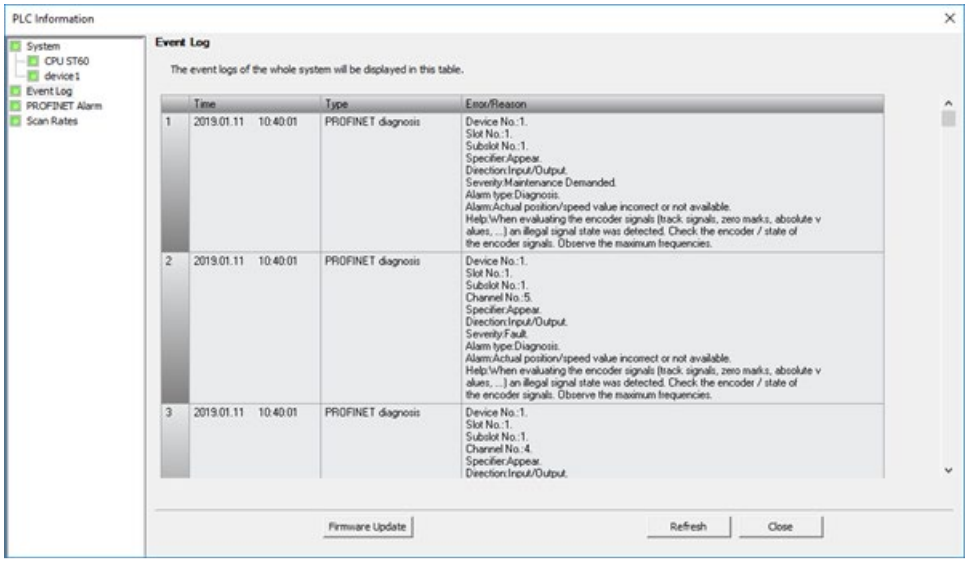
  For each module in a slot, the dialog displays the status. The module status is classified as follows:
  - OK
  - Error: If you click "Error" in the "Status" column, the corresponding detailed error information shows at the right.

5.7.2 Event Log

The "Event Log" dialog displays the CPU's stored event history, including events such as power up, power down, errors, and mode transitions. The time of events is also listed.

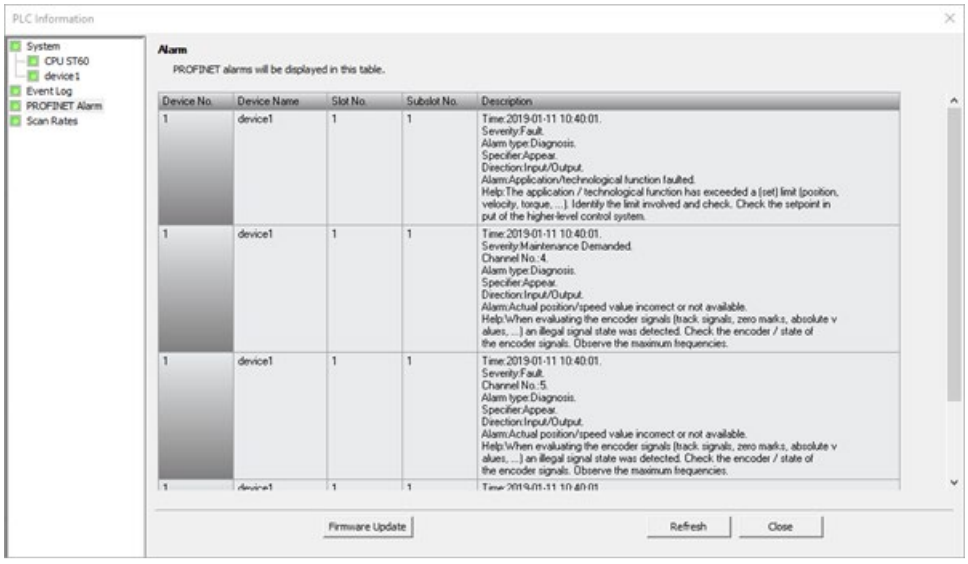
If the CPU firmware version is V2.3 or earlier, the maximum number of displayed event log is 16.

If the CPU firmware version is V2.4 or later, the maximum number of displayed event log is 32.



5.7.3 PROFINET Alarm

The "PROFINET Alarm" dialog displays the PROFINET-related alarms information: device number, device name, slot number, subslot number and alarm description.



5.7.4 Scan Rates

The "Scan Rates" dialog is as follows:

The "Scan Rates" dialog displays the following information:

- Last: the last scan rate
- Minimum: the minimum scan rate
- Maximum: the maximum scan rate

If you click "Reset" button, the scan rate information in PLC is cleared. Then you can click "Refresh" button to display the updated scan rate information.

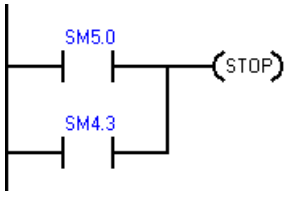
5.7.5 Non-fatal errors and I/O errors

The CPU does not change to STOP mode when it detects a non-fatal error. It only logs the event in SM memory and continues with the execution of your program. However, you can design your program to force the CPU to STOP mode when a non-fatal error is detected.

The following sample program shows a network of a program that is monitoring two of the global non-fatal error bits and changes the CPU to STOP whenever either of these bits = 1.

Table 5- 3 Example logic for detecting a non-fatal error condition

LAD		STL

	When an I/O error or a run-time error occurs, go to STOP mode	`Network 1`
		`LD SM5.0`
		`O SM4.3`
		`STOP`

Non-fatal errors are those indicating problems with the construction of the user program or with certain instruction execution problems in the user program. I/O errors are those indicating problems with the I/O of the CPU, signal board, and expansion modules. You can use STEP 7-Micro/WIN SMART to view the error codes that were generated by the non-fatal and I/O errors.

Click the PLC button from the Information section of the PLC menu ribbon strip, to see the current error status of a PLC connected to STEP 7-Micro/WIN SMART.

Table 5- 4 Non-fatal error types

	Description
Program-compile errors in the CPU	The CPU compiles the program as it downloads. If the CPU detects that the program violates a compilation rule, it aborts the download and generates an error code. (A program that was already downloaded to the CPU would still exist in the permanent memory and would not be lost.) After you correct your program, you can download it again.
I/O device errors	After power-up and after a system block download, the CPU verifies that the I/O configuration stored in the system block matches the CPU, signal board, and expansion modules that are actually present. Any mismatch results in the generation of a configuration error for the device. During runtime, devices can detect other I/O problems (such as missing user power or input value exceeding limits) that cause the CPU to generate an I/O error. The CPU stores module status information in special memory (SM) bits. Your program can monitor and evaluate these bits. SM5.0 is the global I/O error bit and remains set while any I/O error condition exists.
Program execution errors	Your program can create error conditions during execution. These errors can result from improper use of an instruction or from the processing of invalid data by an instruction. For example, an indirect-address pointer that was valid when the program compiled could point to an out-of-range address if the program modified the pointer during execution. Modifying a pointer to an invalid address is an example of a run-time programming problem. The CPU sets SM4.3 upon the occurrence of a run-time programming problem. SM4.3 remains set while the CPU is in RUN mode. The program can execute the GET_ERROR instruction (Page 338) to get the current non-fatal error code and reset SM4.3 to OFF.

Refer to the non-fatal error code list (Page 860) for a description of compile rule violations and run-time programming problems.

Refer to the description of the SM bits (Page 865) for more information about the SM bits used for reporting I/O and program execution errors.

5.7.6 Fatal errors

Fatal errors cause the PLC to stop the execution of your program. Depending upon the severity of the fatal error, it can render the PLC incapable of performing any or all functions. The objective for handling fatal errors is to bring the PLC to a safe state from which the PLC can respond to interrogations about the existing error conditions.

When a fatal error is detected, the PLC changes to STOP mode, turns on the STOP and the ERROR LED, overrides the output table, and turns off the outputs. The PLC remains in this condition until the fatal error condition is corrected.

Once you have made the changes to correct the fatal error condition, use one of the following methods to restart the PLC:

- Turn the PLC power off and then on.
- Using STEP 7-Micro/WIN SMART, click the "Warm Start" button in the modify area of the PLC ribbon strip. This forces the PLC to restart and clear any fatal errors.

Restarting the PLC clears the fatal error condition and performs power-up diagnostic testing to verify that the fatal error has been corrected. If another fatal error condition is found, the PLC again sets the ERROR LED, indicating that an error still exists. Otherwise, the PLC begins normal operation.

Some error conditions can render the PLC incapable of communication. In these cases, you cannot view the error code from the PLC. These types of errors indicate hardware failures that require the PLC to be repaired; they cannot be fixed by changes to the program or clearing the memory of the PLC.

Refer to the fatal error code list (Page 863) for details.

5.8 Program edit in RUN mode

WARNING

Risks when downloading a program in RUN mode

When you download program changes to a PLC in RUN mode, your changes immediately affect process operation. You have no margin for error; mistakes in your programming edits can cause death or serious injury to personnel, and/or damage to equipment. Only qualified personnel should perform a program edit in RUN mode.

Overview

The "program edit in RUN mode" feature allows you to make changes to a program and download them to your PLC without switching to STOP mode.

- You can make minor changes to your current process without having to shut down.

  Example: Change a parameter value.

- You can debug a program more quickly with this feature.

  Example: Invert the logic for a normally open or normally closed switch.

If you download changes to a real process (as opposed to a simulated process, which you might do in the course of debugging a program), be sure to think through the possible safety consequences to machines and machine operators before you download.

You can download only the program block (OB1, subroutines, and interrupts) during a program edit in RUN mode. You cannot download the system block or the data block during a program edit in RUN mode.

Prerequisites for editing in RUN mode

You cannot download your program edits to a PLC that is in RUN mode unless you have met these prerequisites:

- Your program must compile successfully.
- You must have successfully established communications between the computer where you are running STEP 7-Micro/WIN SMART and the PLC.
- The firmware of the target PLC must support the program edit in RUN mode feature. Only S7-200 SMART CPUs with version V2.0 or later firmware support the program edit in RUN mode feature.
- You must provide a password for a protected POU to open the block (for normal editing, edit in RUN mode, and program status operations).

If you change the PLC to STOP mode while a program edit in RUN mode is in progress, the PLC aborts the editing session.

Possible complications

To help you decide whether to download your program modifications to the PLC in RUN mode or STOP mode, consider the following effects from various types of program modifications made during a RUN mode edit:

- If you delete the control logic for an output, the output maintains its last state until the next power cycle or transition to STOP mode.
- If you delete HSC, Motion, or PLS functions that were running at the time of the edit in RUN mode, then these functions continue to run until the next power cycle or transition to STOP mode.
- If you delete ATCH or DTCH instructions in a RUN mode edit but do not delete the associated interrupt routine, then the interrupt routine continues to execute whenever the controlling event occurs until the next power cycle or transition to STOP mode.
- If you add ATCH instructions that are conditional on the first scan flag, the CPU does not enable these events until the next power cycle or STOP-to-RUN mode transition.
- If you delete an ENI or DISI instruction, activated interrupt routines events still continue to operate until the next power cycle or transition from RUN to STOP mode.

- If you modify the table address of a RCV instruction and the RCV instruction is active at the time of the edit in RUN mode, then the PLC writes the received data to the old table address. The PLC does not use the new address until the current receive request (to the old address) completes. Because you have edited your program, if the program looks for the data in the new address, the data will not be there. GET and PUT instructions function similarly.

  Note

  The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

- The PLC does not execute logic that is conditional on the first scan flag until after a power cycle or a transition from STOP to RUN mode. The startup of the modified program after a RUN mode edit does not set the first scan flag.

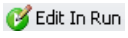
Handling positive or negative transitions

To minimize the process impact of changes that involve the relocation of positive transition (EU) and negative transition (ED) instructions in your program during a RUN mode edit, STEP 7-Micro/WIN SMART temporarily allocates a number to each transition instruction included in your program. For each transition instruction that you add in your program during a RUN mode edit, you must assign it a unique identification number. To assist you in selecting an unused number, STEP 7-Micro/WIN SMART provides an edge usage tab on the cross reference window, available when you activate the Program Edit in RUN Mode feature. This table lists all EU/ ED instructions that are currently in use in your program, so you can use this list to guide you in making changes to your program.

Performing a program edit and download in RUN mode

To initiate a program edit in RUN mode, follow these steps:

1. From the Settings area of the Debug menu ribbon strip, click the Edit In Run button.

   Note

   If you have not saved your current program in the program editor, STEP 7-Micro/WIN SMART prompts you to save your project. You can use the same name or you can change the name.

2. From the warning dialog, click the "Continue" button to confirm that you want to proceed with editing your program in RUN mode. STEP 7-Micro/WIN SMART uploads the program that is currently stored in the CPU and displays it in the program editor, where you can make the changes you need.

After you make the desired editing changes, you must download them before they can take effect in the CPU. Once you initiate a download, you cannot perform other tasks in STEP 7-Micro/WIN SMART until the download completes.

Examine the output window to see whether any compile errors exist (for instance, duplicate EU or ED numbers). You can double-click the error message to edit the offending network in the program editor.

Specifying CPU allocation (background time)

During a program edit in RUN mode, the CPU requires time to compile the modified program in the background while it continues to execute the currently loaded program. In the system block (Page 131), you can configure the amount of background time that is available for the compilation. Note that you can only download the system block when the CPU is in STOP mode.

5.9 Features for debugging your program

STEP 7-Micro/WIN SMART provides the following features to help you debug your program:

- Adding bookmarks in your program to make it easy to move back and forth between lines of a long program
- Tracking references in your program with the cross reference table (Page 647)
- Using a status chart (Page 652) to display PLC data values and status
- Displaying status in the program editor (Page 649)

For more information about debugging your program, refer to the chapter on diagnostics and troubleshooting (Page 647).

PLC device configuration

6

6.1 Configuring the operation of the PLC system

6.1.1 System block

The system block provides configuration of the S7-200 SMART CPU, signal boards, and expansion modules.

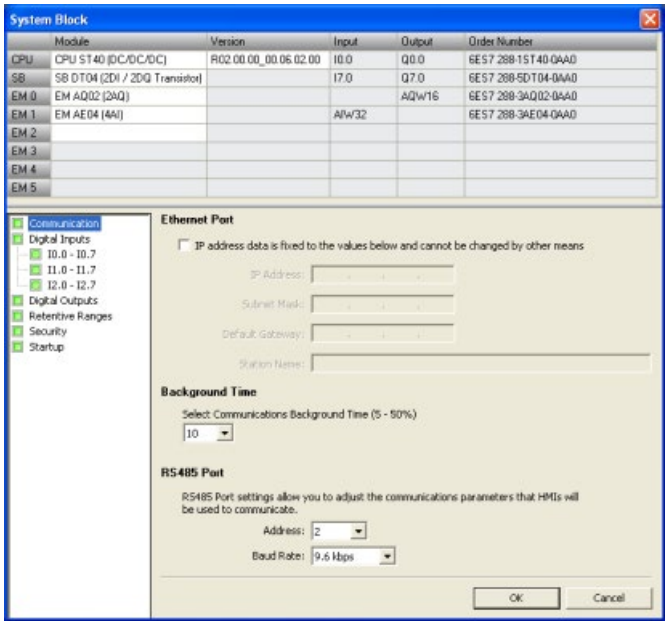
Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

Use one of the following methods to view and edit the system block to set up CPU options:

- Click the "System Block" button on the navigation bar (Page 26).
- Select "System Block" from the Component drop-down list (Page 26) in the Windows area of the View menu ribbon strip.
- Select the "System Block" node, then press Enter; or double-click the "System Block" node in the project tree (Page 26).

STEP 7-Micro/WIN SMART opens the system block, and displays the configuration options that are applicable for your CPU type.



Hardware configuration

The top part of the System Block dialog displays the modules that you have configured and allows you to add or delete modules. Use the drop-down lists to change, add, or delete the CPU model, signal board, and expansion modules. As you add modules, the input and output columns display the assigned input and output addresses.

Note

Optimally, select the CPU model and firmware version (V1 or V2) in the system block to be the model and firmware version of the actual CPU you plan to use. When downloading your project, if the CPU model and firmware version in the project does not match the model and firmware version of the connected CPU, STEP 7-Micro/WIN SMART issues a warning message. You can continue with the download, but if the connected CPU does not support the resources and capabilities that the project requires, a download error occurs.

Module options

The bottom part of the system block dialog displays options for the module that you select in the top part. Click any node in the configuration options tree to modify the project configuration for the selected module.

The system block includes the following configuration options for CPU modules:

- Communication (Page 131)
- Digital inputs and pulse catch bits (Page 133)
- Digital outputs (Page 136)
- Retentive Ranges (Page 137)
- Security (Page 138)
- Startup (Page 142)

Configuration options specific to other devices such as analog inputs (Page 143), analog outputs (Page 146), RTD analog inputs (Page 147), Thermocouple (TC) analog inputs (Page 152), RS485/RS232 CM01 communications signal board (Page 155), Battery BA01 signal board (Page 156), and additional digital inputs and outputs are accessible from the system block when you add those modules.

You must establish communications between STEP 7-Micro/WIN SMART and your CPU before you can download or upload a system bock.

You can then download a modified system block in order to provide the CPU with a new system configuration. New properties that you enter take effect when you download (Page 40) the modifications to the CPU.

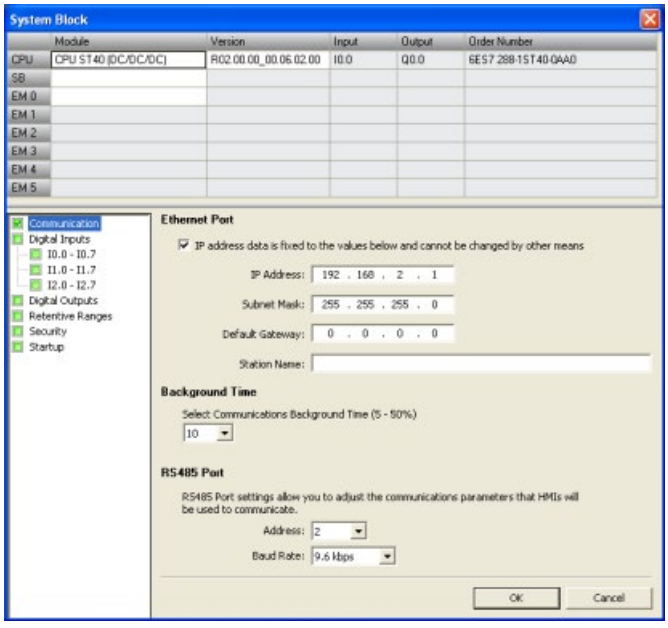
Alternatively, you can upload an existing system block from the CPU in order to make your STEP 7-Micro/WIN SMART project configuration match that of the CPU.

6.1.2 Configuring communication

Click the Communication node of the system block (Page 129) dialog to configure the Ethernet port, background time, and RS485 port.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.



Ethernet port

If you want your CPU to obtain its Ethernet network port information from the project, click the "IP address data is fixed to the values below and cannot be changed by other means" checkbox. You can then enter the following Ethernet network information:

- IP address: Each device must have an Internet Protocol (IP) address. The device uses this address to deliver data on a more complex, routed network.
- Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet are usually located in close physical proximity to each other on a Local Area Network (LAN). The subnet mask defines the boundaries of an IP subnet. A subnet mask of 255.255.255.0 is generally suitable for a local network.

- Default gateway: Gateways (or IP routers) are the link between LANs. Using a gateway, a computer in a LAN can send messages to other networks, which might have other LANs behind them. If the destination of the data is not within the LAN, the gateway forwards the data to another network or group of networks where it can be delivered to its destination. Gateways rely on IP addresses to deliver and receive data packets.
- Station name: The station name is the name by which this CPU is identified on the network. Use a name that helps you identify the CPU on the Communications dialog.

Note

The station name follows the standard DNS (Domain Name System) naming conventions. The S7-200 SMART CPUs limit the station name to a maximum of 63 characters, which can consist of the lower case letters a through z, the digits 0 through 9, the hyphen character (minus sign) and the period character.

The CPU prohibits certain names:

- The station name must not have the format n.n.n.n where n is a value of 0 through 999.
- You cannot begin the station name with the string port-nnn or the string port-nnn-nnnnn where n is a digit 0 through 9. For example, port-123 and port-123-45678 are illegal station names. A station name cannot start or end with a hyphen or period.

Background time

You can configure the percentage of the scan cycle time that is dedicated to processing the communication requests. As you increase the percentage of time that is dedicated to processing communication requests, you are increasing scan time, which makes your control process run more slowly. The scan time only increases if there are communication requests to process.

The default percentage of the scan time dedicated to processing communication requests is set to 10%. This setting provides a reasonable compromise for processing compilation/ status operations, while minimizing the impact to your control process. You can adjust this value by 5% increments up to a maximum of 50%.

As you add more communication partners to the S7-200 SMART CPU, additional background time is required to handle the requests from those partners. GET and PUT instructions need additional resources to create and maintain connections to other devices. The EM DP01 PROFIBUS DP module requires additional communication background time if you have HMIs or other CPUs communicating with the S7-200 SMART CPU through the EM DP01. Open User Communication (OUC) also adds an additional load to the CPU and may require additional background time.

RS485 port

Use these settings to adjust the communication parameters for system protocols for the onboard RS485 port. The system protocols are used when connecting to a programming device or HMI devices:

- RS485 port address: Click the scroll buttons to enter the desired CPU address (1-126). The default port address is 2.
- Baud Rate: Choose the desired data baud rate from the dropdown list (9.6 Kbps, 19.2 Kbps, or 187.5 Kbps).

Note

You can make the following RS485 communication connections for S7-200 SMART CPUs:

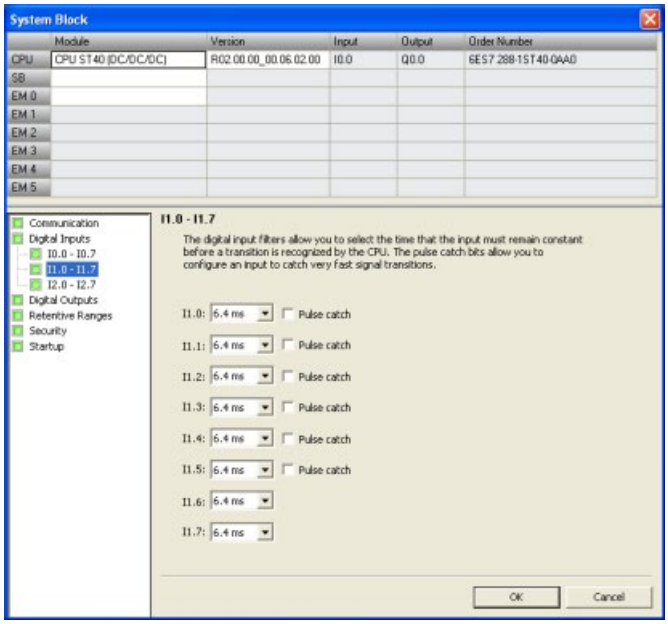
- Use a USB-PPI cable to program all CPU models through any serial port, including the RS485 port, the signal board port, and the DP01 PROFIBUS port.
- Use the RS485 and RS232 ports for HMI access (Data read/write) and Freeport communications.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

6.1.3 Configuring the digital inputs

Click the Digital Inputs node of the system block (Page 129) dialog to configure digital input filters and pulse catch bits.



Digital input filters

You can filter digital input signals by setting an input delay time. This delay helps to filter noise on the input wiring that could cause inadvertent changes to the states of the inputs. When an input state change occurs, the input must remain at the new state for the duration of the delay time in order to be accepted as valid. The filter rejects noise impulses and forces input lines to stabilize before the data is accepted.

The S7-200 SMART CPU allows you to select an input delay time for all of its digital input points. The quantity of input points available is dependent upon your CPU model (Page 19).

The first fourteen input points (I0.0 through I0.7 and I1.0 through I1.5) support an expanded set of delay time choices (selectable to one of seven settings in the range of 0.2 ms to 12.8 ms or one of seven settings in the range of 0.2 µs to 12.8 µs). The remaining input points (I1.6 and greater) support only a limited set of input delay choices (6.4 ms, 12.8 ms, or no filtering).

For example, all twelve input points of a CPU SR20 support the expanded list of input delay settings. In a CPU ST40, the expanded list of input delay choices is available for the first fourteen input points, while only the limited list of input delay choices is available for the remaining ten input points.

The default filter time for all input points is 6.4 ms.

To set an input delay, follow these steps:

1. Select the time of the delay from the drop-down list beside one or more inputs.
2. Click the OK button to enter the selections.

WARNING

Risks with changes to filter time for digital input channel

If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 12.8 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 12.8 ms may not be detected or counted.

This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

To ensure that a new filter time goes immediately into effect, a power cycle of the CPU must be applied.

Pulse catch bits

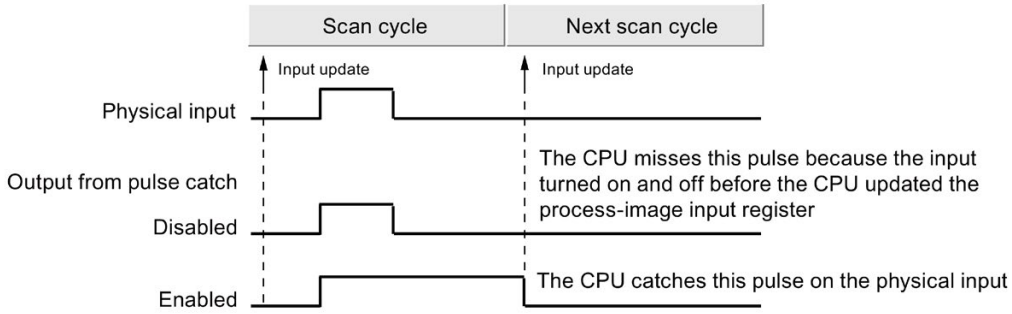
The S7-200 SMART CPU provides a pulse catch feature for digital input points. The pulse catch feature allows you to capture high-going pulses or low-going pulses that are of such a short duration that they would not always be seen when the CPU reads the digital inputs at the beginning of the scan cycle.

When pulse catch is enabled for an input, a change in state of the input is latched and held until the next input cycle update. This ensures that a pulse which lasts for a short period of time will be caught and held until the S7-200 SMART CPU reads the inputs.

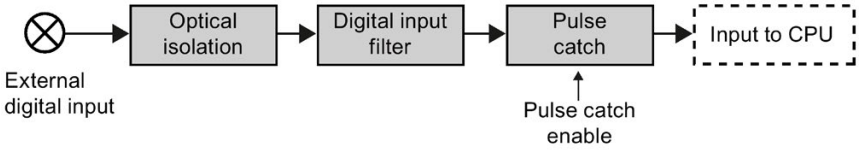
You can enable individual pulse catch operation for the first fourteen digital input points (I0.0 through I0.7 and I1.0 through I1.5), dependent upon the CPU model (Page 19).

If your configuration includes an SB DT04, you can enable the two additional digital input points available on this signal board for pulse catch operation.

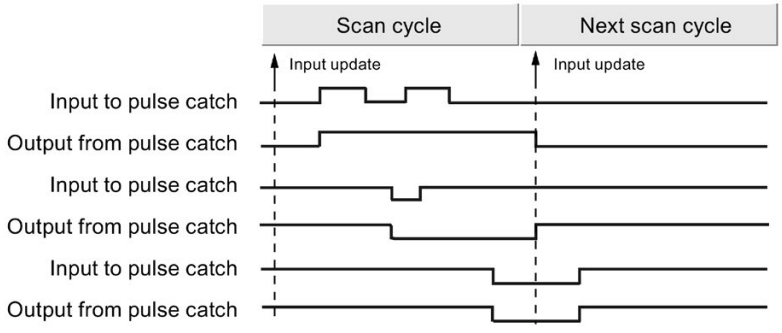
The figure below shows the basic operation of the S7-200 SMART CPU with and without pulse catch enabled:



Because the pulse catch function operates on the input after it passes through the input filter, you must adjust the input filter time so that the pulse is not removed by the filter. The figure below shows a block diagram of the digital input circuit:

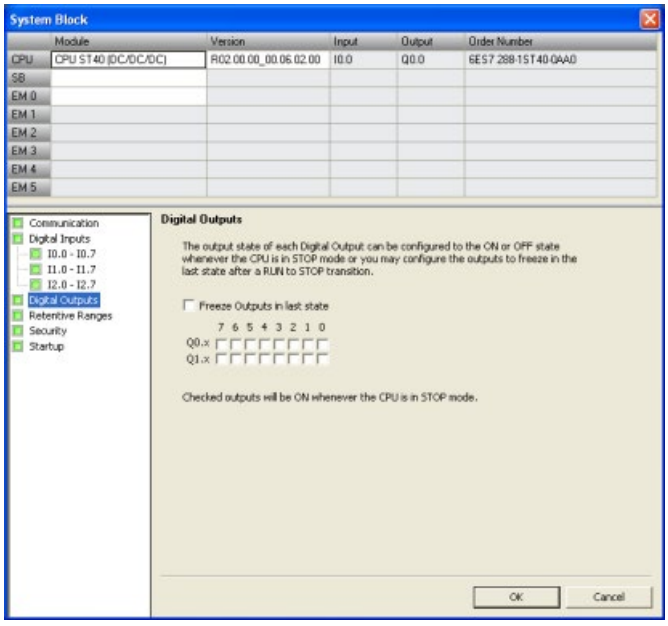


The figure below shows the response of an enabled pulse catch function to various input conditions. If you have more than one pulse in a given scan, only the first pulse is read. If you have multiple pulses in a given scan, you should use the rising/falling edge interrupt events:



6.1.4 Configuring the digital outputs

Click the Digital Outputs node of the system block (Page 129) to configure options for the digital outputs of the selected module.



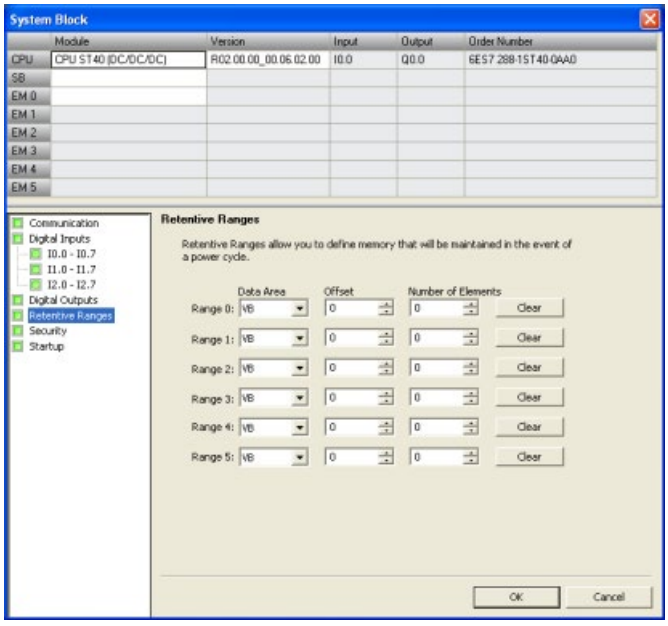
You can set digital output points to a specific value when the CPU is in STOP mode, or preserve the output states that existed before the transition to STOP mode.

You have two ways to set the digital output behavior in STOP mode:

- **Freeze Outputs in last state:** Click this checkbox to have all digital outputs frozen in their last states at the time of a RUN-to-STOP transition.
- **Substitute value:** If the Freeze Outputs in last state checkbox is not checked, this table allows you to select the desired state of each output whenever the CPU is in STOP mode. Click the checkbox for each output you want set to ON (1). The default substitute value for digital outputs is OFF (0).

6.1.5 Configuring the retentive ranges

Click the Retentive Ranges node of the system block (Page 129) dialog to configure ranges of memory that will be retained following a power cycle.



Configure the areas of memory you want to retain through power cycles. Enter new values for V, M, T, or C memory.

You can define ranges of addresses in the following memory areas to be retentive: V, M, T, and C. For timers, only the retentive timers (TONR) can be retained, and, for both timers and counters, only the current value can be retained (timer and counter bits are cleared on each power-up).

By default, the CPU has no defined retentive memory areas, but you can configure the retentive ranges:

- The S7-200 SMART CPU models CPU SR20, CPU ST20, CPU SR30, CPU ST30, CPU SR40, CPU ST40, CPU SR60, and CPU ST60 have a maximum of 10 Kbytes of retentive memory.
- The S7-200 SMART CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have a maximum of 2 Kbytes of retentive memory.

Data retention after CPU power interruption

The CPU performs the following actions regarding retentive memory at power down and power up:

- **At power down:**

  The CPU saves the memory ranges designated as retentive to permanent memory.

- **At power up:**

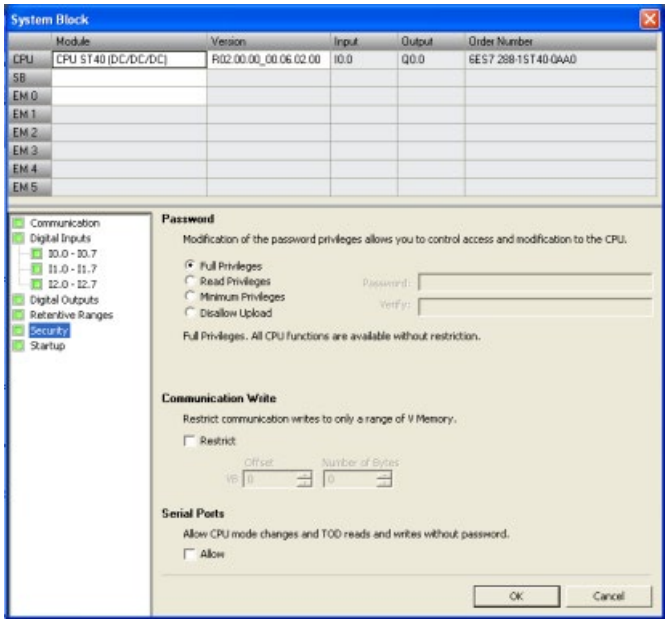
  The CPU first clears V, M, C, and T memory, copies any initial values from the data block to V memory, and then copies the saved retentive values from permanent memory to RAM.

S7-200 SMART CPU memory addresses for retentive ranges

Data type	Desc.	CPU CR20s CPU CR30s CPU CR40s CPU CR60s	CPU SR20 CPU ST20	CPU SR30 CPU ST30	CPU SR40 CPU ST40	CPU SR60 CPU ST60
V	Data Memory	VB0-VB8191	VB0-VB8191	VB0-VB12281	VB0-VB16383	VB0-VB20479
T	Timers	T0-T31, T64-T95	T0-T31, T64-T95	T0-T31, T64-T95	T0-T31, T64-T95	T0-T31, T64-T95
C	Counters	C0-C255	C0-C255	C0-C255	C0-C255	C0-C255
M	Flag bits	MB0-MB31	MB0-MB31	MB0-MB31	MB0-MB31	MB0-MB31

6.1.6 Configuring system security

Click the Security node of the system block (Page 129) dialog to configure a password and security settings for the CPU.



The password can be any combination of letters, numbers, and symbols and is case-sensitive.

Password-protected privilege levels

The CPU offers four levels of password protection, with "Full Privileges" (Level 1) providing unrestricted access and "Disallow Upload" (Level 4) providing the most restricted access. The default condition for the S7-200 SMART CPU is "Full Privileges" (Level 1).

A CPU password authorizes access to CPU functions and memory. With no CPU password downloaded ("Full privileges" (Level 1)), the S7-200 SMART CPU allows unrestricted access. If you have configured higher than "Full Privileges" (Level 1) access and downloaded a CPU password, the S7-200 SMART CPU requires password entry for access to CPU operations as defined in the table below.

The "Disallow Upload" (Level 4) password restriction protects the user program (your intellectual property) even if the password becomes known. You can never upload in Level 4 and can only change the privilege level if there is no user program present in the CPU. As a result, you can always protect your user program, even if someone discovers your password.

Table 6- 1 S7-200 SMART CPU password-protected privilege levels

Description of operation	Full privileges (Level 1)	Read privileges (Level 2)	Minimum privileges (Level 3)	Disallow upload (Level 4)
Read and write user data	Permitted	Permitted	Permitted	Permitted
Start, stop, and power-up reset of the CPU	Permitted	Restricted	Restricted	Restricted
Read the time-of-day clock	Permitted	Permitted	Permitted	Permitted
Write the time-of-day clock	Permitted	Restricted	Restricted	Restricted
Upload the user program, data, and the CPU configuration	Permitted	Permitted	Restricted	Never Permitted
Download of program block, data block, or system block	Permitted	Restricted	Restricted	Restricted Note: Never permitted for the system block if the user program block is present. Password verification is required if the user downloads the program block or the data block.
Reset to factory defaults	Permitted	Restricted	Restricted	Restricted
Delete of program block, data block, or system block	Permitted	Restricted	Restricted	Restricted Note: Never permitted for the system block if the user program block is present.
Copy of program block, data block, or system data block to a memory card	Permitted	Restricted	Restricted	Restricted
Forcing of data in status chart	Permitted	Restricted	Restricted	Restricted

Description of operation	Full privileges (Level 1)	Read privileges (Level 2)	Minimum privileges (Level 3)	Disallow upload (Level 4)
Execute single or multiple scan operations.	Permitted	Restricted	Restricted	Restricted
Writing of output in STOP mode.	Permitted	Restricted	Restricted	Restricted
Reset of scan rates in PLC information	Permitted	Restricted	Restricted	Restricted
Program status	Permitted	Permitted	Restricted	Never Permitted
Project compare	Permitted	Permitted	Restricted	Never Permitted

Communication write restrictions

You can restrict communication writes to a specific range of V memory and disallow communication writes to other memory areas (I, Q, AQ, and M). To restrict communication writes to a specific range of V memory, select the "Restrict" checkbox, and configure the range in bytes of V memory.

This area can be as small as no bytes to as large as all V memory.

With this functionality, the user program can validate the data written into this subset of memory before using it in your application for even better security. Note that these restrictions pertain only to communication writes (for example, writes from HMIs, STEP 7-Micro/WIN SMART, or PC Access and PUTs from other CPUs,), not writes from the user program.

Note

If you restrict write access to a specific range of V memory, be sure that Text Display modules or HMIs only write within the writable range of V memory. Also, if you use the PID wizard, PID control panel, motion wizard, or motion control panel be sure that the V memory that the wizards or panels use are within the writable range of V memory.

With this restriction disabled, you can write to the full ranges of memory areas, including I, Q, M, V, and AQ.

Serial ports mode changes and Time-of-Day (TOD) writes

You can allow CPU mode changes (go-to-RUN, go-to-STOP) and TOD writes through the serial ports (both the RS485 built-in and RS485/RS232 signal board if your CPU model supports it) without a password. To do so, select the "Allow" checkbox in the "Serial Ports" section.

This checkbox provides backward compatibility with older HMIs that do not prompt for a password for these functions. The following selections are available:

- If this box is checked and the CPU is password protected, then you can change operating modes and make TOD writes with these older HMIs.
- If this box is unchecked and the CPU is password protected, you cannot change operating modes or make TOD writes with these older HMIs.
- If the CPU is not password protected, you can change operating modes and make TOD writes with these older HMIs, regardless of the setting of the checkbox.

Accessing a password-protected CPU

Note

When you enter the password for a password-protected CPU, the authorization level for that password remains effective for up to one minute after the programming device has been disconnected from the S7-200 SMART CPU. Always exit STEP 7-Micro/WIN SMART before disconnecting the cable to prevent another user from unauthorized access.

Entering the password over a network does not compromise the password protection for the S7-200 SMART CPU. If one authorized user is accessing restricted functions across a network, that does not authorize other users to access those functions. Only one user is allowed unrestricted access to the S7-200 SMART CPU at a time.

Disabling a password

You can disable the password by changing the privilege level 4, 3, or 2 to "Full privileges" (Level 1), since Level 1 allows all unrestricted CPU access.

Note

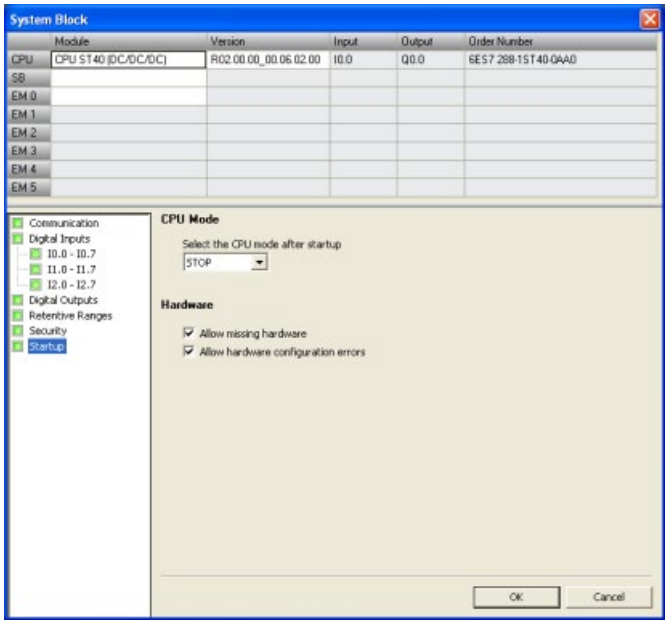
If the privilege level is at "Disallow Upload" (Level 4), you cannot download a new system block with a new password level if a valid user program exists. You must delete the user program first, and then you can download an updated system block.

What to do if you forget the password

If you forget your password, you must reset your PLC to factory defaults. (Refer to clear PLC memory (Page 158) for more information.)

6.1.7 Configuring the startup options

Click the Startup node of the system block (Page 129) dialog to configure startup options for the PLC.



CPU mode

From this dialog you can select the mode for the CPU after a startup. You have one of the following three choices:

- STOP

  The CPU shall always enter STOP mode after a power up or restart (default selection).

- RUN

  The CPU shall always enter RUN mode after a power up or restart. For most applications, especially those where the CPU operates independently without a connection to STEP 7-Micro/WIN SMART, the RUN startup mode selection is the correct choice.

- LAST

  The CPU shall enter the operating mode that existed prior to the last power up or restart. This selection can be useful during program development or commissioning. Be aware that a running CPU can enter STOP mode for a variety of reasons, such as the failure of an expansion module, occurrence of a scan watchdog timeout, the insertion of a memory card, or an erratic power up event. Once the CPU enters STOP mode, it will continue to enter STOP mode each time the CPU powers up. You must restore the CPU back to RUN mode (Page 41) from STEP 7-Micro/WIN SMART.

Hardware options

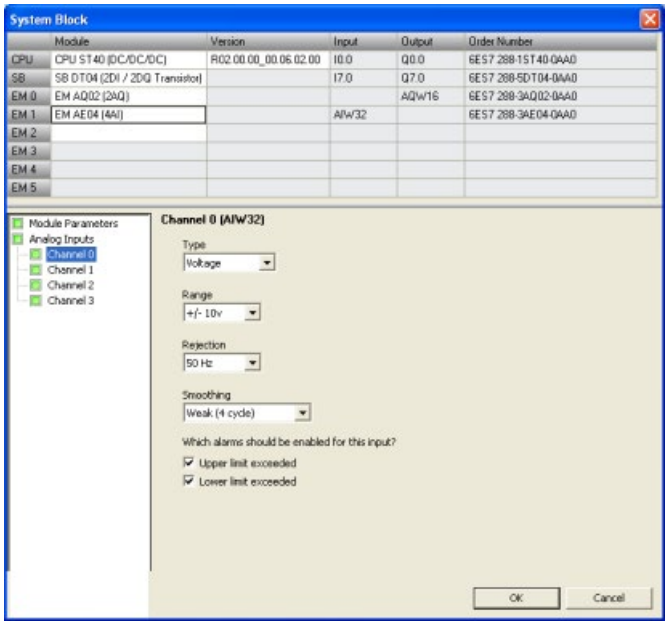
You can also configure the CPU to allow RUN mode operation under the following hardware conditions:

- One or more devices specified in the hardware configuration stored in the CPU are missing.
- A difference exists between the hardware configuration stored in the CPU and the devices actually present, resulting in configuration errors (for example, discrete input module in place of a configured discrete output module).

If you deselect one or both of the selections, the CPU is prohibited from entering RUN mode if any of the disallowed conditions are true.

6.1.8 Configuring the analog inputs

Click the Analog Inputs node of the system block (Page 129) dialog to configure options for an analog input module that you have selected in the top section.



Analog type configuration

For each analog input channel, you configure the type to be either voltage or current. The type selected for the even-numbered channels also applies to the odd-numbered channels: the type selection for Channel 0 also applies to Channel 1, and the type selection for Channel 2 also applies to Channel 3.

Range

You then configure either the voltage range or the current range for the channel. You can choose one of the following value ranges:

- +/- 2.5v
- +/- 5v
- +/- 10v
- 0 - 20ma

Rejection

Fluctuations in analog input values can also be caused by the response time of the sensor, or the length and condition of the wires carrying the analog signal to the module. In such cases, the fluctuating values could be changing too rapidly for the program logic to respond effectively. You can configure the module to reject signals to eliminate or minimize noise at the following frequencies:

- 10 Hz
- 50 Hz
- 60 Hz
- 400 Hz

Smoothing

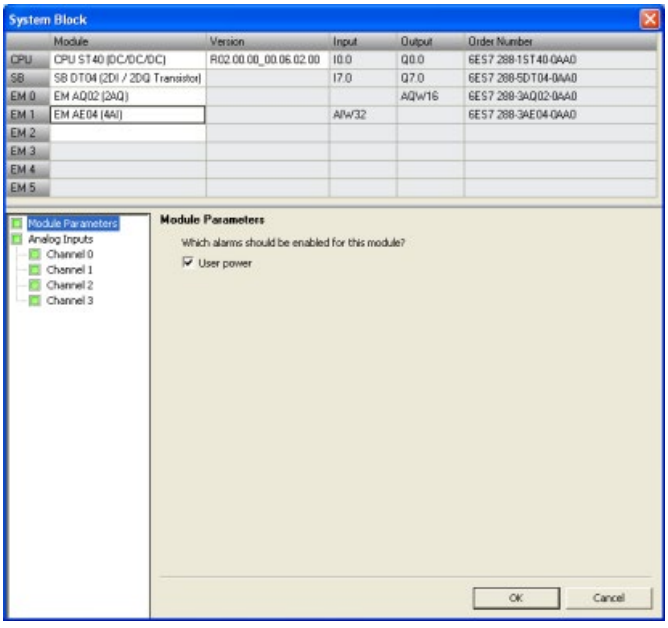
You can also configure the module to smooth the analog input signal over a configured number of cycles, thus presenting an averaged value to the program logic. You have four choices for the smoothing algorithm:

- None (no smoothing)
- Weak
- Medium
- Strong

Alarm configuration

You select whether to enable or disable the following alarms for the selected channel of the selected module:

- Upper limit exceeded (value > 32511)
- Lower limit exceeded (value < -32512)
- User power (Configured in the system block "Module Parameters" node; see the figure below.)



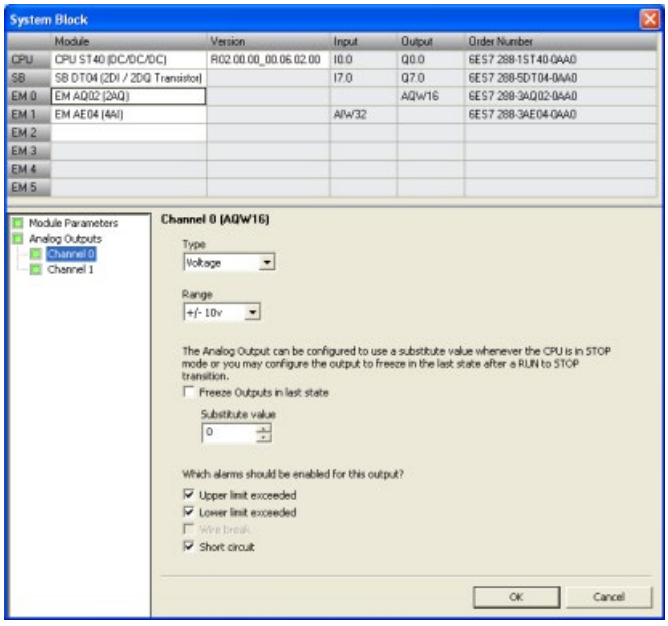
6.1.9 Reference to the analog inputs technical specifications

For further information on analog Input configuration options, refer to the following technical specifications:

- Range: "Measurement ranges of the analog inputs for voltage and current (SB and SM)" (Page 823)
- Rejection: "Sample time and update times for the analog inputs" (Page 822)
- Smoothing: "Step response of the analog inputs" (Page 822)

6.1.10 Configuring the analog outputs

Click the Analog Outputs node of the system block (Page 129) dialog to configure options for an analog output module that you have selected in the top section.



Analog type configuration

For each analog output channel you configure the type to be either voltage or current.

Range

You then configure either the voltage range or the current range for the channel. You can choose one of the following value ranges:

- +/- 10 V
- 0 - 20 mA

Output behavior in STOP mode

You can set analog output points to a specific value when the CPU is in STOP mode or preserve the output states that existed before the transition to STOP mode.

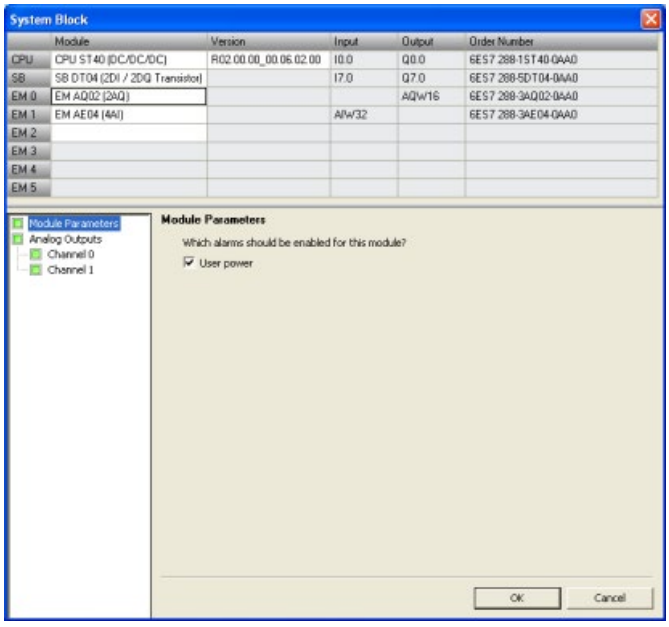
You have two ways to set the analog output behavior in STOP mode:

- Freeze outputs in last state: Click this checkbox to have all analog outputs frozen to their last values on a RUN-to-STOP transition.
- Substitute value: If the "Freeze outputs in last state" checkbox is not checked, you can enter a value (-32512 to 32511) that is applied to the output whenever the CPU is in STOP mode. The default substitute value is 0.

Alarm configuration

You select whether to enable or disable the following alarms for the selected channel of the selected module:

- Upper limit exceeded (value > 32511)
- Lower limit exceeded (value < -32512)
- Wire break (for current channels only)
- Short circuit (for voltage channels only)
- User power (Configured in the system block "Module Parameters" node; see the figure below.)



6.1.11 Reference to the analog outputs technical specifications

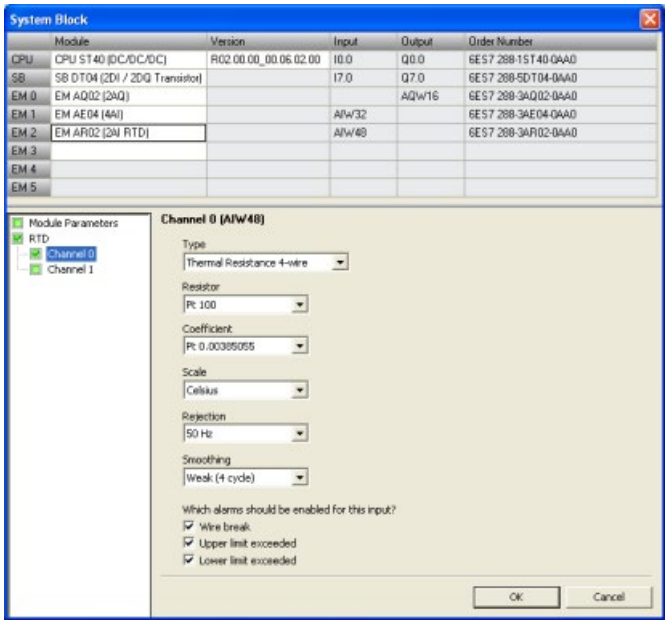
For further information on analog output range configuration, refer to the "Measurement ranges of the analog outputs for voltage and current (SB and SM)" (Page 824) technical specification.

6.1.12 Configuring the RTD analog inputs

Click the RTD analog Input node of the system block (Page 129) dialog to configure options for an RTD analog input module that you have selected in the top section.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.



The RTD analog input module provides a current at terminals I+ and I- for resistance measurements. The current is fed to the resistance for measuring its voltage potential. The current cables must be wired directly to the resistance thermometer/resistor.

Measurements programmed for 4-or 3-wire connections compensate for line resistance and return considerably higher accuracy compared to 2-wire connections.

RTD type configuration

For each RTD input channel, you configure the type, choosing one of the following options:

- Resistance 4-wire
- Resistance 3-wire
- Resistance 2-wire
- Thermal Resistance 4-wire
- Thermal Resistance 3-wire
- Thermal Resistance 2-wire

Resistor

Depending upon the RTD type that you select, you can configure the following RTD resistors for the channel:

Table 6- 2 RTD types and available resistors

RTD types	RTD resistors	
• Resistance 4-wire • Resistance 3-wire • Resistance 2-wire Note: For these RTD types and resistors, you cannot configure temperature coefficients or temperature scales.	• 48 ohms • 150 ohms • 300 ohms • 600 ohms • 3000 ohms	
• Thermal Resistance 4-wire • Thermal Resistance 3-wire • Thermal Resistance 2-wire	• Pt 10 • Pt 50 • Pt 100 • Pt 200 • Pt 500 • Pt 1000 • LG-Ni 1000	• Ni 100 • Ni 120 • Ni 200 • Ni 500 • Ni 1000 • Cu 10 • Cu 50 • Cu 100

Coefficient

Depending upon the RTD resistor that you select, you can configure the following RTD temperature coefficients for the channel:

RTD resistors	RTD temperature coefficients
• 48 ohms • 150 ohms • 300 ohms • 600 ohms • 3000 ohms	Note: For these RTD resistors, you cannot configure temperature coefficients or temperature scales.
• Pt 10 • Pt 50	• Pt 0.00385055 • Pt 0.003910
• Pt 100 • Pt 500	• Pt 0.00385055 • Pt 0.003916 • Pt 0.003902 • Pt 0.003920 • Pt 0.003910

RTD resistors	RTD temperature coefficients
• Pt 200 • Pt 1000	• Pt 0.00385055 • Pt 0.003916 • Pt 0.003902 • Pt 0.003920
• Ni 100	• Ni 0.006170 • Ni 0.006180 • Ni 0.006720
• Ni 120 • Ni 200 • Ni 500 • Ni 1000	• Ni 0.006180 • Ni 0.006720
• Cu 10	• Cu 0.00426 • Cu 0.00428 • Cu 0.00427
• Cu 50 • Cu 100	• Cu 0.00426 • Cu 0.00428
• LG-Ni 1000	• LG-Ni 0.005000

Scale

You configure a temperature scale for the channel, choosing one of the following options:

- Celsius
- Fahrenheit

Note

For the "Resistance 4-wire", "Resistance 3-wire", and "Resistance 2-wire" RTD types and associated resistors, you cannot configure temperature coefficients or temperature scales.

Rejection

Fluctuations in RTD analog input values can also be caused by the response time of the sensor, or the length and condition of the wires carrying the RTD analog signal to the module. In such cases, the fluctuating values could be changing too rapidly for the program logic to respond effectively. You can configure the module to reject signals to eliminate or minimize noise at the following frequencies:

- 10 Hz
- 50 Hz

- 60 Hz
- 400 Hz

Smoothing

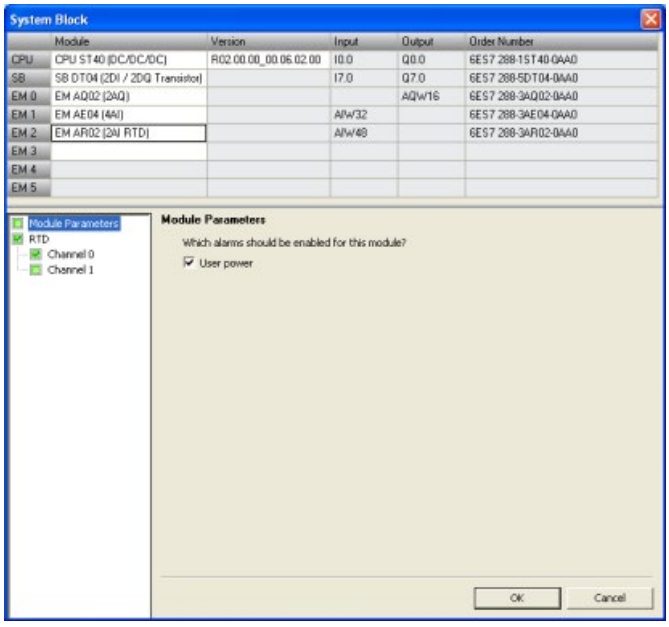
You can also configure the module to smooth the RTD analog input signal over a configured number of cycles, thus presenting an averaged value to the program logic. You have four choices for the smoothing algorithm:

- None
- Weak
- Medium
- Strong

Alarm configuration

You select whether to enable or disable the following alarms for the selected channel of the selected RTD module:

- Wire break
- Upper limit exceeded
- Lower limit exceeded
- User power (Configured in the system block "Module Parameters" node; see the figure below.)

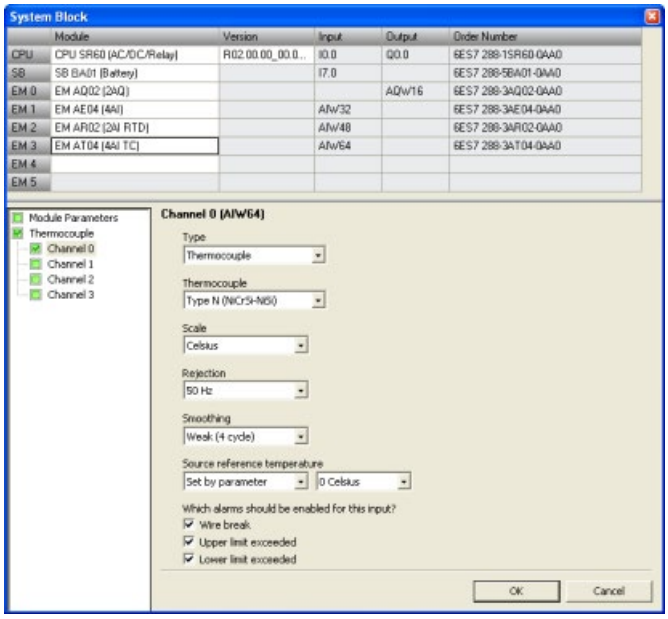


6.1.13 Configuring the TC analog inputs

Click the TC (Thermocouple) analog input node of the system block (Page 129) dialog to configure options for a TC analog input module that you have selected in the top section.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.



The TC analog expansion module measures the value of voltage connected to the module inputs.

Thermocouple type configuration

For each TC analog input module channel, you configure the type, choosing one of the following options:

- Thermocouple
- Voltage

Thermocouple

Depending upon the thermocouple type that you select, you can configure the following thermocouples for the channel:

- Type B (PtRh-PtRh)
- Type N (NiCrSi-NiSi)
- Type E (NiCr-CuNi)
- Type R (PtRh-Pt)

- Type S (PtRh-Pt)
- Type J (Fe-CuNi)
- Type T (Cu-CuNi)
- Type K (NiCr-Ni)
- Type C (W5Re-W26Re)
- TXK/XK (TXK/XK(L))

Scale

You configure a temperature scale for the channel, choosing one of the following options:

- Celsius
- Fahrenheit

Rejection

Fluctuations in thermocouple analog input values can also be caused by the response time of the sensor, or the length and condition of the wires carrying the thermocouple analog signal to the module. In such cases, the fluctuating values could be changing too rapidly for the program logic to respond effectively. You can configure the TC analog input module to reject signals to eliminate or minimize noise at the following frequencies:

- 10 Hz
- 50 Hz
- 60 Hz
- 400 Hz

Smoothing

You can also configure the module to smooth the thermocouple analog input signal over a configured number of cycles, thus presenting an averaged value to the program logic. You have four choices for the smoothing algorithm:

- None
- Weak
- Medium
- Strong

Source reference temperature

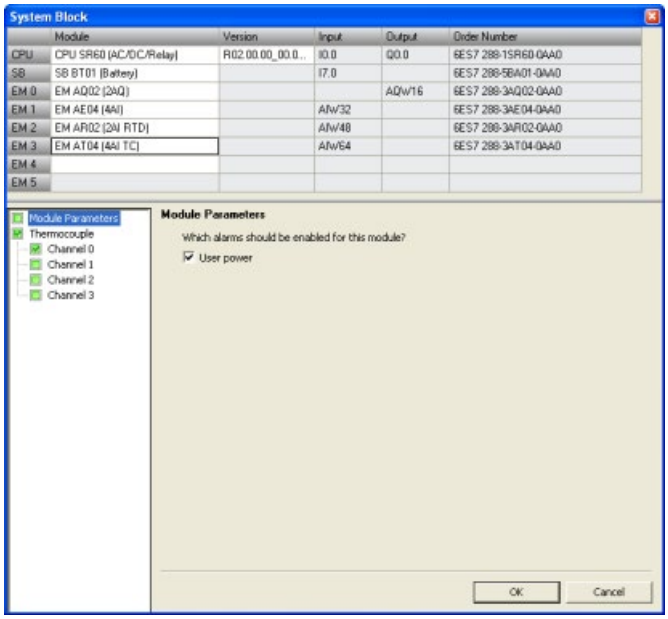
You configure a source reference temperature for each TC analog input module channel, choosing one of the following options:

- Set by parameter
- Internal reference

Alarm configuration

You select whether to enable or disable the following alarms for the selected channel of the selected TC analog input module:

- Wire break
- Upper limit exceeded
- Lower limit exceeded
- User power (Configured in the system block "Module Parameters" node; see the figure below.)



Basic operation of a thermocouple

Thermocouples are formed whenever two dissimilar metals are electrically bonded to each other. A voltage is generated that is proportional to the junction temperature. This voltage is small; one microvolt could represent many degrees. Measuring the voltage from a thermocouple, compensating for extra junctions, and then linearizing the result forms the basis of temperature measurement using thermocouples.

When you connect a thermocouple to the TC analog input module, the two dissimilar metal wires are attached to the module at the module signal connector. The place where the two dissimilar wires are attached to each other forms the sensor thermocouple.

Two more thermocouples are formed where the two dissimilar wires are attached to the signal connector. The connector temperature causes a voltage that adds to the voltage from the sensor thermocouple. If this voltage is not corrected, then the temperature reported will deviate from the sensor temperature.

Cold junction compensation is used to compensate for the connector thermocouple. Thermocouple tables are based on a reference junction temperature, usually zero degrees Celsius. The cold junction compensation compensates the connector to zero degrees Celsius. The cold junction compensation restores the voltage added by the connector thermocouples. The temperature of the module is measured internally, and then converted to

a value to be added to the sensor conversion. The corrected sensor conversion is then linearized using the thermocouple tables.

For optimum operation of the cold junction compensation, the thermocouple module must be located in a thermally stable environment. Slow variation (less than 0.1 °C/minute) in ambient module temperature is correctly compensated within the module specifications. Air movement across the module will also cause cold junction compensation errors.

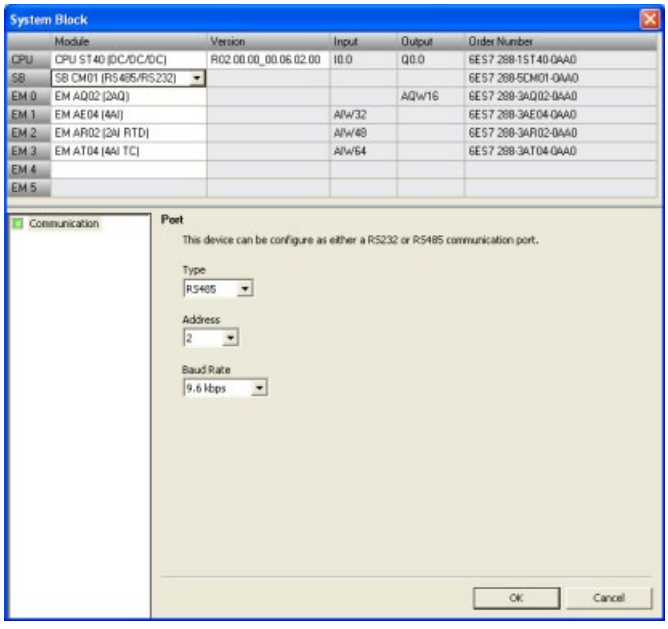
If better cold junction error compensation is needed, an external iso-thermal terminal block may be used. The thermocouple module provides for use of a 0 °C referenced or 50 °C referenced terminal block.

6.1.14 Configuring the RS485/RS232 CM01 communications signal board

Click the CM01 communications signal board node of the system block (Page 129) dialog to configure options for an RS485/RS232 CM01 communications signal board that you have selected in the top section.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.



CM01 signal board type configuration

You configure the CM01 signal board type from the dropdown list, choosing one of the following options:

- RS485
- RS232

Address

Click the scroll buttons to enter the desired port address (1-126), for the RS485 or RS232 port: The default port address is 2.

Baud rate

Choose the desired data baud rate from the dropdown list:

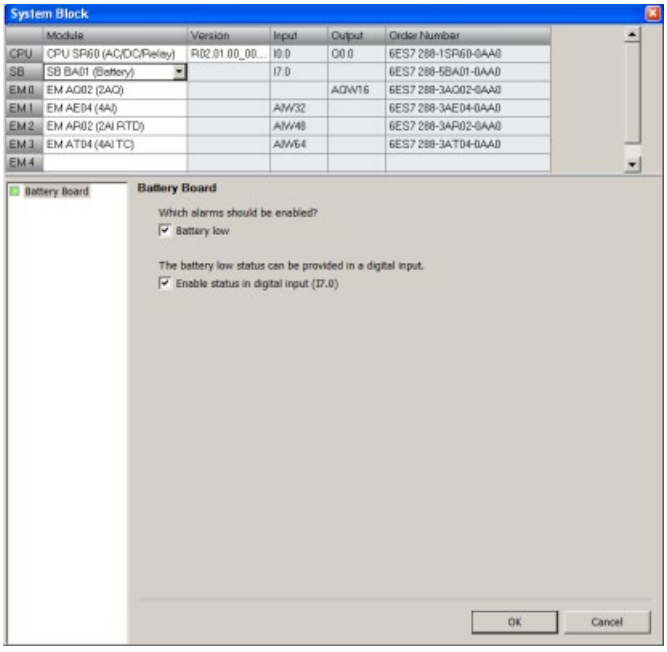
- 9.6 Kbps
- 19.2 Kbps
- 187.5 Kbps

6.1.15 Configuring the BA01 battery signal board

Click the BA01 battery signal board node of the system block (Page 129) dialog to configure options for a BA01 battery signal board that you have selected in the top section.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.



Enable bad diagnostic alarm

Click the “Enable bad diagnostic alarm” checkbox to trigger an alarm when the battery fails.

Enable status in digital input

Click the "Enable status in digital input" to enable a digital input to monitor the status of the signal board.

Operation of the Battery (BA01) Signal Board

The battery signal board contains a red LED that provides the customer a visual indication of the battery health. An Illuminated LED indicates a battery low condition.

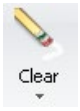
The CPU automatically utilizes the real-time clock on the signal board and performs the battery test and battery health LED operation, whether or not the System block contains a configuration for the signal board.

The battery signal board System block configuration contains selections that allow the customer to report the battery low state as a diagnostic alarm and/or to report the battery state (1=battery low, 0 = battery OK) in the LSB of the configured image register input byte for the device (for example, I7.0). The customer must select the battery signal board in the System block configuration in order to gain access to the additional battery health reporting options.

6.1.16 Clearing PLC memory

To clear designated areas of PLC memory, follow these steps:

1. Ensure that the PLC is in STOP mode.
2. Click the Clear button from the Modify area of the PLC menu ribbon strip.



WARNING

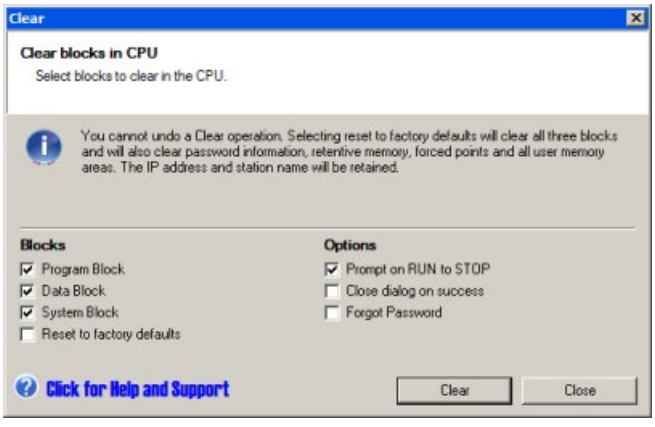
Effect of clearing PLC memory on outputs

Clearing the PLC memory affects the state of digital and analog outputs. The default is for digital and analog outputs to use a substitute value of 0. If you have defined substitute values other than 0 or chosen "Freeze" for your digital or analog outputs, you need to be aware that when you delete the system block, you are deleting the substitute and freeze information and, as a result, your outputs shall return to the default value of 0. Furthermore, if you perform a selective clear such that you keep your system block but delete your program block, then your analog outputs are frozen at their current value. Until you download a new program block, the only way to make changes to the state of the analog outputs is by means of the status chart.

If the S7-200 SMART PLC is connected to equipment when you clear the PLC memory, changes to the state of the digital outputs can be transmitted to the equipment. If you clear PLC memory without planning for the consequences to your digital and analog outputs, your equipment could operate in an unpredictable fashion, which could result in death or serious injury to personnel and/or damage to equipment.

Always follow appropriate safety precautions and ensure that your process is in a safe state before clearing the PLC memory.

3. Select what to clear - Program Block, Data Block, System Block, or all blocks, or select "Reset to factory defaults".
4. Click the Clear button.



Clearing the PLC memory requires the PLC to be in STOP mode and then deletes the selected blocks or resets the PLC to the factory-set defaults, depending on your selection. A clear operation does not clear the IP address, station name, or reset the time-of-day clock.

When executed, the "Reset to factory defaults" setting deletes all blocks, resets all user memory to the initial powerup state, and resets all Special Memory (Page 865) to initial values.

What to do if you forget the PLC password

If you forget the PLC password (Page 138), you can clear the PLC memory using one of two methods:

- Use a reset-to-factory-defaults memory card (Page 160) that you have made for this purpose (standard CPU models).
- Select the "Reset to factory defaults" choice under "Blocks", the "Forgot Password" choice under "Options", and power cycle the CPU.

Clearing the PLC using a reset-to-factory defaults memory card

For standard CPUs, you can clear the PLC using a previously-made reset-to-factory defaults memory card.

WARNING

Inserting a memory card into a CPU

Inserting a memory card into a CPU in RUN mode causes the CPU to automatically transition to STOP mode. You cannot change the CPU to RUN mode if a memory card is inserted.

Inserting a memory card into a running CPU can cause disruption to process operation, possibly resulting in death or severe personal injury.

Always ensure that the CPU is in STOP mode (Page 41) prior to inserting a memory card.

To clear the PLC using this card follow these steps:

1. Insert the reset-to-factory-defaults memory card. The CPU goes to STOP mode and flashes the STOP LED.
2. Power cycle the CPU. The CPU flashes the RUN/STOP LEDs until the reset is complete (about one second), and then flashes the STOP LED indicating that the reset is finished.
3. Remove the memory card.
4. Power cycle the CPU. The CPU is reset to the factory defaults. The former IP address and baud rate settings are cleared, but the time-of-day clock is unaffected.

After the CPU is reset, you can assign a new password and begin programming, or load a program from another program transfer memory card (Page 87) or from your hard disk.

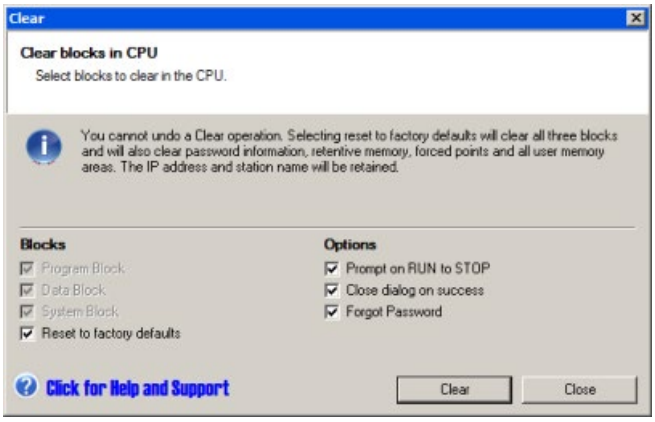
Note

If you load a password-protected program from a memory card or file on your hard disk, you must enter the password to access the protected areas. You cannot access a password-protected program component without a password, nor can you clear an assigned password without entry of the password.

Clearing the PLC by reset command followed by PLC power cycle

To clear the PLC when you have forgotten the password, follow these steps:

1. Click the Clear button from the Modify area of the PLC menu ribbon strip.
2. Select the "Reset to factory defaults" choice under "Blocks" and the "Forgot Password" choice under "Options".



3. Click the Clear button and power cycle the CPU within 60 seconds. Note that you must physically cycle the power within 60 seconds; a warm start or other reboot is not sufficient.

After you perform these steps within the required time frame, the CPU resets to factory defaults.

6.1.17 Creating a reset-to-factory-defaults memory card

You can create a memory card that will return a standard S7-200 SMART CPU to a factory default state. You can use this reset-to-factory-defaults memory card if you ever want to clear the contents of a standard CPU. To create a reset-to-factory-defaults memory card, follow these steps:

1. Using a card reader and Windows explorer, delete all contents from a microSDHC card.
2. Create a simple text file with an editor such as Notepad that contains one line with the string "RESET_TO_FACTORY". (Do not enter the quotation marks.)

3. Save this file to the microSDHC card root level under the file name "S7_JOB.S7S".
4. Label the card and store it in a safe place for future use.

Note

A reset-to-factory-defaults card is for resetting standard CPUs only

Because the compact serial (CRs) model CPUs do not have a microSD card interface, you cannot use a reset-to-factory-defaults card to clear the PLC and reset it to factory faults. See Clearing PLC memory (Page 158) for instructions on how to clear a PLC without the use of a reset-to-factory-defaults card.

6.2 High-speed I/O

High-speed counters

The CPU provides integrated high-speed counter functions that count high speed external events without degrading the performance of the CPU. Refer to the "Product overview" (Page 19) chapter for the rates supported by your CPU. Dedicated inputs exist for clocks, direction control, and reset, where these functions are supported. You can select single phase, dual phase, or AB quadrature phase for varying the counting rate. For more information, refer to the description of the high-speed counter instructions (Page 248).

High-speed pulse output

The standard CPU models support high-speed pulse outputs that generate either a high-speed pulse train output (PTO) or pulse width modulation (PWM) on certain outputs. Refer to the "Product overview" (Page 19) chapter for the quantity and rates supported by your CPU.

The PTO function provides a square wave (50% duty cycle) output for a specified number of pulses (from 1 to 2,147,483,647 pulses) and a specified frequency (in Hz). You can program the PTO function to produce either one train of pulses or a pulse profile consisting of multiple trains of pulses. For example, you can use a pulse profile to control a stepper motor through a simple ramp up, run, and ramp down sequence or more complicated sequences.

The PWM function provides a fixed cycle time with a variable duty cycle output, with the cycle time and the pulse width specified in either microsecond or millisecond increments. When the pulse width is equal to the cycle time, the duty cycle is 100 percent, and the output is turned on continuously. When the pulse width is zero, the duty cycle is 0 percent, and the output is turned off.

Refer to the pulse output instruction (Page 273) for more information. The chapter on open-loop motion control provides additional information using PWM (Page 673).

Open-loop motion control

The standard CPU models support an open-loop motion control capability. Motion profiles can be constructed and executed, interactive movement can be performed under user program control, and a number of built-in reference point seek sequences are available.

Depending upon configuration, open-loop motion support in the CPU requires the use of certain CPU resources, such as high-speed outputs, high-speed counters, and edge interrupts.

Refer to the "Product overview" (Page 19) chapter for the quantity of motion axes and pulse rates supported by your CPU.

Refer to the chapter on open-loop motion control (Page 673) for a full description of the motion capabilities in your CPU.

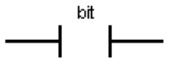
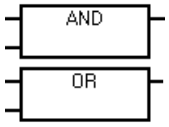
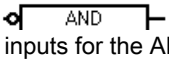
Program instructions

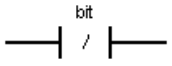
7

7.1 Bit logic

7.1.1 Standard inputs

LAD	FBD	STL	Description

		`LD bit`
		`A bit`
		`O bit`
			Test a bit value in memory (M, SM, T, C, V, S, L) or process image register (I or Q).
			LAD: Normally open and normally closed switches are represented by a contact symbol. If power flow is present on the left-side and the contact is closed, then power flows through the contact to the right-side connector and to the next connected element.
			• The Normally Open (N.O.) LAD contact is closed (ON) when the bit is equal to 1.
			• The Normally Closed (N.C.) LAD contact is closed (ON) when the bit is equal to 0.
			
			FBD: Normally open instructions are represented by AND/OR boxes. Box instructions can be used to evaluate Boolean signals in the same manner as ladder contact networks. Normally closed instructions are also represented by boxes. A normally closed instruction is created by placing the negation circle  , on a binary input signal connector. The number of inputs for the AND/OR boxes can be expanded to a maximum of 31 inputs.
			
			STL: The normally open contact is represented by the LD (load), A (AND), and O (OR) instructions. These instructions load, AND, or OR the value of the addressed bit with the top bit of the logic stack. The normally closed contact is represented by the LDN (Load NOT), AN (AND NOT), and ON (OR NOT) instructions. These instructions load, AND, or OR the logical NOT of the addressed bit value with the top bit of the logic stack.

		`LDN bit`
		`AN bit`
		`ON bit`	

Input / output	Data type	Operand
bit (LAD, STL)	BOOL	I, Q, V, M, SM, S, T, C, L,
Input (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
Output (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

FBD AND/OR input assignment

The editor feature described in the following table is active only if an input stub is selected and colored red, inside the FBD box cursor.

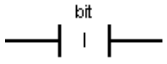
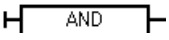
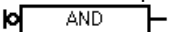
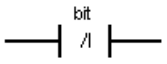
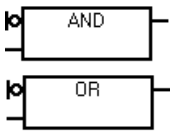
Input option	Place cursor	Tool button	Shortcut key
Add input	On box		+
Remove input	On box and bottom input		-

See also

Bit logic input examples (Page 176)

Logic stack overview (Page 166)

7.1.2 Immediate inputs

LAD	FBD	STL	Description
		**LDI bit** **AI bit** **OI bit**	The immediate instruction obtains the physical input value when the instruction is executed, but the process image register is not updated. An immediate contact does not wait on the PLC scan cycle to update; it updates immediately. The Normally Open Immediate contact is closed (ON) when the physical input point (bit) state is 1. The Normally Closed Immediate contact is closed (ON) when the physical input point (bit) state is 0. **LAD**: Normally open and normally closed immediate instructions are represented by contacts. **FBD**: Normally open immediate instructions are represented by the vertical immediate indicator  in front of an input connection. The Normally closed immediate instruction is represented by the immediate indicator and negation circle  in front of an input connection. The immediate indicator cannot be used when a logic flow connection is used instead of a physical input (I) bit address. FBD box instructions can be used to evaluate physical signals in the same manner as ladder contacts. The number of inputs for the AND/OR boxes can be expanded to a maximum of 31 inputs. **STL**: The Normally Open Immediate contact is represented by the LDI (Load Immediate), AI (AND Immediate), and OI (OR Immediate) instructions. These instructions load, AND, or OR the physical input value with the top of the logic stack. A normally Closed Immediate contact is represented by the LDNI (Load NOT Immediate), ANI (AND NOT Immediate), and ONI (OR NOT Immediate) instructions. These instructions immediately load, AND, or OR the logical NOT of the value of the physical input value with the top of the logic stack.
		**LDNI bit** **ANI bit** **ONI bit**	

Input / output	Data type	Operand
bit (LAD, STL)	BOOL	I
Input (FBD)	BOOL	I

FBD editor input assignment

The editor feature described in the following table is active only if an input stub is selected and colored red, inside the FBD box cursor.

Input option	Place cursor	Tool button	Shortcut key
Add input	On box		+
Remove input	On box and bottom input		-
Toggle negate input	On box and input		F11
Toggle immediate input	On box and input		CTRL F11

See also

Bit logic input examples (Page 176)

Logic stack overview (Page 166)

7.1.3 Logic stack overview

The STEP 7-Micro/WIN SMART program compiler uses the logic stack to transform the graphical I/O networks of LAD and FBD programs into STL (statement list) programs. The resultant STL program is logically the same as the original LAD or FBD graphical network and can be executed as a program list. All successfully compiled LAD and FBD programs have generated the underlying STL program and can be viewed as LAD, FBD, or STL.

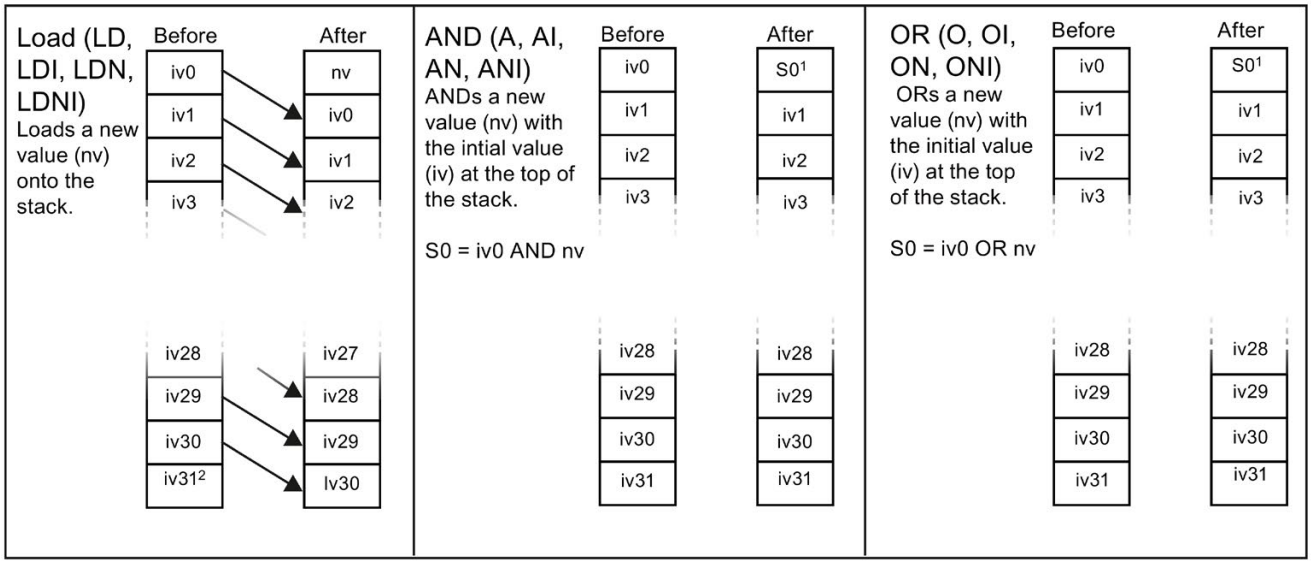
For LAD and FBD editing, the STL logic stack instructions are automatically generated and the programmer does not need to use the logic stack instructions.

You can also create STL programs directly with the STL editor. An STL programmer uses the logic stack instructions directly. Combination logic can be created in the STL editor that is too complex to be viewed in the LAD or FBD editor, but may be necessary for special applications.

All successfully compiled LAD and FBD programs can be viewed in STL, but not all successfully compiled STL programs can be viewed in LAD or FBD.

Input networks and the logic stack

As shown in the following figure, the CPU uses a logic stack to combine the logic states of STL inputs. In these examples, "iv0" to "iv31" identify the initial values of the logic stack levels, "nv" identifies a new value provided by the instruction, and "S0" identifies the calculated value that is stored in the logic stack.



[1] S0 identifies the calculated value that is stored in the logic stack.
[2] After the execution of a Load, the value iv31 is lost.

Output networks and the logic stack

ENO is a binary output for boxes in LAD and FBD. If a LAD box has power flow at the EN input and is executed without error, the ENO output passes power flow to the next LAD element. You can use the ENO as an enable bit that indicates the successful completion of an instruction. The ENO bit is used with the top of stack to affect power flow for execution of subsequent instructions. STL instructions do not have an EN input. The top of the stack must have a value of logic 1 for conditional instructions to be executed. In STL there is no ENO output. However, the STL instructions that correspond to LAD and FBD instructions with ENO outputs set a special ENO bit. This bit is accessible with the AND ENO (AENO) instruction.

STL	Description
AENO	AENO is used in the STL representation of LAD/FBD box ENO bit. AENO performs a logical AND of the ENO bit with the top of stack for the same effect as the ENO bit of a LAD/FBD box. The result of the AND operation is the new top of stack value.

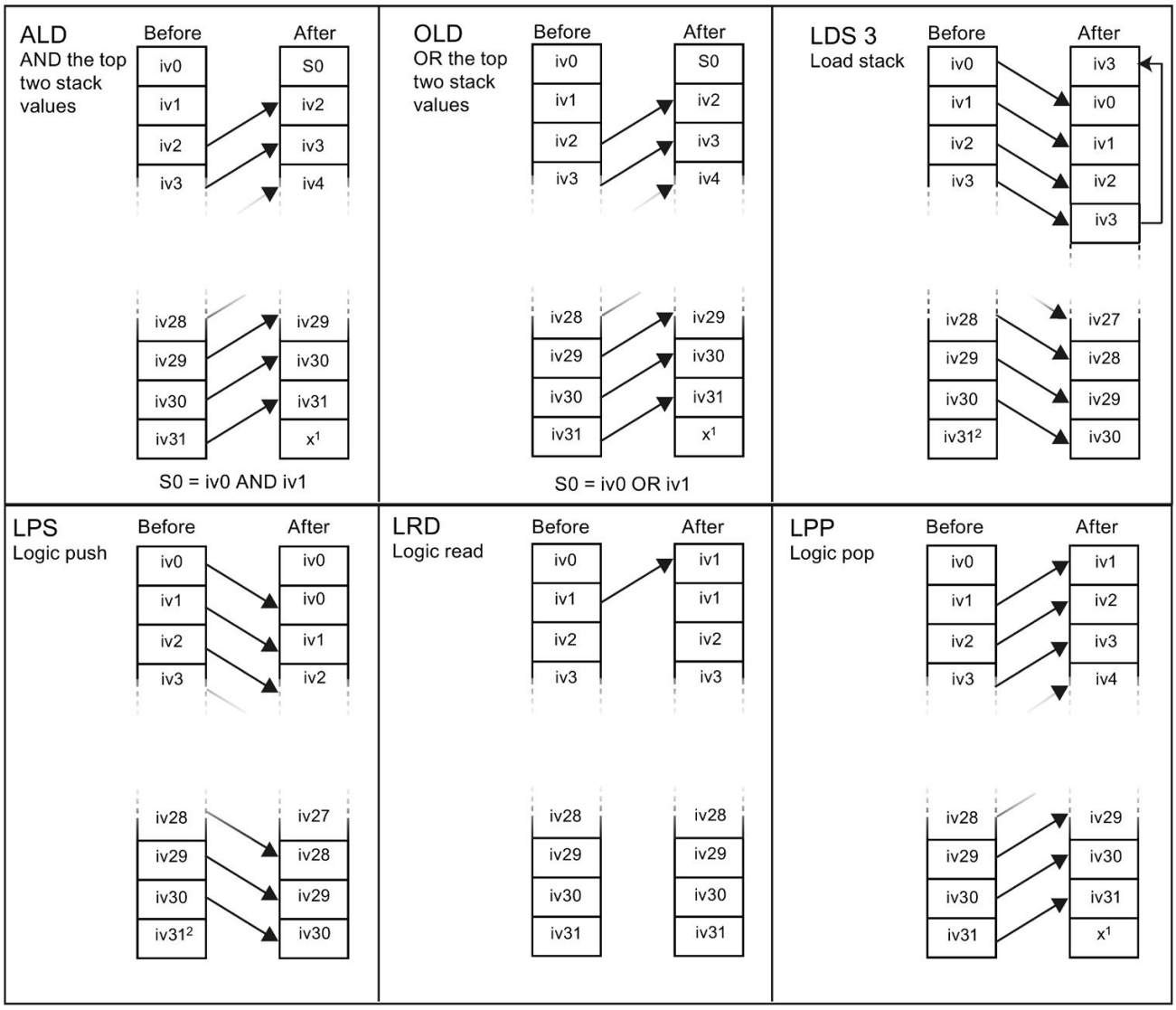
7.1.4 STL logic stack instructions

STL[1]	Description
ALD	The AND Load instruction (ALD) combines the values in the first and second levels of the stack using a logical AND operation. The result is loaded in the top of stack. After the ALD is executed, the stack depth is decreased by one.
OLD	The OR Load instruction (OLD) combines the values in the first and second levels of the stack, using a logical OR operation. The result is loaded in the top of the stack. After the OLD is executed, the stack depth is decreased by one.
LPS	The Logic Push instruction (LPS) duplicates the top value on the stack and pushes this value onto the stack. The bottom of the stack is pushed off and lost.
LRD	The Logic Read instruction (LRD) copies the second stack value to the top of stack. The stack is not pushed or popped, but the old top-of-stack value is destroyed by the copy.
LPP	The Logic Pop instruction (LPP) pops one value off of the stack. The second stack value becomes the new top of stack value.
LDS N	The Load Stack instruction (LDS) duplicates the stack bit (N) on the stack and places this value on top of the stack. The bottom of the stack is pushed off and lost.
AENO	AENO is used in the STL representation of the LAD/FBD box ENO bit. AENO performs a logical AND of the ENO bit with the top of stack for the same effect as the ENO bit of a LAD/FBD box. The result of the AND operation is the new top of stack.

[1] Not applicable for LAD or FBD

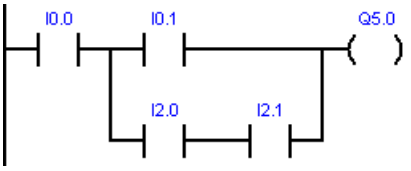
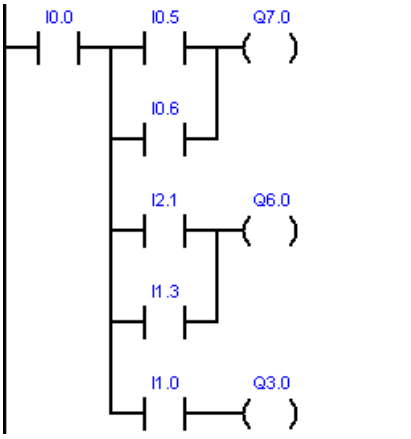
LDS (Load Stack) Input	Data type	Operands
N	BYTE	Constant (0 to 31)

As shown in the following figure, the CPU uses a logic stack to resolve the control logic. In these examples, "iv0" to "iv31" identify the initial values of the logic stack, "nv" identifies a new value provided by the instruction, and "S0" identifies the calculated value that is stored in the logic stack.

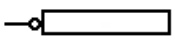


[1] The value is unknown (it could be either a 0 or a 1).
[2] After the execution of a Logic push or a Load stack instruction, value iv31 is lost.

Logic Stack example: LAD networks transformed into STL code

LAD	STL
	Network 1 LD I0.0 LD I0.1 LD I2.0 A I2.1 OLD ALD = Q5.0
	Network 2 LD I0.0 LPS LD I0.5 O I0.6 ALD = Q7.0 LRD LD I2.1 O I1.3 ALD = Q6.0 LPP A I1.0 = Q3.0

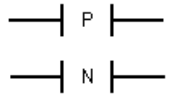
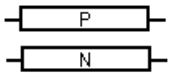
7.1.5 NOT

LAD	FBD	STL	Description
		NOT	The Not instruction (NOT) inverts the state of the power flow input. **LAD**: The NOT contact changes the state of power flow input. When power flow reaches the NOT contact, it stops. When power flow does not reach the NOT contact, it supplies power flow. **FBD**: The NOT instruction is represented as a graphical negation (bubble) symbol on Boolean box input connectors and functions as a logic state inverter. **STL**: The NOT instruction changes the value on the top of the stack from 0 to 1, or from 1 to 0.

See also

Bit logic input examples (Page 176)

7.1.6 Positive and negative transition detectors

LAD	FBD	STL	Description
		**EU** **ED**	The positive transition contact instruction (Edge Up) allows power to flow for one scan for each OFF-to-ON transition. The negative transition contact instruction (Edge Down) allows power to flow for one scan for each ON-to-OFF transition. S7-200 SMART CPUs support a combined total (positive and negative) of 1024 edge detector instructions in your program. **LAD**: Positive and negative transition instructions are represented by contacts. **FBD**: The transition instructions are represented by the P and N boxes. **STL**: The positive transition is detected by the EU (Edge Up) instruction. Upon detection of a 0-to-1 transition in the value on the top of the stack, the top of the stack value is set to 1; otherwise, it is set to 0. The negative transition is detected by the ED (Edge Down) instruction. Upon detection of a 1-to-0 transition in the value on the top of the stack, the top of the stack value is set to 1; otherwise, it is set to 0.

Input / output	Data type	Operand
IN (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
OUT (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

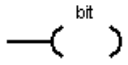
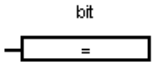
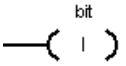
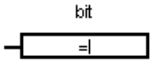
Note

Because the Positive Transition and Negative Transition instructions require an on-to-off or an off-to-on transition, you cannot detect an edge-up or edge-down transition on the first scan. During the first scan, the CPU saves the initial input state in a memory bit. On subsequent scans, these instructions compare the current state and the state of the memory bit, to detect a transition.

See also

Bit logic input examples (Page 176)

7.1.7 Coils: output and output immediate instructions

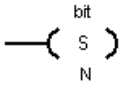
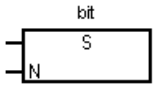
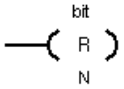
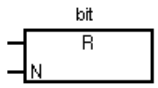
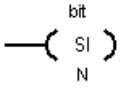
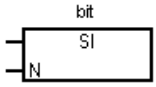
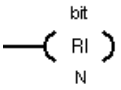
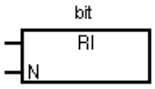
LAD	FBD	STL	Description
		`= bit`	The Output instruction writes the new value for the output bit to the process image register. **LAD** and **FBD**: When the output instruction is executed, the S7-200 turns the output bit in the process image register ON or OFF. The assigned bit is set equal to power flow state. **STL**: The value on the top of the stack is copied to the assigned bit.
		`=I bit`	The Output Immediate instruction writes the new value to both the physical output and the corresponding process image register location when the instruction is executed. **LAD** and **FBD**: When the output immediate instruction is executed, the physical output point (bit) is immediately set equal to power flow. The "I" indicates an immediate reference; the new value is written to both the physical output point and the corresponding process image register address. This differs from the non-immediate references, which only write the new value to the process image register. **STL**: The instruction immediately copies the value on the top of the stack to the assigned physical output bit and process image address.

Input / output	Data type	Operand
Bit	BOOL	I, Q, V, M, SM, S, T, C, L
Bit (immediate)	BOOL	Q
Input (LAD)	BOOL	Power flow
Input (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

See also

Bit logic output examples (Page 177)

7.1.8 Set, reset, set immediate, and reset immediate functions

LAD	FBD	STL	Description
		`S bit, N`	The Set (S) and Reset (R) instructions set (ON) or reset (OFF) the number of bits (N), starting at the address (bit). You can set or reset from 1 to 255 bits. If the Reset instruction specifies either a timer bit (T address) or counter bit (C address), the instruction resets the timer or counter bit and clears the current value of the timer or counter.
		`R bit, N`	
		`SI bit, N`	The Set Immediate and Reset Immediate instructions immediately set (ON) or immediately reset (OFF) the number of points (N), starting at address (bit). You can set or reset from 1 to 255 points immediately. The "I" indicates an immediate reference; when the instruction is executed, the new value is written to both the physical output point and the corresponding process image register location. This differs from the non-immediate references, which write the new value to the process image register only.
		`RI bit, N`	

Non-fatal errors with ENO = 0	SM bits affected
• N = 0 (zero) • 0006H Indirect address • 0091H Operand out of range	None

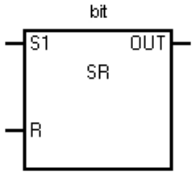
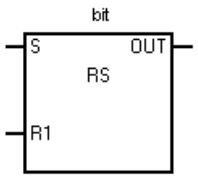
Input / output	Data type	Operand
Bit	BOOL	I, Q, V, M, SM, S, T, C, L
Bit (immediate)	BOOL	Q
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, Constant, *VD, *AC, *LD
Input (LAD)	BOOL	Power flow
Input (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

See also

Bit logic input examples (Page 176)

Bit logic output examples (Page 177)

7.1.9 Set and reset dominant bistable

LAD/FBD [1]	Description
	The bit parameter assigns the Boolean address that is set or reset. The optional OUT connection reflects the signal state of the Bit parameter. **SR** (Set dominant bistable) is a latch where the set dominates. If the set (S1) and reset (R) signals are both true, the output (OUT) is true.
	**RS** (Reset dominant bistable) is a latch where the reset dominates. If the set (S) and reset (R1) signals are both true, the output (OUT) is false.

[1] Not applicable for STL

Input / outputs	Data type	Operand
bit	BOOL	I, Q, V, M, S
S1, R (LAD SR)	BOOL	Power flow
S, R1 (LAD RS)	BOOL	Power flow
OUT (LAD)	BOOL	Power flow
S1, R (FBD SR)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
S, R1 (FBD RS)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
OUT (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

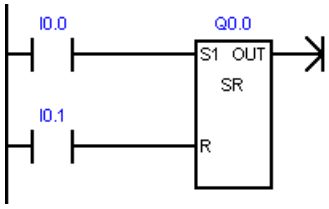
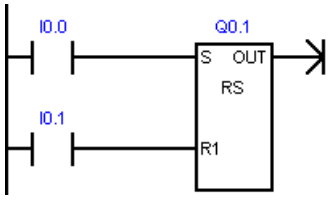
SR truth table

S1	R	Out (bit)
0	0	Previous state
0	1	0
1	0	1
1	1	1

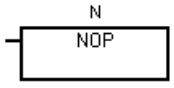
RS truth table

S	R1	Out (bit)
0	0	Previous state
0	1	0
1	0	1
1	1	0

Example SR and RS

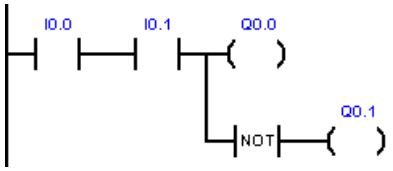
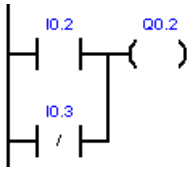
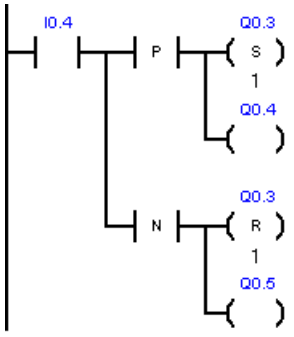
LAD	STL
	**NETWORK 1** **LD I0.0** **LD I0.1** **NOT** **A Q0.0** **OLD** **= Q0.0**
	**NETWORK 2** **LD I0.0** **LD I0.1** **NOT** **LPS** **A Q0.1** **= Q0.1** **LPP** **ALD** **O Q0.1** **= Q0.1**

7.1.10 NOP (No operation) instruction

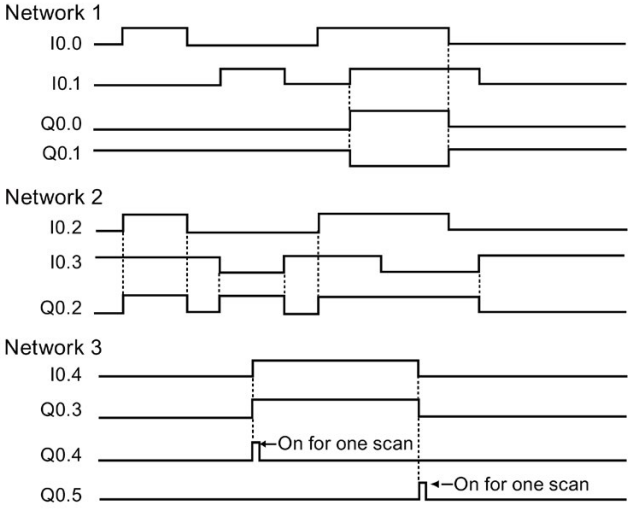
LAD	STL	Description
	**NOP N**	The No Operation (NOP) instruction has no effect on the user program execution. This instruction is not available in FBD mode. The operand N is a number from 0 to 255.

Inputs / Output	Data type	Operand
N (LAD, STL)	BYTE	N: Constant (0 to 255)

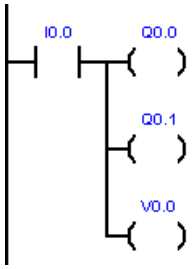
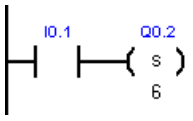
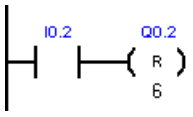
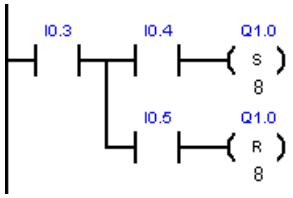
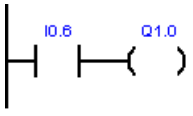
7.1.11 Bit logic input examples

LAD		STL
	Normally-open contacts I0.0 AND I0.1 must be ON (closed) to activate Q0.0. The NOT instruction acts as an inverter. In RUN mode, Q0.0 and Q0.1 have opposite logic states.	**Network 1** **LD I0.0** **A I0.1** **= Q0.0** **NOT** **= Q0.1**
	(Normally-open contact I0.2 must be ON) or (Normally-closed contact I0.3 must be OFF), to activate Q0.2. One or more parallel LAD branches (OR logic) must be true to make the output active.	**Network 2** **LD I0.2** **ON I0.3** **= Q0.2**
	A positive Edge Up input on a P contact or a negative Edge Down input on an N contact outputs a pulse with a 1 scan cycle duration. In RUN mode, the pulsed state changes of Q0.4 and Q0.5 are too fast to be visible in program status view. The Set and Reset outputs latch the pulse state into Q0.3 and make the state change visible in program status view.	**Network 3** **LD I0.4** **LPS** **EU** **S Q0.3, 1** **= Q0.4** **LPP** **ED** **R Q0.3, 1** **= Q0.5**

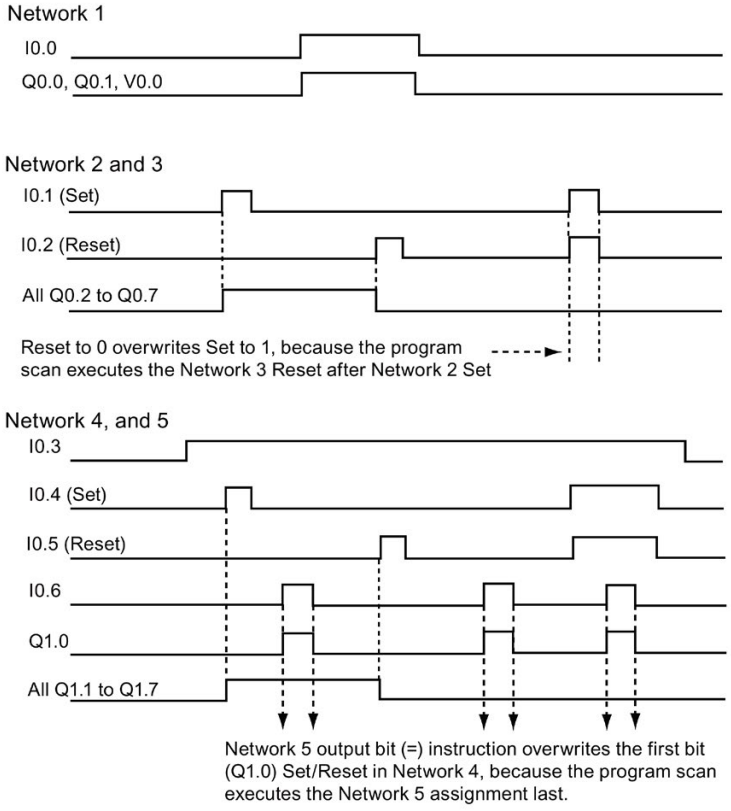
Run-mode timing for input example



7.1.12 Bit logic output examples

LAD		STL
	Output instructions assign bit values to external I/O (I, Q) and internal memory (M, SM, T, C, V, S, L).	**Network 1** **LD I0.0** **= Q0.0** **= Q0.1** **= V0.0**
	Set a sequential group of 6 bits to a value of 1. Specify a starting bit address and how many bits to set. The program status indicator for Set is ON when the value of the first bit (Q0.2) is 1.	**Network 2** **LD I0.1** **S Q0.2, 6**
	Reset a sequential group of 6 bits to a value of 0. Specify a starting bit address and how many bits to reset. The program status indicator for Reset is ON when the value of the first bit (Q0.2) is 0.	**Network 3** **LD I0.2** **R Q0.2, 6**
	Sets and resets 8 output bits (Q1.0 to Q1.7) as a group.	**Network 4** **LD I0.3** **LPS** **A I0.4** **S Q1.0, 8** **LPP** **A I0.5** **R Q1.0, 8**
	The Set and Reset instructions perform the function of a latched relay. To isolate the Set/Reset bits, make sure they are not overwritten by another assignment instruction. In this example, Network 4 sets and resets eight output bits (Q1.0 to Q1.7) as a group. In RUN mode, Network 5 can overwrite the Q1.0 bit value and control the Set/Reset program status indicators in Network 4.	**Network 5** **LD I0.6** **= Q1.0**

Run-mode timing for output examples



7.2 Clock

7.2.1 Read and set real-time clock

LAD / FBD	STL	Description
READ_RTC EN ENO T	**TODR T**	The Read real-time clock instruction reads the current time and date from the CPU and loads it in an 8 byte Time buffer starting at byte address T.
SET_RTC EN ENO T	**TODW T**	The Set real-time clock instruction writes a new time and date to the CPU using the 8 byte Time buffer data that is assigned by T.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0007H T data error	None

Input	Data type	Operand
T	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

Note

READ_RTC, SET_RTC programming tips

These instructions do not accept Invalid dates. If you enter February 30, for example, a time-of-day non-fatal error occurs (0007H).

Do not use the READ_RTC / SET_RTC instructions in both the main program and in an interrupt routine. A READ_RTC / SET_RTC instruction in an interrupt routine cannot execute while another READ_RTC / SET_RTC instruction is executing. In this case, the CPU sets system flag bit SM4.3, indicating that two simultaneous accesses to the clock were attempted resulting in a T data error (non-fatal error 0007H).

The time-of-day clock in the CPU uses only the least significant two digits for the year, so 00 represents the year 2000. User programs that use the year's value must take into account the two-digit representation.

The CPU handles leap year correctly through year 2099.

Format of 8 byte time buffer, beginning at byte address T

You must assign all date and time values in BCD format (for example, 16#12 for the year 2012). The BCD value range of 00 to 99 can assign years, in 2000 to 2099 range.

T byte	Description	Data value
0	Year	00 to 99 (BCD value) Year 20xx: where xx is the two digit BCD value in T-byte 0
1	Month	01 to 12 (BCD value)
2	Day	01 to 31 (BCD value)
3	Hour	00 to 23 (BCD value)
4	Minute	00 to 59 (BCD value)
5	Second	00 to 59 (BCD value)
6	reserved	Always set to 00
7	Day of week	Value ignored when written with the SET_RTC / TODW instruction. Value reports correct day of week when read with the READ_RTC / TODR instruction based upon current Year/Month/Day values. 1 to 7, 1 = Sunday, 7 = Saturday (BCD value)

Extended power outage effect on the CPU clock

See the S7-200 SMART system manual appendix A CPU specifications, for how long the real-time clock can maintain the correct time during power outages.

A CPU initializes with the time values shown in the following table, after an extended power outage.

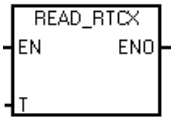
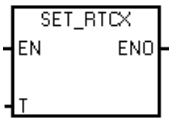
Date	Time	Day of week
01-Jan-2000	00:00:00	Saturday

Note

Compact serial (CRs) CPU models do not have a RTC (Real-time Clock)

You can use the READ_RTC and SET_RTC instructions to set the year, date, and time values in compact serial (CRs) CPU models, but the values will be lost on the next CPU power off-on cycle. On power-up, the date and time will initialize to January 1, 2000.

7.2.2 Read and set real-time clock extended

LAD / FBD	STL	Description
	**TODRX T**	The Read real-time clock extended instruction reads the current time, date, and daylight saving configuration from the PLC and loads it in a 19-byte buffer beginning at the address assigned by T.
	**TODWX T**	The Set real-time clock instruction writes a new time, date, and daylight saving configuration to the PLC using the 19-byte buffer data that is assigned by byte address T.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0007H T data error • 0091H Operand out of range	None

Input	Data type	Operand
T	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

Note

READ_RTCX, SET_RTCX programming tips

These instructions do not accept Invalid dates. If you enter February 30, for example, a time-of-day non-fatal error occurs (0007H).

Do not use the READ_RTCX / SET_RTCX instructions in both the main program and in an interrupt routine. A READ_RTCX / SET_RTCX instruction in an interrupt routine cannot execute while another READ_RTCX / SET_RTCX instruction is executing. In this case, the CPU sets system flag bit SM4.3, indicating that two simultaneous accesses to the clock were attempted resulting in a T data error (non-fatal error 0007H).

The time-of-day clock in the CPU uses only the least significant two digits for the year, so 00 represents the year 2000. User programs that use the year's value must take into account the two-digit representation.

The CPU handles leap year correctly through year 2099.

Format of 19 byte time buffer, beginning at byte address T

Note

T bytes (9 to18) or (9 to 20) are used only when a time correction mode is assigned in byte 8. Otherwise, the last values written to bytes (9 to18) or (9 to 20) by STEP 7-Micro/WIN SMART or the SET_RTCX instruction are returned.

You must assign all date and time values in BCD format (for example, 16#12 for the year 2012). The BCD value range of 00 to 99 can assign the year in the range of 2000 to 2099.

T byte	Description	Data value
0	Year	00 to 99 (BCD value) Year 20xx: where xx is the two digit BCD value in T-byte 0
1	Month	01 to 12 (BCD value)
2	Day	01 to 31 (BCD value)
3	Hour	00 to 23 (BCD value)
4	Minute	00 to 59 (BCD value)
5	Second	00 to 59 (BCD value)
6	reserved	Always set to 00
7	Day of week	Value ignored when written with the SET_RTCX / TODWX instruction. Value reports correct day of week when read with the READ_RTCX / TODRX instruction based upon current Year/Month/Day values. 1 to 7, 1 = Sunday, 7 = Saturday (BCD value)
8	Correction mode: For Daylight saving time (DST)	00H = correction disabled 01H = EU (time zone offset from UTC = 0 hrs) [1] 02H = EU (time zone offset from UTC = +1 hrs) [1] 03H = EU (time zone offset from UTC = +2 hrs) [1] 04H-07H = reserved 08H = EU (time zone offset from UTC = -1 hrs) [1] 09H-0FH = reserved 10H = US [2] 11H = Australia [3] 12H = reserved 13H = New Zealand [4] 14H-EDH = reserved EEH = user defined (day of week) (using values in bytes 9-20) EFH-FDH reserved FEH = reserved FFH = user defined (day of month) (using values in bytes 9-18)
The following bytes 9-18 are used only for correction mode = FFH (legacy user assigned)		
9	DST correction hours	0 to 23 (BCD value)
10	DST correction minutes	0 to 59 (BCD value)
11	DST beginning month	1 to 12 (BCD value)
12	DST beginning day	1 to 31 (BCD value)
13	DST beginning hour	0 to 23 (BCD value)
14	DST beginning minute	0 to 59 (BCD value)

T byte	Description	Data value
15	DST ending month	1 to 12 (BCD value)
16	DST ending day	1 to 31 (BCD value)
17	DST ending hour	0 to 23 (BCD value)
18	DST ending minute	0 to 59 (BCD value)
The following bytes 9-20 are used only for correction mode = EEH (extended user assigned)		
9	DST correction hours	0 to 23 (BCD value)
10	DST correction minutes	0 to 59 (BCD value)
11	DST beginning month	1 to 12 (BCD value)
12	DST beginning week	1 to 5 (BCD value) [5]
13	DST beginning weekday	1 to 7 (BCD value)
14	DST beginning hour	0 to 23 (BCD value)
15	DST beginning minute	0 to 59 (BCD value)
16	DST ending month	1 to 12 (BCD value)
17	DST ending week	1 to 5 (BCD value) [5]
18	DST ending weekday	1 to 7 (BCD value)
19	DST ending hour	0 to 23 (BCD value)
20	DST ending minute	0 to 59 (BCD value)

[1] EU convention: Adjust time ahead one hour on last Sunday in March at 1:00 a.m. UTC. Adjust time back one hour on last Sunday in October at 2:00 a.m. UTC. (The local time when the correction is made depends upon the time zone offset from UTC).

[2] US convention: 2007 standard - Adjust time ahead one hour on second Sunday in March at 2:00 a.m. local time. Adjust time back one hour on first Sunday in November at 2:00 a.m. local time.

[3] Australia convention: 2007 standard - Adjust time ahead one hour on first Sunday in October at 2:00 a.m. local time. Adjust time back one hour on first Sunday in April at 2:00 a.m. local time (also for Australia - Tasmania).

[4] New Zealand convention: 2007 standard - Adjust time ahead one hour on last Sunday in September at 2:00 a.m. local time. Adjust time back one hour on first Sunday in April at 2:00 a.m. local time.

[5] To assign the last occurrence of the weekday in the month (for example, the last Monday in April), set the week = 5.

Extended power outage effect on the CPU clock

See the S7-200 SMART system manual appendix A CPU specifications, for how long the real-time clock can maintain the correct time during power outages.

A CPU initializes with the time values shown in the following table, after an extended power outage.

Date	Time	Day of week
01-Jan-2000	00:00:00	Saturday

Note

Compact serial (CRs) CPU models do not have a RTC (Real-time Clock)

You can use the READ_RTCX and SET_RTCX instructions to set the year, date, and time values in compact serial (CRs) CPU models, but the values will be lost on the next CPU power off-on cycle. On power-up, the date and time will initialize to January 1, 2000.

7.3 Communication

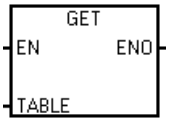
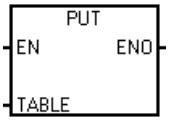
7.3.1 GET and PUT (Ethernet)

You can use the GET and PUT instructions for communication between S7-200 SMART CPUs through the Ethernet connection.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

Table 7- 1 GET and PUT instructions

LAD/FBD	STL	Description
	`GET table`	The GET instruction initiates a communications operation on the Ethernet port to gather data from a remote device, as defined in the description table (TABLE). The GET instruction can read up to 222 bytes of information from a remote station.
	`PUT table`	The PUT instruction initiates a communications operation on the Ethernet port to write data to a remote device, as defined in the description table (TABLE). The PUT instruction can write up to 212 bytes of information to a remote station.

You can have any number of GET and PUT instructions in the program, but only a maximum of 16 GET and PUT instructions can be activated at any one time. For example, you can have eight GET and eight PUT instructions, or six GET and ten PUT instructions, active at the same time in a given CPU.

When you execute a GET or PUT instruction, the CPU makes an Ethernet connection to the remote IP address in the GET or PUT table. The CPU maintains a maximum of eight connections at a time. Once a connection is established, that connection is maintained until the CPU goes to STOP mode.

The CPU utilizes a single connection for all GET/PUT instructions that are directed to the same IP address. For example, if there are three GET instructions enabled at the same time when the remote IP address is 192.168.2.10, then the GET instructions execute sequentially on one Ethernet connection to IP address 192.168.2.10.

If you try to create a ninth connection (a ninth IP address), the CPU searches through the connections to find the connection that has been inactive for the longest period of time. The CPU disconnects this connection and then creates a connection to the new IP address.

The GET and PUT instructions require additional communication background time (refer to "Configuring communication" (Page 131)) when they are processing/active/busy and also when they are just maintaining the connection to the other device. The amount of communication background time required depends on the number of GET and PUT instruction that are active/busy, how often the GET and PUT instructions are executed, and the number of connections that are currently open. You should adjust the communication background time to a higher value if the communication performance is slow.

Table 7- 2 Valid operands for the GET and PUT instructions

Inputs/Outputs	Data Type	Operands
TABLE	BYTE	IB, QB, VB, MB, SMB, SB, *VD, *LD, *AC

Error conditions that set ENO = 0:

- 0006 (indirect address)
- If the function returns an error and sets the E bit of table status byte (see the figure below)

The following figure describes the table that is referenced by the TABLE parameter, and the following table lists the error codes.

Table 7- 3 Definition of GET and PUT instructions TABLE parameter

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	D [1]	A [2]	E [3]	0	Error code			
1	Remote station IP Address [4]							
2								
3								
4								
5	Reserved = 0 (Must be set to zero)							
6	Reserved = 0 (Must be set to zero)							
7	Pointer to the data area in the remote station (I, Q, M, V, or DB1) [5]							
8								
9								
10								

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
11	Data length [6]							
12	Pointer to the data area in the local station (this CPU) (I, Q, M, V, or DB1) [7]							
13								
14								
15								

[1] D - Done (function has been completed)

[2] A - Active (function has been queued)

[3] E - Error (function returned an error)

[4] Remote station IP address: The address of the CPU whose data is to be accessed.

[5] Pointer to the data area in the remote station: An indirect pointer to the data that is to be accessed in the remote station.

[6] Data length: The number of bytes of data that are to be accessed in the remote station (1 to 212 bytes for PUT and 1 to 222 bytes for GET).

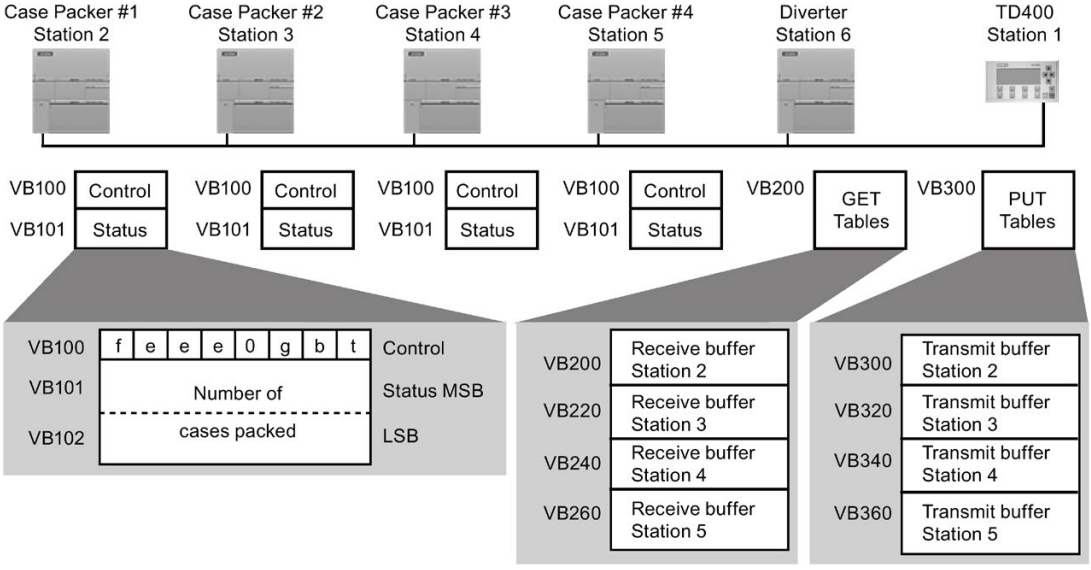
[7] Pointer to the data area in the local station: An indirect pointer to the data that is to be accessed in the local station (this CPU).

Table 7- 4 Error codes for the GET and PUT instructions TABLE parameter

Code	Definition
0	No error
1	Illegal parameter in the PUT/GET table: • Local area is not I, Q, M, or V • Local area is not large enough for the data length requested • Data length is zero or greater than 222 bytes for a GET or greater than 212 bytes for a PUT • Remote area is not I, Q, M, or V • Remote IP address is illegal (0.0.0.0) • Remote IP address is a broadcast address or a multicast address • Remote IP address is the same as the Local IP address • Remote IP address is on a different subnet
2	Too many PUT/GET instructions are currently active (only 16 allowed)
3	No connection available. All connections are currently active with outstanding requests
4	Error returned from remote CPU: • Too much data was requested or sent • Writing to Q memory is not allowed in STOP mode • Memory area is write-protected (see SDB configuration)

Code	Definition
5	No connection available to the remote CPU: • Remote CPU does not have an available server connection • Connection to remote CPU was lost (CPU powered off, physical disconnect)
6 to 9, A to F	Not used (Reserved for future use)

The following figure shows an example to illustrate the utility of the GET and PUT instructions. For this example, consider a production line where tubs of butter are being filled and sent to one of four boxing machines (case packers). The case packer packs eight tubs of butter into a single cardboard box. A diverter machine controls the flow of butter tubs to each of the case packers. Four CPUs control the case packers, and a CPU with a TD 400 operator interface controls the diverter.



t Out of butter tubs to pack; t=1, out of butter tubs

b Box supply is low; b=1, must add boxes in the next 30 minutes

g Glue supply is low; g=1, must add glue in the next 30 minutes

eee Error code identifying the type of fault experienced

f Fault indicator; f=1, the case packer has detected an error

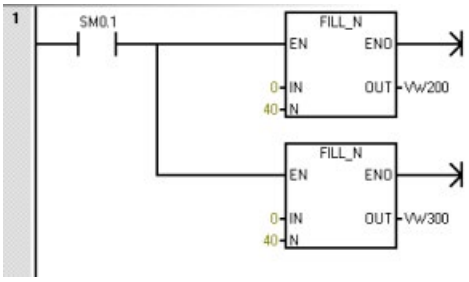
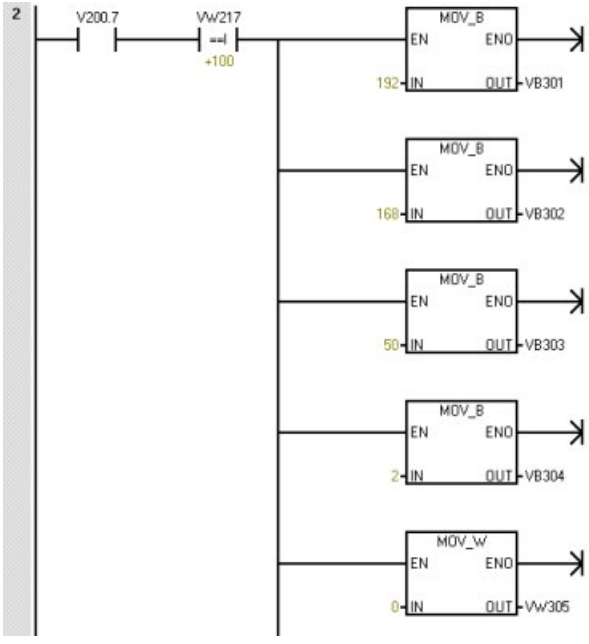
The following figure shows the GET table (VB200) and PUT table (VB300) for accessing the data in station 2. The Diverter CPU uses a GET instruction to read the control and status information on a continuous basis from each of the case packers. Each time a case packer has packed 100 cases, the diverter notes this and sends a message to clear the status word using a PUT instruction.

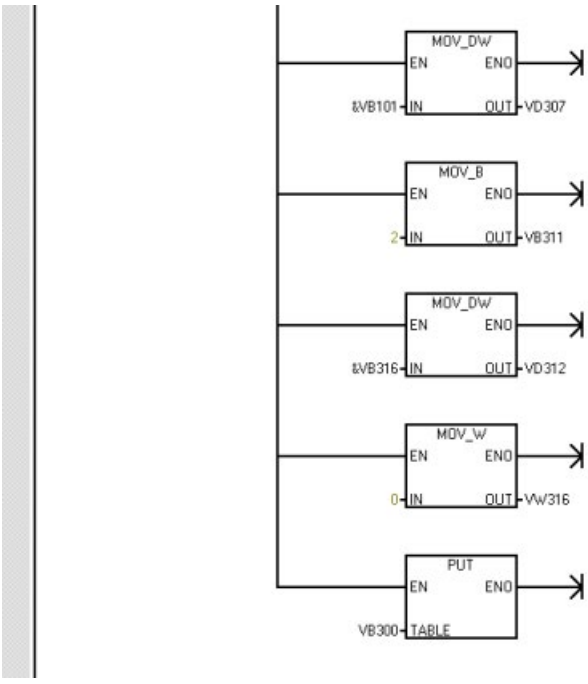
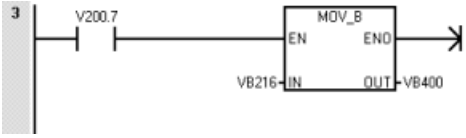
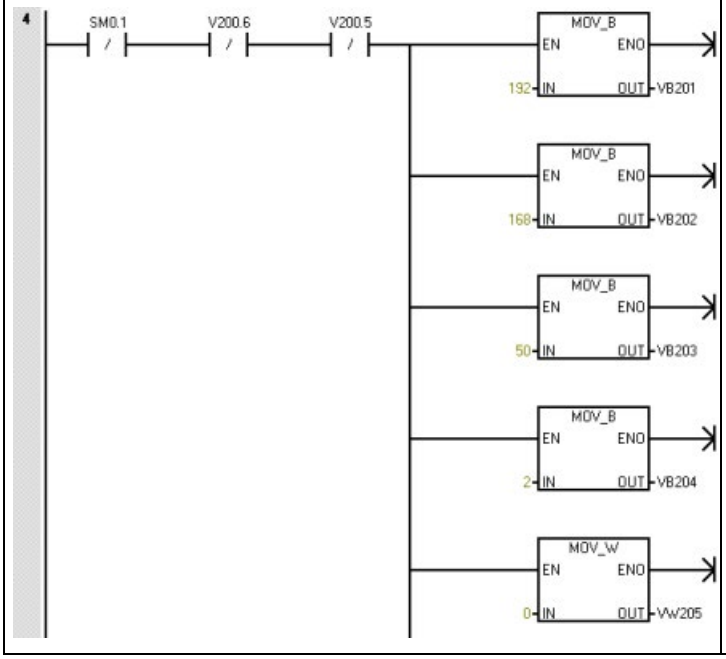
Table 7- 5 GET and PUT instructions buffer for reading from and clearing the count of Case Packer 1

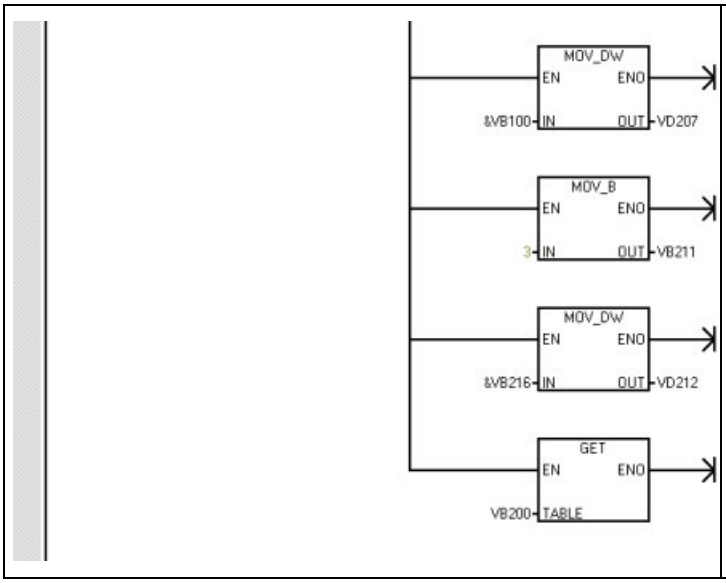
GET_ TABLE buffer	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**PUT_ TABLE buffer**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
VB200	D	A	E	0	Error code				VB300	D	A	E	0	Error code			
VB201	Remote station IP address = 192.								VB301	Remote station IP address = 192.							
VB202	168.								VB302	168.							
VB203	50.								VB303	50.							
VB204	2								VB304	2							
VB205	Reserved = 0 (Must be set to zero)								VB305	Reserved = 0 (Must be set to zero)							
VB206	Reserved = 0 (Must be set to zero)								VB306	Reserved = 0 (Must be set to zero)							
VB207	Pointer to the data								VB307	Pointer to the data							
VB208	area in the								VB308	area in the							
VB209	remote station =								VB309	remote station =							
VB210	(&VB100)								VB310	(&VB101)							
VB211	Data length = 3 bytes								VB311	Data length = 2 bytes							
VB212	Pointer to the data								VB312	Pointer to the data							
VB213	area in the								VB313	area in the							
VB214	local station (this CPU) =								VB314	local station (this CPU) =							
VB215	(&VB216)								VB315	(&VB316)							
VB216	Control								VB316	0							
VB217	Status MSB								VB317	0							
VB218	Status LSB																

In this example, the data immediately follows the PUT and GET tables. This data can be placed anywhere in the CPU memory since it is pointed to by the local station pointer in a table (for example, VB212 - VB215).

Table 7- 6 Example: GET and PUT instructions

	**Network 1** **LD SM0.1** **FILL +0, VW200, 40** **FILL +0, VW300, 40**	On the first scan, clear all receive and transmit buffers.
	**Network 2** **LD V200.7** **AW= VW217, +100** **MOVB 192, VB301** **MOVB 168, VB302** **MOVB 50, VB303** **MOVB 2, VB304** **MOVW 0, VB305** **MOVD &VB101, VD307** **MOVB 2, VB311** **MOVD &VB316, VD312** **MOVW 0, VW316** **PUT VB300**	When the GET "Done" bit (V200.7) is set and 100 cases have been packed: 1. Load the station address of case packer 1. 2. Load a pointer to the data in the remote station. 3. Load the length of data to be transmitted. 4. Load the data to transmit. Reset the number of cases packed by case packer 1

		
	**Network 3** **LD V200.7** **MOVB VB216, VB400**	When the GET "Done" bit is set, save the control data from case packer 1.
	**Network 4** **LDN SM0.1** **AN V200.6** **AN V200.5** **MOVB 192, VB201** **MOVB 168, VB202** **MOVB 50, VB203** **MOVB 2, VB204** **MOVW 0, VB205** **MOVD &VB100, VD207** **MOVB 3, VB211** **MOVD &VB216, VD212** **GET VB200**	If not the first scan and there are no errors: 1. Load the station address of case packer 1. 2. Load a pointer to the data in the remote station. 3. Load the length of data to be received. 4. Read the control and status data in case packer 1.

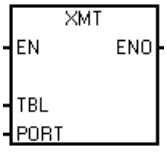
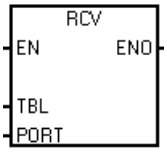
		

7.3.2 Transmit and receive (Freeport on RS485/RS232)

You can use the Transmit (XMT) and Receive (RCV) instructions for communication between a S7-200 SMART CPU and other devices through the CPU serial port(s). Each S7-200 SMART CPU provides an integrated RS485 port (Port 0). The standard CPUs additionally support an optional CM01 Signal Board (SB) RS232/RS485 port (Port 1). The communication protocol must be implemented in the user program.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of signal boards.

LAD / FBD	STL	Description
	**XMT TBL, PORT**	The Transmit instruction (XMT) is used in Freeport mode to transmit data by means of the communications port(s).
	**RCV TBL, PORT**	The Receive instruction (RCV) initiates or terminates the receive message function. You must specify a start and an end condition for the Receive box to operate. Messages received through the specified port (PORT) are stored in the data buffer (TBL). The first entry in the data buffer specifies the number of bytes received.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0009H Simultaneous Transmit/Receive on port 0 • 000BH Simultaneous Transmit/Receive on port 1 • 0090H Port number is invalid • Receive parameter error sets SM86.6 or SM186.6 • CPU is not in Freeport mode	• SM 86.6 Receive message terminated on port 0 • SM 186.6 Receive message terminated on port 1

Input / output	Data type	Operand
TBL	BYTE	IB, QB, VB, MB, SMB, SB, *VD, *LD, *AC
PORT	BYTE	Constant: 0 or 1 Note: The two available ports are as follows: • Integrated RS485 port (Port 0), • CM01 Signal Board (SB) RS232/RS485 port (Port 1)

Using Freeport mode to control the serial communications port

You can select the Freeport mode to control the serial communications port of the CPU by means of your user program. When you select Freeport mode, your program controls the operation of the communications port through the use of the receive interrupts, the transmit interrupts, the Transmit instruction, and the Receive instruction and entirely controls the communications protocol while in Freeport mode. You use SMB30 and SMB130 to select the baud rate and parity.

The CPU assigns two special memory bytes to the two physical ports:

- SMB30 to the integrated RS485 port (Port 0)
- SMB130 to the CM01 RS232/RS485 Signal Board (SB) port (Port 1)

The Freeport mode is disabled and normal communications are re-established (for example, HMI device access) when the CPU is in STOP mode.

In the simplest case, you can send a message to a printer or a display using only the Transmit (XMT) instruction. Other examples include a connection to a bar code reader, a weigh scale, and a welder. In each case, you must write your program to support the protocol that is used by the device with which the CPU communicates while in Freeport mode.

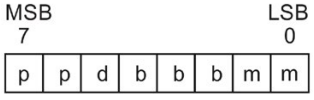
You can only use Freeport communications when the CPU is in RUN mode. Enable the Freeport mode by setting a value of 01 in the protocol select field of SMB30 (Port 0) or SMB130 (Port 1). While in Freeport mode, you cannot communicate with an HMI on the same port.

Note

The serial CR model CPUs disable Freeport mode when you connect a USB-PPI cable to the CPU. Likewise, the CPU inhibits the switch to Freeport mode if you connect a USB-PPI cable to the CRs CPUs.

Changing PPI communications to Freeport mode

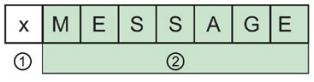
SMB30 and SMB130 configure the communications ports, 0 and 1 respectively, for Freeport operation and provide selection of baud rate, parity, and number of data bits. The following figure describes the Freeport control byte. One stop bit is generated for all configurations.



pp	Parity select		d	Data bits per character	
	00 = 01 = 10 = 11 =	No parity Even parity No parity Odd parity		0 = 1 =	8 bits per character 7 bits per character
bbb	Freeport baud rate		mm	Protocol selection	
	000 = 001 = 010 = 011 = 100 = 101 = 110 = 111 =	38400 19200 9600 4800 2400 1200 115200 57600		00 = 01 = 10 = 11 =	PPI slave mode Freeport mode Reserved (defaults to PPI slave mode) Reserved (defaults to PPI slave mode)

Transmit data

The Transmit instruction lets you send a buffer of one or more characters, up to a maximum of 255. The following figure shows the format of the Transmit buffer.



① Number of bytes to transmit

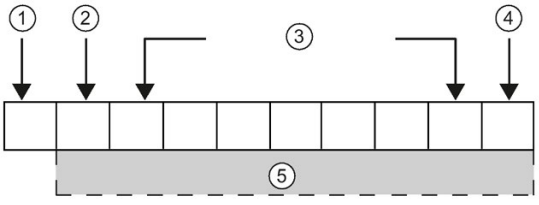
② Characters of the message

If an interrupt routine is attached to the transmit complete event, the CPU generates an interrupt (interrupt event 9 for port 0 and interrupt event 26 for port 1) after the last character of the buffer is sent.

You can transmit without using interrupts (for example, sending a message to a printer) by monitoring SM4.5 (port 0) or SM4.6 (port 1) to signal when transmission is complete.

You can use the Transmit instruction to generate a BREAK condition by setting the number of characters to zero and then executing the Transmit instruction. This generates a BREAK condition on the line for 16-bit times at the current baud rate. Transmitting a BREAK is handled in the same manner as transmitting any other message, in that a Transmit interrupt is generated when the BREAK is complete and SM4.5 or SM4.6 signals the current status of the Transmit operation.

Receive data



The Receive instruction lets you receive a buffer of one or more characters, up to a maximum of 255. The following figure shows the format of the Receive buffer.

① Number of bytes received (byte field)

② Start character

③ Message

④ End character

⑤ Characters of the message

If an interrupt routine is attached to the receive message complete event, the CPU generates an interrupt (interrupt event 23 for port 0 and interrupt event 24 for port 1) after the last character of the buffer is received.

You can receive messages without using interrupts by monitoring SMB86 (port 0) or SMB186 (port 1). This byte is non-zero when the Receive instruction is inactive or has been terminated. It is zero when a receive is in progress.

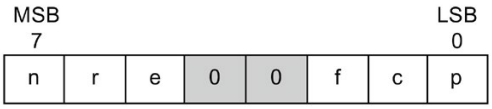
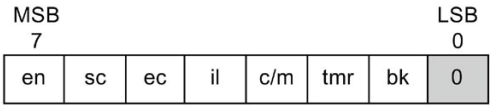
As shown in the following table, the Receive instruction allows you to select the message start and message end conditions, using SMB86 through SMB94 for port 0 and SMB186 through SMB194 for port 1.

Note

The receive message function is automatically terminated in case of a framing, parity, overrun, or break error.

You must define a start condition and an end condition (maximum character count) for the receive message function to operate.

Receive buffer format (SMB86 to SMB94, and SMB186 to SMB194)

Port 0	Port 1	Description
SMB86	SMB186	Receive message status byte  n: 1 = Receive message function terminated; user issued disable command. r: 1 = Receive message function terminated; error in input parameters or missing start or end condition. e: 1 = End character received. t: 1 = Receive message function terminated; timer expired. c: 1 = Receive message function terminated; maximum character count achieved. p: 1 = Receive message function terminated; a parity error.
SMB87	SMB187	Receive message control byte  en: 0 = Receive message function is disabled. 1 = Receive message function is enabled. The enable/disable receive message bit is checked each time the RCV instruction is executed. sc: 0 = Ignore SMB88 or SMB188. 1 = Use the value of SMB88 or SMB188 to detect start of message. ec: 0 = Ignore SMB89 or SMB189. 1 = Use the value of SMB89 or SMB189 to detect end of message. il: 0 = Ignore SMB90 or SMB190. 1 = Use the value of SMB90 or SMB190 to detect start of message. c/m: 0 = Timer is an inter-character timer. 1 = Timer is a message timer. tmr: 0 = Ignore SMW92 or SMW192. 1 = Terminate receive if the time period in SMW92 or SMW192 is exceeded. bk: 0 = Ignore break conditions. 1 = Use break condition as start of message detection.
SMB88	SMB188	Start of message character.
SMB89	SMB189	End of message character.
SMW90	SMW190	Idle line time period given in milliseconds. The first character received after idle line time has expired is the start of a new message.

Port 0	Port 1	Description
SMW92	SMW192	Inter-character/message timer time-out value given in milliseconds. If the time period is exceeded, the receive message function is terminated.
SMB94	SMB194	Maximum number of characters to be received (1 to 255 bytes). This range must be set to the expected maximum buffer size, even if the character count message termination is not used.

Start and End conditions for the Receive instruction

The Receive instruction uses the bits of the receive message control byte (SMB87 or SMB187) to define the message start and end conditions.

Note

If there is traffic present on the communications port from other devices when the Receive instruction is executed, the receive message function could begin receiving a character in the middle of that character, resulting in a possible parity or framing error and termination of the receive message function. If parity is not enabled the received message could contain incorrect characters. This situation can occur when the start condition is specified to be a specific start character or any character, as described in item 2 and item 6 below.

The Receive instruction supports several message start conditions. Specifying a start condition involving a break or an idle line detection avoids the problem of starting a message in the middle of a character by forcing the receive message function to synchronize the start of the message with the start of a character before placing characters into the message buffer.

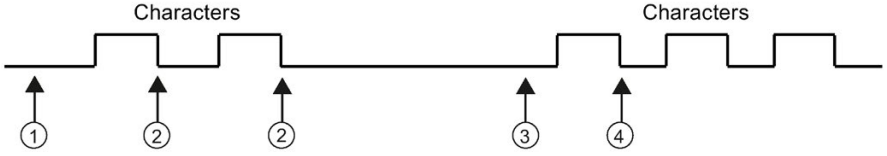
The Receive instruction supports several start conditions:

1. *Idle line detection:* The idle line condition is defined as a quiet or idle time on the transmission line. A receive is started when the communications line has been quiet or idle for the number of milliseconds specified in SMW90 or SMW190. When the Receive instruction in your program is executed, the receive message function initiates a search for an idle line condition. If any characters are received before the idle line time expires, the receive message function ignores those characters and restarts the idle line timer with the time from SMW90 or SMW190. See the following figure. After the idle line time expires, the receive message function stores all subsequent characters received in the message buffer.

   The idle line time should always be greater than the time to transmit one character (start bit, data bits, parity and stop bits) at the specified baud rate. A typical value for the idle line time is three character times at the specified baud rate.

   You use idle line detection as a start condition for binary protocols, protocols where there is not a particular start character, or when the protocol specifies a minimum time between messages.

   Setup: il = 1, sc = 0, bk = 0, SMW90/SMW190 = idle line timeout in milliseconds

   ① Receive instruction is executed: Starts the idle time
   ② Restarts the idle time
   ③ Idle time is detected: Starts the Receive Message function
   ④ First character is placed in the message buffer

2. *Start character detection:* The start character is any character which is used as the first character of a message. A message is started when the start character specified in SMB88 or SMB188 is received. The receive message function stores the start character in the receive buffer as the first character of the message. The receive message function ignores any characters that are received before the start character. The start character and all characters received after the start character are stored in the message buffer.

   Typically, you use start character detection for ASCII protocols in which all messages start with the same character.

   Setup: il = 0, sc = 1, bk = 0, SMW90/SMW190 = don't care, SMB88/SMB188 = start character

3. *Idle line and start character:* The Receive instruction can start a message with the combination of an idle line and a start character. When the Receive instruction is executed, the receive message function searches for an idle line condition. After finding the idle line condition, the receive message function looks for the specified start

character. If any character but the start character is received, the receive message function restarts the search for an idle line condition. All characters received before the idle line condition has been satisfied and before the start character has been received are ignored. The start character is placed in the message buffer along with all subsequent characters.

The idle line time should always be greater than the time to transmit one character (start bit, data bits, parity and stop bits) at the specified baud rate. A typical value for the idle line time is three character times at the specified baud rate.

Typically, you use this type of start condition when there is a protocol that specifies a minimum time between messages, and the first character of the message is an address or something which specifies a particular device. This is most useful when implementing a protocol where there are multiple devices on the communications link. In this case the Receive instruction triggers an interrupt only when a message is received for the specific address or devices specified by the start character.

Setup: il = 1, sc = 1, bk = 0, SMW90/SMW190 > 0, SMB88/SMB188 = start character

4. *Break detection:* A break is indicated when the received data is held to a zero value for a time greater than a full character transmission time. A full character transmission time is defined as the total time of the start, data, parity and stop bits. If the Receive instruction is configured to start a message on receiving a break condition, any characters received after the break condition are placed in the message buffer. Any characters received before the break condition are ignored.

   Typically, you use break detection as a start condition only when a protocol requires it.

   Setup: il = 0, sc = 0, bk = 1, SMW90/SMW190 = don't care, SMB88/SMB188 = don't care

5. *Break and a start character:* The Receive instruction can be configured to start receiving characters after receiving a break condition, and then a specific start character, in that sequence. After the break condition, the receive message function looks for the specified start character. If any character but the start character is received, the receive message function restarts the search for a break condition. All characters received before the break condition has been satisfied and before the start character has been received are ignored. The start character is placed in the message buffer along with all subsequent characters.

   Setup: il = 0, sc = 1, bk = 1, SMW90/SMW190 = don't care, SMB88/SMB188 = start character

6. *Any character:* The Receive instruction can be configured to immediately start receiving any and all characters and placing them in the message buffer. This is a special case of the idle line detection. In this case the idle line time (SMW90 or SMW190) is set to zero. This forces the Receive instruction to begin receiving characters immediately upon execution.

   Setup: il = 1, sc = 0, bk = 0, SMW90/SMW190 = 0, SMB88/SMB188 = don't care

   Starting a message on any character allows the message timer to be used to time out the receiving of a message. This is useful in cases where Freeport is used to implement the master or host portion of a protocol and there is a need to time out if no response is received from a slave device within a specified amount of time. The message timer starts when the Receive instruction executes because the idle line time was set to zero. The message timer times out and terminates the receive message function if no other end condition is satisfied.

   Setup: il = 1, sc = 0, bk = 0, SMW90/SMW190 = 0, SMB88/SMB188 = don't care, c/m = 1, tmr = 1, SMW92 = message timeout in milliseconds

The Receive instruction supports several ways to terminate a message. The message can be terminated on one or a combination of the following:

1. *End character detection:* The end character is any character which is used to denote the end of the message. After finding the start condition, the Receive instruction checks each character received to see if it matches the end character. When the end character is received, it is placed in the message buffer and the receive is terminated.

   Typically, you use end character detection with ASCII protocols where every message ends with a specific character. You can use end character detection in combination with the intercharacter timer, the message timer or the maximum character count to terminate a message.

   Setup: ec = 1, SMB89/SMB189 = end character

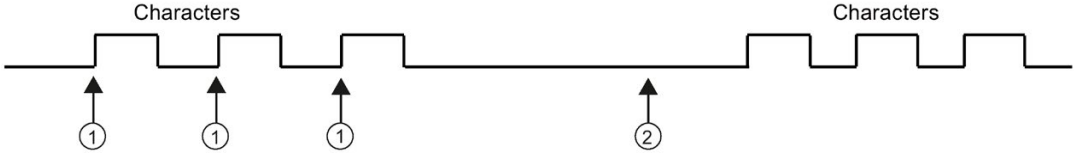
2. *Intercharacter timer:* The intercharacter time is the time measured from the end of one character (the stop bit) to the end of the next character (the stop bit). If the time between

characters (including the second character) exceeds the number of milliseconds specified in SMW92 or SMW192, the receive message function is terminated. The intercharacter timer is restarted on each character received. See the following figure.

You can use the intercharacter timer to terminate a message for protocols which do not have a specific end-of-message character. This timer must be set to a value greater than one character time at the selected baud rate since this timer always includes the time to receive one entire character (start bit, data bits, parity and stop bits).

You can use the intercharacter timer in combination with the end character detection and the maximum character count to terminate a message.

Setup: c/m = 0, tmr = 1, SMW92/SMW192 = timeout in milliseconds



① Restarts the intercharacter timer

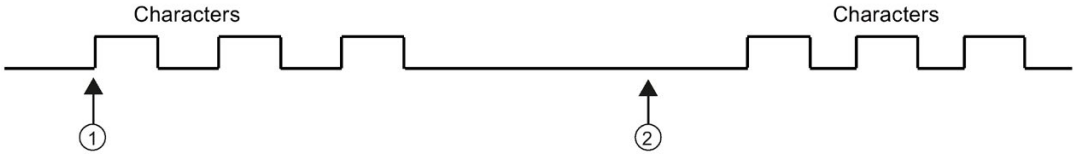
② The intercharacter timer expires: Terminates the message and generates the Receive message interrupt

3. *Message timer:* The message timer terminates a message at a specified time after the start of the message. The message timer starts as soon as the start condition(s) for the receive message function have been met. The message timer expires when the number of milliseconds specified in SMW92 or SMW192 has passed. See the following figure.

   Typically, you use a message timer when the communications devices cannot guarantee that there will not be time gaps between characters or when operating over modems. For modems, you can use a message timer to specify a maximum time allowed to receive the message after the message has started. A typical value for a message timer would be about 1.5 times the time required to receive the longest possible message at the selected baud rate.

   You can use the message timer in combination with the end character detection and the maximum character count to terminate a message.

   Setup: c/m = 1, tmr = 1, SMW92/SMW192 = timeout in milliseconds



① Start of the message: Starts the message timer

② The message timer expires: Terminates the message and generates the Receive message interrupt

4. *Maximum character count:* The Receive instruction must be told the maximum number of characters to receive (SMB94 or SMB194). When this value is met or exceeded, the receive message function is terminated. The Receive instruction requires that the user specify a maximum character count even if this is not specifically used as a terminating condition. This is because the Receive instruction needs to know the maximum size of the receive message so that user data placed after the message buffer is not overwritten.

   The maximum character count can be used to terminate messages for protocols where the message length is known and always the same. The maximum character count is always used in combination with the end character detection, intercharacter timer, or message timer.

5. *Parity errors:* The Receive instruction automatically terminates when the hardware signals a parity, framing, or overrun error; or if a break condition is detected after the start of a message. Parity errors occur only if parity is enabled in SMB30 or SMB130. Framing errors occur if the stop bit is not correct. Overrun errors occur if characters come in to quickly for the hardware to handle. A break condition terminates a message because it resembles a parity or framing error to the hardware. There is no way to disable this function.

6. *User termination:* The user program can terminate a receive message function by executing another Receive instruction with the enable bit (EN) in SMB87 or SMB187 set to zero. This immediately terminates the receive message function.

Using character interrupt control to receive data

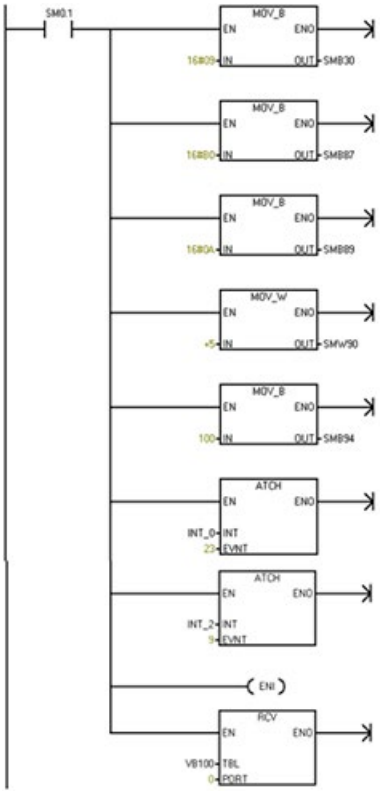
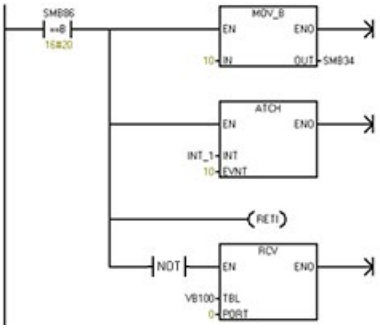
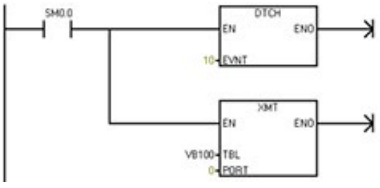
To allow complete flexibility in protocol support, you can also receive data using character interrupt control. Each character received generates an interrupt. The received character is placed in SMB2, and the parity status (if enabled) is placed in SM3.0 just prior to execution of the interrupt routine attached to the receive character event. SMB2 is the Freeport receive character buffer. Each character received while in Freeport mode is placed in this location for easy access from the user program. SMB3 is used for Freeport mode and contains a parity error bit that is turned on when a parity, framing, overrun, or break error is detected on a received character. All other bits of the byte are reserved. Use the parity bit either to discard the message or to generate a negative acknowledgement to the message.

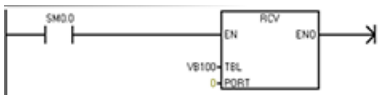
When the character interrupt is used at high baud rates (38.4 Kbps to 115.2 Kbps), the time between interrupts is very short. For example, the character interrupt for 38.4 Kbps is 260 microseconds, for 57.6 Kbps is 173 microseconds, and for 115.2 Kbps is 86 microseconds. Ensure that you keep the interrupt routines very short to avoid missing characters, or else use the Receive instruction.

Note

SMB2 and SMB3 are shared between Port 0 and Port 1. When the reception of a character on Port 0 results in the execution of the interrupt routine attached to that event (interrupt event 8), SMB2 contains the character received on Port 0, and SMB3 contains the parity status of that character. When the reception of a character on Port 1 results in the execution of the interrupt routine attached to that event (interrupt event 25), SMB2 contains the character received on Port 1 and SMB3 contains the parity status of that character.

Example: Transmit and Receive instructions

MAIN	Network 1	Network 1 //This program receives a string of characters until a line feed character is received. The message is then transmitted back to the sender.	
		LD SM0.1 MOVB 16#09, SMB30	On the first scan: 1. Initialize Freeport: - Select 9600 baud. - Select 8 data bits. - Select no parity.
		MOVB 16#B0, SMB87	2. Initialize RCV message control byte: - RCV enabled. - Detect end of message character. - Detect idle line condition as the message start condition.
		MOVB 16#0A, SMB89	3. Set end of message character to hex 0A (line feed).
		MOVW +5, SMW90	4. Set idle line timeout to 5 ms.
		MOVB 100, SMB94	5. Set maximum number of characters to 100.
		ATCH INT_0, 23	6. Attach interrupt 0 to the Receive Complete event.
		ATCH INT_2, 9	7. Attach interrupt 2 to the Transmit Complete event.
		ENI	8. Enable user interrupts.
		RCV VB100, 0	9. Enable receive box with buffer at VB100.
INT 0	Network 1	Network 1	
		LDB= SMB86, 16#20 MOVB 10, SMB34 ATCH INT_1, 10 CRETI NOT RCV VB100, 0	Receive complete interrupt routine: 1. If receive status shows receive of end character, then attach a 10 ms timer to trigger a transmit and return. 2. If the receive completed for any other reason, then start a new receive.
INT 1	Network 1	Network 1	
		LD SM0.0 DTCH 10 XMT VB100, 0	10-ms Timer interrupt: 1. Detach timer interrupt. 2. Transmit message back to user on port.
INT 2	Network 1	Network 1	

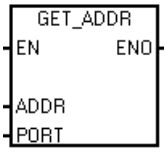
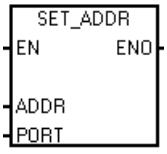
		LD SM0.0 RCV VB100, 0	Transmit Complete interrupt: Enable another receive.

7.3.3 Get port address and set port address (PPI protocol on RS485/RS232)

You can use the GET_ADDR and SET_ADDR instructions to read and set the PPI network address of the selected port.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of signal boards.

LAD / FBD	STL	Description
	`GPA ADDR, PORT`	The GET_ADDR instruction reads the station address of the CPU port specified in PORT and places the value in the address specified in ADDR.
	`SPA ADDR, PORT`	The SET_ADDR instruction sets the port station address (PORT) to the value specified in ADDR. The new address is not saved permanently. After a power cycle, the affected port returns to the network address downloaded in the system block.

Non-fatal error conditions with ENO = 0	SM bits affected
• 006H Indirect address • 0004H Attempted to perform a SET_ADDR instruction in an interrupt routine • 0090H Port number is invalid • 0091H Port address is invalid	None

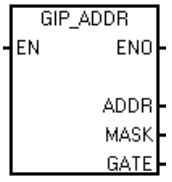
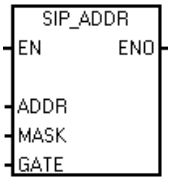
Input / output	Data type	Operand
ADDR	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant (A constant value is valid only for the Set Port Address instruction.)
PORT	BYTE	Constant: 0 or 1 Note: The two available ports are as follows: • Integrated RS485 port (Port 0), • CM01 Signal Board (SB) RS232/RS485 port (Port 1)

7.3.4 Get IP address and set IP address (Ethernet)

You can use the GIP_ADDR and SIP_ADDR instructions to read and set the Ethernet IP address, the subnet mask, and gateway address for the Ethernet port.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

LAD / FBD	STL	Description
	**GIP ADDR, MASK, GATE**	The GIP_ADDR instruction copies the CPU's IP address into ADDR, the CPU's subnet mask into MASK, and the CPU's gateway into GATE.
	**SIP ADDR, MASK, GATE**	The SIP_ADDR instruction sets the CPU's IP address to the value found in ADDR, the CPU's subnet mask to the value found in MASK, and the CPU's gateway to the value found in GATE.

Non-fatal errors with ENO = 0	SM bits affected
• 006H Indirect address • 0004H Attempted to execute a SIP_ADDR instruction in an interrupt routine • IP address cannot be changed (see following note) • IP address is invalid for the current subnet	None

Input / output	Data type	Operand
ADDR	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
MASK	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
GATE	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Note

To use the SIP_ADDR instruction, do not select the **"IP address data is fixed to the values below and cannot be changed by other means"** option for the Ethernet Port in the Communication section of the System Block.

Execution of the SIP_ADDR instruction causes the CPU to store the IP address, subnet mask, and gateway values in persistent memory.

Example

Note that STEP 7-Micro/WIN SMART displays the outputs for the GIP_ADDR instruction, ADDR, MASK, and GATE, as string values. For the SIP_ADDR instruction, however, you provide the ADDR, MASK, and GATE inputs as hexadecimal values. For the SIP_ADDR input values, think of each octet of the IP address, MASK and GATE as a hexadecimal number.

For the SIP_ADDR instruction, consider the octets of the IP address "192.168.2.150":

Octet decimal value	Hexadecimal value
192	C0
168	A8
2	02
150	96

You would use the combined hexadecimal values of the octets to form the ADDR input to the SIP_ADDR instruction: 16#C0A80296. (You could convert this number to a decimal value, but the hexadecimal value is representative of the values of the octets.)

Similarly, consider the octets of the subnet mask "255.255.255.0":

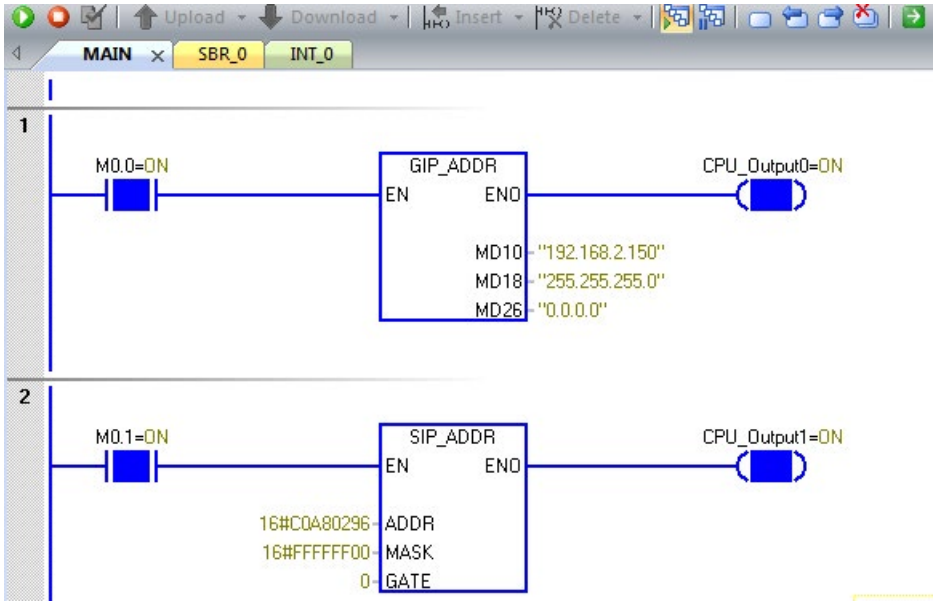
Octet decimal value	Hexadecimal value
255	FF
255	FF
255	FF
0	00

You would use the combined hexadecimal values of the octets to form the MASK input to the SIP_ADDR instruction: 16#FFFFFF00. You could also use the decimal equivalent, but not a string representation.

The following program status display shows two networks:

- Network 1: The GIP_ADDR reads the IP address of 192.168.2.150 with a subnet mask of 255.255.255.0.
- Network 2: The SIP_ADDR sets the IP address to 192.168.2.150 (16#C0A80296) and sets the subnet mask to 255.255.255.0 (16#FFFFFF00).

Note that the default gateway is 0.



7.3.5 Open user communication

The Open User Communication (OUC) instructions give your program a way to communicate over an Ethernet network to another Ethernet capable device. The other Ethernet device can be another S7-200 SMART CPU or another third party device that supports either UDP, TCP, or ISO-on-TCP protocol. Your program controls all aspects of the communication from selecting the protocol, initiating the connection, sending data, receiving data, and terminating the connection.

Note

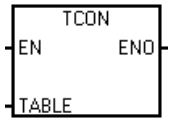
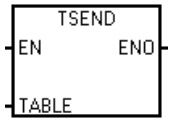
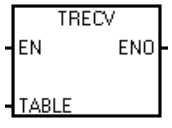
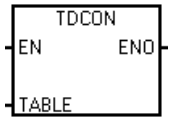
The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

7.3.5.1 OUC instructions

There are four Open User Communications (OUC) instructions to control the communication process:

- TCON opens the UDP, TCP, or ISO-on-TCP (RFC 1006) connection between the S7-200 SMART CPU and the remote device.
- TSEND and TRCV send and receive data.
- TDCON closes the connection.

Table 7- 7 OUC instructions

LAD/FBD	STL	Description
	**TCON table**	TCON initiates a UDP, TCP, or ISO-on-TCP communications connection from the CPU to a communication partner.
	**TSEND table**	TSEND sends data to another device.
	**TRECV table**	TRECV retrieves data over an existing communication connection.
	**TDCON table**	TDCON terminates a UDP, TCP, or ISO-on-TCP communications connection.

The OUC instructions maintain information about the connections so that your program does not need to permanently allocate V memory for the OUC Tables. The data in the tables must be kept constant while the OUC instruction is active.

The OUC instructions require additional communication background time when they are processing/active/busy and also when they are just maintaining the connection to the other device. The amount of communication background time required depends on the number of OUC instructions that are active/busy, how often the OUC instructions are executed, and the number of connections that are currently open. You should adjust the communication background time to a higher value if the communication performance is slow. Refer to "Configuring communication" (Page 131) for further information.

All of the OUC instructions use a table to store the parameters for the instructions. The content of the tables for each instruction is described below.

The S7-200 SMART CPU uses the input table parameters to determine the instance of the OUC instructions. The table parameters must be kept the same during an operation so that the S7-200 SMART CPU knows that the particular instruction (instance) is the same as during the previous scans.

Note

Siemens also offers the Open User Communication (OUC) library instructions for your convenience. The OUC Library instructions build the tables for you based upon the inputs to the library instructions. The Library instructions also retrieve the response information from the tables and provide this information on the outputs of the library instructions. Refer to "Open user communication library" (Page 529) for further information.

Table 7- 8 Valid operands for the OUC instructions

Inputs/Outputs	Data Type	Operands
TABLE	BYTE	IB, QB, VB, MB, SMB, SB, *VD, *LD, *AC

Error conditions that set ENO = 0:

- 0006 (indirect address)
- If the function returns an error and sets the E bit of table status byte (see the figure below)

TCON instruction

You use the TCON instruction to set up and establish a communication connection. Once the CPU establishes the connection, it is automatically maintained and monitored by the CPU. The TCON instruction has only one parameter which is the address of the TCON table. The TCON table contains the connection parameters. There are two formats for the TCON table based upon the protocol selected for the connection. UDP and TCP share a common table format. ISO-on-TCP has a special TCON table format. Refer to the TCON instruction tables below for further information.

Set the REQ bit in the table to TRUE to initiate a connection. The CPU ignores the REQ bit while the TCON instruction is active and the connection is initializing and the Active bit is TRUE. The TCON instruction set the Done bit when the CPU has established the

connection. The Error bit is set if there is a problem with the connection parameters or if the CPU cannot establish a connection to the remote device. The Error Code describes the reason for the connection failure if the Error bit is set.

The TCON instruction is asynchronous and can take several scans to complete. The Active bit will be set while the connection operation is pending.

The TCON instruction creates either an active (client) connection or a passive (server) connection. The CPU initiates contact with the remote devices for an active connection. Passive connections cause the CPU to wait on the remote device to contact the CPU.

You can use the TCON instruction to determine the current status of a connection. If your program calls the TCON instruction with the REQ bit set to FALSE, the CPU reports the status of the connection:

- The instruction sets the Done bit (without Error) if the CPU establishes the connection and the connection is operational.
- The instruction sets the Active bit if the connection is still in the process of being connected.
- The instruction sets the Done and Error bits if no connection can be established. The Error Code gives the reason for the connection failure.

The REQ bit in the table is level-triggered. It is recommended that you put a positive edge trigger on the REQ input to initiate a connection so that the CPU only requests the connection establishment one time.

During the connection process (the call to the TCON instruction), your program assigns a Connection ID to the given connection. The Connection ID is a user-selected, 16-bit value passed into the TCON instruction. The Connection ID can be any number 0 to 65534 inclusive. The CPU does not allow the Connection ID to be 65535 (0xFFFF). The Connection ID value is an input to all of the OUC instructions to identify the connection to be utilized for the given operation.

You can select your own Connection ID, allowing you to use a number that is logical for your situation. For example, you can use part of your IP address as your connection ID. You can name your connection to IP address 192.168.2.10, connection ID 10.

Note that the S7-200 SMART does not automatically attempt to reconnect to a device after the connection has been closed. After a connection has been disconnected, your program must execute another TCON instruction to reconnect the device. This is true for both active and passive connections.

TCON instruction tables

The following tables contain the format and definitions for the TCON instruction. Refer to "OUC instruction error codes" (Page 219) for the error code listing. Refer to "Ports and TSAPs" (Page 402) for port number restrictions and further information:

- Status: The first byte of the table returns the status of the operation to the user. The OUC instructions ignore the value of the status byte as an input. The status byte is valid on the return of the instruction. These are the status bit definitions:
  - D = Done (Complete)
  - A = Active (In progress, in other words, Busy)
  - E = Error (Complete with error)
  - Error Code

  If there is an error, the Done and Error bits are both set. The error codes are listed in "OUC instruction error codes" (Page 219).
- REQ: You use the REQ bit to initiate a new operation. The REQ bit is a level-triggered value. Your program code must provide the one shot operation if required (a positive edge contact). If the operation is not busy, a REQ value of TRUE initiates a new operation. For example, if there is not currently a TSEND instruction in progress, a TRUE value in the REQ bit causes the program to initiate a new TSEND instruction operation.
- Connection ID: The Connection ID is a 16-bit value that you select to pass into the function. The range is 0 to 65534 (65535 is reserved). The Connection ID parameter is an input to the OUC instructions. The TSEND, TRECV, and TDCON instructions use the Connection ID that you select for the TCON instruction as a reference.

Table 7- 9 Definition of TCON instruction TABLE parameter structure for UDP and TCP

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	D	A	E	Error code (5 bits)				
1	A/P [1]							REQ
2	Connection ID (2 bytes)							
3								
4	Connection Type [2]							
5	Remote IP Address [3]							
6								
7								
8								

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
9	Remote port [4]							
10								
11	Local port [5]							
12								

[1] A/P: Active/Passive selection (1 = Active, 0 = Passive)

[2] Connection Type: The Connection Type informs the TCON instruction of the desired type of connection: UDP = 19 and TCP = 11

[3] Remote IP address: This is the IP address of the remote device in the case of an active connection. You should set the Remote IP Address to 0.0.0.0 for UDP connections. The IP address must be different than that of the local CPU and cannot be a multicast or broadcast address. Since the S7-200 SMART supports routing, the IP address may be on a different subnet than the local CPU. If you set the IP address for a passive (server) connection, then the CPU only accepts a connection from the specified IP address. If you set the IP address to 0.0.0.0 for a passive connection, the CPU accepts a connection from any IP address.

[4] Remote Port: This is the port number in the remote device. You do not use the remote port number for UDP or passive connections, and you should set the Remote Port to zero.

[5] Local Port: This is the port number for the connection in the local CPU.

Table 7- 10 Definition of TCON instruction TABLE parameter structure for ISO-on-TCP

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	D	A	E	Error code (5 bits)				
1	A/P [1]							REQ
2	Connection ID (2 bytes)							
3								
4	Connection Type [2]							
5	Remote IP Address [3]							
6								
7								
8								

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
9	Remote TSAP [4]							
to	String of 2 to 16 characters (3 to 17 bytes)							
25								
26	Local TSAP [5]							
to	String of 2 to 16 characters (3 to 17 bytes)							
42								

1 A/P: Active/Passive selection (1 = Active, 0 = Passive)

2 Connection Type: The Connection Type informs the TCON instruction of the desired type of connection: ISO-on-TCP = 12

3 Remote IP address: This is the IP address of the remote device in the case of an active connection. The IP address must be different than that of the local CPU and cannot be a multicast or broadcast address. Since the S7-200 SMART supports routing, the IP address may be on a different subnet than the local CPU.
If you set the IP address for a passive (server) connection, then the CPU only accepts a connection from the specified IP address. If you set the IP address to 0.0.0.0 for a passive connection, the CPU accepts a connection from any IP address.

4 Remote TSAP: This is the Transport Service Access Point (TSAP) of the remote device. You use the remote TSAP for ISO-on-TCP connections only. The remote TSAP is a string of 2 to 16 ASCII characters.

5 Local TSAP: This is the Transport Service Access Point (TSAP) for the connection in local CPU. You only use the local TSAP for ISO-on-TCP connections. The local TSAP is a string of 2 to 16 ASCII characters. If you use two characters, the TSAP must start with a hex "E0" character ($E0), followed by another hex character (for example, "$E0$01"). You cannot use the string "SIMATIC-".

TSEND

You use the TSEND instruction to send data over an existing communication connection. The TSEND table contains the connection parameters. There are two formats for the TSEND table based upon the protocol selected for the connection. TCP and ISO-on-TCP share a common table format. UDP has a special TSEND table format. Refer to the TSEND and TRECV instruction tables below for further information.

The TSEND instruction initiates sending the specified number of bytes when your program calls the TSEND instruction with the REQ bit set and the connection is not currently busy with some other operation.

The REQ bit is level-triggered. It is recommended that you put a positive edge trigger on the REQ input so that the CPU does not initiate unintended send operations. The CPU ignores the REQ bit while the TSEND is Active. The status bits and error code show the status of the TSEND for each call:

- Done without Error means the TSEND instruction completed with no errors.
- Active means that the TSEND instruction is still busy.
- Done with Error means there is a problem with the TSEND instruction. The Error Code contains the reason for the failure.

The Done/Active/Error status is shown for one call of the TSEND instruction after the send operation is complete. After that, the TSEND instruction responds with error code 24, which means no operation pending, if your program calls the instruction with REQ set to FALSE. If the REQ bit remains set, the TSEND instruction initiates another send operation.

The maximum amount of data that you can send in one message is 1024 bytes. Only one TSEND instruction can be active at a time on a given connection. The program copies the data from your send buffer in user memory to an internal buffer when the TSEND instruction executes with REQ set, so you can change your send buffer after the TSEND instruction executes.

TRECV

You use the TRECV instruction to retrieve data that the CPU has received over an existing communication connection. You assign the receive area/buffer and the maximum length of the receive area so that there is no possibility of a buffer overrun. The TRECV table contains the parameters needed for the TRECV instruction. There are two formats for the TRECV table based upon the protocol selected for the connection. TCP and ISO-on-TCP share a common table format. UDP has a special TRECV table format. Refer to the TSEND and TRECV tables below for further information.

The TRECV instruction does not have a REQ bit. After the first execution of the TRECV instruction, the status bits show the instruction as Active. All subsequent calls to the TRECV instruction show an Active status if no data has been received by the CPU for this connection.

After a successful receipt of data, the instruction sets the Done bit in the status byte of the table, and the returned Data Length value is the actual number of bytes received. The TRECV instruction copies the received data from an internal buffer to the your receive buffer only when the TRECV instruction executes and the Done bit is set to TRUE.

The maximum amount of data that you can receive in one message is 1024 bytes. Because TCP acts as a "streaming" protocol, the program can collect multiple messages in one receive message if the TRECV instruction is not called frequently. The UDP and ISO-on-TCP protocols guarantee that each message is delineated as a separate message.

For example, let us suppose that there is a TCP client that sends four 20-byte messages to the S7-200 SMART in rapid succession, and your program is not calling the TRECV instruction. If your program calls the TRECV instruction after all four messages have been accepted by the CPU, the program sees this as one receive message of 80 bytes. Your program is responsible for calling the TRECV instruction as often as needed to receive each message as it is sent.

Assuming the same client and the same messages as in the above example, ISO-on-TCP and UDP delivers the four messages during four subsequent calls to the TRECV instruction. These protocols delineate the messages and keep them separate in the CPU until your program calls the TRECV instruction to retrieve them.

If the CPU receives more bytes than will fit into the user buffer, the TRECV instruction copies in the maximum number of bytes allowed (Data Length in the table) and discards the rest of the received bytes. In this situation, the TRECV instruction completes with an error to tell the user that bytes were discarded.

TSEND and TRECV instruction tables

The following tables contain the format and definitions for the TSEND and TRECV instructions. Refer to "OUC instruction error codes" (Page 219) for the error code listing. Refer to "Ports and TSAPs" (Page 402) for port number restrictions and further information:

- Status: The first byte of the table returns the status of the operation to the user. The OUC instructions ignore the value of the status byte as an input. The status byte is valid on the return of the instruction. These are the status bit definitions:
  - D = Done (Complete)
  - A = Active (In progress, in other words, Busy)
  - E = Error (Complete with error)
  - Error Code

  If there is an error, the Done and Error bits are both set. The error codes are listed in "OUC instruction error codes" (Page 219).
- REQ: You use the REQ bit to initiate a new operation. The REQ bit is a level-triggered value. Your program code must provide the one shot operation if required (a positive edge contact). If the operation is not busy, a REQ value of TRUE initiates a new operation. For example, if there is not currently a TSEND instruction in progress, a TRUE value in the REQ bit causes the program to initiate a new TSEND instruction operation.
- Connection ID: The Connection ID is a 16-bit value that you select to pass into the function. The range is 0 to 65534 (65535 is reserved). The Connection ID parameter is an input to the OUC instructions. The TSEND, TRECV, and TDCON instructions use the Connection ID that you select for the TCON instruction as a reference.

Table 7- 11 Definition of TSEND and TRECV instruction TABLE parameter structure for TCP and ISO-on-TCP

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	D	A	E	Error code (5 bits)				
1								REQ [1]
2	Connection ID (2 bytes)							
3								

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
4	Data Length [2]							
5								
6	Data Pointer [3]							
7								
8								
9								

[1] REQ: You set the REQ bit to TRUE to initiate a new TSEND instruction operation. The TRECV instruction ignores the REQ status bit. The REQ bit is only used for the TSEND instruction.
For the TRECV instruction, the Done bit means that the CPU received data (New Data Ready) and the Data_Length value returns the actual number of bytes received. If there is no data available when called, the TRECV instruction returns with the Active flag set and a Data_Length value of zero. If the number of received bytes exceeds the size of the receive buffer (data length input), the program copies the maximum number of bytes into the buffer and returns an error to the TRECV instruction.

[2] Data Length: The Data Length in the TRECV instruction table is both an input and output parameter. The input value is the maximum size of the receive buffer. The output value is the number of bytes actually received.
The Data Length is an input value only for the TSEND instruction.

[3] Data Pointer: An S7-200 SMART pointer to the data in the local CPU.

Table 7- 12 Definition of TSEND and TRECV instruction TABLE parameter structure for UDP

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	D	A	E	Error code (5 bits)				
1								REQ [1]
2	Connection ID (2 bytes)							
3								
4	Data Length [2]							
5								
6	Data Pointer [3]							
7								
8								
9								

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
10	Remote IP Address [4]							
11								
12								
13								
14	Remote port [5]							
15								

[1] REQ: You set the REQ bit to TRUE to initiate a new TSEND instruction operation. The TRECV instruction ignores the REQ status bit. The REQ bit is only used for the TSEND instruction.
For the TRECV instruction, the Done bit means that the CPU received data (New Data Ready) and the Data_Length value returns the actual number of bytes received. If there is no data available when called, the TRECV instruction returns with the Active flag set and a Data_Length value of zero. If the number of received bytes exceeds the size of the receive buffer (data length input), the program copies the maximum number of bytes into the buffer and returns an error to the TRECV instruction.

[2] Data Length: The Data Length in the TRECV instruction structures is both an input and output parameter. The input value is the maximum size of the receive buffer. The output value is the number of bytes actually received.
The Data Length is an input value only for the TSEND instruction.

[3] Data Pointer to the data area: An S7-200 SMART pointer to the data in the local CPU.

[4] Remote IP address: This is the IP address of the remote device for a TSEND instruction. The IP address must be different than that of the local CPU and cannot be a multicast or broadcast address. Since the S7-200 SMART supports routing, the IP address may be on a different subnet than the local CPU. (The IP address must be supplied for each UDP send operation.)
The IP address is a returned value for a UDP receive operation. The IP address is the address of the sender of UDP message.

[5] Remote Port: This is the port number in the remote device.
The remote port is a returned value for a UDP receive operation. The port is the port number of the sender of the UDP message.
UDP requires the remote port number for each TSEND instruction message.

TDCON

You use the TDCON instruction to terminate an existing communication connection. The instruction terminates the connection when the REQ bit is set. It is recommended that you put a positive edge trigger on the REQ input. If the your program calls the TDCON instruction and the connection is already disconnected, then the instruction responds with error code 24, which means no operation pending.

TDCON instruction table

The following table contains the format and definitions for the TDCON instruction. Refer to "OUC instruction error codes" (Page 219) for the error code listing. Refer to "Ports and TSAPs" (Page 402) for port number restrictions and further information:

- Status: The first byte of the table returns the status of the operation to the user. The OUC instructions ignore the value of the status byte as an input. The status byte is valid on the return of the instruction. These are the status bit definitions:
  - D = Done (Complete)
  - A = Active (In progress, in other words, Busy)
  - E = Error (Complete with error)
  - Error Code

  If there is an error, the Done and Error bits are both set. The error codes are listed in "OUC instruction error codes" (Page 219).
- REQ: You use the REQ bit to initiate a new operation. The REQ bit is a level-triggered value. Your program code must provide the one shot operation if required (a positive edge contact). If the operation is not busy, a REQ value of TRUE initiates a new operation. For example, if there is not currently a TSEND instruction in progress, a TRUE value in the REQ bit causes the program to initiate a new TSEND instruction operation.
- Connection ID: The Connection ID is a 16-bit value that you select to pass into the function. The range is 0 to 65534 (65535 is reserved). The Connection ID parameter is an input to the OUC instructions. The TSEND, TRECV, and TDCON instructions use the Connection ID that you select for the TCON instruction as a reference.

Table 7- 13 Definition of TDCON instruction TABLE parameter structure

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	D	A	E	Error code (5 bits)				
1								REQ
2	Connection ID (2 bytes)							
3								

7.3.5.2 OUC instruction error codes

The following table lists the Open User Communication (OUC) error codes:

Error code	Description	TCON	TSEND	TRECV	TDCON
0	No error	X	X	X	X
1	The data length parameter is greater than the maximum allowed (1024 bytes).		X	X	
2	The data buffer is not in I, Q, M, or V memory areas.		X	X	
3	The data buffer does not fit in the memory area.		X	X	
4	The table parameter does not fit into the memory area.	X	X	X	X
5	The connection is locked in another context. You are attempting to access the same connection in both the background (the Main) and in an interrupt routine at the same time.	X	X	X	X
6	A UDP IP address or port error		X		
7	An instance mismatch: The connection is busy with another instance or the input data does not match the data stored for the requested connection ID when the request was initiated.	X	X	X	X
8	The Connection ID does not exist because the connection has never been created, or the connection was terminated at your request (using the TDCON instruction).	X	X	X	X
9	A TCON operation is in progress with this Connection ID.		X	X	X
10	A TDCON operation is in progress with this Connection ID.	X	X	X	
11	A TSEND instruction is in progress with this Connection ID.		X		X
12	A temporary communication error has occurred. The connection cannot be started at this time. Try again later.	X	X	X	
13	The connection partner refused or actively dropped the connection (the partner issued a disconnect to this CPU).	X	X	X	
14	The connection partner cannot be reached (no answer to the connect request).	X	X	X	
15	The connection aborted due to inconsistencies. Disconnect and reconnect to correct the situation.	X	X	X	X
16	The Connection ID is already in use with a different IP address, port, or TSAP combination.	X			
17	No connection resource is available. All connections of the requested type (active/passive) are in use.	X			
18	The local or remote port number is reserved or the port number is already in use for another server (passive) connection.	X			

Error code	Description	TCON	TSEND	TRECV	TDCON
19	One of the following IP address errors have occurred: • The IP address is invalid (for example, address 0.0.0.0). • This IP address is the IP address of this CPU. • This CPU has IP address 0.0.0.0. • The IP address is a broadcast or multicast address.	X			
20	A local or remote TSAP error (ISO-on-TCP only)	X			
21	An invalid connection ID (65535 is reserved)	X			
22	An active/passive error (UDP only allows passive)	X			
23	The connection type is not one of the allowed types.	X			
24	There is no operation pending so there is no status to report.		X		X
25	The receive buffer is too small: The CPU received more bytes than the buffer length supports. The CPU discards the extra bytes.			X	
31	Unknown error	X	X	X	X

7.4 Compare

7.4.1 Compare number values

The compare instructions can compare two number values with the same data type. You can compare bytes, integers, double integers, and real numbers.

For **LAD and FBD**: When the comparison is TRUE, the compare instruction sets ON a contact (LAD network power flow), or output (FBD logic flow).

For **STL**: When the comparison is TRUE, the compare instructions can load, AND, or OR a 1 with the value on the top of the logic stack.

Types of comparison

Six comparison types are available:

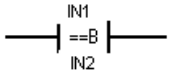
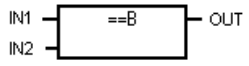
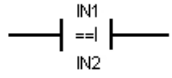
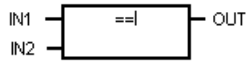
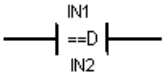
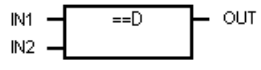
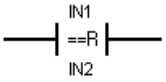
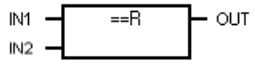
Comparison type	The output is TRUE only if
== (LAD/FBD) = (STL)	IN1 is equal to IN2
<>	IN1 is not equal to IN2
>=	IN1 is greater than or equal to IN2

Comparison type	The output is TRUE only if
<=	IN1 is less than or equal to IN2
>	IN1 is greater than IN2
<	IN1 is less than IN2

Selecting the data types to be compared

The data type identifier that you choose determines the required data type for the IN1 and IN2 parameters.

Data type identifier	Required IN1, IN2 data type
B	Unsigned byte
W	Signed word integer
D	Signed double word integer
R	Signed real

LAD contacts, FBD boxes	STL	Comparison result
 	**LDB= IN1, IN2** **OB= IN1, IN2** **AB= IN1, IN2**	Compare two unsigned byte values: The result is TRUE, if IN1 = IN2
 	**LDW= IN1, IN2** **OW= IN1, IN2** **AW= IN1, IN2**	Compare two signed integer values: The result is TRUE, if IN1 = IN2
 	**LDD= IN1, IN2** **OD= IN1, IN2** **AD= IN1, IN2**	Compare two signed double integer values: The result is TRUE, if IN1 = IN2
 	**LDR= IN1, IN2** **OR= IN1, IN2** **AR= IN1, IN2**	Compare two signed real values: The result is TRUE, if IN1 = IN2

Note

The following conditions cause a non-fatal error, set power flow to OFF (ENO bit = 0), and use value 0 as the result of the comparison

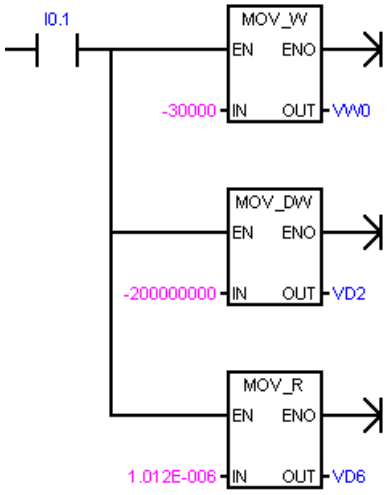
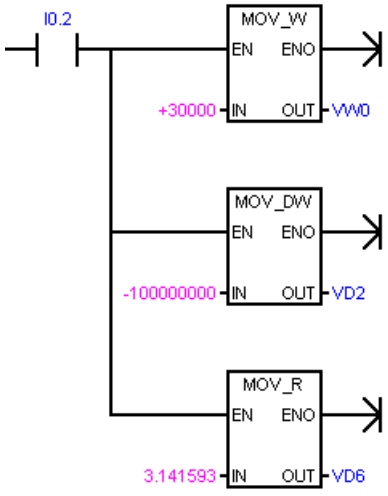
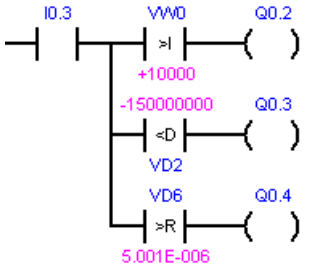
- Illegal indirect address is encountered (any compare instruction)
- Illegal real number (for example, NaN) is encountered for compare real instruction

To prevent these conditions from occurring, ensure that you properly initialize pointers and values that contain real numbers before executing compare instructions that use these values.

Compare instructions are executed regardless of the state of power flow.

Input / output	Data type	Operand
IN1, IN2	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	BOOL	LAD: Power flow FBD: I, Q, V, M, SM, S, T, C, L, Logic Flow

Example compare values

LAD		STL
	Activate I0.1 to load V memory addresses with low values that make the comparisons FALSE and that set the status indicators OFF.	**Network 1** **LD I0.1** **MOVW -30000, VW0** **MOVD -200000000, VD2** **MOVR 1.012E-006, VD6**
	Activate I0.2 to load V memory addresses with high values that make the comparisons TRUE and that set the status indicators ON.	**Network 2** **LD I0.2** **MOVW +30000, VW0** **MOVD -100000000, VD2** **MOVR 3.141593, VD6**
	Activate I0.3 to perform comparisons. The Integer Word comparison tests to find if VW0 > +10000 is TRUE. You can also compare two values stored in variable memory like VW0 > VW100.	**Network 3** **LD I0.3** **LPS** **AW> VW0, +10000** **= Q0.2** **LRD** **AD< -150000000, VD2** **= Q0.3** **LPP** **AR> VD6, 5.001E-006** **= Q0.4**

See also

Constants (Page 75)

7.4.2 Compare character strings

The compare string instructions can compare two ASCII character strings.

For **LAD** and **FBD:** When the comparison is TRUE, the compare instruction turns ON the contact (LAD), or output (FBD).

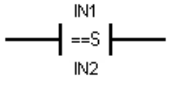
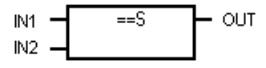
For **STL**: When the comparison is TRUE, the compare instruction loads, ANDs, or ORs a 1 with the value on the top of the logic stack.

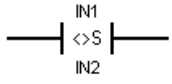
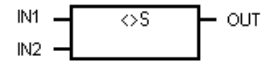
Comparisons can be made between two variables, or between a constant and a variable. If a constant is used in a comparison, then it must be the top parameter (LAD contact / FBD box) or the first parameter (STL).

In the program editor, a constant string parameter assignment must begin and end with a double quote character. The maximum length of a constant string entry is 126 characters (bytes).

In contrast, a variable string is referenced by the byte address of the initial length byte with the character bytes stored the next byte addresses. A variable string has a maximum length of 254 characters (bytes) and can be initialized in the data block editor (with beginning and ending double quote character).

LAD contact FBD box	STL	Description


 
	`LDS=  IN1, IN2`
	`OS=  IN1, IN2`
	`AS=  IN1, IN2`
		Compare two character strings of STRING data type:
		The result is TRUE, if string IN1 **equals** string IN2.


 
	`LDS<>  IN1, IN2`
	`OS<>  IN1, IN2`
	`AS<>  IN1, IN2`
		Compare two character strings of STRING data type:
		The result is TRUE, if string IN1 **does not equal** string IN2.

Note

The following conditions cause a non-fatal error, set power flow to OFF (ENO bit = 0), and use value 0 as the result of the comparison:

- Illegal indirect address is encountered (any compare instruction)
- A variable string with a length greater than 254 characters is encountered (Compare String instruction)
- A variable string whose starting address and length are such that it will not fit in the specified memory area (Compare String instruction)

To prevent these conditions from occurring, ensure that you properly initialize pointers and memory locations that are intended to hold ASCII strings prior to executing compare instructions that use these values. Ensure that the buffer reserved for an ASCII string can reside completely within the specified memory area.

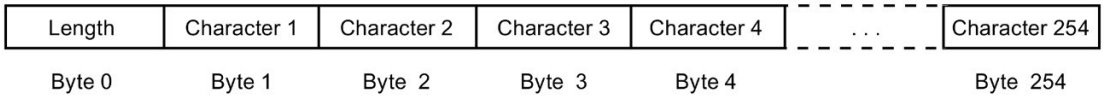
Compare instructions are executed regardless of the state of power flow.

Input / output	Data type	Operand
IN1	STRING	VB, LB, *VD, *LD, *AC, Constant string
IN2	STRING	VB, LB, *VD, *LD, *AC
OUT	BOOL	LAD: Power flow FBD: I, Q, V, M, SM, S, T, C, L, Logic Flow

Format of the STRING data type

A string variable is a sequence of characters, with each character stored as a byte. The first byte of the STRING data type defines the length of the string, which is the number of character bytes.

The diagram below shows the STRING data type stored as a variable in memory. The string can have a length of 0 to 254 characters. The maximum storage requirement for a variable string is 255 bytes (the length byte plus 254 characters).



If a constant string parameter is entered directly in the program editor (126 characters maximum) or a variable string is initialized in the data block editor (254 characters maximum), the string assignment must begin and end with double quote characters.

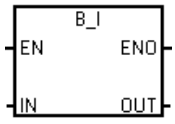
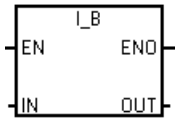
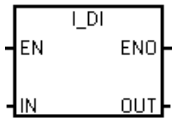
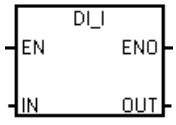
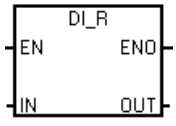
See also Constants (Page 75)

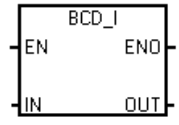
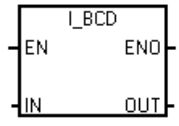
7.5 Convert

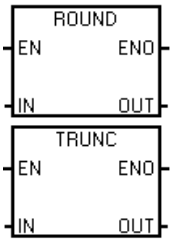
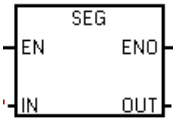
7.5.1 Standard conversion instructions

These instructions convert an input value IN to the assigned format and store the output value in the memory location assigned by OUT. For example, you can convert a double integer value to a real number. You can also convert between integer and BCD formats.

Standard conversions

LAD / FBD	STL	Description
	`BTI IN, OUT`	**Byte to integer**: Convert the byte value IN to an integer value and place the result at the address assigned to OUT. The byte is unsigned; therefore, there is no sign extension.
	`ITB IN, OUT`	**Integer to byte**: Convert the word value IN to a byte value and place the result at the address assigned to OUT. Values 0 to 255 are converted. All other values result in overflow and the output is not affected. **Note:** To change an integer to a real number, execute the Integer to Double Integer instruction and then the Double Integer to Real instruction.
	`ITD IN, OUT`	**Integer to double integer**: Convert the integer value IN to a double integer value and place the result at the address assigned to OUT. The sign is extended.
	`DTI IN, OUT`	**Double Integer to integer**: Convert the double integer value IN to an integer value and place the result at the address assigned to OUT. If the value that you convert is too large to be represented in the output, then the overflow bit is set and the output is not affected.
	`DTR IN, OUT`	**Double integer to real**: Convert a 32-bit, signed integer IN into a 32-bit real number and place the result at the address assigned to OUT.


 
	`BCDI OUT`
	
	`IBCD OUT`
		BCD to Integer:
		Convert the binary-coded decimal WORD data type value IN to an integer WORD data type value and load the result in the address assigned to OUT. The valid range for IN is 0 to 9999 BCD.
		
		Integer to BCD:
		Convert the input integer WORD data type value IN to a binary-coded decimal WORD data type and load the result at the address assigned to OUT. The valid range for IN is 0 to 9999 integer.
		For STL, the IN and OUT parameters use the same address.

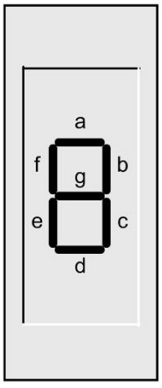
LAD / FBD	STL	Description
	**ROUND IN, OUT** **TRUNC IN, OUT**	**Round**: Convert the 32-bit real-number value IN to a double integer value and place the rounded result at the address assigned to OUT. If the fraction portion is 0.5 or greater, the number is rounded up. **Truncate**: Convert the 32-bit real-number value IN into a double integer value and place the result at the address assigned to OUT. Only the whole number portion of the real number is converted, and the fraction is discarded. **Note:** If the value that you are converting is not a valid real number or is too large to be represented in the output, then the overflow bit is set and the output is not affected.
	**SEG IN, OUT**	**SEG:** To illuminate the segments of a seven-segment display, the Segment instruction converts the character byte specified by IN to generate a bit pattern byte at the address assigned to OUT. The illuminated segments represent the character in the least significant digit of the input byte.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Indirect address • SM1.1 Overflow • SM1.6 Invalid BCD	• SM1.1 Overflow • SM1.6 Invalid BCD

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD (BCD_I, I_BCD), INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AIW, AC, *VD, *LD, *AC, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, HC, AC, *VD, *LD, *AC, Constant
	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD (BCD_I, I_BCD)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	INT (B_I, DI_I)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC,, AQW, *VD, *LD, *AC
	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

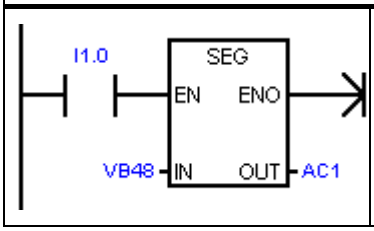
Coding for a seven-segment display

(IN) LSD	Segment Display	(OUT) - gfe dcba
0	0	0011 1111
1	1	0000 0110
2	2	0101 1011
3	3	0100 1111
4	4	0110 0110
5	5	0110 1101
6	6	0111 1101
7	7	0000 0111



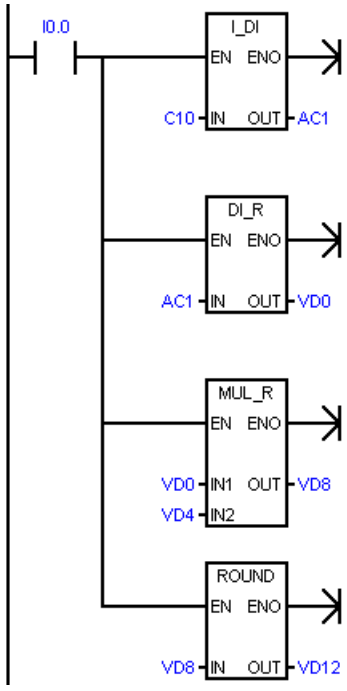
(IN) LSD	Segment Display	(OUT) - gfe dcba
8	8	0111 1111
9	9	0110 0111
A	A	0111 0111
B	b	0111 1100
C	C	0011 1001
D	d	0101 1110
E	E	0111 1001
F	F	0111 0001

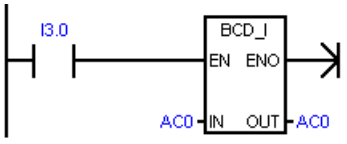
Example: Using SEG to display the numeral 5 on a seven-segment display

LAD		STL
		**Network 1** **LD I1.0** **SEG VB48, AC1**

Examples: I_DI, DI_R, and BCD_I

LAD		STL

	Convert inches to centimeters:
	1. Load a counter value (inches) into AC1 (ex. C10=101).
	2. Convert the value to a real number (ex. VD0=101.0).
	3. Multiply by 2.54 to convert to centimeters (ex. VD4=2.54, VD8=256.54).
	4. Convert the value back to an integer (ex. VD12=257).
		`Network 1`
		`LD I0.0`
		`ITD C10, AC1`
		`DTR AC1, VD0`
		`MOVR VD0, VD8`
		`*R VD4, VD8`
		`ROUND VD8, VD12`

	Convert a BCD value to an integer (ex. AC0=1234, execute BCD_I, then AC0=04D2).	`Network 2`
		`LD I0.3`
		`BCDI AC0`

See also

Assigning a constant value for instructions

See also

Assigning a constant value for instructions (Page 75)

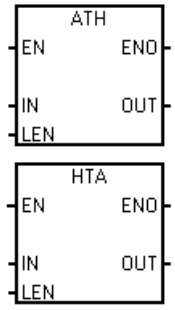
7.5.2 ASCII character array conversion

Converting from or to ASCII character byte arrays

The ASCII character array instructions use the BYTE data type for character input or output. An array of ASCII characters is referenced a sequence of byte addresses.

This is not the STRING data type, as no length byte is used. Use the ASCII string instructions to work with the variables of the STRING data type.

ASCII to Hex and Hex to ASCII

LAD / FBD	STL	Description
	**ATH IN, OUT, LEN** **HTA IN, OUT, LEN**	ATH converts a number LEN of ASCII characters, starting at IN, to hexadecimal digits starting at OUT. The maximum number of ASCII characters that can be converted is 255 characters. HTA converts the hexadecimal digits, starting with the input byte IN, to ASCII characters starting at OUT. The number of hexadecimal digits to be converted is assigned by length LEN. The maximum number of ASCII characters or hexadecimal digits that can be converted is 255. Valid ASCII input characters are the alphanumeric characters 0 to 9 with a hexadecimal code value of 30 to 39, and uppercase characters A to F with a hex code value of 41 to 46.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • SM1.7 ATH: Illegal ASCII value	• SM1.7 ATH: Illegal ASCII value

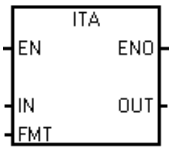
Input / output	Data type	Operand
IN, OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC
LEN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

Converting number values to the ASCII character representation (ITA, DTA, and RTA)

ASCII character output number format:

- Positive values are written to the output buffer without a sign.
- Negative values are written to the output buffer with a leading minus sign (-).
- Leading zeros to the left of the decimal point (except the digit adjacent to the decimal point) are suppressed.
- Values are right-justified in the output buffer.
- Real numbers: Values to the right of the decimal point are rounded to fit in the assigned number of digits to the right of the decimal point.
- Real numbers: The size of the output buffer must be a minimum of three bytes more than the number of digits to the right of the decimal point.

Integer to ASCII

LAD / FBD	STL	Description
	**ITA IN, OUT, FMT**	The Integer to ASCII instruction converts the integer value IN to an array of ASCII characters. The format parameter FMT assigns the conversion precision to the right of the decimal, and whether the decimal point is to be shown as a comma or a period. The resulting conversion is placed in 8 consecutive bytes beginning with the address assigned by OUT.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • FMT bit is not zero for 4 most significant bits of the FMT byte • nnn > 5	None

Input / output	Data type	Operand
IN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

The size of the output buffer is always 8 bytes. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted with no decimal point. For values of nnn greater than 5, the output buffer is filled with ASCII space characters. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction. The most significant 4 bits must always be zero.

The following figure shows examples of values that are formatted using a decimal point (c=0) with three digits to the right of the decimal point (nnn=011).

FMT operand for the integer to ASCII (ITA) instruction

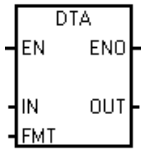
FMT

MSB							LSB
7	6	5	4	3	2	1	0
0	0	0	0	c	n	n	n

c = comma (1) or decimal point (0)
nnn= digits to right of decimal point

	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6	Out +7
in = 12				0	.	0	1	2
in = -123			-	0	.	1	2	3
in = 1234				1	.	2	3	4
in = -12345		-	1	2	.	3	4	5

Double integer to ASCII

LAD / FBD	STL	Description
	`DTA IN, OUT, FMT`	The **Double Integer to ASCII** instruction converts a double word IN to an array of ASCII characters. The format parameter FMT specifies the conversion precision to the right of the decimal. The resulting conversion is placed in 12 consecutive bytes beginning with OUT.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Invalid indirect address • 0091H Operand out of range • FMT bit is not zero for 4 most significant bits, of the FMT byte • nnn > 5	• None

Input / output	Data type	Operand
IN	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

The size of the output buffer is always 12 bytes. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted with no decimal point. For values of nnn bigger than 5, the output buffer is filled with ASCII spaces. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction. The most significant 4 bits must always be zero.

The following figure shows examples of values that are formatted using a decimal point (c=0) with four digits to the right of the decimal point (nnn=100).

FMT operand for the double integer to ASCII (DTA) instruction

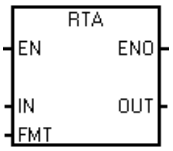
FMT

MSB							LSB
7	6	5	4	3	2	1	0
0	0	0	0	c	n	n	n

c = comma (1) or decimal point (0)
nnn= digits to right of decimal point

	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6	Out +7	Out +8	Out +9	Out +10	Out +11
in = 12						-	0	.	0	0	1	2
in = 1234567					1	2	3	.	4	5	6	7

Real to ASCII

LAD / FBD	STL	Description
	`RTA IN, OUT, FMT`	The **Real to ASCII** instruction converts a real-number value IN to ASCII characters. The format parameter FMT specifies the conversion precision to the right of the decimal, whether the decimal point is shown as a comma or a period, and the output buffer size. The resulting conversion is placed in an output buffer beginning with OUT.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Invalid indirect address • 0091H Operand out of range • nnn > 5 • ssss < 3 • ssss < number of characters in OUT	None

Input / output	Data type	Operand
IN	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

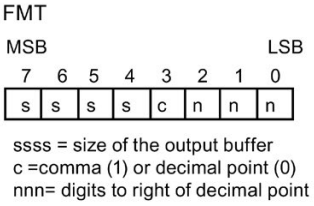
The number (or length) of the resulting ASCII characters is the size of the output buffer and can be assigned from 3 to 15 bytes or characters.

The real-number format supports a maximum of 7 significant digits. Attempting to display more than 7 significant digits produces a rounding error.

The following figure describes the format operand (FMT) for the RTA instruction. The size of the output buffer is assigned by the ssss field. A size of 0, 1, or 2 bytes is not valid. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted without a decimal point. The output buffer is filled with ASCII spaces for values of nnn greater than 5 or when the assigned output buffer is too small to store the converted value. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction.

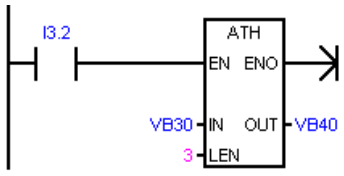
The following figure also shows examples of values that are formatted using a decimal point (c=0) with one digit to the right of the decimal point (nnn=001) and a buffer size of six bytes (ssss=0110).

FMT Operand for the Real to ASCII (RTA) instruction



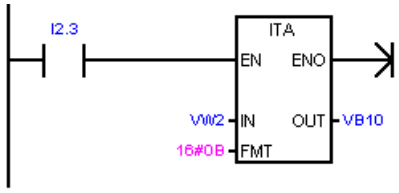
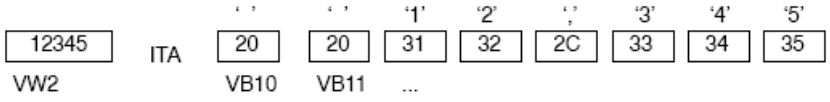
	Out	Out +1	Out +2	Out +3	Out +4	Out +5
in = 1234.5	1	2	3	4	.	5
in = -0.0004				0	.	0
in = -3.67526			-	3	.	7
in = 1.95				2	.	0

Example: ASCII to Hexadecimal

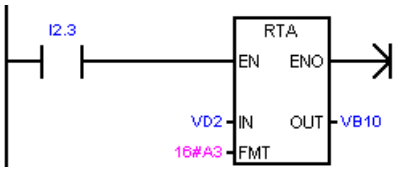
LAD	STL
	**Network 1** **LD I3.2** **ATH VB30, VB40, 3**

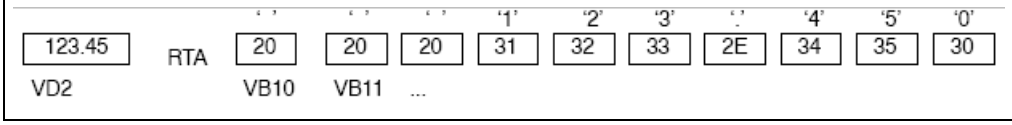
[1] The "x" indicates that the "nibble" (half of a byte) is unchanged.

Example: Integer to ASCII

LAD		STL
	Convert the integer value at VW2 to 8 ASCII characters starting at VB10, using a format of 16#0B (a comma for the decimal point, followed by 3 digits).	**Network 1** **LD I2.3** **ITA VW2, VB10, 16#0B**
		

Example: Real to ASCII

LAD		STL
	Convert the real value at VD2 to 10 ASCII characters starting at VB10, using a format of 16#A3 (a period for the decimal point, followed by 3 digits).	**Network 1** **LD I2.3** **RTA VD2, VB10, 16#A3**



See also

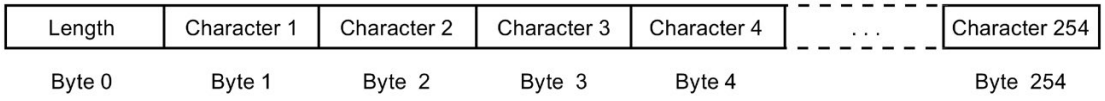
Assigning a constant value for instructions (Page 75)

7.5.3 Number value to ASCII string conversion

Format of the STRING data type

A string variable is a sequence of characters, with each character stored as a byte. The first byte of the STRING data type defines the length of the string, which is the number of character bytes.

The diagram below shows the STRING data type stored as a variable in memory. The string can have a length of 0 to 254 characters. The maximum storage requirement for a variable string is 255 bytes (the length byte plus 254 characters).



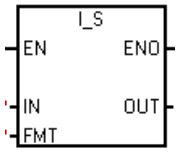
If a constant string parameter is entered directly in the program editor (126 characters maximum) or a variable string is initialized in the data block editor (254 characters maximum), the string assignment must begin and end with double quote characters.

ASCII output number format

- Positive values are written to the output buffer without a sign.
- Negative values are written to the output buffer with a leading minus sign (-).
- Leading zeros to the left of the decimal point (except the digit adjacent to the decimal point) are suppressed.
- Values are right-justified in the output string.

- Real numbers: Values to the right of the decimal point are rounded to fit in the specified number of digits to the right of the decimal point.
- Real numbers: The size of the output string must be a minimum of three bytes more than the number of digits to the right of the decimal point.

Integer to string conversion

LAD / FBD	STL	Description
	**ITS IN, OUT, FMT**	The Integer to String instruction converts an integer word IN to an ASCII string with a length of 8 characters. The format (FMT) assigns the conversion precision to the right of the decimal, and whether the decimal point is to be shown as a comma or a period. The resulting string is written to 9 consecutive bytes starting at OUT.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H indirect address • 0091H operand out of range • Illegal format (nnn > 5) • FMT bit is not zero for the four most significant bits of the FMT byte	None

Input / output	Data type	Operand
IN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	STRING	VB, LB, *VD, *LD, *AC

The length of the output string is always 8 characters. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted without a decimal point. For values of nnn greater than 5, the output is a string of 8 ASCII space characters. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction. The most significant 4 bits of the format must be zero.

The following figure also shows examples of values that are formatted using a decimal point (c= 0) with three digits to the right of the decimal point (nnn = 011). The value at OUT is the length of the string stored in the next byte addresses.

FMT parameter for the integer to string instruction

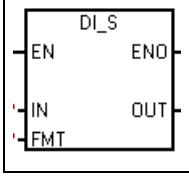
FMT

MSB							LSB
7	6	5	4	3	2	1	0
0	0	0	0	c	n	n	n

c = comma (1) or decimal point (0)
nnn= digits to right of decimal point

	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6	Out +7	Out +8
in = 12	8				0	.	0	1	2
in = -123	8			-	0	.	1	2	3
in = 1234	8				1	.	2	3	4
in = -12345	8		-	1	2	.	3	4	5

Double integer to string conversion

LAD / FBD	STL	Description
	`DTS IN, OUT, FMT`	The Double Integer to String instruction converts a double integer IN to an ASCII string with a length of 12 characters. The format (FMT) assigns the conversion precision to the right of the decimal, and whether the decimal point is to be shown as a comma or a period. The resulting string is written to 13 consecutive bytes starting at OUT.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H indirect address • 0091H operand out of range • Illegal format (nnn > 5) • FMT bit is not zero for the four most significant bits of the FMT byte	None

Input / output	Data type	Operand
IN	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	STRING	VB, LB, *VD, *LD, *AC

The length of the output string is always 12 characters. The number of digits to the right of the decimal point in the output buffer is specified by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point causes, then the value is displayed without a decimal point. For values of nnn greater than 5, the output is a string of 12 ASCII space characters. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between the whole number and the fraction. The upper 4 bits of the format must be zero.

The following figure also shows examples of values that are formatted using a decimal point (c= 0) with four digits to the right of the decimal point (nnn = 100). The value at OUT is the length of the string stored in the next byte addresses.

FMT operand for the double integer to string instruction

FMT

MSB							LSB
7	6	5	4	3	2	1	0
0	0	0	0	c	n	n	n

c = comma (1) or decimal point (0)
nnn= digits to right of decimal point

	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6	Out +7	Out +8	Out +9	Out +10	Out +11	Out +12
in = 12	12							0	.	0	0	1	2
in = 1234567	12					1	2	3	.	4	5	6	7

Real to string conversion

LAD / FBD		Description
R_S EN ENO IN OUT FMT	**RTS IN, OUT, FMT**	The Real to String instruction converts a real value IN to an ASCII string. The format (FMT) assigns the conversion precision to the right of the decimal, whether the decimal point is to be shown as a comma or a period and the length of the output string. The resulting conversion is placed in a string beginning with OUT. The length of the resulting string is specified in the format and can be 3 to 15 characters.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H indirect address • 0091H operand out of range • Illegal format – (nnn > 5) – ssss < 3 – ssss < number of characters required	None

Input / output	Data type	Operand
IN	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	STRING	VB, LB, *VD, *LD, *AC

The real-number format used by the CPU supports a maximum of 7 significant digits. An attempt to display more than the 7 significant digits produces a rounding error.

The length of the output string is specified by the ssss field. A size of 0, 1, or 2 bytes is not valid. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is displayed without a decimal point. The output string is filled with ASCII space characters when nnn is greater than 5 or when the assigned length of the output string is too small to store the converted value. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between the whole number and the fraction.

The following figure also shows examples of values that are formatted using a decimal point (c= 0) with one digit to the right of the decimal point (nnn = 001) and an output string length

of 6 characters (ssss = 0110). The value at OUT is the length of the string stored in the next byte addresses.

FMT operand for the real to string instruction

FMT

MSB							LSB
7	6	5	4	3	2	1	0
s	s	s	s	c	n	n	n

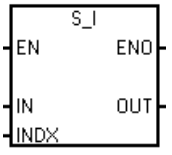
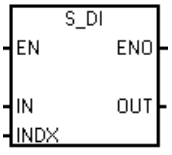
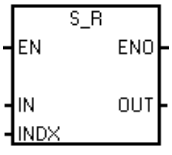
ssss = length of output string
c = comma (1) or decimal point (0)
nnn= digits to right of decimal point

	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6
in = 1234.5	6	1	2	3	4	.	5
in = -0.0004	6				0	.	0
in = -3.67526	6			-	3	.	7
in = 1.95	6				2	.	0

See Also

Assigning a constant value for instructions (Page 75)

7.5.4 ASCII sub-string to number value conversion

LAD / FBD	STL	Description
	`STI IN, INDX, OUT`	ASCII sub-string to integer value conversion
	`STD IN, INDX, OUT`	ASCII sub-string to double integer value conversion
	`STR IN, INDX, OUT`	ASCII sub-string to real value conversion

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • 009BH Index = 0 • SM1.1 Overflow or illegal value	• SM1.1 Overflow or illegal value

Input / output	Data type	Operand
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
INDX	BYTE	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	INT	VW, IW, QW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC
	DINT, REAL	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

String input format for S_I (integer number) and S_DI (double integer number)

[spaces] [+ or -] [digits 0 - 9]

String input format for S_R (real number)

[spaces] [+ or -] [digits 0 - 9] [. or ,] [digits 0 - 9]

INDX parameter

The INDX value is normally set to 1, which starts the conversion with the first character of the string. The INDX value can be set to other values to start the conversion at different points within the string. This can be used when the input string contains text that is not part of the number to be converted. For example, if the input string is "Temperature: 77.8", you set INDX to a value of 13 to skip over the word "Temperature: " at the start of the string.

The Substring to Real instruction does not convert strings using scientific notation or exponential forms of real numbers. The instruction does not produce an overflow error (SM1.1) but converts the string to a real number up to the exponential and then terminates the conversion. For example, the string '1.234E6' converts without errors to a real value of 1.234.

The conversion is terminated when the end of the string is reached or when the first invalid character is found. An invalid character is any character that is not a digit (0 - 9), or one of the following characters: plus (+), minus (-), comma (,), or period (.).

The overflow error (SM1.1) is set whenever the conversion produces an integer value that is too large for the output value. For example, the Substring to Integer instruction sets the overflow error if the input string produces a value greater than 32767 or less than -32768.

The overflow error (SM1.1) is also set if no conversion is possible when the input string does not contain a valid value. For example, if the input string contains 'A123', the conversion instruction sets SM1.1 (overflow) and the output value remains unchanged.

Examples of valid and invalid input strings

Valid Input Strings
for Integer and Double Integer

Input String	Output Integer
'123'	123
'-00456'	-456
'123.45'	123
'+2345'	2345
'000000123ABCD'	123

Valid Input Strings
for Real Numbers

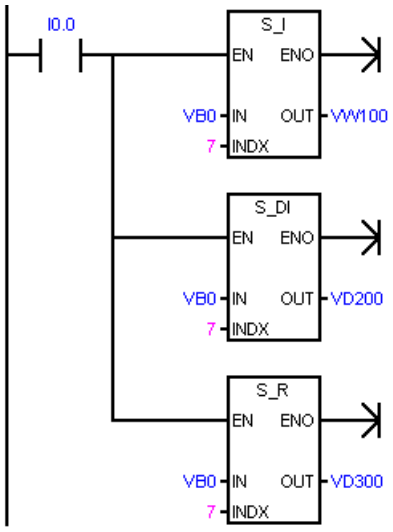
Input String	Output Real
'123'	123.0
'-00456'	-456.0
'123.45'	123.45
'+2345'	2345.0
'00.000000123'	0.000000123

Invalid Input Strings

Input String
'A123'
' '
'++123'
'+-123'
'+ 123'

Example string conversion: Substring to integer, double integer, and real

LAD



S_I converts the numeric string to an integer value.

S_DI converts the numeric string to a double integer value.

S_R converts the numeric string to a real value.

STL

```
Network 1
LD I0.0
STI VB0, 7, VW100
STD VB0, 7, VD200
STR VB0, 7, VD300
```

VB0 VB11

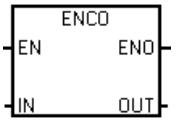
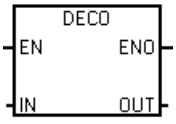
11	'T'	'e'	m'	'p'	' '	' '	'9'	'8'	'.'	'6'	'F'

After executing the network:
VW100 (integer) = 98
VD200 (double integer) = 98
VD300 (real) = 98.6

See also

Assigning a constant value for instructions (Page 75)

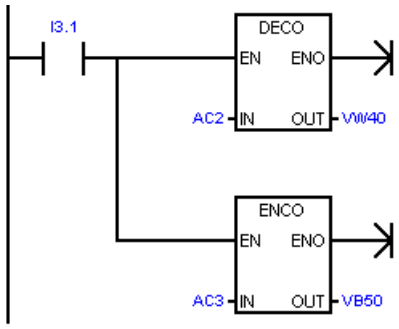
7.5.5 Encode and decode

LAD / FBD	STL	Description
	**ENCO IN, OUT**	Encode writes the bit number of the least significant bit set in the input word IN, to the least significant "nibble" (4 bits) of the output byte OUT.
	**DECO IN, OUT**	Decode sets the bit in the output word OUT that corresponds to the bit number represented by the least significant "nibble" (4 bits) of the input byte IN. All other bits of the output word are set to 0.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Indirect address	None

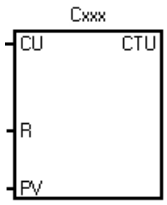
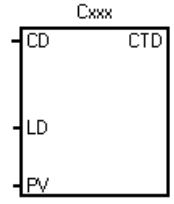
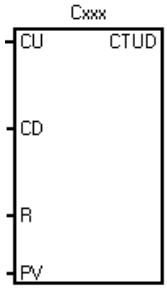
Input / output	Data type	Operand
IN	WORD (ENCO)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	BYTE (DECO)	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE (ENCO)	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD (DECO)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC

Example: Encode and decode

<table>
<tr><th>LAD</th><th></th><th>STL</th></tr>
<tr>
<td> 
</td>
<td>If AC2 contains error bits:
1. The DECO instruction sets the bit in VW40 that corresponds to this error code
2. The ENCO instruction converts the least significant bit set to an error code that is stored in VB50.</td>
<td>Network 1
LD I3.1
DECO AC2, VW40
ENCO AC3, VB50</td>
</tr>
<tr>
<td colspan="3">
</td>
</tr>
</table>

7.6 Counters

7.6.1 Counter instructions

LAD / FBD	STL	Description
	`CTU Cxxx, PV`	**LAD/FBD:** The **CTU** count up instruction counts up from the current value each time the count up CU input makes the transition from OFF to ON. When the current value Cxxx is greater than or equal to the preset value PV, the counter bit Cxxx is set ON. The current count value is reset when the reset input R is set ON, or when the reset instruction is executed for the Cxxx address. The counter stops counting when it reaches the maximum value 32,767. **STL:** R reset input is the top of stack value. CU count up input is loaded in the second stack level
	`CTD Cxxx, PV`	**LAD/FBD:** The **CTD** count down instruction counts down from the current value of that counter each time the CD count down input makes the transition from OFF to ON. When the current value Cxxx is equal to 0, the counter bit Cxxx turns ON. The counter resets the counter bit Cxxx and loads the current value with the preset value PV when the LD load input is set ON. The counter stops upon reaching zero, and the counter bit Cxxx is set ON. **STL:** LD load input is the top of stack value. CD count down input value is loaded in the second stack level
	`CTUD Cxxx, PV`	**LAD/FBD:** The **CTUD** count up/down instruction counts up each time the CU count up input makes the transition from OFF to ON, and counts down each time the CD count down input makes the transition from OFF to ON. The current value Cxxx of the counter maintains the current count. The PV preset value is compared to the current value each time the counter instruction is executed. Upon reaching maximum value 32,767, the next rising edge at the count up input causes the current count to wrap around to the minimum value -32,768. On reaching the minimum value -32,768, the next rising edge at the count down input causes the current count to wrap around to the maximum value 32,767. When the current value Cxxx is greater than or equal to the PV preset value, the counter bit Cxxx is set ON. Otherwise, the counter bit is OFF. The counter is reset when the R reset input is set ON, or when the Reset instruction is executed for the Cxxx address. **STL:** R reset input is the top of stack value. CD count down input value is loaded in the second stack level. CU count Up input value is loaded in the third stack level

Input / output	Data type	Operand
Cxxx	WORD	Constant (C0 to C255)
CU, CD (LAD)	BOOL	Power flow
CU, CD (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
R (LAD)	BOOL	Power Flow
R (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
LD (LAD)	BOOL	Power Flow
LD (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
PV	INT	IW, QW, VW, MW, SMW, SW, LW, T, C, AC, AIW, *VD, *LD, *AC, Constant

Note

Since there is one current value for each counter, do not assign the same counter number to more than one counter. (Up Counters, Up/Down Counters, and Down counters with the same number access the same current value.)

When you reset a counter using the Reset instruction, the counter bit is reset and the counter current value is set to zero. Use the counter number to reference both the current value and the counter bit of that counter.

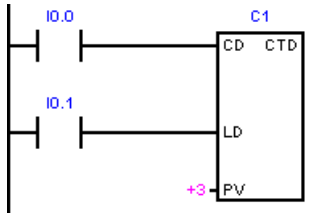
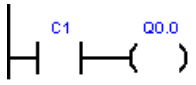
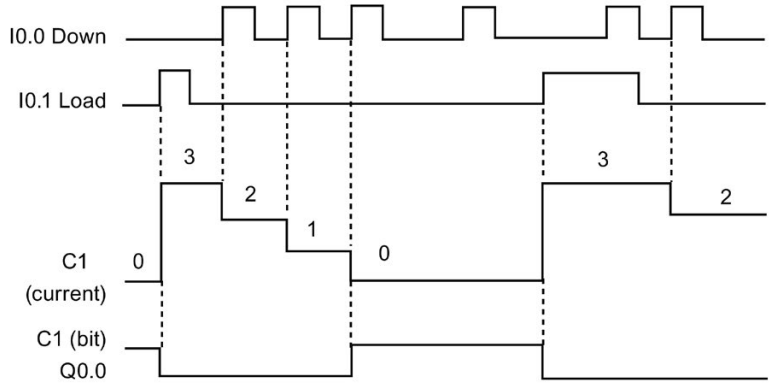
See also Configuring the retentive ranges - system block configuration (Page 137)

Counter operation

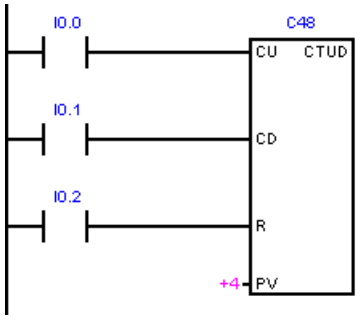
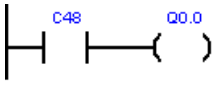
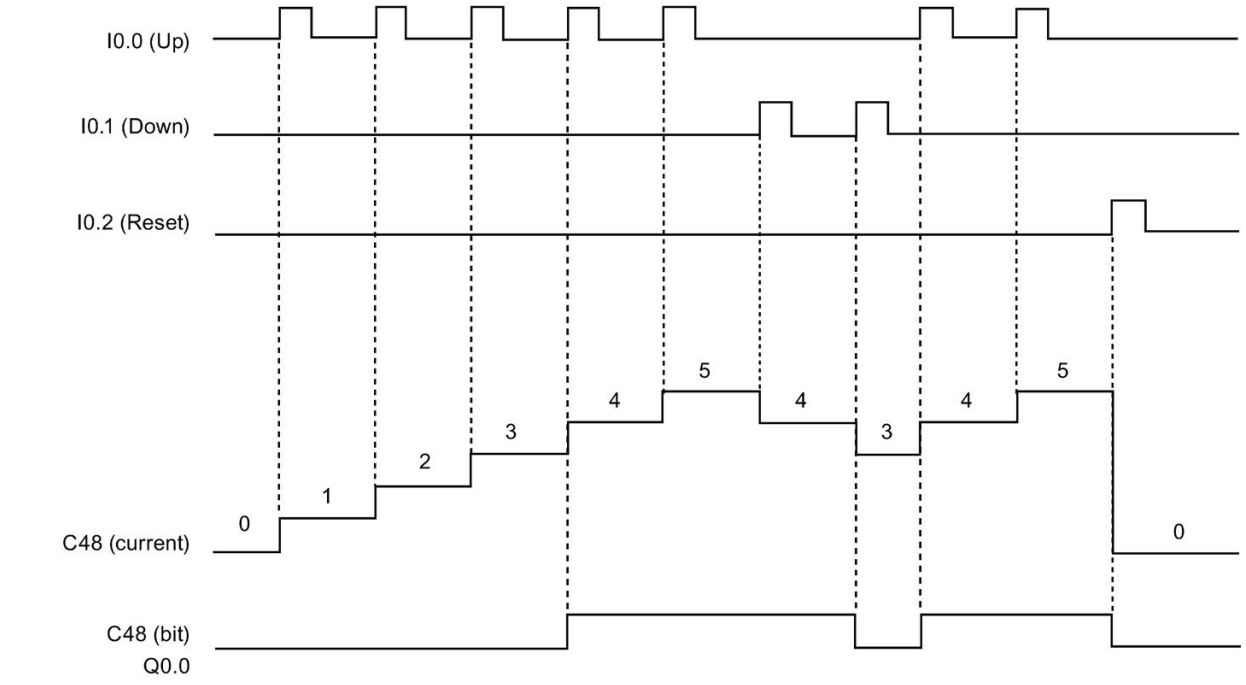
Type	Operation	Counter bit	Power cycle / first scan
CTU	• CU increments the current value. • Current value continues to increment until it reaches 32,767.	The counter bit is set ON when: Current value >= Preset	• Counter bit is OFF. • Current value can be retained [1]
CTD	• CD decrements the current value until the current value reaches 0.	The counter bit is set ON when: Current value = 0	• Counter bit is OFF. • Current value can be retained [1]
CTUD	• CU increments the current value. • CD decrements the current value. • Current value continues to increment or decrement until the counter is reset.	The counter bit is set ON when: Current value >= Preset	• Counter bit is OFF. • Current value can be retained [1]

[1] You can select the current value for the counter to be retentive, but not the counter bit value.

Example CTD count down

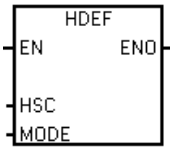
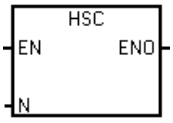
<table>
<tr><th>LAD</th><th></th><th>STL</th></tr>
<tr><td> 
</td><td>Count down counter C1 current value counts from 3 to 0
With I0.1 OFF, I0.0 OFF-ON decrements C1 current value
I0.1 ON loads countdown preset value 3</td><td>Network 1
LD I0.0
LD I0.1
CTD C1, +3</td></tr>
<tr><td> 
</td><td>C1 bit is ON when counter C1 current value = 0</td><td>Network 2
LD C1
= Q0.0</td></tr>
<tr><td colspan="3">Timing diagram
 
</td></tr>
</table>

Example CTUD count up/down

LAD		STL
	I0.0 counts up I0.1 counts down I0.2 resets current value to 0	**Network 1** **LD I0.0** **LD I0.1** **LD I0.2** **CTUD C48, +4**
	Count Up/Down counter C48 turns on C48 bit when current value >= 4	**Network 2** **LD C48** **= Q0.0**
Timing diagram 		

7.6.2 High-speed counter instructions

High-speed counters can count high-speed events that cannot be controlled by standard counters. Standard counters operate at a slower rate that is limited by the PLC scan time. You can use the HDEF and HSC instructions and create your own HSC routines, or you can simplify the programming tasks by using the High Speed Counter wizard.

LAD / FBD	STL	Description
	**HDEF HSC, MODE**	The High-Speed Counter Definition instruction (HDEF) selects the operating mode of a specific high-speed counter (HSC0-5). The mode selection defines the clock, direction, and reset functions of the high-speed counter. You must use one High-Speed Counter Definition instruction for each of up to six active high-speed counters. The S model CPUs[1] have six HSCs. The C model CPUs[2] have four HSCs.
	**HSC N**	The High-Speed Counter (HSC) instruction configures and controls the high-speed counter, based on the state of the HSC special memory bits. The parameter N specifies the high-speed counter number. The high-speed counters can be configured for up to eight different modes of operation. Each counter has dedicated inputs for clocks, direction control, and reset where these functions are supported. In AB quadrature phase, you can select one times (1x) or four times (4x) the maximum counting rate. All counters run at maximum rates without interfering with one another.

[1] S model CPUs: SR20, ST20, SR30, ST30, SR40, ST40, SR60, and ST60

[2] C model CPUs: CR20s, CR30s, CR40s, and CR60s

Error conditions with ENO = 0		SM bits affected
HDEF: • 0003H Input point conflict • 0004H Illegal instruction in interrupt • 000AH HSC redefinition • 0016H Attempted to use HSC or Edge Interrupt on Input that is allocated for use by Motion Functionality • 0090H Invalid HSC number	HSC: • 0001H HSC before HDEF • 0005H Simultaneous HSC/PLS • 0090H Invalid HSC number	None

Input / output	Data type	Operand
HSC	BYTE	HSC number constant (0, 1, 2, 3, 4, or 5)
MODE	BYTE	Mode number constant: Eight possible modes (0, 1, 3, 4, 6, 7, 9, or 10)
N	WORD	HSC number constant (0, 1, 2, 3, 4, or 5)

HSC operation

A high-speed counter can be used as the drive for a drum timer, where a shaft rotating at a constant speed is fitted with an incremental shaft encoder. The shaft encoder provides a specified number of counts per revolution and a reset pulse that occurs once per revolution. The clock(s) and the reset pulse from the shaft encoder provide the inputs to the high-speed counter.

The high-speed counter is loaded with the first of several presets, and the desired outputs are activated for the time period where the current count is less than the current preset. The counter is set up to provide an interrupt when the current count is equal to preset and also when reset occurs.

As each current-count-value-equals-preset-value interrupt event occurs, a new preset is loaded and the next state for the outputs is set. When the reset interrupt event occurs, the first preset and the first output states are set, and the cycle is repeated.

Since the program interrupts occur at a much lower rate than the counting rates of the high-speed counters, precise control of high-speed operations can be implemented with relatively minor impact to the overall PLC scan cycle time. The method of interrupt attachment allows each load of a new preset to be performed in a separate interrupt routine for easy state control. (Alternatively, all interrupt events can be processed in a single interrupt routine.)

HSC input assignments and capabilities

All high-speed counters function the same way for the same mode of operation, but every mode is not supported for every HSC number. The HSC input connections (clock, direction, and reset) must use the CPU's integrated input channels as shown in the High-speed counter summary (Page 251) table. Input channels located on a signal board or an expansion module cannot be used for high-speed counters.

Note

You must ensure that high-speed counter inputs are correctly filtered and wired, for counting high frequency signals.

In an S7-200 SMART CPU, all high-speed counter inputs are connected to internal input filter circuits. The S7-200 SMART default input filter setting is 6.4 ms, which limits the maximum counting rate to 78 Hz. You must change the filter settings to count higher frequencies.

Refer to "Noise reduction for high-speed inputs (Page 252)" for details about system block filter options, maximum counting frequencies, shielding requirements, and external pull-down circuits.

HSC counting mode support

- The compact models support a total of four HSC devices (HSC0, HSC1, HSC2, and HSC3).
- The SR and ST models support a total of six HSC devices (HSC0, HSC1, HSC2, HSC3, HSC4, and HSC5).

- HSC0, HSC2, HSC4, and HSC5 support eight counter modes (mode 0, 1, 3, 4, 6, 7, 9, and 10).
- HSC1 and HSC3 support only one counter mode (mode 0).

Available HSC counter types

- Single-phase clock counter with internal direction control:
  - Mode 0:
  - Mode 1: with external reset
- Single-phase clock counter with external direction control:
  - Mode 3:
  - Mode 4: with external reset
- Two-phase clock counter with 2 clock inputs (clock-up and clock-down):
  - Mode 6:
  - Mode 7: with external reset
- AB quadrature phase counter:
  - Mode 9:
  - Mode 10: with external reset

HSC operating rules

- Before you use a high-speed counter, you must execute the HDEF instruction (High-Speed Counter Definition) to select a counter mode. Use the first scan memory bit, SM0.1 (this bit is ON for the first scan and OFF for subsequent scans) to execute HDEF directly, or call a subroutine that contains the HDEF instruction.
- You can use all counter types with or without a reset input.
- When you activate the reset input, it clears the current value and holds it clear until you deactivate the reset input.

Reference information

Refer to the following sections for further information:

- High-speed counter programming (Page 254)
- High-speed counter summary (Page 251)
- Example initialization sequences for high-speed counters (Page 265)
- Noise reduction for high-speed inputs (Page 252)

7.6.3 High-speed counter summary

	Clock A	Direction / Clock B	Reset	Single phase / Dual phase maximum clock / input rate	AB quadrature phase maximum clock / input rate
HSC0	I0.0	I0.1	I0.4	S model CPUs:[1] • 200 kHz	S model CPUs: • 100 kHz = Maximum 1x count rate • 400 kHz = Maximum 4x count rate
				C model CPUs:[2] • 100 kHz	C model CPUs: • 50 kHz = Maximum 1x count rate • 200 kHz = Maximum 4x count rate
HSC1	I0.1			S model CPUs: • 200 kHz	
				C model CPUs: • 100 kHz	
HSC2	I0.2	I0.3	I0.5	S model CPUs: • 200 kHz	S model CPUs: • 100 kHz = Maximum 1x count rate • 400 kHz = Maximum 4x count rate
				C model CPUs: • 100 kHz	C model CPUs: • 50 kHz = Maximum 1x count rate • 200 kHz = Maximum 4x count rate
HSC3	I0.3			S model CPUs: • 200 kHz	
				C model CPUs: • 100 kHz	
HSC4	I0.6	I0.7	I1.2	SR30 and ST30 model CPUs: • 200 kHz	SR30 and ST30 model CPUs: • 100 kHz = Maximum 1x count rate • 400 kHz = Maximum 4x count rate
				SR20, ST20, SR40, ST40, SR60, and ST60 model CPUs: • 30 kHz	SR20, ST20, SR40, ST40, SR60, and ST60 model CPUs: • 20 kHz = Maximum 1x count rate • 80 kHz = Maximum 4x count rate
				C model CPUs: • n/a	C model CPUs: • n/a
HSC5	I1.0	I1.1	I1.3	S model CPUs: • 30 kHz	S model CPUs • 20 kHz = Maximum 1x count rate • 80 kHz = Maximum 4x count rate
				C model CPUs: • n/a	C model CPUs: • n/a

	Clock A	Direction / Clock B	Reset	Single phase / Dual phase maximum clock / input rate	AB quadrature phase maximum clock / input rate

[1] S model CPUs: SR20, ST20, SR30, ST30, SR40, ST40, SR60, and ST60

[2] C model CPUs: CR20s, CR30s, CR40s, and CR60s

7.6.4 Noise reduction for high-speed inputs

Counting high-speed pulses with HSC inputs

Note

High-speed input wiring must use shielded cables

Use shielded cable with a maximum length of 50 m, when connecting HSC input channels I0.0, I0.1, I0.2, I0.3, I0.6. I0.7, I1.0, and I1.1.

One or both of the following actions may be necessary to correctly operate a high-speed counter.

- Adjust the System Block digital input filter time of the input channels used by the HSC channel. The S7-200 SMART CPU applies input filtering before the counting of pulses by the HSC channel. This means that if an HSC input pulse occurs at a rate that is filtered out by the input filtering, then the HSC does not detect any pulses on the input. You must make sure that you configure the filter time of each input of the HSC to a value that allows counting at the rate your application requires. This includes direction and reset inputs. The following table shows the maximum input frequency that an HSC can detect for each input filter configuration:

Input filter time	Maximum detectable frequency
0.2 µs	200 kHz (S model CPUs)[1] 100 kHz (C model CPUs)[2]
0.4 µs	200 kHz (S model CPUs) 100 kHz (C model CPUs)
0.8 µs	200 kHz (S model CPUs) 100 kHz (C model CPUs)
1.6 µs	200 kHz (S model CPUs) 100 kHz (C model CPUs)
3.2 µs	156 kHz (S model CPUs) 100 kHz (C model CPUs)
6.4 µs	78 kHz
12.8 µs	39 kHz
0.2 ms	2.5 kHz
0.4 ms	1.25 kHz
0.8 ms	625 Hz
1.6 ms	312 Hz
3.2 ms	156 Hz

Input filter time	Maximum detectable frequency
6.4 ms	78 Hz
12.8 ms	39 Hz

1 S model CPUs: SR20, ST20, SR30, ST30, SR40, ST40, SR60, ST60

2 C model CPUs: CR20s, CR30s, CR40s, and CR60s

- If the device generating the HSC input signals does not drive the input signals both high and low, then signal distortion can occur at high speeds. This can occur if the output of the device is an open-collector transistor. When the transistor turns off, there is nothing driving the signal to a low state. The signal transitions to a low state, but the time to do so is dependent on the input resistance and capacitance of the circuitry. This condition can result in missed pulses. You can prevent this condition by wiring a pull-down resistor to the input signals, as seen in the following figure. Since the input voltage of the CPU is 24 V DC, the resistor has to be rated for a high wattage. A 100 ohm, 5 Watt resistor is a suitable choice.

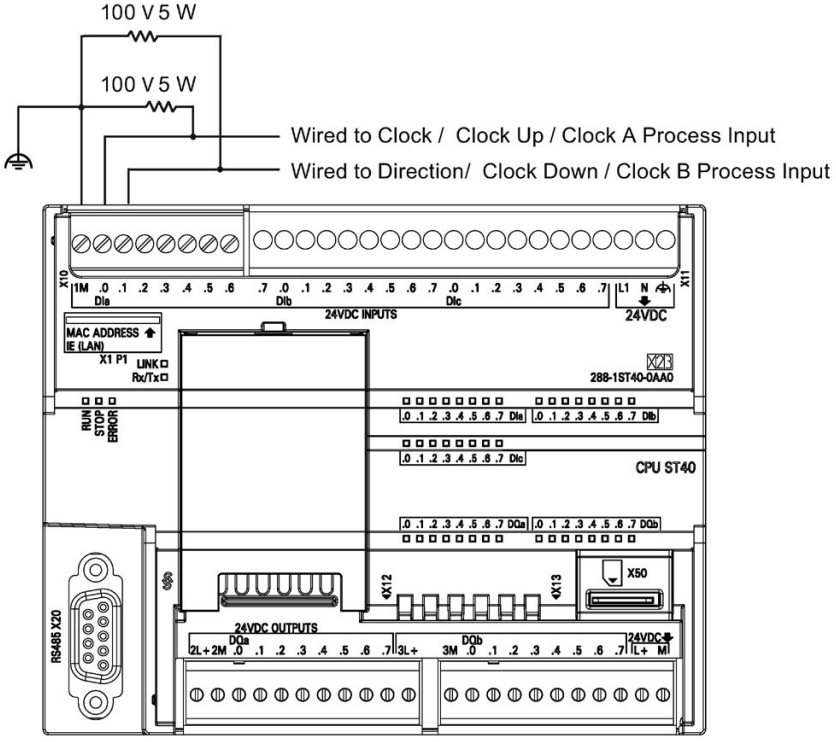


Figure 7-1 Pull-down resistor wiring for open-collector HSC input drivers

7.6.5 High-speed counter programming

You can use the high-speed counter wizard to simplify HSC programming tasks. The wizard helps you select the counter type/mode, preset/current values, counter options, and generates the necessary special memory assignments, subroutines, and interrupt routines.

Note

You must ensure that high-speed counter inputs are correctly filtered and wired for counting high frequency signals.

In an S7-200 SMART CPU, all high-speed counter inputs are connected to internal input filter circuits. The S7-200 SMART CPU default input filter setting is 6.4 ms, which limits the maximum counting rate to 78 Hz. You must change the filter settings to count higher frequencies.

Refer to "Noise reduction for high-speed inputs (Page 252)" for details about system block filter options, maximum counting frequency, shielding requirements, and external pull-down circuits.

Configuring high-speed counters

Use one of the following actions to configure the high-speed counter wizard:

- Open the wizard: Select "High-Speed Counter" in the wizards area of the Tools menu ribbon strip.
- Open the wizard: Double-click "High-Speed Counter" node in the wizards folder, from the project tree.

With the wizard open, assign the HSC setup values. You can navigate through the wizard setup pages, modify parameters, and then generate new wizard program code.

Your program must perform the following basic tasks to use a high-speed counter:

- Define the counter and mode (execute the HDEF instruction exactly once for each counter).
- Set the control byte in SM memory.
- Set the current value (starting value) in SM memory.
- Set the preset value (target value) in SM memory.
- Assign and enable appropriate interrupt routines.
- Activate the high-speed counter (execute the HSC instruction).

HDEF instruction sets the counting mode

The HDEF instruction assigns HSC counter mode. The following table shows the physical inputs assigned for clock, direction control, and reset functions. The same input cannot be used for two different functions, but you can use any input not being used by the present mode of its high-speed counter for another purpose. For example, if the present mode of HSC0 is mode 1, which uses I0.0 and I0.4; then you can use I0.1, I0.2, and I0.3 for edge interrupts, HSC3, or motion control inputs.

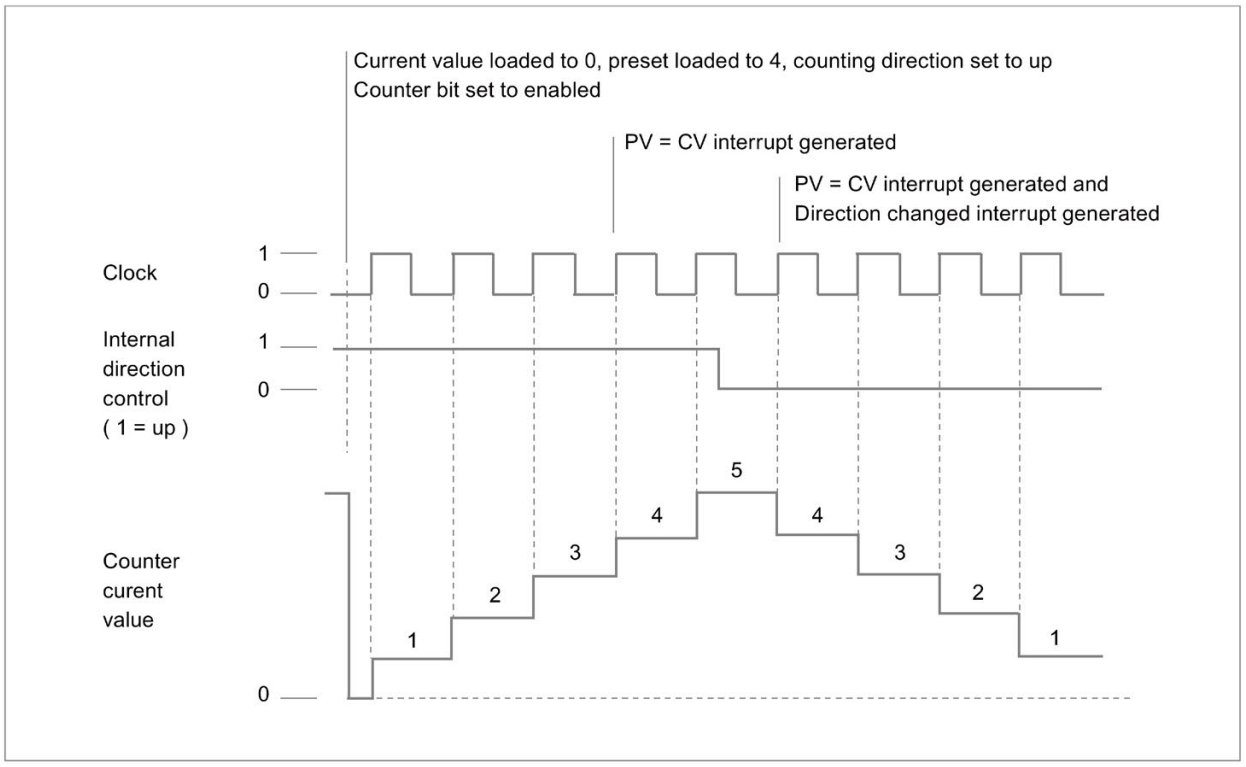
Note

All counting modes of HSC0 always use I0.0 and all counting modes of HSC2 always use I0.2, so you cannot use these inputs for other purposes when these counters are in use.

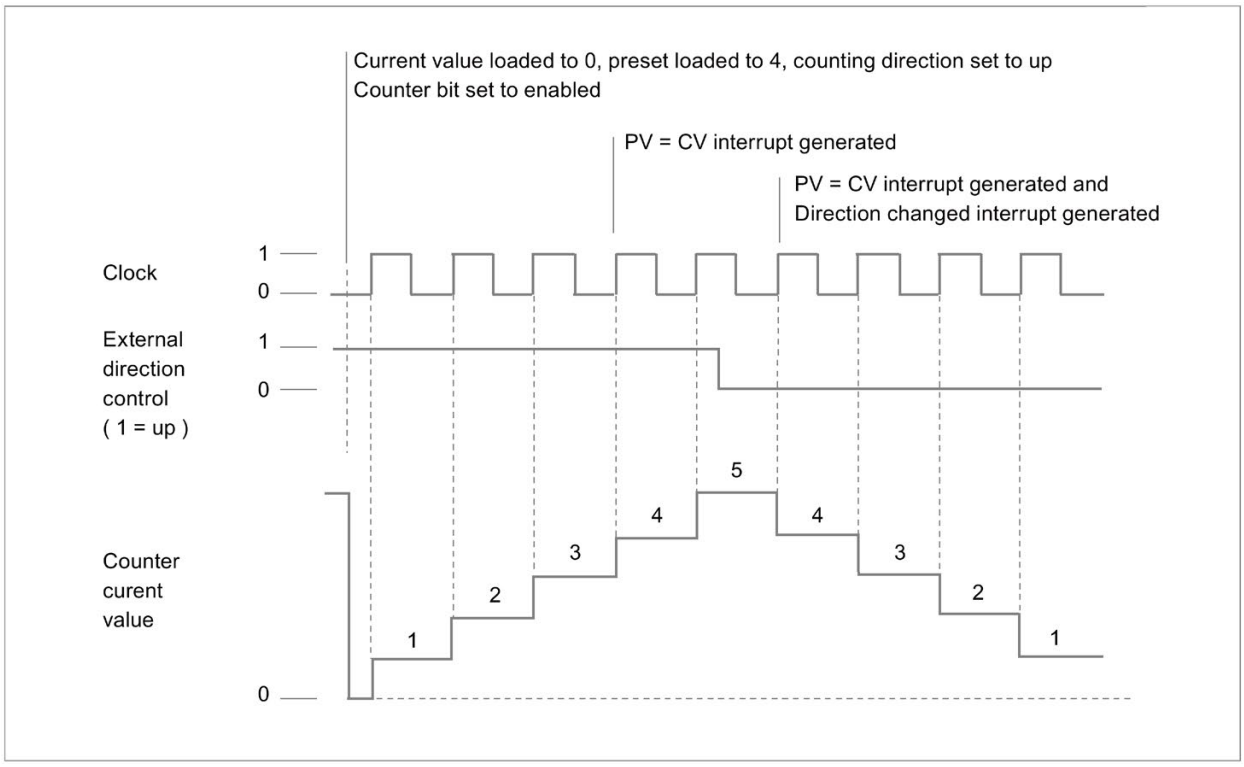
Mode	Description	Input assignment		
	HSC0	I0.0	I0.1	I0.4
	HSC1	I0.1		
	HSC2	I0.2	I0.3	I0.5
	HSC3	I0.3		
	HSC4	I0.6	I0.7	I1.2
	HSC5	I1.0	I1.1	I1.3
0	Single-phase counter with internal direction control	Clock		
1		Clock		Reset
3	Single-phase counter with external direction control	Clock	Direction	
4		Clock	Direction	Reset
6	Two-phase counter with 2 clock inputs	Clock Up	Clock Down	
7		Clock Up	Clock Down	Reset
9	AB quadrature phase counter	Clock A	Clock B	
10		Clock A	Clock B	Reset

How mode selection affects counter operation

HSC modes 0 and 1

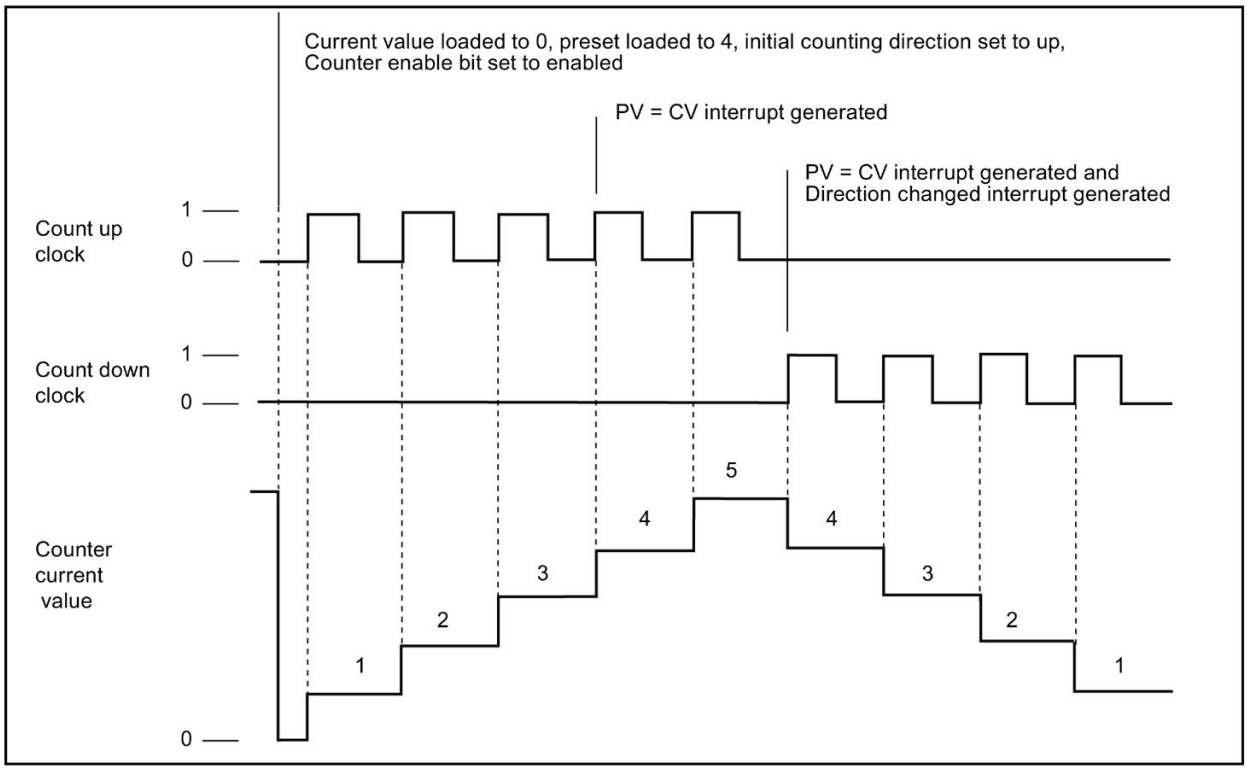


HSC modes 3 and 4

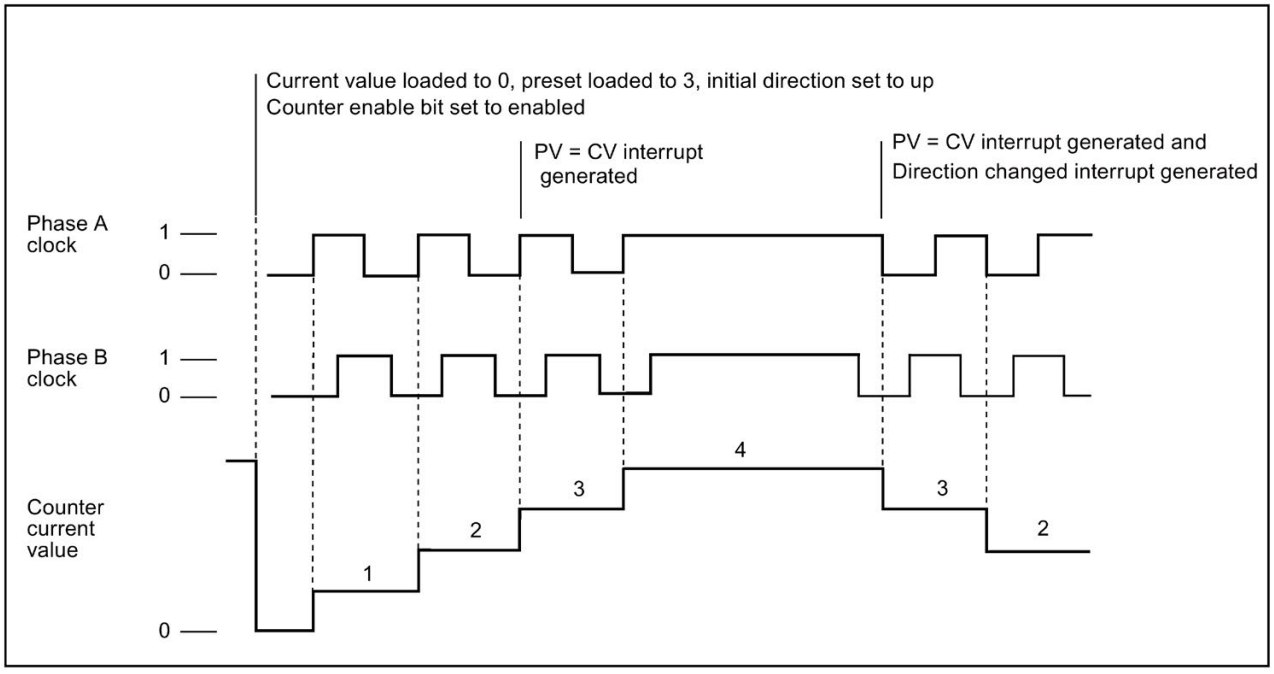


HSC modes 6 and 7

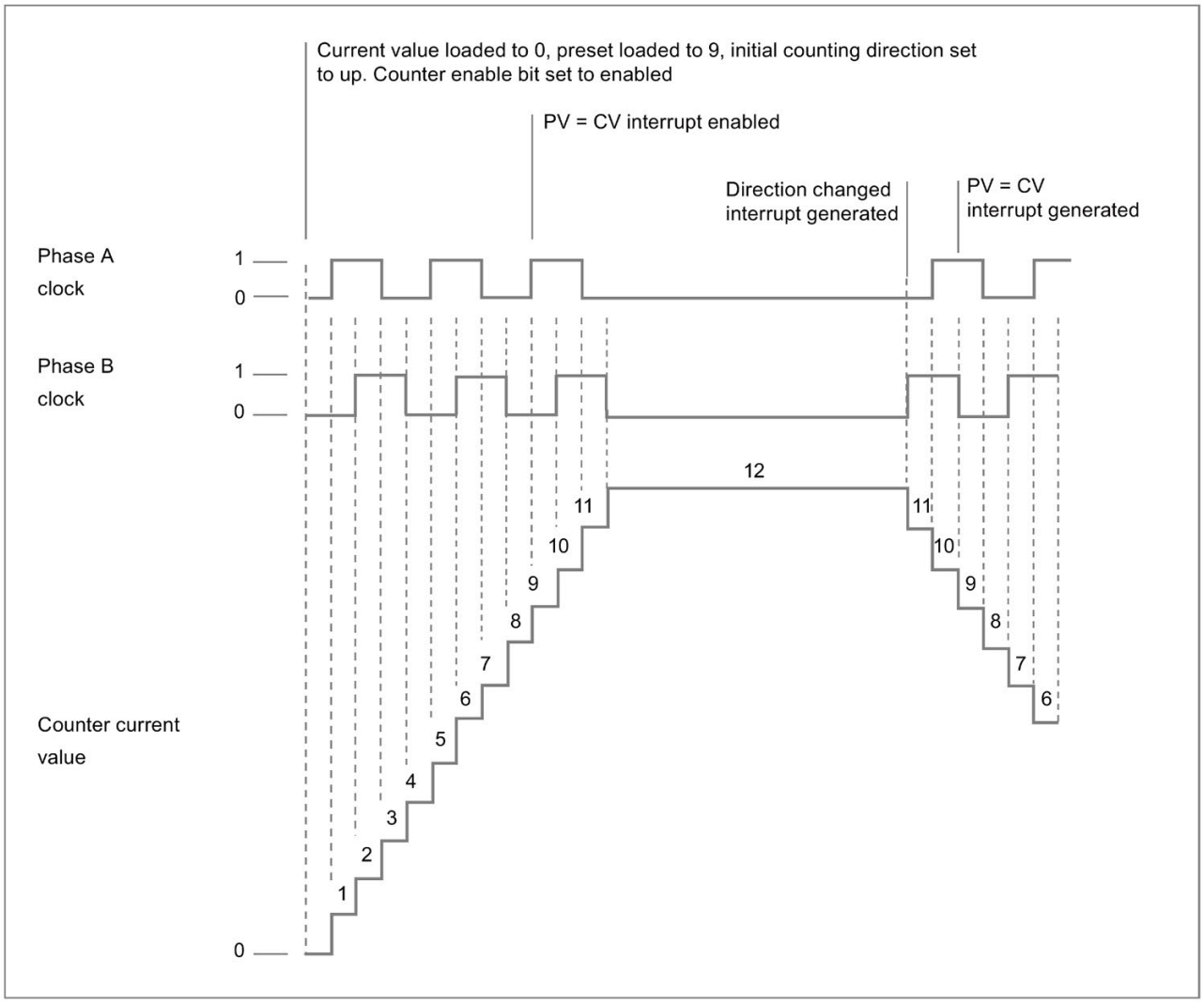
When you use counting modes 6 or 7, and rising edges on both the up clock and down clock inputs occur within 0.3 microseconds of each other, the high-speed counter could see these events as happening simultaneously. If this happens, the current value is unchanged and no change in counting direction is indicated. As long as the separation between rising edges of the up and down clock inputs is greater than this time period, the high-speed counter captures each event separately. In either case, the program generates no error, and the counter maintains the correct count value.



HSC modes 9 and 10 (AB quadrature phase 1x)



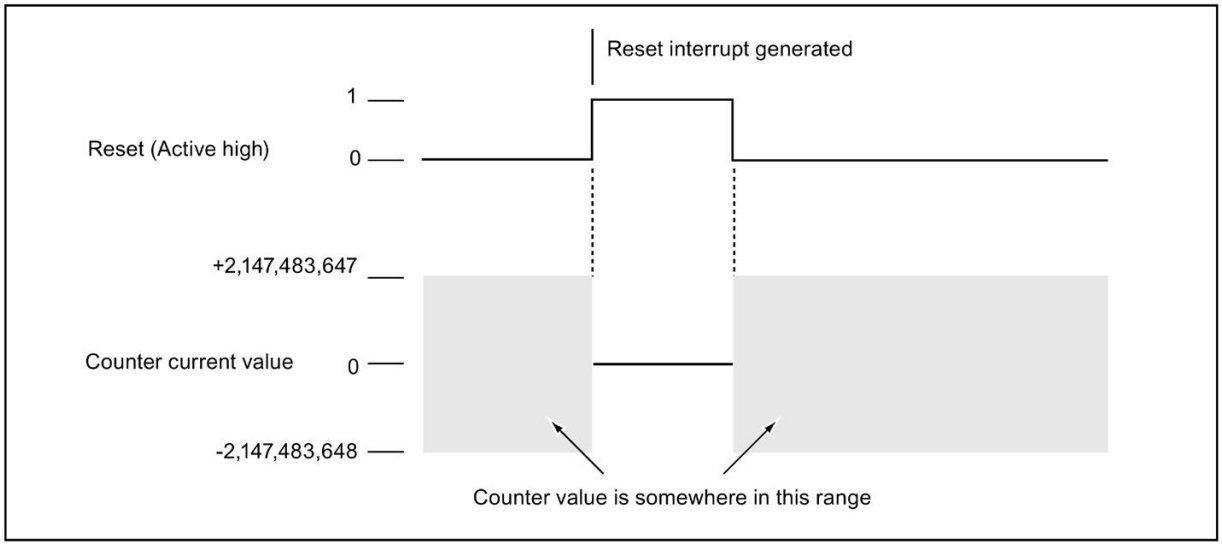
HSC modes 9 and 10 (AB quadrature phase 4x)



Reset operation

The operation of reset shown in the following figure applies to all modes that use the reset input. In the figure below, the reset operation is shown with the active state assigned as the high level.

HSC reset



HDEF instruction sets the reset active level and counting rate

HSC0, HSC2, HSC4, and HSC5 counters have two control bits that are used to configure the active state of the reset and to select 1x or 4x counting modes (AB quadrature phase counters only). These bits are located in the HSC control byte for the respective counter and are only used when the HDEF instruction is executed. These bits are defined in the following table.

Note

You must set these two control bits to the desired state before the HDEF instruction is executed. Otherwise, the counter takes on the default configuration for the counter mode selected.

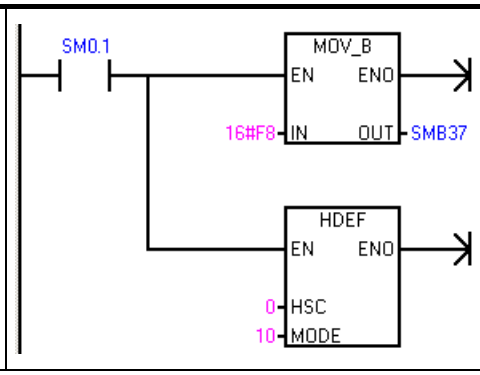
Once the HDEF instruction has been executed, you cannot change the counter setup unless you first place the CPU in STOP mode.

HSC0	HSC1	HSC2	HSC3	HSC4	HSC5	Description (used only when HDEF is executed)
SM37.0	Not supported	SM57.0	Not supported	SM147.0	SM157.0	**Active level control bit for Reset:*** • 0 = Reset is active high • 1 = Reset is active low
SM37.2	Not supported	SM57.2	Not supported	SM147.2	SM157.2	**Counting rate selection for AB quadrature phase counters:*** • 0 = 4X counting rate • 1 = 1X counting rate

* The default setting of the reset input is active high, and the AB quadrature phase counting rate is 4x (or four times the input clock frequency).

Example: High-speed counter definition

LAD			STL

MAIN		On the first scan:
		1. Select the reset input to be active high and select 4x mode.
		2. Configure HSC0 for AB quadrature phase with reset input (mode 10).
			`Network 1`
			`LD SM0.1`
			`MOVB 16#F8, SMB37`
			`HDEF 0, 10`

HSC instruction enables counters, sets counting direction, and loads preset/current count values

The HSC instruction uses the control byte during execution. After you assign the counter and the counter mode, you can program the dynamic parameters of the counter. Each high-speed counter has a control byte in SM memory that allows the following actions:

- Enabling or disabling the counter
- Controlling the direction (modes 0 and 1 only), or the initial counting direction for all other modes
- Loading the current value
- Loading the preset value

HSC Control bytes

HSC0	HSC1	HSC2	HSC3	HSC4	HSC5	Description
SM37.3	SM47.3	SM57.3	SM137.3	SM147.3	SM157.3	Counting direction control bit: • 0 = Count down • 1 =Count up
SM37.4	SM47.4	SM57.4	SM137.4	SM147.4	SM157.4	Write the counting direction to the HSC: • 0 = No update • 1 =Update direction
SM37.5	SM47.5	SM57.5	SM137.5	SM147.5	SM157.5	Write the new preset value to the HSC: • 0 = No update • 1 = Update preset

HSC0	HSC1	HSC2	HSC3	HSC4	HSC5	Description
SM37.6	SM47.6	SM57.6	SM137.6	SM147.6	SM157.6	Write the new current value to the HSC: • 0 = No update • 1 =Update current value
SM37.7	SM47.7	SM57.7	SM137.7	SM147.7	SM157.7	Enable the HSC: • 0 = Disable the HSC • 1 =Enable the HSC

Read the HSC current value with your program

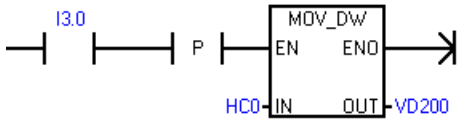
You can only read the current value of each high-speed counter using the data type HC (High-Speed-Counter Current) followed by the counter identifier number (0, 1, 2, 3, 4, or 5) as shown in the following table. Use the HC data type whenever you wish to read the current count, either in a status chart or in the user program. The HC data type is read-only double word value; you cannot write a new current count to the high-speed counter using the HC data type.

Current values of HSC0, HSC1, HSC2, HSC3, HSC4, and HSC5

Value to be read	HSC0 address	HSC1 address	HSC2 address	HSC3 address	HSC4 address	HSC5 address
CV (counter current value)	HC0	HC1	HC2	HC3	HC4	HC5

Example: Reading and saving the current count value

LAD			STL

MAIN		Save the value of HSC0 into VD200 when I3.0 transitions from OFF to ON.	`Network 1`
			`LD I3.0`
			`EU`
			`MOVD HC0, VD200`

Set current values and preset values with your program

Each high-speed counter has a 32-bit current value (CV) and a 32-bit preset value (PV) stored internally. The current value is the actual count value of the counter, while the preset value is a comparison value optionally used to trigger an interrupt when the current value reaches the preset value. You can read the current value using the HC data type as described in the previous section. You cannot read the preset value directly. To load a new current or preset value into the high-speed counter, you must set up the control byte and the special memory double-word(s) that hold the desired new current and/or new preset values, and also execute the HSC instruction to cause the new values to be transferred to the high-speed counter. The table below lists the special memory double-words used to hold the desired new current and preset values.

Use the following steps to write a new current value and/or new preset value to the high-speed counter (steps 1 and 2 can be done in either order):

1. Load the value to be written into the appropriate SM new current value and/or new preset value (see the table below). Loading these new values does not affect the high-speed counter yet.
2. Set or clear the appropriate bits in the appropriate control byte to indicate whether to update the current and/or preset values (bit x.5 for preset and x.6 for current). Manipulating these bits does not affect the high-speed counter yet.
3. Execute the HSC instruction referencing the appropriate high-speed counter number. Executing this instruction causes the control byte to be examined. If the control byte specifies an update for the current, the preset, or both, then the appropriate values are copied from the SM new current value and/or new preset value locations into the high-speed counter internal registers.

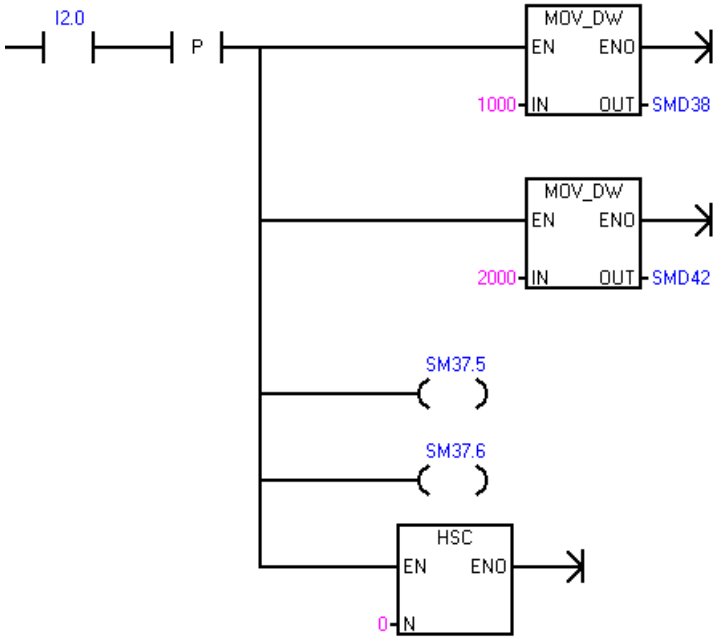
Value to be loaded	HSC0	HSC1	HSC2	HSC3	HSC4	HSC5
New current value (new CV)	SMD38	SMD48	SMD58	SMD138	SMD148	SMD158
New preset value (new PV)	SMD42	SMD52	SMD62	SMD142	SMD152	SMD162

Note

Changes to the control byte and the SM locations for new current value and new preset value does not affect the high-speed counter until the corresponding HSC instruction is executed.

Example: Updating the current and preset values

LAD		STL

MAIN program network
 
	Update the current count to 1000 and the preset value to 2000 for HSC0 when I2.0 transitions from OFF to ON.	`Network 1`
		`LD I2.0`
		`EU`
		`MOVD 1000, SMD38`
		`MOVD 2000, SMD42`
		`= SM37.5`
		`= SM37.6`
		`HSC 0`

Attaching HSC interrupt routines in your program

All high-speed counter modes support an interrupt event when the current value of the HSC is equal to the loaded preset value. Counter modes that use an external reset input support an interrupt on activation of the external reset. All counter modes except modes 0 and 1 support an interrupt on a change in counting direction. Each of these interrupt conditions can be enabled or disabled separately. For a complete discussion on the use of interrupts, see the section about Interrupt instructions (Page 307).

HSC status byte

A status byte for each high-speed counter provides status memory bits that indicate the current counting direction and whether the current value is greater than or equal to the preset value. The following table defines these status bits for each high-speed counter.

Note

Status bits are valid only while the high-speed counter interrupt routine executes. The purpose of monitoring the state of the high-speed counter is to enable interrupts for the events that are of consequence to the operation being performed.

Table 7- 14 Status bits for HSC0, HSC1, HSC2, HSC3, HSC4, and HSC5

HSC0	HSC1	HSC2	HSC3	HSC4	HSC5	Description
SM36.5	SM46.5	SM56.5	SM136.5	SM146.5	SM156.5	**Current counting direction status bit:** • 0 = Counting down • 1 = Counting up
SM36.6	SM46.6	SM56.6	SM136.6	SM146.6	SM156.6	**Current value equals preset value status bit:** • 0 = Not equal • 1 = Equal
SM36.7	SM46.7	SM56.7	SM136.7	SM146.7	SM156.7	**Current value greater than preset value status bit:** • 0 = Less than or equal • 1 = Greater than

Reference information

Refer to the following sections for further information:

- High-speed counter instructions (Page 248)
- High-speed counter summary (Page 251)
- Example initialization sequences for high-speed counters (Page 265)

7.6.6 Example initialization sequences for high-speed counters

HSC0 is used as the counter in the following descriptions of the initialization and operation sequences.

- HSC0, HSC2, HSC4, and HSC5 support counting modes (0, 1), (3, 4), (6, 7), and (9, 10).
- HSC1 and HSC3 only support counting mode 0.

The initialization descriptions assume that you have just placed the CPU in RUN mode, and, for that reason, the first scan memory bit is true. If this is not the case, remember that you can execute the HDEF instruction only one time for each high-speed counter, after entering RUN mode. Executing HDEF for a high-speed counter a second time generates a run-time error and does not change the counter setup from the way it was set up on the first execution of HDEF for that counter.

Note

Although the following sequences show how to change direction, current value, and preset value individually, you can change all or any combination of them in the same sequence by setting the value of SMB37 appropriately and then executing the HSC0 instruction.

Initialization of modes 0 and 1

The following steps describe how to initialize HSC0 for single-phase up/down counter with internal direction (modes 0 and 1):

1. Use the first scan memory bit to call a subroutine in which the initialization operation is performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

   For example: SMB37 = 16#F8 produces the following results:

   – Enables the counter
   – Writes a new current value
   – Writes a new preset value
   – Sets the direction to count up
   – Sets the reset input to be active high
3. Execute the HDEF instruction with the HSC input set to 0 and the MODE input set to one of the following:

   – Mode 0 for no external reset
   – Mode 1 for external reset
4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
5. Load SMD42 (double-word-sized value) with the desired preset value.
6. In order to capture the current value equal to preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section that discusses the Interrupt instructions for complete details on interrupt processing.
7. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
8. Execute the global interrupt enable instruction (ENI) to enable interrupts.
9. Execute the HSC instruction to cause the CPU to program HSC0.
10. Exit the subroutine.

Initialization of modes 3 and 4

The following steps describe how to initialize HSC0 for single-phase up/down counter with external direction control (modes 3 and 4):

1. Use the first scan memory bit to call a subroutine in which the initialization operation is performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

   For example: SMB37 = 16#F8 produces the following results:

   – Enables the counter
   – Writes a new current value
   – Writes a new preset value
   – Sets the initial direction of the HSC to count up
   – Sets the reset input to be active high
3. Execute the HDEF instruction with the HSC input set to 0 and the MODE input set to one of the following:

   – Mode 3 for no external reset
   – Mode 4 for external reset
4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
5. Load SMD42 (double-word-sized value) with the desired preset value.
6. In order to capture the current-value-equal-to-preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section that discusses the Interrupt instructions for complete details on interrupt processing.
7. In order to capture direction changes, program an interrupt by attaching the direction changed interrupt event (event 27) to an interrupt routine.
8. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
9. Execute the global interrupt enable instruction (ENI) to enable interrupts.
10. Execute the HSC instruction to cause the CPU to program HSC0.
11. Exit the subroutine.

Initialization of modes 6 and 7

The following steps describe how to initialize HSC0 for two-phase up/down counter with up/down clocks (modes 6 and 7):

1. Use the first scan memory bit to call a subroutine in which the initialization operations are performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

   For example: SMB37 = 16#F8 produces the following results:

   – Enables the counter
   – Writes a new current value
   – Writes a new preset value
   – Sets the initial direction of the HSC to count up
   – Sets the reset input to be active high
3. Execute the HDEF instruction with the HSC input set to 0 and the MODE set to one of the following:

   – Mode 6 for no external reset
   – Mode 7 for external reset
4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
5. Load SMD42 (double-word-sized value) with the desired preset value.
6. In order to capture the current-value-equal-to-preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section on interrupts.
7. In order to capture direction changes, program an interrupt by attaching the direction changed interrupt event (event 27) to an interrupt routine.
8. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
9. Execute the global interrupt enable instruction (ENI) to enable interrupts.
10. Execute the HSC instruction to cause the CPU to program HSC0.
11. Exit the subroutine.

Initialization of modes 9 and 10

The following steps describe how to initialize HSC0 as an AB quadrature phase counter (modes 9 and 10):

1. Use the first scan memory bit to call a subroutine in which the initialization operations are performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

   Example (1x counting mode): SMB37 = 16#FC produces the following results:

   – Enables the counter
   – Writes a new current value
   – Writes a new preset value
   – Sets the initial direction of the HSC to count up
   – Sets the reset input to be active high

   Example (4x counting mode): SMB37 = 16#F8 produces the following results:

   – Enables the counter
   – Writes a new current value
   – Writes a new preset value
   – Sets the initial direction of the HSC to count up
   – Sets the reset input to be active high
3. Execute the HDEF instruction with the HSC input set to 0 and the MODE input set to one of the following:
   – Mode 9 for no external reset
   – Mode 10 for external reset
4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
5. Load SMD42 (double-word-sized value) with the desired preset value.
6. In order to capture the current-value-equal-to-preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section on enabling interrupts (ENI) for complete details on interrupt processing.
7. In order to capture direction changes, program an interrupt by attaching the direction changed interrupt event (event 27) to an interrupt routine.
8. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
9. Execute the global interrupt enable instruction (ENI) to enable interrupts.
10. Execute the HSC instruction to cause the CPU to program HSC0.
11. Exit the subroutine.

Change direction in modes 0 and 1

The following steps describe how to configure HSC0 for change direction for single-phase counter with internal direction (modes 0 and 1):

1. Load SMB37 to write the desired direction:

   SMB37 = 16#90

   – Enables the counter

   – Sets the direction of the HSC to count down

   SMB37 = 16#98

   – Enables the counter

   – Sets the direction of the HSC to count up

2. Execute the HSC instruction to cause the CPU to program HSC0.

Loading a new current value (any mode)

The following steps describe how to change the counter current value of HSC0 (any mode):

1. Load SMB37 to write the desired current value:

   SMB37 = 16#C0

   – Enables the counter

   – Writes the new current value

2. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).

3. Execute the HSC instruction to cause the CPU to program HSC0.

Loading a new preset value (any mode)

The following steps describe how to change the preset value of HSC0 (any mode):

1. Load SMB37 to write the desired preset value:

   SMB37 = 16#A0

   – Enables the counter

   – Writes the new preset value

2. Load SMD42 (double-word-sized value) with the desired preset value.

3. Execute the HSC instruction to cause the CPU to program HSC0.

Disabling a high-speed counter (any mode)

The following steps describe how to disable the HSC0 high-speed counter (any mode):

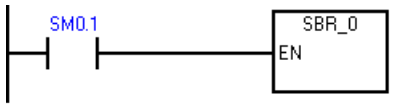
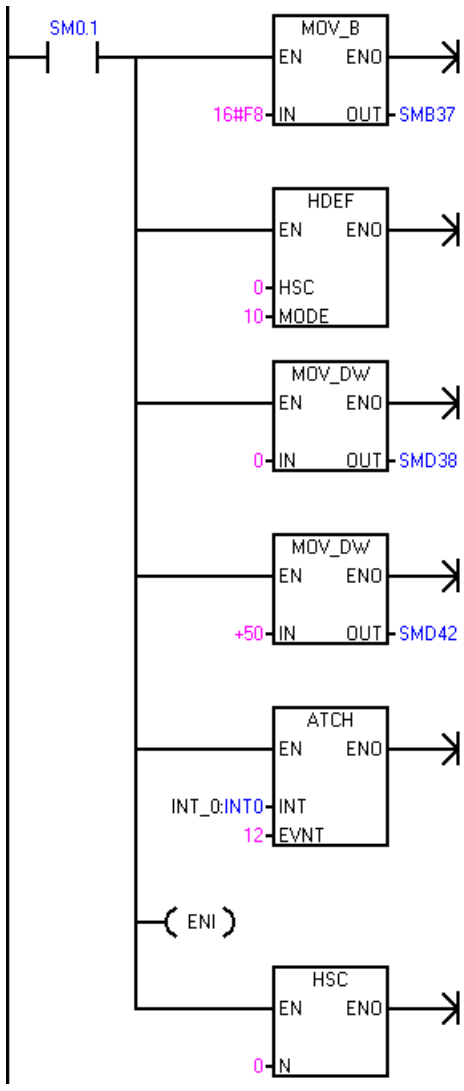
1. Load SMB37 to disable the counter:

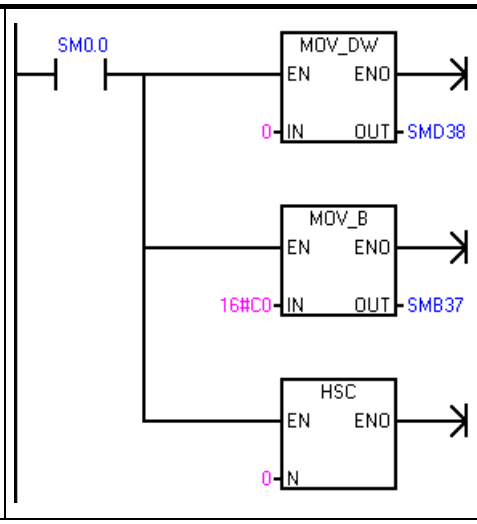
   SMB37 = 16#00

   – Disables the counter

2. Execute the HSC instruction to disable the counter.

Example: high-speed counter instruction

<table>
<tr><th colspan="3">LAD</th><th>STL</th></tr>
<tr><td>MAIN</td><td> 
</td><td>On the first scan, call SBR_0.</td><td>Network 1
LD SM0.1
CALL SBR_0</td></tr>
<tr><td>SBR0</td><td> 
</td><td>On the first scan, configure HSC0:
1. Enable the counter
– Write a new current value.
– Write a new preset value.
– Set the initial direction to count up.
– Select the reset input to be active high.
– Select 4x mode.
2. Configure HSC0 for AB quadrature phase with reset input.
3. Clear the current value of HSC0.
4. Set the HSC0 preset value to 50.
5. Attach event 12 to interrupt routine INT_0. The interrupt is executed when HSC0 current value = preset value.
6. Global interrupt enable
7. Configure HSC0.</td><td>Network 1
LD SM0.1
MOVB 16#F8, SMB37
HDEF 0, 10
MOVD +0, SMD38
MOVD +50, SMD42
ATCH INT_0, 12
ENI
HSC 0</td></tr>
</table>

LAD			STL
INT0		Program HSC0: 1. Clear the current value of HSC0. 2. Select to write only a new current and leave HSC0 enabled. 3. Configure HSC0.	**Network 1** **LD SM0.0** **MOVD +0, SMD38** **MOVB 16#C0, SMB37** **HSC 0**

Reference information

Refer to the following sections for further information:

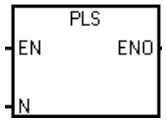
- High-speed counter instructions (Page 248)
- High-speed counter summary (Page 251)
- High-speed counter programming (Page 254)
- Interrupt instructions (Page 307)

7.7 Pulse output

7.7.1 Pulse output instruction (PLS)

The Pulse output (PLS) instruction controls the Pulse train output (PTO) and Pulse width modulation (PWM) functions available on the high-speed outputs (Q0.0, Q0.1, and Q0.3).

When using PWM, you can use an optional wizard to create the PWM instructions.

LAD / FBD	STL	Description
	**PLS N**	You can use the PLS instruction to create up to three PTO or PWM operations. PTO allows the user to control the frequency and number of pulses for a square wave (50% duty cycle) output. PWM allows the user control of a fixed cycle time output with a variable duty cycle.

Error conditions with ENO = 0	SM bits affected
• 0005H: Simultaneous HSC/PLS • 000DH: Attempt to redefine pulse output while it is active • 000EH: Number of PTO profile segments was set to 0 • 0017H: Attempt to assign resource for PTO/PWM that is already assigned to motion control • 001BH: Attempt to change time base on enabled PWM • 0090H: N is not 0, 1, or 2. • 0091H: Range error	None

Input / output	Data type	Operand
N (channel)	WORD	Constant: 0 (= Q0.0), 1 (= Q0.1), or 2 (= Q0.3)

The CPU has three PTO/PWM generators (PLS0, PLS1, and PLS2) that create either a high-speed pulse train or a pulse width modulated waveform. PLS0 is assigned to digital output point Q0.0, PLS1 is assigned to digital output point Q0.1, and PLS2 is assigned to digital output point Q0.3. A designated special memory (SM) location stores the following data for each generator: a PTO status byte (8-bit value), a control byte (8-bit value), a cycle time or frequency (unsigned 16-bit value), a pulse width value (unsigned 16-bit value), and a pulse count value (unsigned 32-bit value).

The PTO/PWM generators and the process image register share the use of Q0.0, Q0.1, and Q0.3. When a PTO or PWM function is active on Q0.0, Q0.1, or Q0.3, the PTO/PWM generator has control of the output, and normal use of the output point is inhibited. The output waveform is not affected by the state of the process image register, the forced value of the point, or the execution of immediate output instructions. When the PTO/PWM generator is inactive, control of the output reverts to the process image register. The process image register determines the initial and final state of the output waveform, causing the waveform to start and end at a high or low level.

Note

PTO/PWM through the PLS instruction is not possible if the selected output point is already configured for use with motion control through use of the Motion wizard.

The PTO/PWM outputs must have a minimum load of at least 10% of rated load to provide crisp transitions from off to on, and from on to off.

Before enabling PTO/PWM operation, set the value of the process image register for Q0.0, Q0.1, and Q0.3 to 0.

Default values for all control bits, cycle time/frequency, pulse width, and pulse count values are 0.

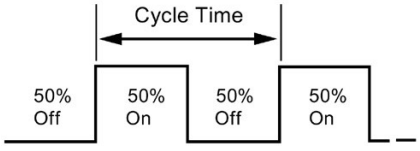
Note

The Pulse Output (PLS) instruction can only be used with the following S7-200 SMART CPUs:

- SR20 / ST20 (Two channels, Q0.0 and Q0.1)
- SR30 / ST30, SR40 / ST40, and SR60 / ST60 (Three channels, Q0.0, Q0.1, and Q0.3)

7.7.2 Pulse train output (PTO)

PTO provides a square wave with a 50% duty cycle output for a specified number of pulses at a specified frequency. Refer to the figure below. PTO can produce either a single train of pulses or multiple trains of pulses using a pulse profile. You specify the number of pulses and the frequency:



- Number of pulses: 1 to 2,147,483,647
- Frequency:
  - 1 to 100,000 Hz (multiple-segment)
  - 1 to 65,535 Hz (single-segment)

Use the following formula to convert from cycle time to frequency:

F = 1 / CT

where:

F	Frequency (Hz)
CT	Cycle time (seconds)

See the following table for pulse count and frequency limitations:

Table 7- 15 Pulse count and frequency in the PTO function

Pulse count / frequency	Reaction
Frequency < 1 Hz	Frequency defaults to 1 Hz
Frequency > 100,000 Hz	Frequency defaults to 100,000 Hz
Pulse count = 0	Pulse count defaults to 1 pulse
Pulse count > 2,147,483,647	Pulse count defaults to 2,147,483,647 pulses

Note

When using a PTO with very short cycle times (high frequencies), you should take into account the switching delay specifications for the output points and how the switching delay can affect the duty cycle. See Appendix A for the digital output switching delay for your CPU.

The PTO function allows the "chaining" or "pipelining" of pulse trains. When the active pulse train is complete, the output of a new pulse train begins immediately. This allows continuity between subsequent output pulse trains.

Single-Segment pipelining of PTO pulses

In single-segment pipelining, you are responsible for updating the SM locations for the next pulse train. After the initial PTO segment has started, you must immediately modify the SM locations with the parameters of the second waveform. After you update the SM values, execute the PLS instruction again. The PTO function holds the attributes of the second pulse train in a pipeline until it completes the first pulse train. The PTO function can store only one entry at a time in the pipeline. When the first pulse train completes, the output of the second waveform begins, and you can store a new pulse train specification in the pipeline. You can then repeat this process to set up the characteristics of the next pulse train. Attempting to load the pipeline while it is still full results in the PTO Overflow bit (SM66.6, SM76.6, or SM566.6) being set and the instruction being ignored.

Smooth transitions between pulse trains occur unless the active pulse train completes before a new pulse train setup is captured by the execution of the PLS instruction.

Note

In single-segment pipelining, the frequency has an upper limit of 65,535 Hz. If a higher frequency is needed (up to 100,000 Hz), multiple-segment pipelining must be used.

Multiple-Segment pipelining of PTO pulses

In multiple-segment pipelining, the S7-200 SMART automatically reads the characteristics of each pulse train segment from a profile table located in V memory. The SM locations used in this mode are the control byte, the status byte, and the starting V memory offset of the profile table (SMW168, SMW178, or SMW578). Execution of the PLS instruction starts the multiple segment operation.

Each segment entry is 12 bytes in length and is composed of a 32 bit starting frequency, a 32 bit ending frequency, and a 32-bit pulse count value. The table below shows the format of the profile table configured in V memory.

The PTO generator automatically increases or decreases the frequency linearly from the starting frequency to the ending frequency. The frequency is increased or decreased by a constant value at a constant rate. Once the number of pulses reaches the specified pulse count, the next PTO segment is loaded. This sequence repeats until it reaches the end of the profile. A segment’s time duration should be greater than 500 microseconds. If the time duration is too small, the CPU may not have enough time to calculate the next PTO segment values. If the next segment cannot be calculated in time, then the PTO pipeline underflow bit (SM66.6, SM76.6, and SM566.6) is set to "1" and the PTO operation terminated.

While the PTO profile is operating, the number of the currently active segment is available in SMB166, SMB176, or SMB576.

Table 7- 16 Profile table format for multiple-segment PTO operation[1]

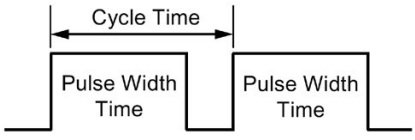
Byte offset	Segment	Description of table entries
0		Number of segments: 1 to 255[2]
1	#1	Starting Frequency (1 to 100,000 Hz)
5		Ending Frequency (1 to 100,000 Hz)
9		Pulse count (1 to 2,147,483,647)
13	#2	Starting Frequency (1 to 100,000 Hz)
17		Ending Frequency (1 to 100,000 Hz)
21		Pulse count (1 to 2,147,483,647)
(Continues)	#3	(Continues)

[1] Entering a profile offset and number of segments that places any part of the profile table outside of V memory generates a non-fatal error. The PTO function does not generate a PTO output.

[2] Entering a value of 0 for the number of segments generates a non-fatal error. No PTO output is generated.

7.7.3 Pulse width modulation (PWM)

PWM provides three channels that allow a fixed cycle time output with a variable duty cycle. Refer to the figure below. You can specify the cycle time and the pulse width in either microsecond or millisecond increments:



- Cycle time: 10 µs to 65,535 µs or 2 ms to 65,535 ms
- Pulse width time: 0 µs to 65,535 µs or 0 ms to 65,535 ms

As shown in the following table, setting the pulse width equal to the cycle time (which makes the duty cycle 100 percent) turns the output on continuously. Setting the pulse width to 0 (which makes the duty cycle 0 percent) turns the output off.

Note

When using a PWM with very short cycle times, you should take into account the switching delay specifications for the output points and how the switching delay can affect the pulse width time. See Appendix A for the digital output switching delay for your CPU.

Pulse width time and cycle time and reactions in the PWM function

Pulse width time / cycle time	Reaction
Pulse width time >= Cycle time value	The duty cycle is 100%: the output is turned on continuously.
Pulse width time = 0	The duty cycle is 0%: the output is turned off continuously.
Cycle time < 2 time units	The cycle time defaults to two time units.

Changing the characteristics of a PWM waveform

You can only use synchronous updates to change the characteristics of a PWM waveform. With a synchronous update, the change in the waveform characteristics occurs on a cycle boundary, providing a smooth transition.

7.7.4 Using SM locations to configure and control the PTO/PWM operation

The PLS instruction reads the data stored in the specified SM memory locations and programs the PTO/PWM generator accordingly. SMB67 controls PTO0 or PWM0, SMB77 controls PTO1 or PWM1, and SMB567 controls PTO2 or PWM2. The "SM locations for the PTO/PWM control registers" table (the first table below) describes the registers used to control the PTO/PWM operation. You can use the "PTO/PWM control byte reference" table (the second table below) as a quick reference to determine the value to place in the PTO/PWM control register to invoke the desired operation.

You can change the characteristics of a PTO or PWM waveform by modifying the locations in the SM area (including the control byte) and then executing the PLS instruction. You can disable the generation of a PTO or PWM waveform at any time by writing 0 to the PTO/PWM enable bit of the control byte (SM67.7, SM77.7, or SM567.7) and then executing the PLS instruction. The output point immediately reverts back to process image register control.

If you disable the PTO or PWM operation while the operation is producing a pulse, that pulse internally completes its full cycle time duration. However, the pulse is not present at the output point because, at that time, the process image register regains control of the output. Your program can enable the pulse generator again with no time delay as long as the following is true: the pulse mode (PTO or PWM) being enabled is the same mode that was disabled. An error may occur if your program first disables a PTO and then enables a PWM on the same output channel or if your program first disables a PWM and then enables a PTO.

The PTO Idle bit in the status byte (SM66.7, SM76.7, or SM566.4) is provided to indicate the completion of the programmed pulse train. In addition, an interrupt routine can be invoked upon the completion of a pulse train. (Refer to the descriptions of the Interrupt instructions (Page 307).) If you are using the single segment operation, the interrupt routine is invoked upon the completion of each PTO. For example, if a second PTO is loaded into the pipeline, the PTO function invokes the interrupt routine upon the completion of the first PTO, and again upon the completion of the second PTO that was loaded into the pipeline. When using the multiple segment operation, the PTO function invokes the interrupt routine upon completion of the profile table.

The following conditions set the bits of the status byte (SMB66, SMB76, and SMB566):

- If an "Add Error" occurs in the pulse generator that results in an invalid frequency value, the PTO function terminates and the Delta Calculation Error bit (SM66.4, SM76.4, or SM566.4) is set to 1. The output reverts to image register control. To correct this issue, try adjusting the PTO profile parameters.
- Manually disabling a PTO profile in progress sets the PTO Profile Disabled bit (SM66.5, SM76.5, or SM566.5) to 1.
- The PTO/PWM overflow/underflow bit (SM66.6, SM76.6, or SM566.6) is set to 1 if either of these situations occur:
  - An attempt is made to load the pipeline while it is full; this is an overflow condition.
  - A PTO profile segment is too short to allow the CPU to calculate the next segment, and an empty pipeline is transferred; this is an underflow condition, and the output reverts to image register control.
- You must clear the PTO/PWM overflow/underflow bit manually after it is set to detect subsequent overflows. The transition to RUN mode initializes this bit to 0.

Note

- Ensure that you understand the definition of the PTO/PWM mode select bit (SM67.6, SM77.6, and SM567.6). The bit definition may not be the same as some legacy products that support a Pulse instruction. In the S7-200 SMART, the user selects PTO or PWM mode with the following definition: 0 = PWM, 1 = PTO.
- When you load a cycle time/frequency (SMW68, SMW78, or SMW568), pulse width (SMW70, SMW80, or SMW570), or pulse count (SMD72, SMD82, or SMD572), also set the appropriate update bits in the control register before you execute the PLS instruction.
- For a multiple segment pulse train operation, you must also load the starting offset (SMW168, SMW178, or SMW578) of the profile table and the profile table values before you execute the PLS instruction.
- If you attempt to change the time base of a PWM output while the PWM is executing, the request is ignored and a non-fatal error (0x001B - ILLEGAL PWM TIMEBASE CHG) is created.

Table 7- 17 SM locations for the PTO/PWM control registers

Q0.0	Q0.1	Q0.3	Status bits
SM66.4	SM76.4	SM566.4	PTO delta calculation error (due to an add error): • 0 = No error • 1 = Aborted due to error
SM66.5	SM76.5	SM566.5	PTO profile disabled (due to user command): • 0 = Profile not manually disabled • 1 = User disabled profile

SM66.6	SM76.6	SM566.6	PTO/PWM pipeline overflow/underflow: • 0 = No overflow/underflow • 1 = Overflow/underflow
SM66.7	SM76.7	SM566.7	PTO idle: • 0 = In progress • 1 = PTO idle
Q0.0	**Q0.1**	**Q0.3**	**Control bits**
SM67.0	SM77.0	SM567.0	PTO/PWM update the frequency/cycle time: • 0 = No update • 1 = Update frequency/cycle time
SM67.1	SM77.1	SM567.1	PWM update the pulse width time: • 0 = No update • 1 = Update pulse width
SM67.2	SM77.2	SM567.2	PTO update the pulse count value: • 0 = No update • 1 = Update pulse count
SM67.3	SM77.3	SM567.3	PWM time base: • 0 = 1 µs/tick • 1 = 1 ms/tick
SM67.4	SM77.4	SM567.4	Reserved
SM67.5	SM77.5	SM567.5	PTO single/multiple segment operation: • 0 = Single • 1 = Multiple
SM67.6	SM77.6	SM567.6	PTO/PWM mode select: • 0 = PWM • 1 = PTO
SM67.7	SM77.7	SM567.7	PWM enable: • 0 = Disable • 1 = Enable
Q0.0	**Q0.1**	**Q0.3**	**Other registers**
SMW68	SMW78	SMW568	PTO frequency or PWM cycle time value: 1 to 65,535 Hz (PTO); 2 to 65,535 (PWM)
SMW70	SMW80	SMW570	PWM pulse width value: 0 to 65,535
SMD72	SMD82	SMD572	PTO pulse count value: 1 to 2,147,483,647
SMB166	SMB176	SMB576	Number of the segment in progress: Multiple-segment PTO operation only
SMW168	SMW178	SMW578	Starting location of the profile table (byte offset from V0): Multiple-segment PTO operation only

Table 7- 18 PTO/PWM control byte reference

		Result of executing the PLS instruction					
Control register (Hex value)	**Enable**	**Select mode**	**PTO Segment operation**	**Time base**	**Pulse count**	**Pulse width**	**Cycle time / frequency**
16#80	Yes	PWM		1 µs/cycle			
16#81	Yes	PWM		1 µs/cycle			Update cycle time
16#82	Yes	PWM		1 µs/cycle		Update	
16#83	Yes	PWM		1 µs/cycle		Update	Update cycle time
16#88	Yes	PWM		1 ms/cycle			
16#89	Yes	PWM		1 ms/cycle			Update cycle time
16#8A	Yes	PWM		1 ms/cycle		Update	
16#8B	Yes	PWM		1 ms/cycle		Update	Update cycle time
16#C0	Yes	PTO	Single				
16#C1	Yes	PTO	Single				Update frequency
16#C4	Yes	PTO	Single		Update		
16#C5	Yes	PTO	Single		Update		Update frequency
16#E0	Yes	PTO	Multiple				

7.7.5 Calculating the profile table values

The multiple-segment pipelining capability of the PTO generators can be useful in many applications, particularly in stepper motor control.

For example, you can use PTO with a pulse profile to control a stepper motor through a simple ramp up (acceleration), run (no acceleration), and ramp down (deceleration) sequence. More complicated sequences can be created by defining a pulse profile that consists of up to 255 segments, with each segment corresponding to a ramp up, run, or ramp down operation.

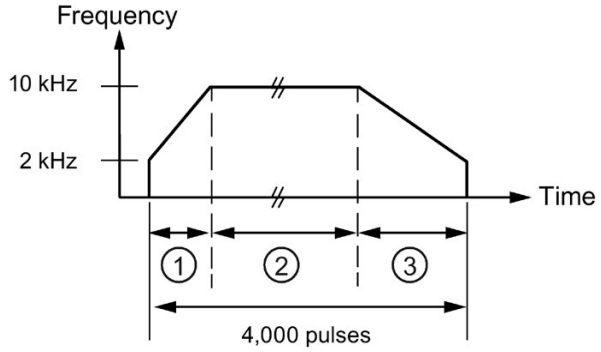
The figure below illustrates sample profile table values required to generate an output waveform:

- Segment 1: Accelerates a stepper motor
- Segment 2: Operates the motor at a constant speed
- Segment 3: Decelerates the motor

To achieve the desired number of motor revolutions for this example, the PTO generator requires the following values:

- Starting and final pulse frequencies of 2 kHz
- A maximum pulse frequency of 10 kHz
- 4000 pulses

During the acceleration portion of the output profile, the output wave form should reach maximum pulse frequency in approximately 200 pulses. The output wave form should complete the deceleration portion of the profile in approximately 400 pulses.



① Segment 1: 200 pulses
② Segment 2: 3400 pulses
③ Segment 3: 400 pulses

The following table lists the values for generating the example waveform. The profile table, for this example, is in V memory and starts at VB500. You can use any block of V memory that is available for a PTO profile table. You can include instructions in your program to load these values into V memory, or you can define the values of the profile in the data block.

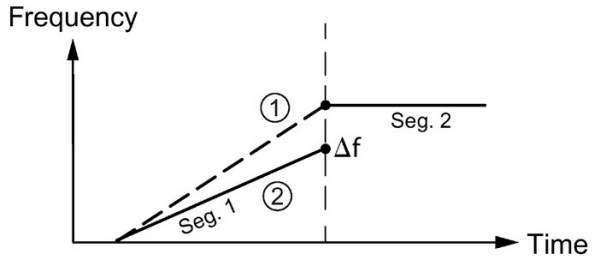
Table 7- 19 Profile table values

Address	Value	Description	
VB500	3	Total number of segments	
VD501	2,000	Starting frequency (Hz)	Segment 1
VD505	10,000	Ending frequency (Hz)	
VD509	200	Number of pulses	
VD513	10,000	Starting frequency (Hz)	Segment 2
VD517	10,000	Ending frequency (Hz)	
VD521	3,400	Number of pulses	
VD525	10,000	Starting frequency (Hz)	Segment 3
VD529	2,000	Ending frequency (Hz)	
VD533	400	Number of pulses	

The PTO generator begins by running Segment 1. After the PTO generator reaches the required number of pulses for Segment 1, it automatically loads Segment 2. This continues until the last segment. After the number of pulses for the last segment is reached, the S7-200 SMART CPU disables the PTO generator.

For each segment of the PTO profile, the pulse train begins at the starting frequency assigned in the table. The PTO generator increases or decreases the frequency at a constant rate to achieve the ending frequency with the correct number of pulses. However, the PTO generator limits the frequency to the starting and ending frequencies specified in the table.

The PTO generator performs repeated additions to the working frequency to create a linear change in frequency over time. The constant value added to the frequency has a limited resolution. This limited resolution can introduce some truncation error into the resulting frequency. Thus, the PTO generator does not guarantee that the pulse train frequency can reach the ending frequency that was specified for that segment. In the figure below, you can see that the truncation error affects the accelerating PTO frequency. The output should be measured to verify that the frequency is within an acceptable frequency range.



① Desired frequency plot

② Actual frequency plot

If the frequency difference (Δf) between the end of a segment and the beginning of the next is not acceptable, try adjusting the ending frequency to compensate for the difference. This adjustment might be an iterative process to get the output within an acceptable frequency range.

Note that changes in segment parameters affect the time it takes the PTO to complete. You can use the equation for the time duration of the segment, found later in this section of the manual, to see what effect this has on the timing. An accurate segment duration time can require some flexibility in the value of the ending frequency or the number of pulses for a given segment.

While the simplified example above is useful as an introduction, real applications can require more complicated waveform profiles. Remember that you can only assign frequencies as an integer number of Hz and perform the frequency modification at a constant rate. The S7-200 SMART CPU selects that constant rate and that rate can be different for each segment.

For legacy projects that were developed in terms of cycle time, instead of frequency, you can use the following formulas to convert to frequency:

$$CT_{Final} = CT_{Initial} + (\Delta CT * PC)$$

$$F_{Initial} = 1 / CT_{Initial}$$

$$F_{Final} = 1 / CT_{Final}$$

where:

$CT_{Initial}$	Starting cycle time (s) for this segment
ΔCT	Delta cycle time (s) of this segment

PC	Quantity of pulses in this segment
CT_{Final}	Ending cycle time (s) for this segment
$F_{Initial}$	Starting frequency (Hz) for this segment
F_{Final}	Ending frequency (Hz) for this segment

The acceleration (or deceleration) and time duration of a given PTO profile segment can be useful in the process of determining correct profile table values. Use the following formulas to calculate the length of time, as well as the acceleration for a given profile:

$\Delta F = F_{Final} - F_{Initial}$

$T_s = PC / (F_{min} + (| \Delta F | / 2))$

$A_s = \Delta F / T_s$

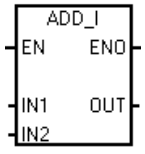
where:

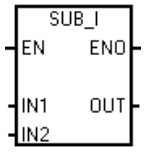
T_s	Time duration (s) of this segment
A_s	Frequency acceleration (Hz/s) of this segment
PC	Quantity of pulses in this segment
F_{min}	Minimum frequency (Hz) for this segment
ΔF	Delta (total change in) frequency (Hz) for this segment

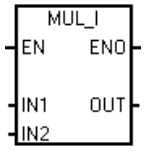
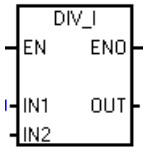
7.8 Math

7.8.1 Add, subtract, multiply, and divide

LAD / FBD	STL	Description


ADD_DI
ADD_R
	`+I IN1, OUT`
	`+D IN1, OUT`
	`+R IN1, OUT`
		The Add Integer instruction adds two 16-bit integers to produce a 16-bit result. The Add Double Integer instruction adds two 32-bit integers to produce a 32-bit result. The Add Real (+R) instruction adds two 32-bit real numbers to produce a 32-bit real number result.
		• LAD and FBD: IN1 + IN2 = OUT
		• STL: IN1 + OUT = OUT


SUB_DI
SUB_R
	`-I IN1, OUT`
	`-D IN1, OUT`
	`-R IN1, OUT`
		The Subtract Integer instruction subtracts two 16-bit integers to produce a 16-bit result. The Subtract Double Integer (-D) instruction subtracts two 32-bit integers to produce a 32-bit result. The Subtract Real (-R) instruction subtracts two 32-bit real numbers to produce a 32-bit real number result.
		• LAD and FBD: IN1 - IN2 = OUT
		• STL: OUT - IN1 = OUT

LAD / FBD	STL	Description
 MUL_DI MUL_R	***I IN1, OUT** ***D IN1, OUT** ***R IN1, OUT**	The Multiply Integer instruction multiplies two 16-bit integers to produce a 16-bit result. The Multiply Double Integer instruction multiplies two 32-bit integers to produce a 32-bit result. The Multiply Real instruction multiplies two 32-bit real numbers to produce a 32-bit real number result. • LAD and FBD: IN1 * IN2 = OUT • STL: IN1 * OUT = OUT
 DIV_DI DIV_R	**/I IN1, OUT** **/D IN1, OUT** **/R IN1, OUT**	The Divide Integer instruction divides two 16-bit integers to produce a 16-bit result. (No remainder is kept.) Divide Double Integer instruction divides two 32-bit integers to produce a 32-bit result. (No remainder is kept.) The Divide Real (/R) instruction divides two 32-bit real numbers to produce a 32-bit real number result. • LAD and FBD: IN1 / IN2 = OUT • STL: OUT / IN1 = OUT

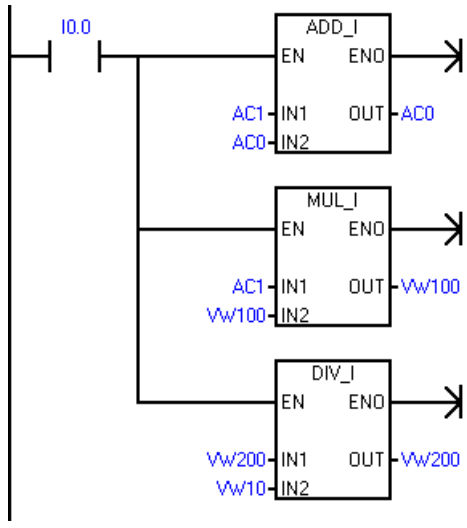
Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • SM1.1 Overflow • SM1.3 Divide by zero	• SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result • SM1.3 Divide by zero

SM1.1 indicates overflow errors and illegal values. If SM1.1 is set, then the status of SM1.0 and SM1.2 is not valid and the original input operands are not altered. If SM1.1 and SM1.3 are not set, then the math operation has completed with a valid result and SM1.0 and SM1.2 contain valid status. If SM1.3 is set during a divide operation, then the other math status bits are left unchanged.

Input / output	Data Type	Operand
IN1, IN2	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *AC, *LD, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
	REAL[1]	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	INT	IW, QW, VW, MW, SMW, SW, LW, T, C, AC, *VD, *AC, *LD
	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

[1] Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

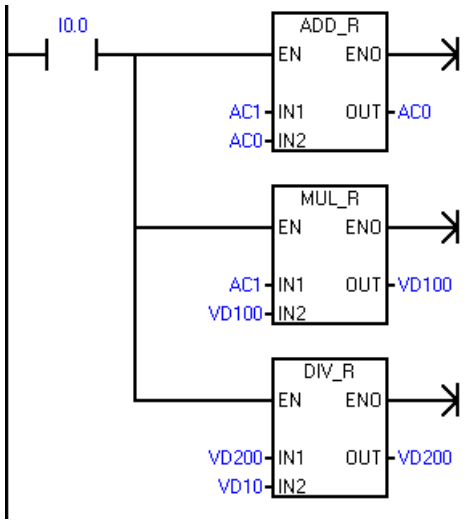
Example: Integer math instructions

LAD	STL
	**Network** **LD I0.0** **+I AC1, AC0** ***I AC1, VW100** **/I VW10, VW200**

Integer operations from the LAD example

	IN1		IN2		OUT
Add data	40	+	60	=	100
Data address	AC1		AC0		AC0
Multiply data	40	*	20	=	800
Data address	AC1		VW100		VW100
Divide data	4000	/	40	=	100
Data address	W200		VW10		VW200

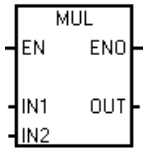
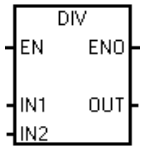
Example: Real math instructions

LAD	STL
	**Network 1** **LD I0.0** **+R AC1, AC0** ***R AC1, VD100** **/R VD10, VD200**

Real number operations from the LAD example

	IN1		IN2		OUT
Add data	4000.0	+	6000.0	=	10000.0
Data address	AC1		AC0		AC0
Multiply data	400.0	*	200.0	=	80000.0
Data address	AC1		VD100		VD100
Divide data	4000.0	/	41.0	=	97.5609
Data address	VD200		VD10		VD200

7.8.2 Multiply integer to double integer and divide integer with remainder

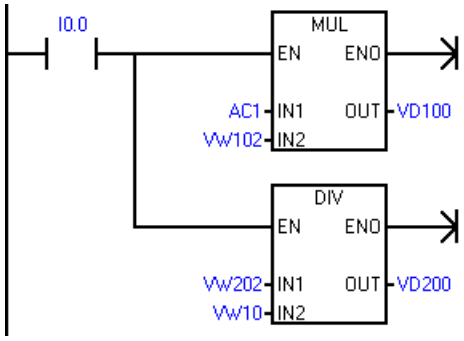
LAD / FBD	STL	Description
	`MUL IN1, OUT`	The multiply integer to double Integer instruction multiplies two 16-bit integers and produces a 32-bit product. In STL, the least-significant word (16 bits) of the 32-bit OUT is used as one of the factors. • LAD and FBD: IN1 * IN2 = OUT • STL: IN1 * OUT = OUT
	`DIV IN1, OUT`	The divide integer with remainder instruction divides two 16-bit integers and produces a 32-bit result consisting of a 16-bit remainder (the most-significant word) and a 16-bit quotient (the least-significant word). In STL, the least-significant word (16 bits) of the 32-bit OUT is used as the dividend. • LAD and FBD: IN1 / IN2 = OUT • STL: OUT / IN1 = OUT

Non-fatal errors with ENO=0	SM bits affected [1]
• 0006H Indirect address • SM1.1 Overflow • SM1.3 Divide by zero	• SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result • SM1.3 Divide by zero

[1] For both of these instructions, SM bits indicate errors and illegal values. If SM1.3 (divide by zero) is set during a divide operation, then the other math status bits are left unchanged. Otherwise, all supported math status bits contain valid status upon completion of the math operation.

Input / output	Data type	Operand
IN1, IN2	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
OUT	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: MUL and DIV instructions

LAD	STL
	**Network 1** **LD I0.0** **MUL AC1, VD100** **DIV VW10, VD200**

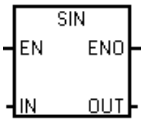
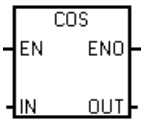
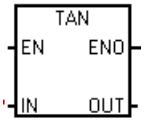
[1] VD100 contains: VW100 and VW102, and VD200 contains: VW200 and VW202.

Real number operations from the LAD example

	IN1		IN2		OUT	
MUL data	400	*	200	=	80000	
Data address	AC1		VW102		VD100	
					remainder	quotient
DIV data	4000	/	41	=	23	97
Data address	VW202		VW10		VW200	VW202 VD200

7.8.3 Trigonometry, natural logarithm/exponential, and square root

Sine (SIN), Cosine (COS), and Tangent (TAN) instructions

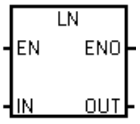
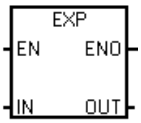
LAD / FBD	STL	Description
	**SIN IN, OUT**	The sine (SIN), cosine (COS), and tangent (TAN) instructions evaluate the trigonometric function of the angle value IN and place the result in OUT. The input angle value is measured in radians. • SIN (IN) = OUT • COS (IN) = OUT • TAN (IN) = OUT To convert an angle from degrees to radians: Use the MUL_R (*R) instruction to multiply the angle in degrees by 1.745329E-2 (approximately by π/180). For the numeric functions instructions, SM1.1 is used to indicate overflow errors and illegal values. If SM1.1 is set, then the status of SM1.0 and SM1.2 is not valid and the original input operands are not altered. If SM1.1 is not set, then the math operation has completed with a valid result and SM1.0 and SM1.2 contain valid status.
	**COS IN, OUT**	
	**TAN IN, OUT**	

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • SM1.1 Overflow	• SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / outputs	Data type	Operand
IN	REAL[1]	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	REAL[1]	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

[1] Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

Natural logarithm (LN) and natural exponential (EXP) instructions

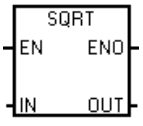
LAD / FBD	STL	Description
	**LN IN, OUT**	The Natural Logarithm instruction (LN) performs the natural logarithm of the value in IN and places the result in OUT. The Natural Exponential instruction (EXP) performs the exponential operation of e raised to the power of the value in IN and places the result in OUT. • LN (IN) = OUT • EXP (IN)= OUT To obtain the base 10 logarithm from the natural logarithm: Divide the natural logarithm by 2.302585 (approximately the natural logarithm of 10). To raise any real number to the power of another real number, including fractional exponents: Combine the Natural Exponential instruction with the Natural Logarithm instruction. For example, to raise X to the Y power, use EXP (Y * LN (X)).
	**EXP IN, OUT**	

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • SM1.1 Overflow	• SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / outputs	Data type	Operand
IN	REAL[1]	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	REAL[1]	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

[1] Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

Square root (SQRT) instruction

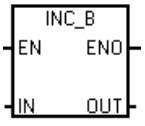
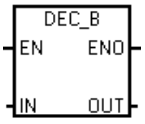
LAD / FBD	STL	Description
	**SQRT IN, OUT**	The Square Root instruction (SQRT) takes the square root of a real number (IN) and produces a real number result OUT. • SQRT (IN)= OUT To obtain other roots: • 5 cubed = 5^3 = EXP(3*LN(5)) = 125 • The cube root of 125 = 125^(1/3) = EXP((1/3)*LN(125))= 5 • The square root of 5 cubed = 5^(3/2) = EXP(3/2*LN(5)) = 11.18034

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • SM1.1 Overflow	• SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / outputs	Data type	Operand
IN	REAL[1]	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	REAL[1]	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

[1] Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

7.8.4 Increment and decrement

LAD / FBD	STL	Description
 INC_W INC_DW	**INCB OUT** **INCW OUT** **INCD OUT**	The increment instructions add 1 to the input value IN and place the result into the location at OUT. • LAD and FBD: IN + 1 = OUT • STL: OUT + 1 = OUT Increment byte (INC_B) operations are unsigned. Increment word (INC_W) operations are signed. Increment double word (INC_DW) operations are signed.
 DEC_W DEC_DW	**DECB OUT** **DECW OUT** **DECD OUT**	The decrement instructions subtract 1 from the input value IN and place the result into the location at OUT. • LAD and FBD: IN - 1 = OUT • STL: OUT - 1= OUT Decrement byte (DEC_B) operations are unsigned. Decrement Word (DEC_W) operations are signed. Decrement double Word (DEC_D) operations are signed.

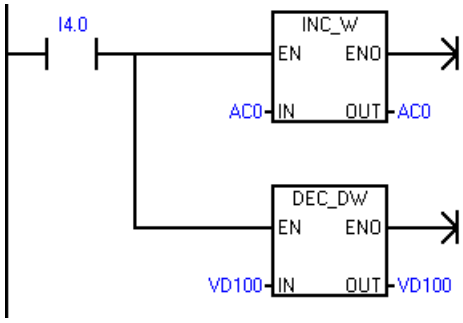
Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • SM1.1 Overflow	• SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant

Input / output	Data type	Operand
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC,*VD, *LD, *AC
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: Increment and decrement

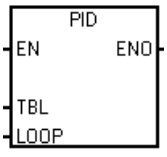
LAD	STL

	`Network 1`
	`LD I4.0`
	`INCW AC0`
	`DECD VD100`

Increment/decrement operations from the LAD example

	IN				OUT
Increment word	125	+	1	=	126
Data address	AC0				AC0
Decrement double word	128000	-	1	=	127999
Data address	VD100				VD100

7.9 PID

LAD / FBD	STL	Description
	`PID TBL, LOOP`	The PID loop instruction (PID) executes a PID loop calculation on the referenced LOOP based upon the input and configuration information in Table (TBL).

Non-fatal errors with ENO = 0	SM bits affected
• 0013H Illegal PID loop table	• SM1.1 Overflow

Input / output	Data type	Operand
TBL	BYTE	VB
LOOP	BYTE	Constant (0 to 7)

The PID loop instruction (Proportional, Integral, Derivative Loop) is provided to perform the PID calculation. The top of the logic stack (TOS) must be ON (power flow) to enable the PID calculation. The instruction has two operands: a TABLE address which is the starting address of the loop table and a LOOP number which is a constant from 0 to 7.

Eight PID instructions can be used in a program. If two or more PID instructions are used with the same loop number (even if they have different table addresses), the PID calculations will interfere with one another and the output will be unpredictable.

The loop table stores nine parameters used for controlling and monitoring the loop operation and includes the current and previous value of the process variable, the setpoint, output, gain, sample time, integral time (reset), derivative time (rate), and the integral sum (bias).

To perform the PID calculation at the desired sample rate, the PID instruction must be executed either from within a timed interrupt routine or from within the main program at a rate controlled by a timer. The sample time must be supplied as an input to the PID instruction via the loop table.

Auto-Tune capability has been incorporated into the PID instruction. Refer to "PID loops and tuning" (Page 661) for a detailed description of auto-tuning. The "PID Tune control panel" (Page 669) only works with PID loops created by the PID wizard.

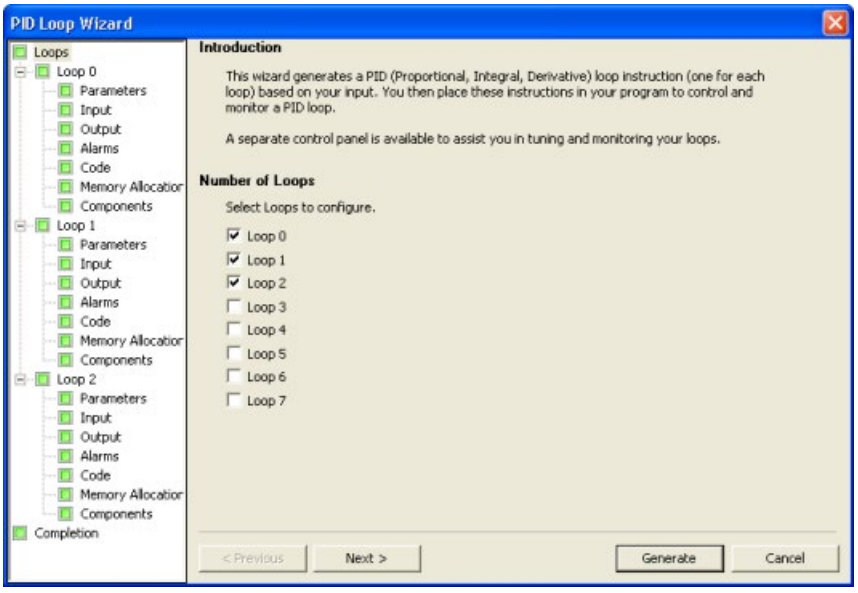
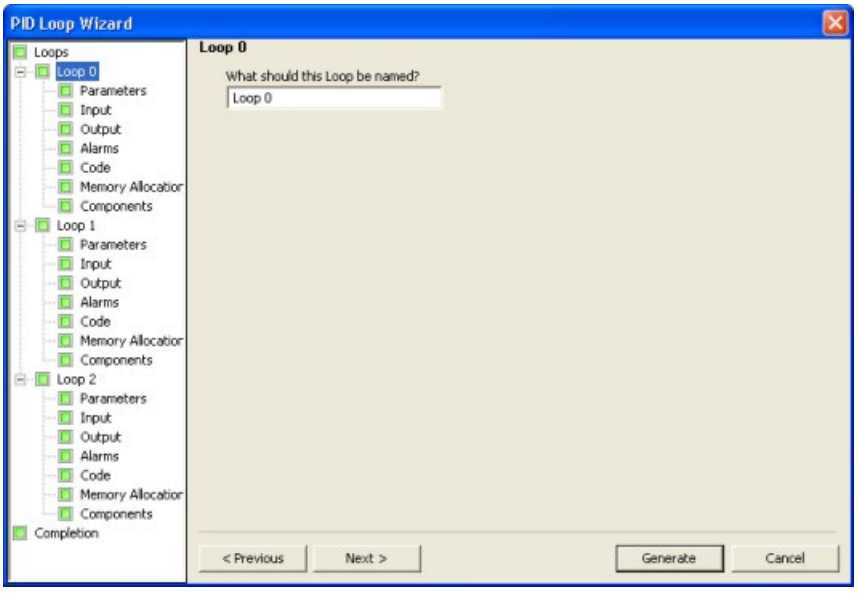
STEP 7-Micro/WIN SMART offers the PID wizard to guide you in defining a PID algorithm for a closed-loop control process. Select the "Instruction wizard" command from the "Tools" menu and then select "PID" from the "Instruction wizard" window.

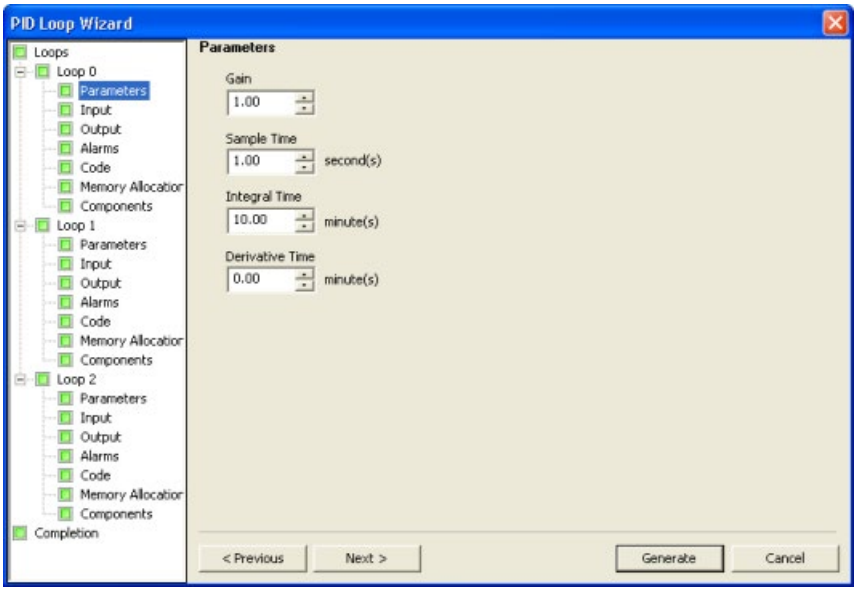
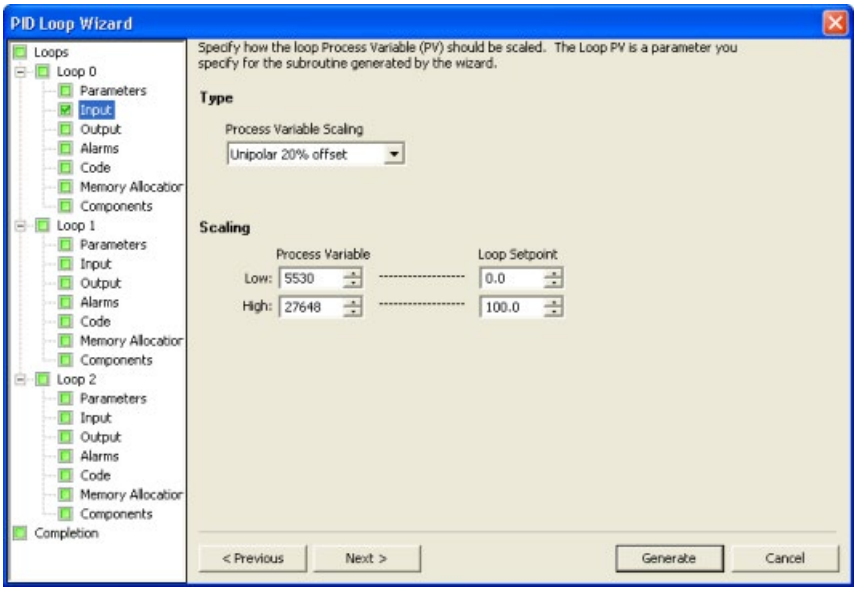
Note

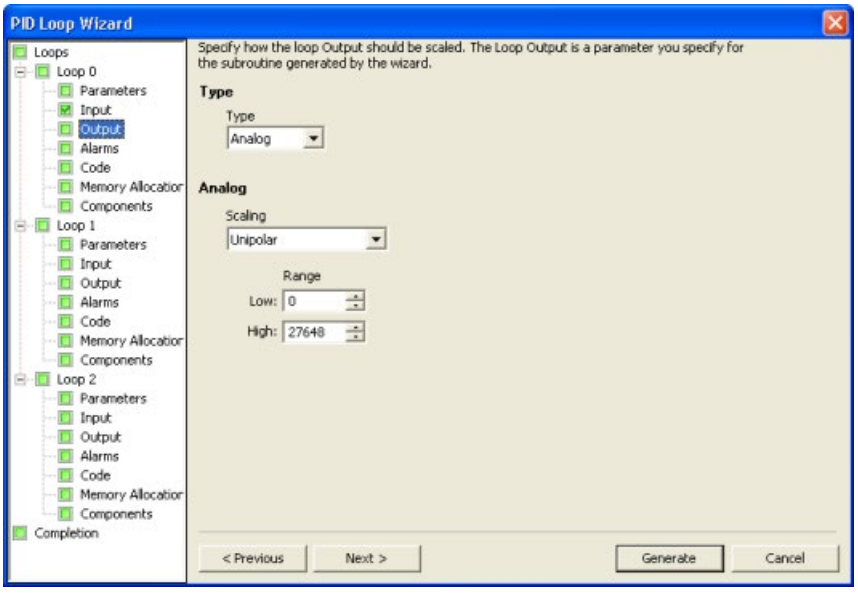
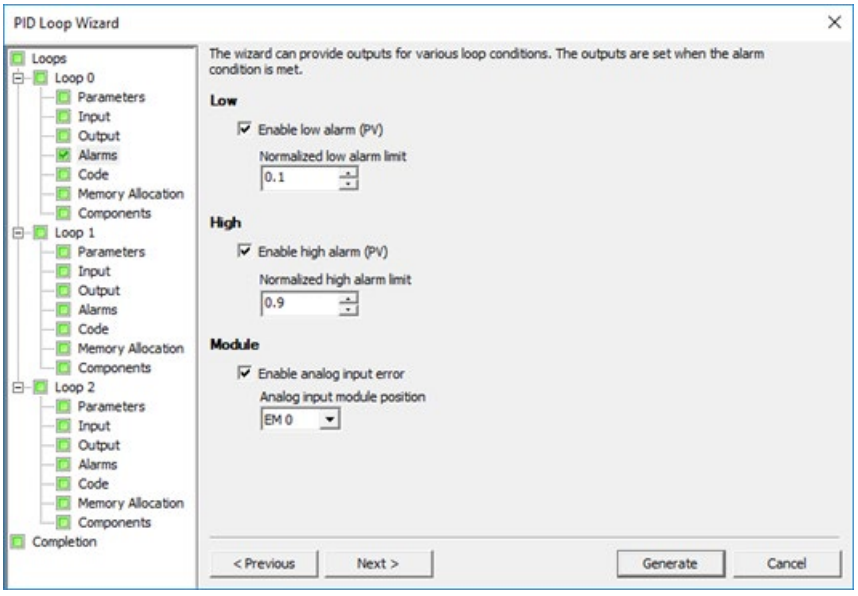
The setpoint of the low range and the setpoint of the high range should correspond to the process variable low range and high range.

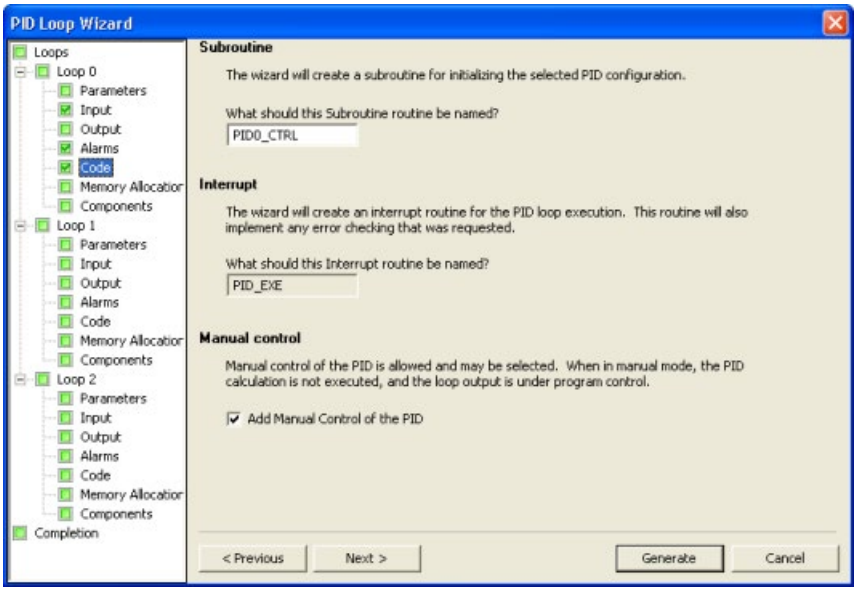
7.9.1 Using the PID wizard

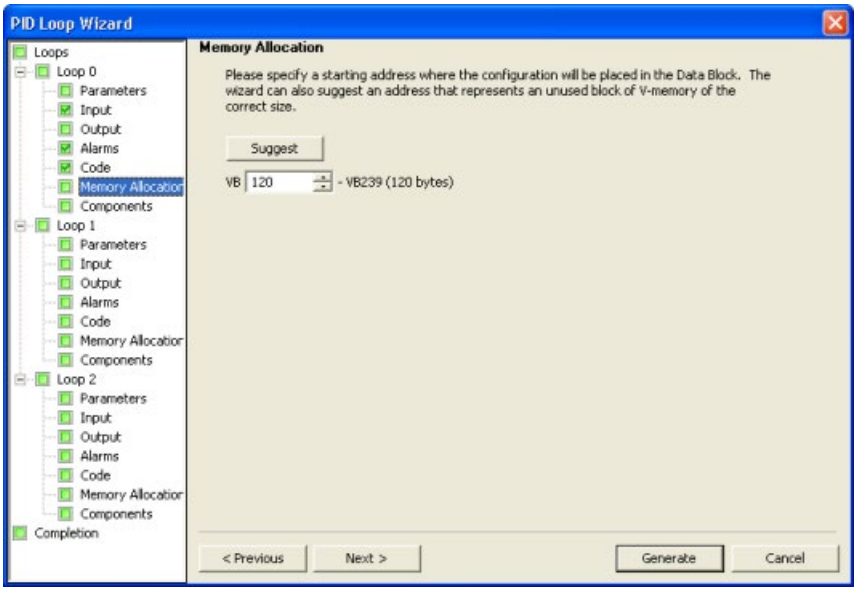
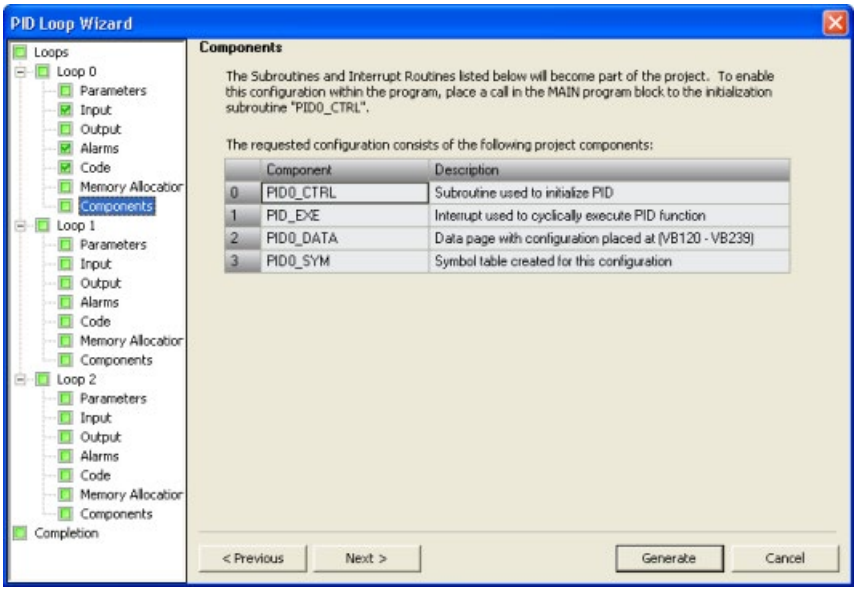
Use the PID wizard to configure your PID loop

Screen	Description
	In this dialog, you select which loops to configure. You can configure a maximum of eight loops. When you select a loop on this dialog, the tree view on the left side of the PID wizard updates with all nodes necessary for configuring that loop.
	You can configure a custom name for your loop. The default name of this screen is "Loop x", where "x" is equal to the loop number.

Screen	Description
	Set the following loop parameters: • Gain (Default value = 1.00) • Sample Time (Default value = 1.00) • Integral Time (Default value = 10.00) • Derivative Time (Default value = 0.00)
	You assign how the loop Process Variable (PV) is to be scaled. You can choose from the following options: • Unipolar (default: 0 to 27648; can edit) • Bipolar (default: -27648 to 27648; can edit) • Unipolar 20% offset (range: 5530 to 27648; set and unchangeable) • Temperature x 10 °C • Temperature x 10 °F In the Scaling parameter, you assign how the loop setpoint (SP) is to be scaled. Default is a real number between 0.0 and 100.0.

Screen	Description
	You enter the loop output options: • How the loop output is to be scaled: – Analog – Digital • Analog scaling parameter: – Unipolar (default: 0 to 27648; can edit) – Bipolar (default: -27648 to 24678; can edit) – Unipolar 20% offset (range: 5530 to 27648; is set and un-changeable • Analog range parameter: Assign the loop output range. The possible range is -27648 to +27648, de-pending on your scaling selection.
	You can assign what conditions to recognize with alarm inputs. Use the checkboxes to enable the alarms as required: • Low Alarm (PV): Set normalized low alarm limit from 0.0 to high alarm limit; default is 0.10. • High Alarm (PV): Set normalized high alarm limit from low alarm limit to 1.00; default is 0.90. • Analog Input Error: Assign where the input module is attached to the PLC.

Screen	Description
	You can make the following code selections: • Subroutine: The PID wizard creates a subroutine for initializing the selected PID configuration. • Interrupt: The PID wizard creates an interrupt routine for the PID loop execution. Note: The wizard assigns a default name for the subroutine and the interrupt routine; you can edit the default names. • Manual control: Use the "Add Manual Control of the PID" checkbox to allow manual control of your PID loops.

Screen	Description
	You can assign the starting address of the V memory byte where the configuration is placed in the data block. The wizard can suggest an address that represents an unused block of V memory of the correct size.
	This screen shows a list of the subroutines and interrupt routines generated by the PID wizard and gives a brief description of how these should be integrated into your program.

STEP 7-Micro/WIN SMART includes a PID tune control panel (Page 669) that allows you to graphically monitor the behavior of your PID loops. In addition, the control panel allows you to initiate the auto-tune sequence, abort the sequence, and apply the suggested tuning values or your own tuning values.

7.9.2 PID algorithm

In steady state operation, a PID controller regulates the value of the output so as to drive the error (e) to zero. A measure of the error is given by the difference between the setpoint (SP) (the desired operating point) and the process variable (PV) (the actual operating point). The principle of PID control is based upon the following equation that expresses the output, M(t), as a function of a proportional term, an integral term, and a differential term:

Output = Proportional term + Integral term + Differential term

$$M(t) = K_C * e + K_C \int_0^t e\,dt + M_{initial} + K_C * de/dt$$

where:

M(t)	Loop output as a function of time
K_C	Loop gain
e	Loop error (the difference between setpoint and process variable)
$M_{initial}$	Initial value of the loop output

In order to implement this control function in a digital computer, the continuous function must be quantized into periodic samples of the error value with subsequent calculation of the output. The corresponding equation that is the basis for the digital computer solution is:

Output = Proportional term + Integral term + Differential term

$$M_n = K_c * e_n + K_I * \sum_1^n e_x + M_{initial} + K_D * (e_n - e_{n-1})$$

where:

M_n	Calculated value of the loop output at sample time n
K_c	Loop gain
e_n	Value of the loop error at sample time n
e_{n-1}	Previous value of the loop error (at sample time n - 1)
K_I	Proportional constant of the integral term
$M_{initial}$	Initial value of the loop output
K_D	Proportional constant of the differential term

From this equation, the integral term is shown to be a function of all the error terms from the first sample to the current sample. The differential term is a function of the current sample and the previous sample, while the proportional term is only a function of the current sample. In a digital computer, it is not practical to store all samples of the error term, nor is it necessary.

Since the digital computer must calculate the output value each time the error is sampled beginning with the first sample, it is only necessary to store the previous value of the error and the previous value of the integral term. As a result of the repetitive nature of the digital computer solution, a simplification in the equation that must be solved at any sample time can be made. The simplified equation is:

Output = Proportional term + Integral term + Differential term

$$M_n = K_C * e_n + K_I * e_n + MX + K_D * (e_n - e_{n-1})$$

where:

M_n	Calculated value of the loop output at sample time n
K_c	Loop gain
e_n	Value of the loop error at sample time n
e_{n-1}	Previous value of the loop error (at sample time n - 1)
K_I	Proportional constant of the integral term
MX	Previous value of the integral term (at sample time n - 1)
K_D	Proportional constant of the differential term

The CPU uses a modified form of the above simplified equation when calculating the loop output value. This modified equation is:

Output = Proportional term + Integral term + Differential term

$$M_n = MP_n + MI_n + MD_n$$

where:

M_n	Calculated value of the loop output at sample time n
MP_n	Value of the proportional term of the loop output at sample time n
MI_n	Value of the integral term of the loop output at sample time n
MD_n	Value of the differential term of the loop output at sample time n

Understanding the elements of the PID equation

Proportional term of the PID equation: The proportional term MP is the product of the gain (K_C), which controls the sensitivity of the output calculation, and the error (e), which is the difference between the setpoint (SP) and the process variable (PV) at a given sample time. The equation for the proportional term as solved by the CPU is:

$$MP_n = K_C * (SP_n - PV_n)$$

where:

MP_n	Value of the proportional term of the loop output at sample time n
K_C	Loop gain
SP_n	Value of the setpoint at sample time n
PV_n	Value of the process variable at sample time n

Integral term of the PID equation: The integral term MI is proportional to the sum of the error (e) over time. The equation for the integral term as solved by the CPU is:

$$MI_n = K_1 e_n + MX = K_C * (T_S / T_I) * (SP_n - PV_n) + MX$$

where:

MI_n	Value of the integral term of the loop output at sample time n
K_C	Loop gain
T_S	Loop sample time
T_I	Integral time (also called the integral time or reset)

SP_n	Value of the setpoint at sample time n
PV_n	Value of the process variable at sample time n
MX	Value of the integral term at sample time n-1 (also called the integral sum or the bias)

The integral sum or bias (MX) is the running sum of all previous values of the integral term. After each calculation of MI_n, the bias is updated with the value of MI_n which might be adjusted or clamped (see the section "Variables and Ranges" for details). The initial value of the bias is typically set to the output value ($M_{initial}$) just prior to the first loop output calculation. Several constants are also part of the integral term, the gain (K_C), the sample time (T_S), which is the cycle time at which the PID loop recalculates the output value, and the integral time or reset (T_I), which is a time used to control the influence of the integral term in the output calculation.

Differential term of the PID equation: The differential term MD is proportional to the change in the error. The CPU uses the following equation for the differential term:

$$MD_n = K_C * (T_D / T_S) * ((SP_n - PV_n) - (SP_{n-1} - PV_{n-1}))$$

To avoid step changes or bumps in the output due to derivative action on setpoint changes, this equation is modified to assume that the setpoint is a constant ($SP_n = SP_{n-1}$). This results in the calculation of the change in the process variable instead of the change in the error as shown:

$$MD_n = K_C * (T_D / T_S) * ((SP_n - PV_n) - (SP_{n-1} - PV_{n-1}))$$

or just:

$$MD_n = K_C * (T_D / T_S) * (PV_{n-1} - PV_n)$$

where:

MD_n	Value of the differential term of the loop output at sample time n
K_C	Loop gain
T_S	Loop sample time
T_D	Differentiation period of the loop (also called the derivative time or rate)
SP_n	Value of the setpoint at sample time n
SP_{n-1}	Value of the setpoint at sample time n - 1
PV_n	Value of the process variable at sample time n - 1
PV_{n-1}	Value of the process variable at sample time n - 1

The process variable rather than the error must be saved for use in the next calculation of the differential term. At the time of the first sample, the value of PV_{n-1} is initialized to be equal to PV_n.

Selecting the type of loop control

In many control systems, it might be necessary to employ only one or two methods of loop control. For example, only proportional control or proportional and integral control might be required. The selection of the type of loop control desired is made by setting the value of the constant parameters.

If you do not want integral action (no "I" in the PID calculation), then a value of infinity "INF", should be specified for the integral time (reset). Even with no integral action, the value of the integral term might not be zero, due to the initial value of the integral sum MX.

If you do not want derivative action (no "D" in the PID calculation), then a value of 0.0 should be assigned for the derivative time (rate).

If you do not want proportional action (no "P" in the PID calculation) and you want I or ID control, then a value of 0.0 should be specified for the gain. Since the loop gain is a factor in the equations for calculating the integral and differential terms, setting a value of 0.0 for the loop gain will result in a value of 1.0 being used for the loop gain in the calculation of the integral and differential terms.

7.9.3 Converting and normalizing the loop inputs

A loop has two input variables, the setpoint and the process variable. The setpoint is generally a fixed value such as the speed setting on the cruise control in your automobile. The process variable is a value that is related to loop output and therefore measures the effect that the loop output has on the controlled system. In the example of the cruise control, the process variable would be a tachometer input that measures the rotational speed of the tires.

Both the setpoint and the process variable are real world values whose magnitude, range, and engineering units could be different. Before these real world values can be operated upon by the PID instruction, the values must be converted to normalized, floating-point representations.

The first step is to convert the real world value from a 16-bit integer value to a floating-point or real number value. The following instruction sequence is provided to show how to convert from an integer value to a real number.

```
ITD    AIW0, AC0    //Convert an input value to a double word
DTR    AC0, AC0     //Convert the 32-bit integer to a real number
```

The next step is to convert the real number value representation of the real world value to a normalized value between 0.0 and 1.0. The following equation is used to normalize either the setpoint or process variable value:

$$R_{Norm} = ((R_{Raw} / Span) + Offset)$$

where:

R_{Norm}	is the normalized, real number value representation of the real world value
R_{Raw}	is the un-normalized or raw, real number value representation of the real world value
Offset	is 0.0 for unipolar values is 0.5 for bipolar values
Span	is the maximum possible value minus the minimum possible value: = 27,648 for unipolar values (typical) = 55,296 for bipolar values (typical)

The following instruction sequence shows how to normalize the bipolar value in AC0 (whose span is 55,296) as a continuation of the previous instruction sequence:

```
/R    55296.0, AC0  //Normalize the value in the accumulator
+R    0.5, AC0      //Offset the value to the range from 0.0 to 1.0
MOVR  AC0, VD100    //Store the normalized value in the loop TABLE
```

7.9.4 Converting the loop output to a scaled integer value

The loop output is the control variable, such as the throttle setting of the cruise control on an automobile. The loop output is a normalized, real number value between 0.0 and 1.0. Before the loop output can be used to drive an analog output, the loop output must be converted to a 16-bit, scaled integer value. This process is the reverse of converting the PV and SP to a normalized value. The first step is to convert the loop output to a scaled, real number value using the formula given below:

$$R_{Scal} = (M_n - Offset) * Span$$

R_{Scal} is the scaled, real number value of the loop output

M_n is the normalized, real number value of the loop output

Offset is 0.0 for unipolar values
is 0.5 for bipolar values

Span is the maximum possible value minus the minimum possible value
= 27,648 for unipolar values (typical)
= 55,296 for bipolar values (typical)

The following instruction sequence shows how to scale the loop output:

```
MOVR  VD108, AC0    //Moves the loop output to the accumulator
-R    0.5, AC0      //Include this statement only if the value is bipolar
*R    55296.0, AC0  //Scales the value in the accumulator
```

Next, the scaled, real number value representing the loop output must be converted to a 16-bit integer. The following instruction sequence shows how to do this conversion:

```
ROUND  AC0, AC0     //Converts the real number to a 32-bit integer
DTI    AC0, LW0     //Converts the value to a 16-bit integer
MOVW   LW0, AQW0    //Writes the value to the analog output
```

7.9.5 Forward- or reverse-acting loops

The loop is forward-acting if the gain is positive and reverse-acting if the gain is negative. (For I or ID control, where the gain value is 0.0, specifying positive values for integral and derivative time will result in a forward-acting loop, and specifying negative values will result in a reverse-acting loop.)

Variables and ranges

The process variable and setpoint are inputs to the PID calculation. Therefore the loop table fields for these variables are read but not altered by the PID instruction.

The output value is generated by the PID calculation, so the output value field in the loop table is updated at the completion of each PID calculation. The output value is clamped between 0.0 and 1.0. The output value field can be used as an input by the user to specify an initial output value when making the transition from manual control to PID instruction (auto) control of the output. (See the discussion in the "Modes" section below.)

If integral control is being used, then the bias value is updated by the PID calculation and the updated value is used as an input in the next PID calculation. When the calculated output value goes out of range (output would be less than 0.0 or greater than 1.0), the bias is adjusted according to the following formulas:

- when the calculated output $M_n > 1.0$

$$MX = 1.0 - (MP_n + MD_n)$$

- when the calculated output $M_n < 0$

$$MX = -(MP_n + MD_n)$$

MX is the value of the adjusted bias

MP_n is the value of the proportional term of the loop output at sample time n

MD_n is the value of the differential term of the loop output at sample time n

M_n is the value of the loop output at sample time n

By adjusting the bias as described, an improvement in system responsiveness is achieved once the calculated output comes back into the proper range. The calculated bias is also clamped between 0.0 and 1.0 and then is written to the bias field of the loop table at the completion of each PID calculation. The value stored in the loop table is used in the next PID calculation.

The bias value in the loop table can be modified by the user prior to execution of the PID instruction in order to address bias value problems in certain application situations. Care must be taken when manually adjusting the bias, and any bias value written into the loop table must be a real number between 0.0 and 1.0.

A comparison value of the process variable is maintained in the loop table for use in the derivative action part of the PID calculation. You should not modify this value.

Modes

There is no built-in mode control for PID loops. The PID calculation is performed only when power flows to the PID box. Therefore, "automatic" or "auto" mode exists when the PID calculation is performed cyclically. "Manual" mode exists when the PID calculation is not performed.

The PID instruction has a power-flow history bit, similar to a counter instruction. The instruction uses this history bit to detect a 0-to-1 power-flow transition. When the power-flow transition is detected, it will cause the instruction to perform a series of actions to provide a bumpless change from manual control to auto control. In order for change to auto mode control to be bumpless, the value of the output as set by the manual control must be supplied as an input to the PID instruction (written to the loop table entry for M_n) before switching to auto control. The PID instruction performs the following actions to values in the

loop table to ensure a bumpless change from manual to auto control when a 0-to-1 power-flow transition is detected:

- Sets setpoint (SP_n) = process variable (PV_n)
- Sets old process variable (PV_{n-1}) = process variable (PV_n)
- Sets bias (MX) = output value (M_n)

The default state of the PID history bits is "set" and that state is established at startup and on every STOP-to-RUN mode transition of the controller. If power flows to the PID box the first time that it is executed after entering RUN mode, then no power-flow transition is detected and the bumpless mode change actions are not performed.

Alarm checking and special operations

The PID instruction is a simple but powerful instruction that performs the PID calculation. If other processing is required such as alarm checking or special calculations on loop variables, these must be implemented using the basic instructions supported by the CPU.

Error conditions

When it is time to compile, the CPU will generate a compile error (range error) and the compilation will fail if the loop table start address or PID loop number operands specified in the instruction are out of range.

Certain loop table input values are not range checked by the PID instruction. You must take care to ensure that the process variable and setpoint (as well as the bias and previous process variable if used as inputs) are real numbers between 0.0 and 1.0.

If any error is encountered while performing the mathematical operations of the PID calculation, then SM1.1 (overflow or illegal value) is set and execution of the PID instruction is terminated. (Update of the output values in the loop table could be incomplete, so you should disregard these values and correct the input value causing the mathematical error before the next execution of the loop's PID instruction.)

Loop table

The loop table is 80 bytes long and has the format shown in the following table.

Offset	Field	Format	Type	Description
0	Process variable (PV_n)	REAL	In	Contains the process variable, which must be scaled between 0.0 and 1.0.
4	Setpoint (SP_n)	REAL	In	Contains the setpoint, which must be scaled between 0.0 and 1.0.
8	Output (M_n)	REAL	In/Out	Contains the calculated output, scaled between 0.0 and 1.0.
12	Gain (K_C)	REAL	In	Contains the gain, which is a proportional constant. Can be a positive or negative number.
16	Sample time (T_S)	REAL	In	Contains the sample time, in seconds. Must be a positive number.

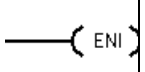
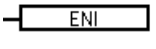
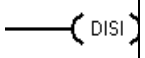
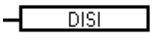
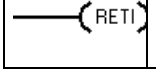
Offset	Field	Format	Type	Description
20	Integral time or reset (T_I)	REAL	In	Contains the integral time or reset, in minutes. Must be a positive number.
24	Derivative time or rate (T_D)	REAL	In	Contains the derivative time or rate, in minutes. Must be a positive number.
28	Bias (MX)	REAL	In/Out	Contains the bias or integral sum value between 0.0 and 1.0.
32	Previous process variable (PV_{n-1})	REAL	In/Out	Contains the value of the process variable stored from the last execution of the PID instruction.
36 to 79	Reserved for auto-tuning variables. Refer to PID loop definition table (Page 662) for details.			

7.10 Interrupt

7.10.1 Interrupt instructions

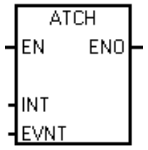
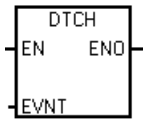
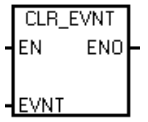
When you make the transition to RUN mode, interrupts are initially disabled. In RUN mode, you can enable interrupt processing by executing the ENI (enable Interrupt) instruction. Executing the DISI (disable interrupt) instruction inhibits the processing of interrupts; however, active interrupt events will continue to be queued.

ENI, DISI, and CRETI

LAD	FBD	STL	Description
		**ENI**	The enable interrupt instruction globally enables processing of all attached interrupt events.
		**DISI**	The disable interrupt instruction globally disables processing of all interrupt events.
		**CRETI**	The conditional return from interrupt instruction can be used to return from an interrupt, based upon the condition of the preceding program logic.

Non-fatal errors with ENO = 0	SM bits affected
• 0004H Attempted ENI or DISI execution disallowed in an interrupt routine	None

ATCH, DTCH, and CEVENT

LAD / FBD	STL	Description
	**ATCH INT, EVNT**	The attach interrupt instruction associates an interrupt event EVNT with an interrupt routine number INT and enables the interrupt event.
	**DTCH EVNT**	The detach interrupt instruction disassociates an interrupt event EVNT from all interrupt routines and disables the interrupt event.
	**CEVENT EVNT**	The clear interrupt event instruction removes all interrupt events of type EVNT from the interrupt queue. Use this instruction to clear the interrupt queue of unwanted interrupt events. If this instruction is being used to clear out spurious interrupt events, then you should detach the event before clearing the events from the queue. Otherwise new events will be added to the queue after the clear event instruction has been executed.

Non-fatal errors with ENO = 0	SM bits affected
• 0002H Conflicting assignment of inputs to an HSC • 0016H Attempt to use HSC or edge interrupt on input channel that is already allocated to a motion control function • 0019H Attempt to use a signal board function without a signal board installed and configured • 0090H Invalid operand (illegal event number)	None

Input / output	Data type	Operand
INT	BYTE	Constant: interrupt routine number (0 to 127)
EVNT	BYTE	Constant: interrupt event number CPUs CR20s, CR30s, CR40s, and CR60s: 0-13, 16-18, 21-23, 27, 28, and 32 CPUs SR20/ST20, SR30/ST30, SR40/ST40, and SR60/ST60: 0-13 and 16-44

7.10.2 Interrupt routine overview and CPU model event support

Before you can invoke an interrupt routine, you must assign an association between the interrupt event and the program segment that you want to execute when the event occurs. Use the attach interrupt instruction to associate an interrupt event (specified by the interrupt event number) and the program segment (specified by an interrupt routine number). You can attach multiple interrupt events to one interrupt routine, but you cannot attach one event to multiple interrupt routines.

When you attach an event and interrupt routine, new occurrences of this event cause execution of the attached interrupt routine only if the program executed the global ENI (enable interrupts) instruction and interrupt event processing is active. Otherwise, the CPU adds the event to the interrupt event queue. If you disable all interrupts using the global DISI (disable interrupts) instruction, the CPU queues each occurrence of the interrupt event until the interrupts are re-enabled, using the global ENI (enable interrupt) instruction or the interrupt queue overflows.

You can disable individual interrupt events by breaking the association between the interrupt event and the interrupt routine with the detach Interrupt instruction. The detach interrupt instruction returns the interrupt to an inactive or ignored state. The following table lists the different types of interrupt events.

Event	Description	CPU CR20s CPU CR30s CPU CR40s CPU CR60s	CPU SR20/ST20 CPU SR30/ST30 CPU SR40/ST40 CPU SR60/ST60
0	I0.0 Rising edge	Y	Y
1	I0.0 Falling edge	Y	Y
2	I0.1 Rising edge	Y	Y
3	I0.1 Falling edge	Y	Y
4	I0.2 Rising edge	Y	Y
5	I0.2 Falling edge	Y	Y
6	I0.3 Rising edge	Y	Y
7	I0.3 Falling edge	Y	Y
8	Port 0 Receive character	Y	Y
9	Port 0 Transmit complete	Y	Y
10	Timed interrupt 0 (SMB34 controls the time interval)	Y	Y
11	Timed interrupt 1 (SMB35 controls the time interval)	Y	Y
12	HSC0 CV=PV (current value = preset value)	Y	Y
13	HSC1 CV=PV (current value = preset value)	Y	Y
14-15	Reserved	N	N
16	HSC2 CV=PV (current value = preset value)	Y	Y
17	HSC2 Direction changed	Y	Y
18	HSC2 External reset	Y	Y
19	PTO0 Pulse count complete	N	Y

Event	Description	CPU CR20s CPU CR30s CPU CR40s CPU CR60s	CPU SR20/ST20 CPU SR30/ST30 CPU SR40/ST40 CPU SR60/ST60
20	PTO1 Pulse count complete	N	Y
21	Timer T32 CT=PT (current time = preset time)	Y	Y
22	Timer T96 CT=PT (current time = preset time)	Y	Y
23	Port 0 Receive message complete	Y	Y
24	Port 1 Receive message complete	N	Y
25	Port 1 Receive character	N	Y
26	Port 1 Transmit complete	N	Y
27	HSC0 Direction changed	Y	Y
28	HSC0 External reset	Y	Y
29	HSC4 CV=PV	N	Y
30	HSC4 direction changed	N	Y
31	HSC4 external reset	N	Y
32	HSC3 CV=PV (current value = preset value)	Y	Y
33	HSC5 CV=PV	N	Y
34	PTO2 Pulse count complete	N	Y
35	I7.0 Rising edge (signal board)	N	Y
36	I7.0 Falling edge (signal board)	N	Y
37	I7.1 Rising edge (signal board)	N	Y
38	I7.1 Falling edge (signal board)	N	Y
43	HSC5 direction changed	N	Y
44	HSC5 external reset	N	Y

7.10.3 Interrupt programming guidelines

Interrupt routine execution

An interrupt routine executes in response to an associated internal or external event. Once the last instruction of an interrupt routine has executed, control returns to the point in the scan cycle at the point of the interruption. You can exit the routine by executing a conditional return from interrupt instruction (CRETI).

Interrupt processing provides quick reaction to special internal or external events. Optimize your interrupt routines to perform a specific task, and then return control to the scan cycle.

Note

- You cannot use the disable interrupt (DISI), enable interrupt (ENI), high-speed counter definition (HDEF), and end (END) instructions in an interrupt routine.
- Keep interrupt routine program logic short and to the point, so execution is quick and other processes are not deferred for long periods of time. If this is not done, unexpected conditions can cause abnormal operation of equipment controlled by the main program.

System support for interrupts

Because interrupts can affect contact, coil, and accumulator logic, the system saves and reloads the logic stack, accumulator registers, and the special memory bits (SM) that indicate the status of accumulator and instruction operations. This avoids disruption to the main user program caused by branching to and from an interrupt routine.

Calling subroutines from interrupt routines

You can call four nesting levels of subroutines from an interrupt routine. The accumulators and the logic stack are shared between an interrupt routine and the four nesting levels of subroutines called from the interrupt routine

Sharing data between the main program and the interrupt routines

You can share data between the main program and one or more interrupt routines. Because it is not possible to predict when the CPU might generate an interrupt, it is desirable to limit the number of variables that are used by both the interrupt routine and elsewhere in the program. Problems with the consistency of shared data can result due to the actions of interrupt routines when the execution of instructions in your main program is interrupted by interrupt events. Use the interrupt block "variable table" (block call interface table) to ensure that your interrupt routine uses only the temporary memory and does not overwrite data used somewhere else in your program.

Ensuring access for a single shared variable

- For an STL program that is sharing a single variable: If the shared data is a single byte, word, or double word variable and your program is written in STL, then correct shared access can be ensured by storing the intermediate values from operations on shared data only in non-shared memory locations or accumulators.
- For a LAD program that is sharing a single variable: If the shared data is a single byte, word, or double word variable and your program is written in LAD, then correct shared access can be ensured by establishing the convention that access to shared memory locations be made using only Move instructions (MOVB, MOVW, MOVD, MOVR). While many LAD instructions are composed of interruptible sequences of STL instructions, these Move instructions are composed of a single STL instruction whose execution cannot be affected by interrupt events.

Ensuring access for multiple shared variables

For an STL or LAD program that is sharing multiple variables: If the shared data is composed of a number of related bytes, words, or double words, then the interrupt disable/enable instructions (DISI and ENI) can be used to control interrupt routine execution. At the point in your main program where operations on shared memory locations are to begin, disable the interrupts. Once all actions affecting the shared locations are complete, re-enable the interrupts. During the time that interrupts are disabled, interrupt routines cannot be executed and therefore cannot access shared memory locations; however, this approach can result in delayed response to interrupt events.

7.10.4 Types of interrupt events that the S7-200 SMART CPU supports

Communication port interrupts

The serial communications port of the CPU can be controlled by your program. This mode of operating the communications port is called Freeport mode. In Freeport mode, your program defines the baud rate, bits per character, parity, and protocol. The Receive and Transmit interrupts are available to facilitate your program-controlled communications. Refer to the Transmit and Receive instructions for more information.

I/O interrupts

I/O interrupts include rising/falling edge interrupts, high-speed counter interrupts, and pulse train output interrupts. The CPU can generate an interrupt on rising and/or falling edges of an input for input channels I0.0, I0.1, I0.2, and I0.3 (and for I7.0 and I7.1 for standard CPUs with an optional digital input signal board). The rising edge and the falling edge events can be captured for each of these input points. These rising/falling edge events can be used to signify a condition that must receive immediate attention when the event happens.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of signal boards.

The high-speed counter interrupts allow you to respond to conditions such as the current value reaching the preset value, a change in counting direction that might correspond to a reversal in the direction in which a shaft is turning, or an external reset of the counter. Each of these high-speed counter events allows action to be taken in real time in response to high-speed events that cannot be controlled at programmable logic controller scan speeds.

The pulse train output interrupts provide immediate notification of completion of the output of the prescribed number of pulses. A typical use of pulse train outputs is stepper motor control.

You enable each of the above interrupts by attaching an interrupt routine to the related I/O event.

Time-based interrupts

Time-based interrupts include timed interrupts and the timer T32/T96 interrupts. You can specify actions to be taken on a cyclic basis using a timed interrupt. The cycle time is set in 1-ms increments from 1 ms to 255 ms. You must write the cycle time in SMB34 for timed interrupt 0, and in SMB35 for timed interrupt 1.

The timed interrupt event transfers control to the appropriate interrupt routine each time the timer expires. Typically, you use timed interrupts to control the sampling of analog inputs or to execute a PID loop at regular intervals.

A timed interrupt is enabled and timing begins when you attach an interrupt routine to a timed interrupt event. During the attachment, the system captures the cycle time value, so subsequent changes to SMB34 and SMB35 do not affect the cycle time. To change the cycle time, you must modify the cycle time value, and then re-attach the interrupt routine to the timed interrupt event. When the re-attachment occurs, the timed interrupt function clears any accumulated time from the previous attachment and begins timing with the new value.

After being enabled, the timed interrupt runs continuously and executes the attached interrupt routine, at the end of each successive time interval. If you exit RUN mode or detach the timed interrupt, the timed interrupt is disabled. If the global DISI (disable interrupt) instruction is executed, timed interrupts continue to occur, but the attached interrupt routine is not processed yet. Each occurrence of the timed interrupt is queued (until either interrupts are enabled or the queue is full).

The timer T32/T96 interrupts allow timely response to the completion of a specified time interval. These interrupts are only supported for the 1-ms resolution on-delay (TON) and off-delay (TOF) timers T32 and T96. The T32 and T96 timers otherwise behave normally. Once the interrupt is enabled, the attached interrupt routine is executed when the active timer's current value becomes equal to the preset time value during the normal 1-ms timer update performed in the CPU. You enable these interrupts by attaching an interrupt routine to the T32 (event 21) and T96 (event 22) interrupt events.

7.10.5 Interrupt priority, queuing, and example program

Interrupt service

Interrupts are serviced by the CPU on a first-come-first-served basis within their respective priority group. Only one user-interrupt routine is ever being executed at any point in time. Once the execution of an interrupt routine begins, the routine is executed to completion. It cannot be pre-empted by another interrupt routine, even by a higher priority routine. Interrupts that occur while another interrupt is being processed are queued for later processing. The following table shows the three interrupt queues and the maximum number of interrupts they can store.

It is possible that more interrupts can occur than a queue can hold. Therefore, queue overflow memory bits (identifying the type of interrupt events that have been lost) are maintained by the system. The following table shows the interrupt queue overflow bits. You should use these bits only in an interrupt routine because they are reset when the queue is emptied, and control is returned to the scan cycle.

If multiple interrupt events occur at the same time, the priority (group and within a group) determines which interrupt event is processed first. Once the highest priority has been

handled, the queue is examined to find the current highest priority event that remains in the queue and the interrupt routine attached to that event is executed. This continues until the queue is empty and control is returned to the scan cycle.

Maximum number of entries per interrupt queue

The following table shows all interrupt events, with their priority and assigned event number.

Queue	Queue depth for all S7-200 SMART CPU models
Communications queue	4
I/O interrupt queue	16
Timed interrupt queue	8

Interrupt queue overflow bits

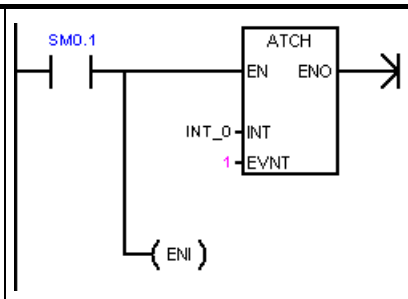
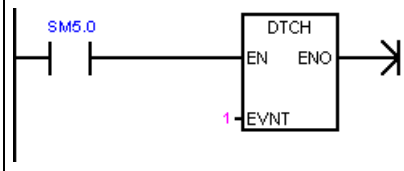
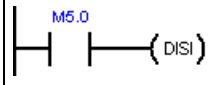
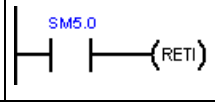
Description (0 = No Overflow, 1 = Overflow)	SM Bit
Communications queue	SM4.0
I/O Interrupt queue	SM4.1
Timed Interrupt queue	SM4.2

Priority order for interrupt events

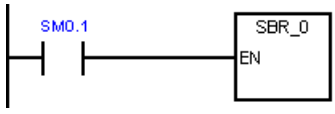
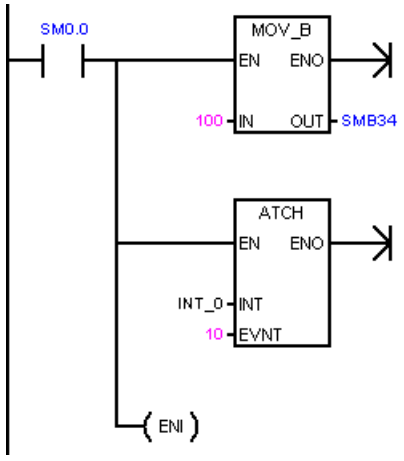
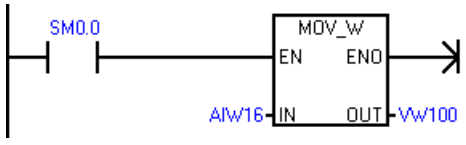
Priority group	Event	Description
Communications Highest Priority	8	Port 0 Receive character
	9	Port 0 Transmit complete
	23	Port 0 Receive message complete
	24	Port 1 Receive message complete
	25	Port 1 Receive character
	26	Port 1 Transmit complete
Discrete Medium Priority	19	PTO0 Pulse count complete
	20	PTO1 Pulse count complete
	34	PTO2 Pulse count complete
	0	I0.0 Rising edge
	2	I0.1 Rising edge
	4	I0.2 Rising edge
	6	I0.3 Rising edge
	35	I7.0 Rising edge (signal board)
	37	I7.1 Rising edge (signal board)
	1	I0.0 Falling edge
	3	I0.1 Falling edge
	5	I0.2 Falling edge
	7	I0.3 Falling edge

Priority group	Event	Description
	36	I7.0 Falling edge (signal board)
	38	I7.1 Falling edge (signal board)
	12	HSC0 CV=PV (current value = preset value)
	27	HSC0 Direction changed
	28	HSC0 External reset
	13	HSC1 CV=PV (current value = preset value)
	16	HSC2 CV=PV (current value = preset value)
	17	HSC2 Direction changed
	18	HSC2 External reset
	32	HSC3 CV=PV (current value = preset value)
	29	HSC4 CV=PV
	30	HSC4 direction changed
	31	HSC4 external reset
	33	HSC5 CV=PV
	43	HSC5 direction changed
	44	HSC5 external reset
Timed Lowest Priority	10	Timed interrupt 0 SMB34
	11	Timed interrupt 1 SMB35
	21	Timer T32 CT=PT interrupt
	22	Timer T96 CT=PT interrupt

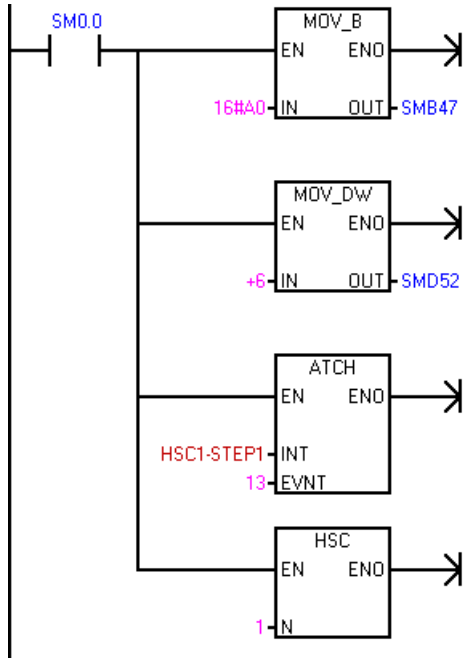
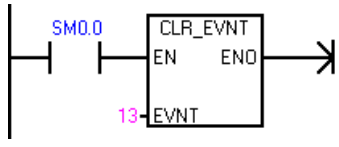
Example 1: Input signal edge detector interrupt

LAD			STL
MAIN Network 1		On the first scan: 1. Define interrupt routine INT_0 to be a falling-edge interrupt for I0.0. 2. Globally enable interrupts.	**Network 1** **LD SM0.1** **ATCH INT_0, 1** **ENI**
Network 2		If an I/O error is detected, disable the falling-edge interrupt for I0.0. (This network is optional.)	**Network 2** **LD SM5.0** **DTCH 1**
Network 3		When M5.0 is on, disable all interrupts. When disabled, attached interrupt events will be queued, but the corresponding interrupt routines will not be executed until interrupts are re-enabled with the ENI instruction.	**Network 3** **LD M5.0** **DISI**
INT 0 Network 1		I0.0 falling-edge interrupt routine: Conditional return based on an I/O error.	**Network 1** **LD SM5.0** **CRETI**

Example 2: Timed interrupt for reading the value of an analog input

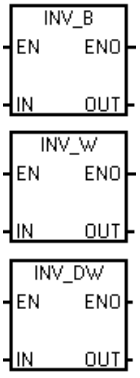
LAD			STL
MAIN Network 1		On the first scan, call subroutine 0.	**Network 1** **LD SM0.1** **CALL SBR_0**
SBR 0 Network 1		Set the interval for the timed interrupt 0 to 100 ms. Attach timed interrupt 0 (Event 10) to INT_0. Global interrupt enable.	**Network 1** **LD SM0.0** **MOVB 100, SMB34** **ATCH INT_0, 10** **ENI**
INT 0 Network 1		Read the value of AIW16 every 100 ms.	**Network 1** **LD SM0.0** **MOVW AIW16, VW100**

Example 3: Clear interrupt event instruction

LAD			STL
SBR 1 Network 1		HSC instruction wizard: Set control bits, write preset. PV = 6 Attach interrupt HSC1_STEP1: CV = PV for HC1 Configure HSC 1.	**Network 1** **LD SM0.0** **MOVB 16#A0, SMB47** **MOVD +6, SMD52** **ATCH HSC1_STEP1, 13**
SBR 1 Network 2		Clear unwanted interrupts caused by machine vibration.	**Network 2** **LD SM0.0** **CEVNT 13**

7.11 Logical operations

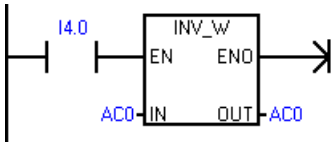
7.11.1 Invert

LAD / FBD	STL	Description
	**INVB OUT** **INVW OUT** **INVD OUT**	The Invert Byte, Invert Word, and Invert Double Word instructions form the one's complement of the input IN and load the result into the memory location OUT

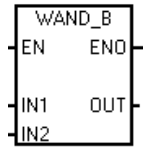
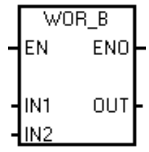
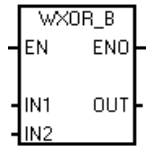
Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	• SM1.0 Result of operation = zero

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC,*VD, *LD, *AC
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: Invert instruction

LAD		STL
	Invert word value in AC0. Result is put in AC0. Invert Word input AC0 1101 0111 1001 0101 Execute one's complement inversion Invert Word output AC0 0010 1000 0110 1010	**Network 1** **LD I4.0** **INVW AC0**

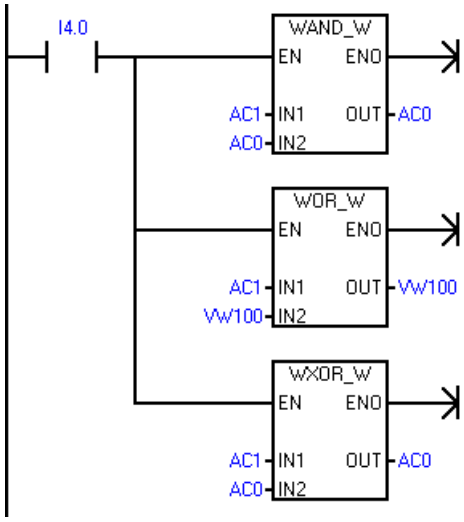
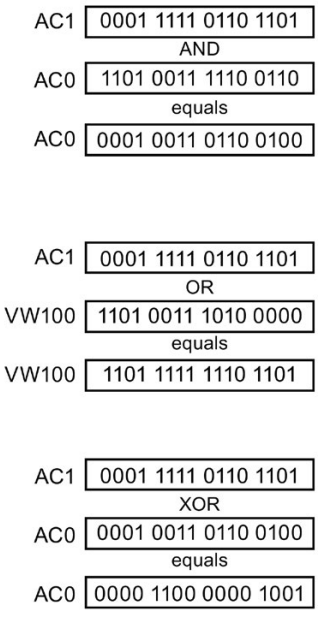
7.11.2 AND, OR, and exclusive OR

LAD / FBD	STL	Description
 WAND_W WAND_DW	**ANDB IN1, OUT** **ANDW IN1, OUT** **ANDD IN1, OUT**	The AND Byte, AND Word, and AND Double Word instructions logically AND the corresponding bits of two input values IN1 and IN2 and load the result in a memory location assigned to OUT. • LAD and FBD: IN1 AND IN2 = OUT • STL: IN1 AND OUT = OUT
 WOR_W WOR_DW	**ORB IN1, OUT** **ORW IN1, OUT** **ORD IN1, OUT**	The OR Byte, OR Word, and OR Double Word instructions logically OR the corresponding bits of two input values IN1 and IN2 and load the result in a memory location assigned to OUT. • LAD and FBD: IN1 OR IN2 = OUT • STL: IN1 OR OUT = OUT
 WXOR_W WXOR_DW	**XORB IN1, OUT** **XORW IN1, OUT** **XORD IN1, OUT**	The Exclusive OR Byte, Exclusive OR Word, and Exclusive OR Double Word instructions logically XOR the corresponding bits of two input values IN1 and IN2 and load the result in a memory location OUT. • LAD and FBD: IN1 XOR IN2 = OUT • STL: IN1 XOR OUT = OUT

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	• SM1.0 Result of operation = zero

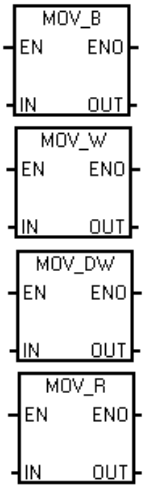
Input / output	Data type	Operand
IN1, IN2	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *AC, *LD

Example: AND, OR, and Exclusive OR instructions

LAD		STL
		**Network 1** **LD I4.0** **ANDW AC1, AC0** **ORW AC1, VW100** **XORW AC1, AC0**

7.12 Move

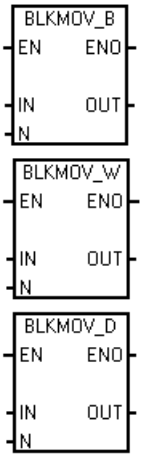
7.12.1 Move byte, word, double word, or real

LAD / FBD	STL	Description
	**MOVB IN, OUT** **MOVW IN, OUT** **MOVD IN, OUT** **MOVR IN, OUT**	The Move Byte, Move Word, Move Double Word, and Move Real instructions move a data value from a source (constant or memory location) IN to a new memory location OUT, without changing the value stored in a source memory location. Use the Move Double Word instruction to create a pointer. For more information, refer to the section on pointers and indirect addressing (Page 76).

Non-fatal error with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *AC, *LD, Constant
	DWORD, DINT	ID, QD, VD, MD, SMD, SD, LD, HC, &VB, &IB, &QB, &MB, &SB, &T, &C, &SMB, &AIW, &AQW, AC, *VD, *LD, *AC, Constant
	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC
	DWORD, DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

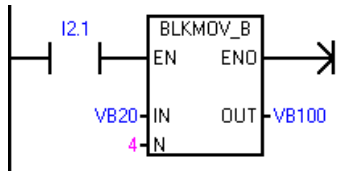
7.12.2 Block move

LAD / FBD	STL	Description
	**BMB IN, OUT, N** **BMW IN, OUT, N** **BMD IN, OUT, N**	The Block Move Byte, Block Move Word, and Block Move Double Word instructions move an assigned block of data values from a source memory location (starting address IN and successive addresses) to a new memory location (starting address OUT and successive addresses). Parameter N assigns the number of bytes, words, or double words to move. The block of data values stored in the source location are not changed. N has a range of 1 to 255.

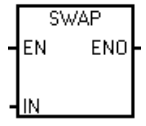
Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range	None

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AIW, *VD, *LD, *AC
	DWORD, DINT	ID, QD, VD, MD, SMD, SD, LD, *VD, *LD, *AC
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AQW, *VD, *LD, *AC
	DWORD, DINT	ID, QD, VD, MD, SMD, SD, LD, *VD, *LD, *AC
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, Constant, *VD, *LD, *AC

Example: Block Move instruction

LAD			STL	
	Move (copy) data in source four byte address sequence (VB20 to VB23) to destination four byte address sequence (VB100 to VB103).		**Network 1** **LD I2.1** **BMB VB20, VB100, 4**	
Source data values	30	31	32	33
Source data addresses	VB20	VB21	VB22	VB23
If I2.1 = 1, then execute BLKMOV_B to move source data values to destination addresses				
Destination data values	30	31	32	33
Destination data addresses	VB100	VB101	VB102	VB103

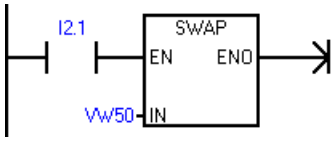
7.12.3 Swap bytes

LAD / FBD	STL	Description
	**SWAP IN**	The Swap Bytes instruction exchanges the most significant byte with the least significant byte of the word IN.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

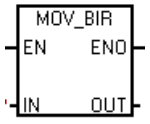
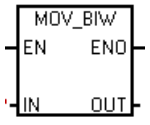
Input / output	Data type	Operand
IN	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC

Example: Swap instructions

LAD	STL
	**Network 1** **LD I2.1** **SWAP VW50**

Hexadecimal data values	D6	C3
Data addresses	VB50	VB51
If I2.1 = 1, then execute SWAP to exchange the byte data in a data word		
Hexadecimal data values	C3	D6
Data addresses	VB50	VB51

7.12.4 Move byte immediate (read and write)

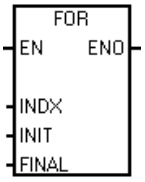
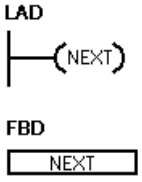
LAD / FBD	STL	Description
	**BIR IN, OUT**	The Move Byte Immediate Read instruction reads the state of physical input IN and writes the result to the memory address OUT, but the process image register is not updated.
	**BIW IN, OUT**	The Move Byte Immediate Write instruction reads the data from the memory address IN and writes to physical output OUT, and the corresponding process image location.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • Unable to access expansion module	None

Input / output	Data type	Operand
IN (BIR)	BYTE	IB, *VD, *LD, *AC
IN (BIW)	BYTE	B, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT (BIR)	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
OUT (BIW)	BYTE	QB, *VD, *LD, *AC

7.13 Program control

7.13.1 FOR-NEXT loop

LAD / FBD	STL	Description
	**FOR INDX, INIT, FINAL**	The FOR instruction executes the instructions between the FOR and the NEXT instructions. You assign the index value or current loop count INDX, the starting loop count INIT, and the ending loop count FINAL.
	**NEXT**	The NEXT instruction marks the end of the FOR loop program segment.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
INDX	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
INIT, FINAL	INT	VW, IW, QW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant

Use the FOR and NEXT instructions to execute a program segment in a loop that is repeated for the assigned count. Each FOR instruction requires one NEXT instruction. You place a FOR-NEXT loop within a FOR-NEXT loop to a maximum nesting depth of eight.

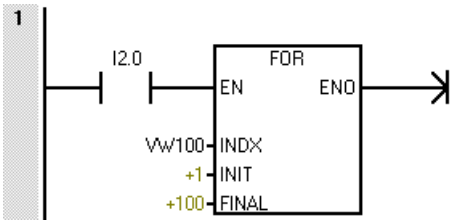
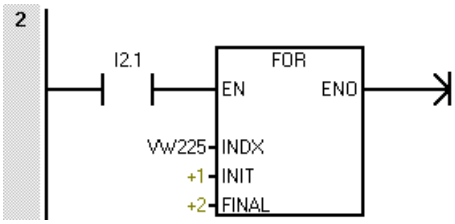
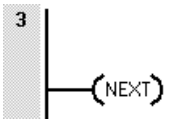
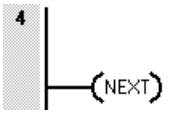
If you enable a FOR-NEXT loop, the execution loop continues until it finishes the iterations, unless you change the FINAL value from within the loop itself. You can change the values while the FOR-NEXT loop is in the looping process. When the loop is enabled again, it copies the INIT value to the INDX value (current loop number).

For example, given an INIT value of 1 and a FINAL value of 10, the instructions between the FOR instruction and the NEXT instruction are executed 10 times with the INDX value being incremented: 1, 2, 3, ... 10.

If the INIT value is greater than the FINAL value, the loop is not executed. After each execution of the instructions between the FOR instruction and the NEXT instruction, the INDX value is incremented and the result is compared to the final value. If the INDX is greater than the final value, the execution loop is terminated.

For STL, if the top of the logic stack value is 1 when your program enters the FOR-NEXT loop, then the top of the logic stack will be 1 when your program exits the FOR-NEXT loop.

Example: FOR-NEXT loop

LAD		STL
1 	When I2.0 is ON, the outside loop (Network 1 - 4) is executed 100 times.	**Network 1** **LD I2.0** **FOR VW100, +1, +100**
2 	The inside loop (Network 2 - 3) is executed twice for each execution of the outside loop when I2.1 is on.	**Network 2** **LD I2.1** **FOR VW225, +1, +2**
3 	End of inside loop	**Network 3** **NEXT**
4 	End of outside loop	**Network 4** **NEXT**

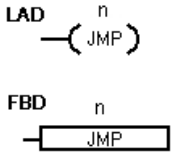
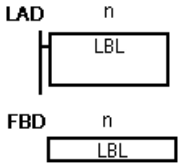
7.13.2 JMP (jump to label)

You can use the JMP (Jump) instruction in the main program, in subroutines, or in interrupt routines. The JMP and its corresponding LBL (Label) instruction must be located within the same program segment either in the main program, a subroutine, or an interrupt routine.

Note

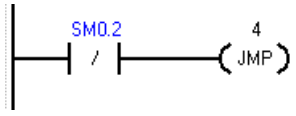
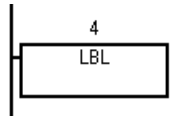
You cannot jump from the main program to a label in either a subroutine or an interrupt routine. Likewise, you cannot jump from a subroutine or interrupt routine to a label outside that subroutine or interrupt routine.

You can use a Jump instruction within an SCR program segment, but the corresponding Label instruction must be located within the same SCR program segment.

LAD / FBD	STL	Description
	**JMP N**	The JMP (Jump) instruction performs a branch to the label N within the program.
	**LBL N**	The LBL (Label) instruction marks the location of the jump destination n.

Input / output	Data type	Operand
n	WORD	Constant (0 to 255)

Example: Jump to Label

LAD		STL
	If the retentive data has not been lost, jump to label 4.	**Network 1** **LDN SM0.2** **JMP 4**
	Label 4	**Network 2** **LBL 4**

7.13.3 SCR (sequence control relay)

SCR (Sequence Control Relay) instructions provide a simple yet powerful state control programming technique for a LAD, FBD, or STL program. Whenever an application consists of a sequence of operations that must be performed repetitively, you can use SCRs to structure the program so that it corresponds directly to your application. As a result, you can program and debug the application quickly and easily.

⚠ WARNING

S bit usage in POUs

Do not use the same S bit in more than one POU. For example, if you use S0.1 in the main program, do not use it in a subroutine.

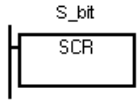
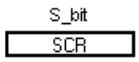
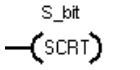
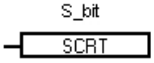
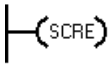
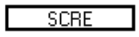
Multiple POUs accessing the same S bit could result in unexpected process operation, possibly resulting in death or severe personal injury.

Check your program to ensure that multiple POUs do not access the same S bit.

Note

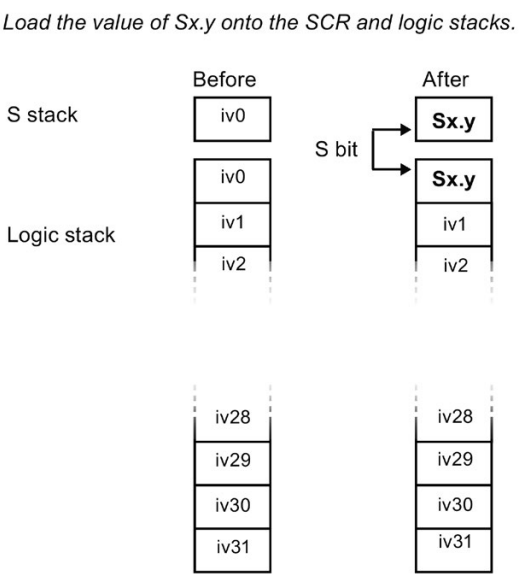
SCR programming restrictions

- You cannot jump into or out of an SCR segment; however, you can use Jump and Label instructions to jump around SCR segments or to jump within an SCR segment.
- You cannot use the END instruction in an SCR segment.

LAD	FBD	STL	Description
		**LSCR S_bit**	The **SCR** instruction loads the SCR and logic stacks with the value of the S bit referenced by the instruction. The resulting value of the SCR stack either energizes or de-energizes the SCR stack. The value of the SCR stack is copied to the top of the logic stack so that LAD boxes or output coils can be tied directly to the left power rail and no preceding contact instruction is required.
		**SCRT S_bit**	The **SCRT** instruction identifies the SCR bit to be enabled (the next S_bit to be set). When power flows to the coil or FBD box, the CPU turns on the referenced S_bit and turns off the S_bit of the LSCR instruction (that enabled this SCR segment).
		CSCRE	The **CSCRE** (conditional SCR end) instruction, for STL and FBD, terminates execution of the SCR segment when enabled. For LAD, a conditional contact placed before a **SCRE** coil performs the conditional SCR end function.
		**SCRE**	The **SCRE** (unconditional SCR end) instruction, for STL and FBD, terminates execution of the SCR segment. For LAD, an SCRE coil connected directly to the power rail performs the unconditional SCR end function.

Input / output	Data type	Operand
S_bit	BOOL	S

S stack and logic stack interaction



The figure shows the S stack and the logic stack and the effect of executing the Load SCR instruction.

Controlling program flow with SCR segments

The main program consists of instructions that execute sequentially once per scan of the PLC. For many applications, it may be appropriate to logically divide the main program into a series of operational steps that mirror steps within a controlled process (for example, a series of machine operations).

One way to logically divide a program into multiple steps is to use SCR segments. SCR segments can divide your program into a single stream of sequential steps, or into multiple streams that can be active simultaneously. It is possible to have a single stream conditionally diverge into multiple streams, and to have multiple streams conditionally re-converge into a single stream.

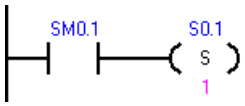
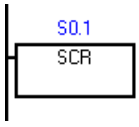
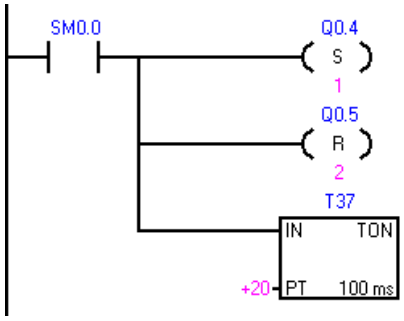
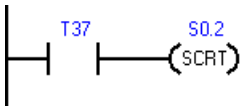
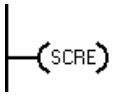
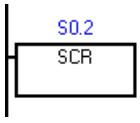
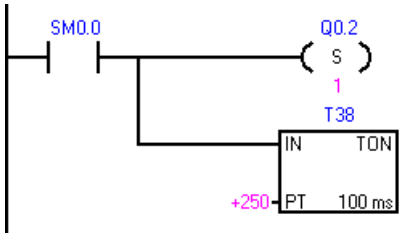
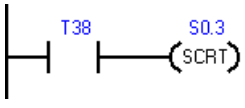
SCR operations

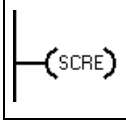
- SCR (Load SCR) marks the beginning of an SCR segment, and the SCRE (End SCR) marks the end of an SCR program segment. All logic between the SCR and the SCRE instructions is dependent upon the value of the S stack for its execution. Logic between SCRE and the next SCR instruction is not dependent on the value of the S stack.
- SCRT (SCR Transition) transfers control from an active SCR segment to another SCR segment.

  Execution of the SCR transition instruction, when it has power flow, will reset the S bit of the currently active SCR segment and will set the S bit of the referenced segment. Resetting the S bit of the active segment does not affect the S stack at the time the SCR Transition instruction executes. Consequently, the SCR segment remains energized until it is exited.
- The STL only instruction CSRE (Conditional SCR End) exits an active SCR segment without executing the instructions between the CSRE and the SCRE (SCR End) instructions. The Conditional SCR End instruction does not affect any S bit nor does it affect the S stack.

Example: SCR sequential control flow

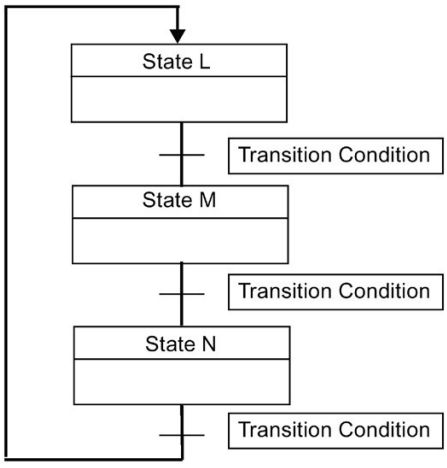
In the following sample program, the first scan bit SM0.1, is used to set S0.1, which will be the active State 1 on the first scan. After a 2-second delay, T37 causes a transition to State 2. This transition deactivates the State 1 SCR (S0.1) segment and activates the State 2 SCR (S0.2) segment.

LAD		STL
	On the first scan enable state 1.	**Network 1** **LD SM0.1** **S S0.1, 1**
	Beginning of state 1 control region.	**Network 2** **LSCR S0.1**
	Control the signals for street 1: 1. Set: Turn on the red light. 2. Reset: Turn off the yellow and green lights. 3. Start a 2-second timer.	**Network 3** **LD SM0.0** **S Q0.4, 1** **R Q0.5, 2** **TON T37, +20**
	After a 2 second delay, transition to state 2.	**Network 4** **LD T37** **SCRT S0.2**
	End of SCR region for state 1.	**Network 5** **SCRE**
	Beginning of state 2 control region.	**Network 6** **LSCR S0.2**
	Control the signals for street 2: 1. Set: Turn on the green light. 2. Start a 25-second timer.	**Network 7** **LD SM0.0** **S Q0.2, 1** **TON T38, +250**
	After a 25 second delay, transition to state 3.	**Network 8** **LD T38** **SCRT S0.3**

LAD		STL
	End of SCR region for state 2.	**Network 9** **SCRE**

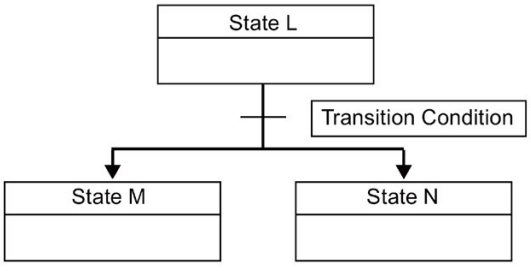
Sequential control flow

A process with a well-defined sequence of steps is easy to model with SCR segments. For example, consider a cyclical process, with 3 steps, that should return to the first step when the third has completed.



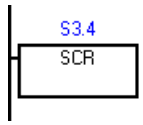
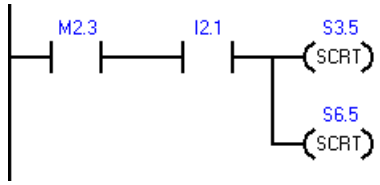
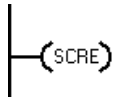
Divergent control flow

In many applications, a single stream of sequential states must be split into two or more different streams. When a control stream diverges into multiple streams, all outgoing streams must be activated simultaneously.



The divergence of control streams can be implemented in an SCR program by using multiple SCRT instructions enabled by the same transition condition, as shown in the following example.

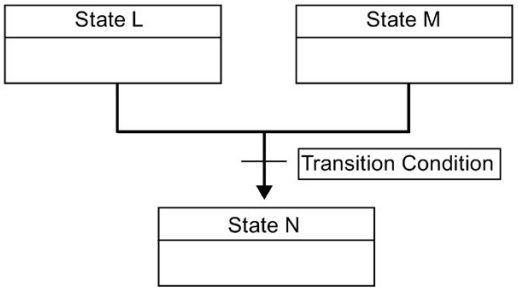
Example: SCR divergent flow control

LAD		STL
	Beginning of state L control region	**Network 1** **LSCR S3.4**
	S3.5: Transition to state M S6.5: Transition to state N	**Network 2** **LD M2.3** **A I2.1** **SCRT S3.5** **SCRT S6.5**
	End of the state region for state L	**Network 3** **SCRE**

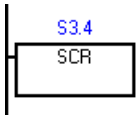
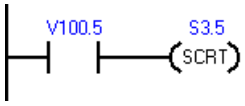
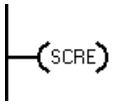
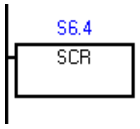
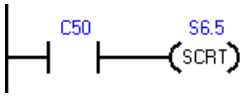
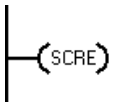
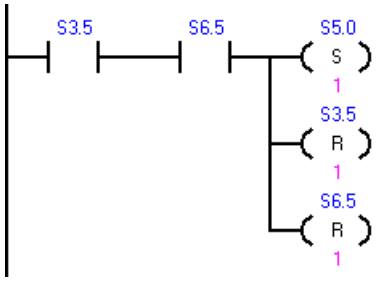
Convergent flow control

When streams converge, all incoming streams must be complete before the next state is executed.

The convergence of control streams can be implemented in an SCR program by making the transition from state L to state L' and by making the transition from state M to state M'. When both SCR bits representing L' and M' are true, state N is enabled as shown in the following example.

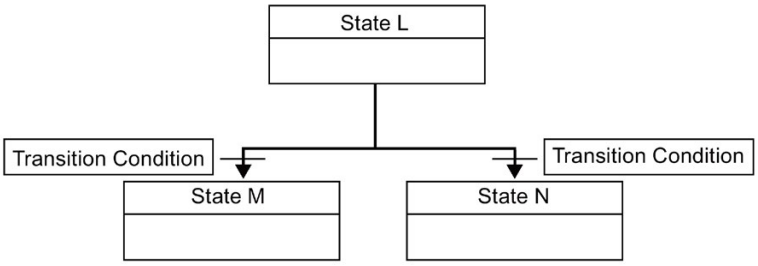


Example: SCR convergent flow control

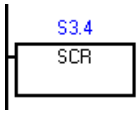
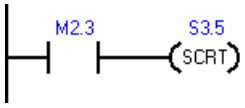
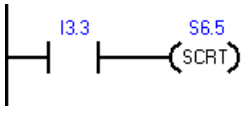
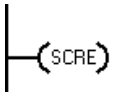
LAD		STL
	Beginning of state L control region	**Network 1** **LSCR S3.4**
	Transition to State L'	**Network 2** **LD V100.5** **SCRT S3.5**
	End of SCR region for state L	**Network 3** **SCRE**
	Beginning of state M control region	**Network 4** **LSCR S6.4**
	Transition to state M'	**Network 5** **LD C50** **SCRT S6.5**
	End of SCR region for state M	**Network 6** **SCRE**
	When both State L' and State M' are activated: • Enable State N (S5.0) • Reset State L' (S3.5) • Reset State M' (S6.5)	**Network 7** **LD S3.5** **A S6.5** **S S5.0, 1** **R S3.5, 1** **R S6.5, 1**

Divergence of a control stream, depending on transition conditions

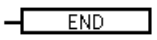
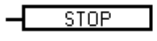
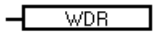
In other situations, a control stream might be directed into one of several possible control streams, depending upon which transition condition becomes true first.



Example: SCR divergent flow control, depending of transition conditions

LAD		STL
	Beginning of state L control region	**Network 1** **LSCR S3.4**
	Transition to state M	**Network 2** **LD M2.3** **SCRT S3.5**
	Transition to state N	**Network 3** **LD I3.3** **SCRT S6.5**
	End of SCR region for state L	**Network 4** **SCRE**

7.13.4 END, STOP, and WDR (watchdog timer reset)

LAD	FBD	STL	Description
		**END**	The conditional END instruction terminates the current scan based upon the condition of the preceding logic. You can use the conditional END instruction in the main program, but you cannot use it in either subroutines or interrupt routines.
		**STOP**	The conditional STOP instruction terminates the execution of your program by causing a transition of the CPU from RUN to STOP mode. If the STOP instruction is executed in an interrupt routine, the interrupt routine is terminated immediately and all pending interrupts are ignored. Remaining actions in the current scan cycle are completed, including execution of the main user program. The transition from RUN to STOP mode is made at the end of the current scan.
		**WDR**	The watchdog reset instruction retriggers the system watchdog timer and adds 500 milliseconds to the time allowed for the scan to complete before a watchdog timeout error occurs.

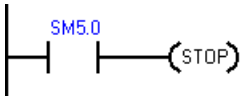
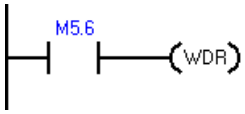
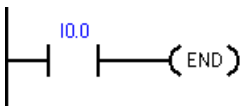
Watchdog timer operation

When the CPU is in RUN mode, the duration of the main scan is limited to 500 milliseconds, by default. If the duration of the main scan exceeds 500 milliseconds, then the CPU automatically transitions to STOP mode and the non-fatal error 001AH (Scan watchdog timeout) is issued.

You can extend the duration of the Main Scan by executing the Watchdog Reset (WDR) instruction in your program. The scan watchdog timeout period is reset to 500 milliseconds each time the WDR instruction is executed.

However, there is an absolute maximum main scan duration of 5 seconds. The CPU will unconditionally transition to STOP mode if the current scan duration reaches 5 seconds.

Example: STOP, END, and WDR (Watchdog reset) instructions

LAD		STL
	When an I/O error is detected, force the transition to STOP mode.	**Network 1** **LD SM5.0** **STOP**
	When M5.6 is ON, execute the watchdog reset instruction to extend the allowed scan time by 500 milliseconds.	**Network 1** **LD SM5.6** **WDR**
	When I0.0 is ON, terminate the current scan.	**Network 1** **LD I.0** **END**

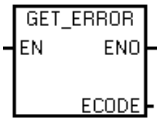
Note

If you expect your scan time to exceed 500 ms, or if you expect a burst of interrupt activity that prevents returning to the main scan for more than 500 ms, you should use the watchdog reset instruction to retrigger the watchdog timer.

Use the watchdog reset instruction carefully. If program execution loops prevent scan completion or excessively delay the completion of the scan, then the following processes are inhibited until the scan cycle is completed.

- Communications (except Freeport mode)
- I/O updating (except Immediate I/O)
- Forced values updating
- SM bit updating (SM0, SM5 to SM29 are not updated)
- RUN-time diagnostics
- STOP instruction, when used in an interrupt routine

7.13.5 GET_ERROR (Get non-fatal error code)

LAD / FBD	STL	Description
	**GERR ECODE**	The get non-fatal error code instruction stores the CPU's current non-fatal error code in the location assigned to ECODE. After the error code is stored, the non-fatal error code is cleared in the CPU.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
ECODE	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC

Non-fatal run-time errors also affect certain special memory error flag addresses that can be evaluated along with the GET_ERROR instruction to determine the cause of a run-time fault. In the event that the generic error flag SM4.3 = 1 (Run-time programming problem) is active, a GET_ERROR execution can be used to identify the specific error.

Non-fatal error code 0000H indicates that no actual error currently exists. In the case of a temporary run-time non-fatal error, a GET_ERROR (ECODE output) produces a non-zero error value and then the next program scan can produce a zero ECODE value.

You should use compare logic to save the ECODE value in another memory location. Your program can then test the saved error code value and begin a programmatic reaction.

Note

The error codes for the ECODE output are listed in the PLC non-fatal error codes table (see reference below). The error code values are in hexadecimal (16#xxxx).

See Also

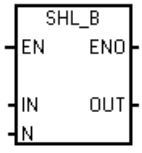
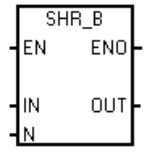
PLC non-fatal error codes (Page 860)

PLC non-fatal error SM flags (Page 862)

7.14 Shift and rotate

7.14.1 Shift and rotate

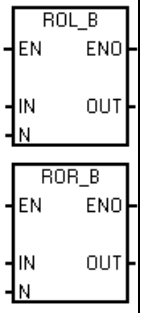
Shift instructions (only the byte size LAD box is illustrated, the others are similar)

LAD / FBD	STL	Shift type	Description
	**SLB OUT, N**	Shift left byte	The shift instructions shift the bit values of input value IN right or left by the bit position shift count N and load the result in the memory location assigned to OUT. The shift instructions fill empty bit positions with zero as each bit is shifted out. If the shift count N is greater than or equal to the maximum allowed (8 for byte operations, 16 for word operations, and 32 for double word operations), the value is shifted the maximum number of times for the operation. If the shift count is greater than 0, the overflow memory bit SM1.1 is set to the value of the last bit shifted out. The SM1.0 zero memory bit is set if the result of the shift operation is zero. Byte operations are unsigned. For word and double word operations, the sign bit is shifted when you use signed data values.
	**SRB OUT, N**	Shift right byte	
SHL_W SHR_W	**SLW OUT, N** **SRW OUT, N**	Shift left word Shift right word	
SHL_DW SHR_DW	**SLD OUT, N** **SRD OUT, N**	Shift left double word Shift right double word	

Non-fatal errors with ENO=0	SM bits affected
• 0006H Indirect address	• SM1.0 Result of operation = zero • SM1.1 Overflow (last bit shifted out)

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

Rotate instructions (only the byte size LAD box is illustrated, the others are similar)

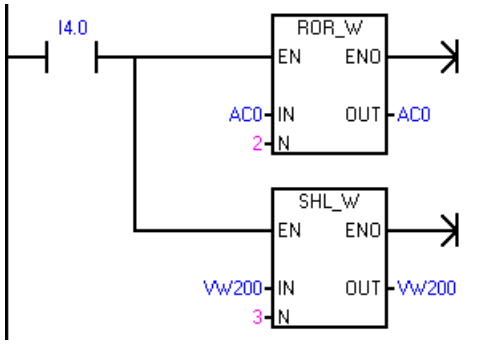
LAD / FBD	STL	Rotate type	Description
	**RLB OUT, N** **RRB OUT, N**	Rotate left byte Rotate right byte	The rotate instructions rotate the bit values of input value IN right or left by the bit position rotate count N and load the result in the memory location assigned to OUT. The rotate operation is circular. If the rotate count is greater than or equal to the maximum for the operation (8 for a byte operation, 16 for a word operation, or 32 for a double-word operation), the CPU performs a modulo operation on the rotate count to obtain a valid shift count before the rotation is executed. This result is a shift count of 0 to 7 for byte operations, 0 to 15 for word operations, and 0 to 31 for double-word operations. If the rotate count is 0, a rotate operation is not performed. If the rotate operation is performed, the overflow bit SM1.1 is set to the value of the last bit rotated out. If the rotate count is not an integer multiple of 8 (for byte operations), 16 (for word operations), or 32 (for double-word operations), the last bit value rotated out is copied to the overflow memory bit SM1.1. The zero memory bit SM1.0 is set when the value to be rotated is zero. Byte operations are unsigned. For word and double word operations, the sign bit is rotated when you use signed data types.
ROL_W ROR_W	**RLW OUT, N** **RRW OUT, N**	Rotate left word Rotate right word	
ROL_DW ROR_DW	**RLD OUT, N** **RRD OUT, N**	Rotate left double word Rotate right double word	

Non-fatal errors with ENO=0	SM bits affected
• 0006H Indirect address	• SM1.0 Result of operation = zero • SM1.1 Overflow (last bit shifted out)

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

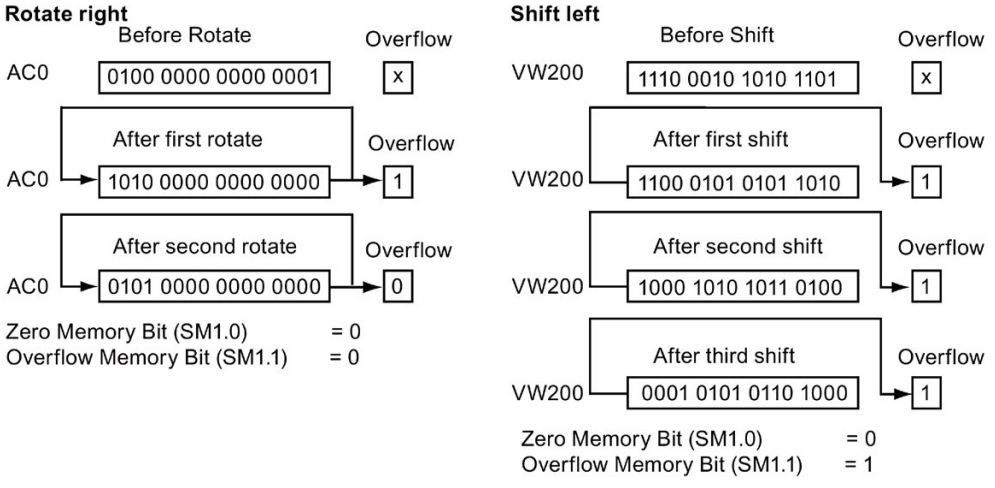
Example: Shift and Rotate instructions

LAD



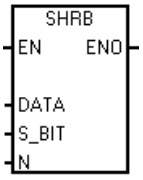
STL

```
Network 1
LD I4.0
RRW AC0, 2
SLW VW200, 3
```



7.14.2 Shift register bit

The Shift Register Bit instruction shifts a bit value into the Shift Register. This instruction provides an easy method for sequencing and controlling product flow or data. Use this instruction to shift the entire register one bit, once per scan.

LAD / FBD	STL	Description
	**SHRB DATA, S_bit, N**	The shift register bit instruction shifts the bit value of DATA into the Shift Register. S_BIT assigns the location of the least significant bit of the shift register. N assigns the length of the Shift Register and the direction of the shift (Shift Plus = N, Shift Minus = -N). Each bit value shifted out by the SHRB instruction is copied to the overflow memory bit SM1.1. The shift register bits are defined by both the least significant bit S_BIT location and the number of bits assigned by the length N.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • 0092H Error in count field	• SM1.1 Overflow (last bit shifted out)

Input / output	Data type	Operand
DATA, S_bit	BOOL	I, Q, V, M, SM, S, T, C, L
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

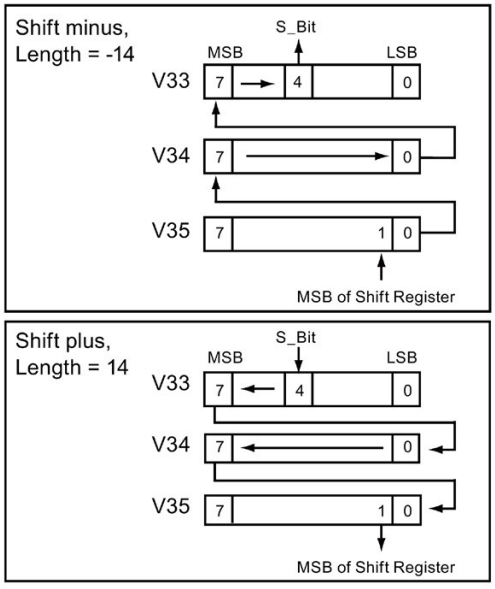
Use the following equation to compute the address of the most significant bit of the Shift Register (MSB.b):

MSB.b = *[(Byte of S_BIT) + ([N] - 1 + (bit of S_BIT)) / 8].[remainder of the division by 8]*

For example: if S_BIT is V33.4 and N is 14, the following calculation shows that the MSB.b is V35.1.

MSB.b = V33 + ([14] - 1 +4)/8
= V33 + 17/8
= V33 + 2 with a remainder of 1
= V35.1

The following figure shows bit shifting for negative and positive values of N.



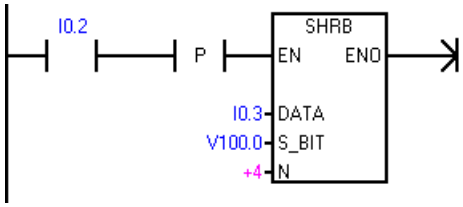
A Shift Minus operation is indicated by a negative value of length N. The input value of DATA shifts into the most significant bit of the shift register, and shifts out of the least significant bit location assigned by S_BIT. The data shifted out is then placed in the overflow memory bit SM1.1.

A Shift Plus operation is indicated by a positive value of length N. The input value of DATA shifts into the least significant bit location assigned by S_BIT and out of the most significant bit of the Shift Register. The bit value shifted out is then placed in the overflow memory bit SM1.1.

The maximum length of the shift register assigned by N is 64 bits (positive or negative).

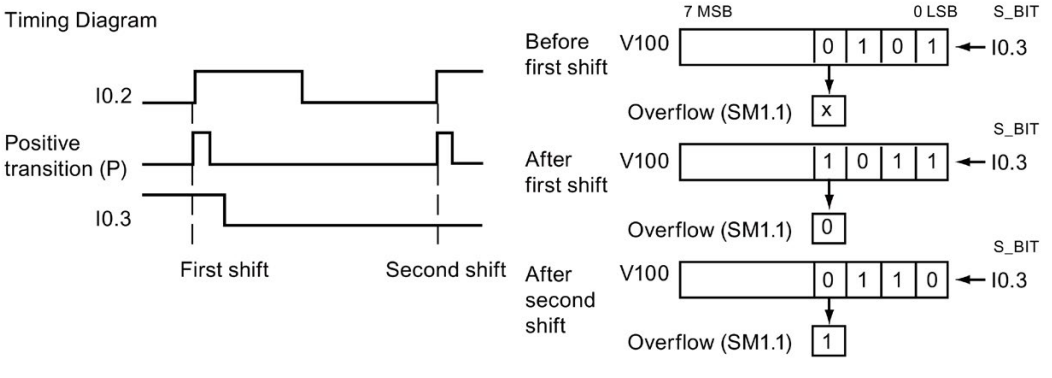
Example: SHRB instruction

LAD



STL

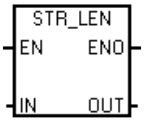
```
Network 1
LD I0.2
EU
SHRB I0.3, V100.0, +4
```



7.15 String

7.15.1 String (Get length, copy, and concatenate)

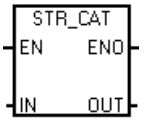
SLEN (String length)

LAD / FBD	STL	Description
	**SLEN IN, OUT**	The string length instruction returns the length in bytes of the string specified by IN. Note: Because Chinese characters are not represented by a single byte, the STR_LEN function does not return the number of characters in a string containing Chinese characters.

Error conditions with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range	None

Input / output	Data type	Operand
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC

SCPY and SCAT (String copy and string concatenate)

LAD / FBD	STL	Description
	SCPY IN, OUT	The copy string instruction copies the string assigned by IN to the string assigned by OUT.
	**SCAT IN, OUT**	The concatenate string instruction appends the string assigned by IN to the end of the string assigned by OUT.

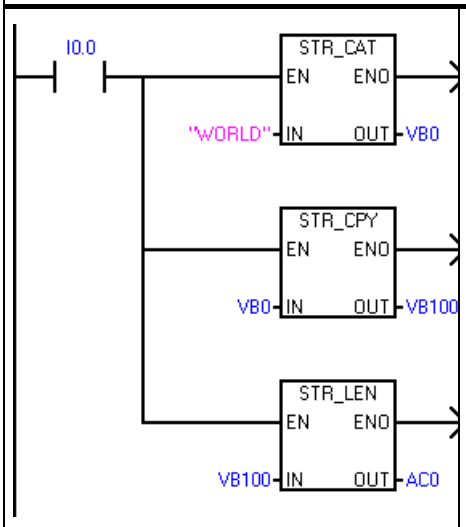
Note: The STR_CPY and STR_CAT instructions operate on bytes and not characters. Because Chinese characters are not represented by a single byte, unexpected results can occur with the STR_CPY and STR_CAT instructions with strings containing Chinese characters. If you know the number of bytes that a character string occupies, you can use the STR_CPY and STR_CAT instructions with the correct number of bytes.

Error conditions with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range	None

Inputs/Outputs	Data types	Operands
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
OUT	STRING	VB, LB, *VD, *LD, *AC

Example: Concatenate string, copy string, and string length Instructions

LAD



1. Append the string "WORLD" to the string at VB0.
2. Copy the string at VB0 to a new string at VB100.
3. Get the length of the string that starts at VB100.

STL

```
Network 1
LD I0.0
SCAT "WORLD", VB0
SCPY VB0, VB100
SLEN VB100, AC0
```

Before executing the program

VB0						VB6
6	'H'	'E'	'L'	'L'	'O'	' '

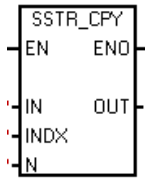
After executing the program

VB0											VB11
11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'

VB100											VB111
11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'

AC0
11

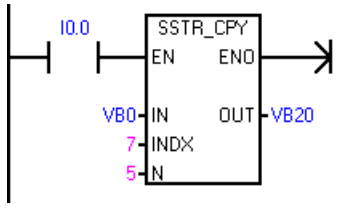
7.15.2 Copy substring from string

LAD / FBD	STL	Description
	**SSCPY IN, INDX, N, OUT**	The copy substring from string instruction copies the assigned number of characters N from the string specified by IN, starting at the index INDX, to a new string assigned by OUT. Note: The SSTR_CPY instruction operates on bytes and not characters. Because Chinese characters are not represented by a single byte, unexpected results can occur with the SSTR_CPY instruction with strings containing Chinese characters. If you know the number of bytes that a character string occupies, you can use the SSTR_CPY instruction with the correct number of bytes.

Non-fatal errors with ENO=0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • 009BH Index = 0	None

Input / output	Data type	Operand
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
OUT	STRING	VB, LB, *VD, *LD, *AC
INDX, N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

Example: Copy substring instruction

AD		STL
	Starting at the seventh byte after the byte count in the string at VB0, copy 5 bytes to a new string at VB20.	**Network 1** **LD I0.0** **SSCPY VB0, 7, 5, VB20**

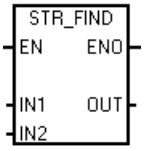
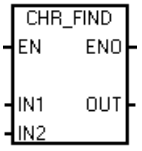
Before executing the program

VB0											VB11
11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'

After executing the program

VB20					VB25
5	'W'	'O'	'R'	'L'	'D'

7.15.3 Find string and first character within string

LAD / FBD	STL	Description
	**SFND IN1, IN2, OUT**	STR_FIND searches for the first occurrence of the string IN2 within the string IN1. The search begins at the starting position assigned by the initial value of OUT (which must be in the range of 1 to the IN1 string length before STR_FIND execution). If a sequence of characters is found that matches exactly the string IN2, the position of the first character in the sequence within the IN1 string is written to OUT. If the string IN2 was not found in the string IN1, then OUT is set to 0.
	**CFND IN1, IN2, OUT**	CHR_FIND searches the string IN1 for the first occurrence of any character from the character set in the string IN2. The search begins at starting position assigned by the initial value of OUT (which must be in the range of 1 to the IN1 string length before CHR_FIND execution). If a matching character is found, then the position of the character is written to OUT. If no matching character is found, OUT is set to 0.

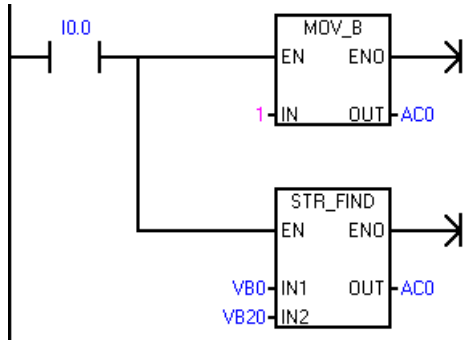
Note: Because Chinese characters are not represented by a single byte, and the string instructions operate on bytes and not characters, unexpected results can occur with the STR_FIND and CHR_FIND instructions with strings containing Chinese characters.

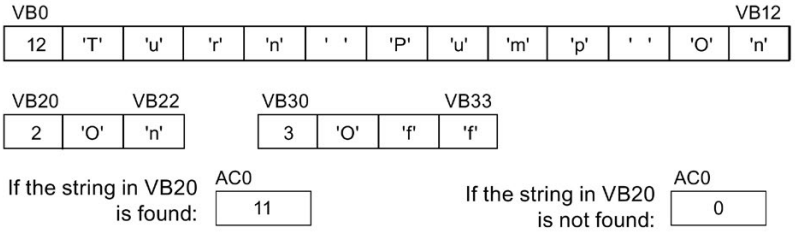
Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • 009BH Index = 0	None

Input / output	Data type	Operand
IN1, IN2	STRING	VB, LB, *VD, *LD, *AC, Constant String
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC

Example: Find string within string instruction

A string stored at VB0 is used as a command for turning a pump on or off. A string "On" is stored at VB20 and a string "Off" is stored at VB30. The result of the find string within string instruction is stored in AC0 (the OUT parameter). If the result is not 0, then the string 'On' was found in the command string (VB12).

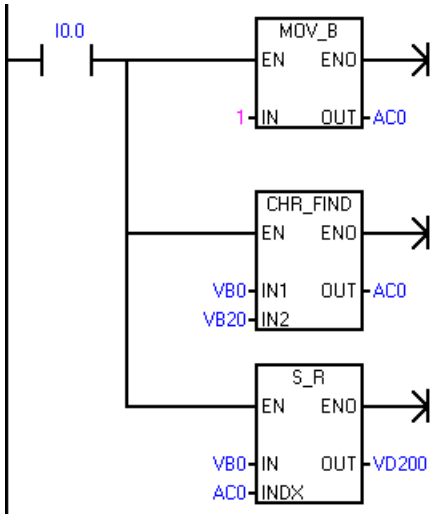
LAD		STL
	1. Set AC0 to 1. (AC0 is used as the OUT parameter.) 2. Search the string at VB0 for the string at VB20 ('On'), starting at the first position (AC0=1).	**Network 1** **LD I0.0** **MOVB 1, AC0** **SFND VB0, VB20, AC0**



Example: Find character within string instruction

A string stored at VB0 contains the temperature. The string constant at IN1 provides all the numeric characters (0-9, +, and -) that can identify a temperature number in a string. CHR_FIND execution finds the starting position of the character "9" in the VB0 string and then S_R execution converts the real number characters into a real number value. VD200 is used to store the real-number value of the temperature.

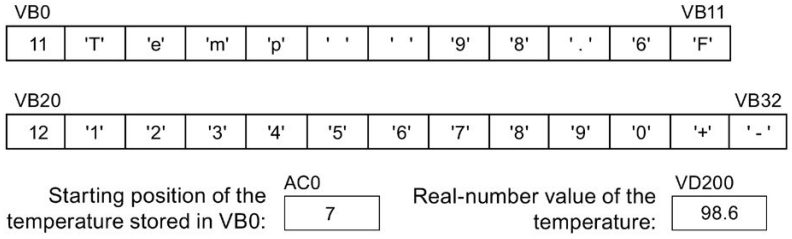
LAD



1. Set AC0 to 1. (AC0 is used as the OUT parameter and points to the first character position in the string.)
2. Find the first numeric character in the string stored at VB0.
3. Convert the string to a real number value.

STL

```
Network 1
LD I0.0
MOVB 1, AC0
CFND VB0, VB20, AC0
STR VB0, AC0, VD200
```

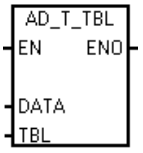


Starting position of the temperature stored in VB0: AC0 = 7

Real-number value of the temperature: VD200 = 98.6

7.16 Table

7.16.1 Add to table

LAD / FBD	STL	Description
	**ATT DATA, TBL**	The add to table instruction adds word values DATA to a table TBL. The first value in the table is the maximum table length TL. The second value is the entry count EC, which stores the number of entries in the table and is updated automatically. New data are added to the table after the last entry. Each time new data are added to the table, the entry count is incremented. A table can have up to 100 data entries.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • SM1.4 Table overfill	• SM1.4 Set to 1 if you try to overfill the table

Input / output	Data type	Operand
DATA	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
TBL	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, *VD, *LD, *AC

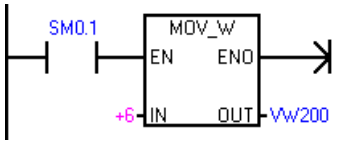
Note

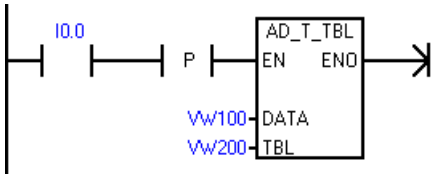
To create a table, first make an entry for the maximum number of table entries. If you do not do this, then you cannot make any entries in the table.

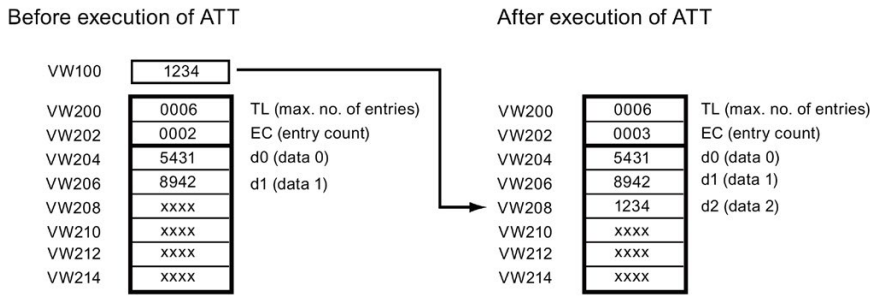
Edge trigger instructions must activate all table read and table write instructions.

Example: Add to Table instruction

LAD		STL

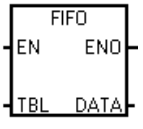
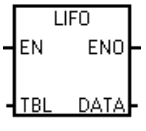
	On the first scan only, load the maximum table length 6 to VW200.	`Network 1`
		`LD SM0.1`
		`MOVW +6, VW200`

	When I0.0 makes a transition to 1, add a third data value (from VW100) to the table at VW200. Two data entries were previously stored in the table which can hold up to six entries.	`Network 2`
		`LD I0.0`
		`ATT VW100, VW200`



7.16.2 First-in-first-out and last-in-first-out

Table 7- 20 FIFO and LIFO instructions

LAD / FBD	STL	Description
	`FIFO TBL, DATA`	The first-in-first-out instruction moves the oldest (or first) entry in a table to an output memory address, by removing the first entry in the assigned table (TBL) and moving the value to the location assigned by DATA. All other entries of the table are shifted up one location. The entry count in the table is decremented for each FIFO execution.
	`LIFO TBL, DATA`	The last-in-first-out instruction moves the newest (or last) entry in the table to an output memory address, by removing the last entry in the table (TBL) and moving the value to the location assigned by DATA. The entry count in the table is decremented for each LIFO execution.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range • SM1.5: Attempt to remove entry from empty table	• SM1.5: Attempt to remove entry from empty table

Input / output	Data type	Operand
TBL	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, *VD, *LD, *AC
DATA	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC

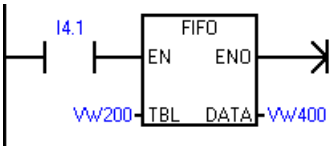
Note

All table read and table write instructions must be activated by edge trigger instructions.

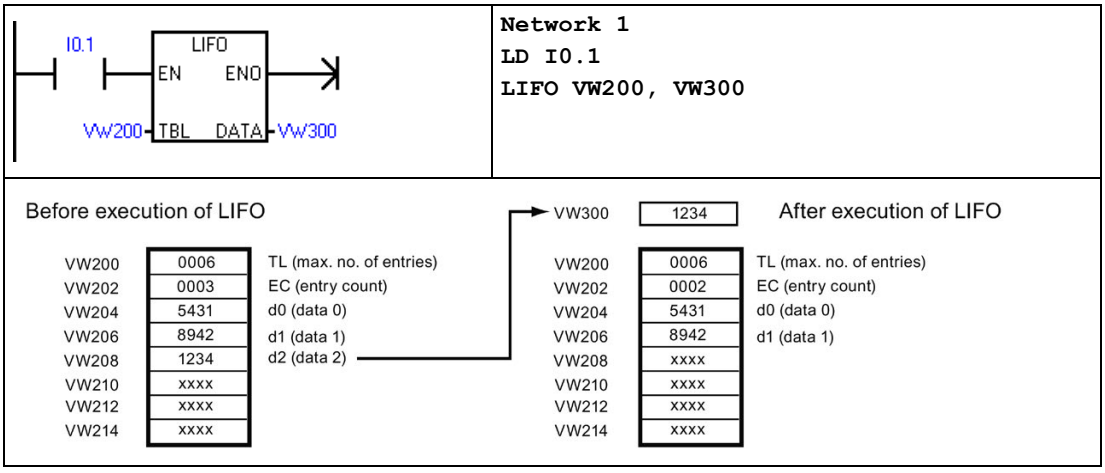
To create a table, you must first make an entry for the maximum number of table entries before any entries can be put in the table.

Example: FIFO instruction

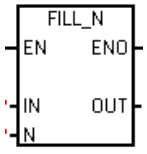
LAD	STL

	`Network 1`
	`LD I4.1`
	`FIFO VW200, VW400`

Example: LIFO instruction



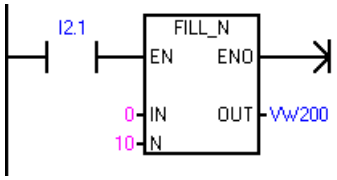
7.16.3 Memory fill

LAD / FBD	STL	Description
	`FILL IN, OUT, N`	The memory fill instruction writes N consecutive words, beginning at address OUT, with the word value contained in address IN. N has a range of 1 to 255.

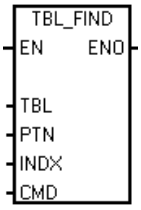
Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range	None

Input / output	Data type	Operand
IN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AQW, *VD, *LD, *AC

Example: Memory fill instruction

LAD	STL
	**Network 1** **LD I2.1** **FILL +0, VW200, 10**

7.16.4 Table find

LAD / FBD	STL	Description
	**FND= TBL, PTN, INDX** **FND<> TBL, PTN, INDX** **FND< TBL, PTN, INDX** **FND> TBL, PTN, INDX**	The Table Find instruction searches a table for data that matches your search criteria. The Table Find instruction searches the table TBL, starting with the table entry INDX, for the data value or pattern PTN that matches the search criteria defined by CMD. The command parameter CMD is given a numeric value of 1 to 4 that corresponds to =, <>, <, and >, respectively. If a match is found, the INDX points to the matching entry in the table. To find the next matching entry, the INDX must be incremented before invoking the table find instruction again. If a match is not found, the INDX has a value equal to the entry count. A table can have up to 100 data entries. The data entries (area to be searched) are numbered from 0 to a maximum value of 99.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • 0091H Operand out of range	None

Input / output	Data type	Operand
TBL	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, *VD, *LD, *AC
PTN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant

Input / output	Data type	Operand
INDX	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
CMD	BYTE	Constant: • 1 = Equal (=) • 2 = Not Equal (<>) • 3 = Less Than (<) • 4 = Greater Than (>)

Note

When you use the table find instruction with tables generated with the Add-to-table, Last-in-first-out, and First-in-first-out instructions, the entry count and the data entries correspond directly. The maximum-number-of-entries word required for the Add-to-table, Last-in-first-out, or First-in-first-out instructions is not required by the Table find instruction. See the following figure.

Consequently, you should set the TBL operand of a Find instruction to one-word address (two bytes) higher than the TBL operand of a corresponding the Add-to-table, Last-in-first-out, or First-in-first-out instruction.

Differences in table formats for ATT, LIFO, FIFO, and TBL_FIND instructions

Table format for ATT, LIFO, and FIFO

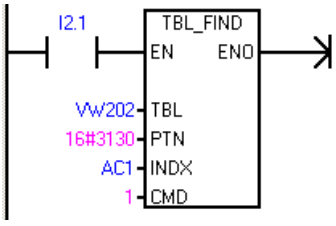
VW200	0006	TL (max. no. of entries)
VW202	0006	EC (entry count)
VW204	xxxx	d0 (data 0)
VW206	xxxx	d1 (data 1)
VW208	xxxx	d2 (data 2)
VW210	xxxx	d3 (data 3)
VW212	xxxx	d4 (data 4)
VW214	xxxx	d5 (data 5)

Table format for TBL_FIND

VW202	0006	EC (entry count)
VW204	xxxx	d0 (data 0)
VW206	xxxx	d1 (data 1)
VW208	xxxx	d2 (data 2)
VW210	xxxx	d3 (data 3)
VW212	xxxx	d4 (data 4)
VW214	xxxx	d5 (data 5)

Example: Table Find instruction

LAD	STL

	`Network 1`
	`LD I2.1`
	`FND= VW202, 16#3130, AC1`

When I2.1 is on, search the table for a value equal to 3130 HEX.

VW202	0006	EC (entry count)
VW204	3133	d0 (data 0)
VW206	4142	d1 (data 1)
VW208	3130	d2 (data 2)
VW210	3030	d3 (data 3)
VW212	3130	d4 (data 4)
VW214	4541	d5 (data 5)

If the table was created using ATT, LIFO, and FIFO instructions, VW200 contains the maximum number of allowed entries and is not required by the Find instructions.

AC1 = 0 — AC1 must be set to 0 to search from the top of table.

Execute a table search

AC1 = 2 — AC1 contains the data entry number corresponding to the first match in the table (d2).

AC1 = 3 — Increment the INDX by one, before searching the remaining enties of the table.

Execute a table search

AC1 = 4 — AC1 contains the data entry number corresponding to the second match in the table (d4).

AC1 = 5 — Increment the INDX by one, before searching the remaining enties of the table.

Execute a table search

AC1 = 6 — AC1 contains a value equal to the entry count. The entire table has been searched without finding another match.

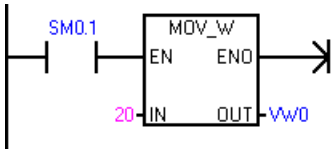
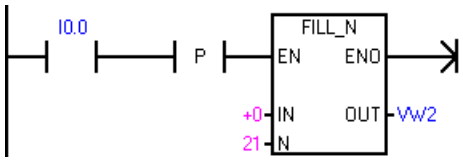
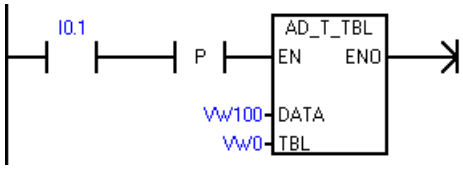
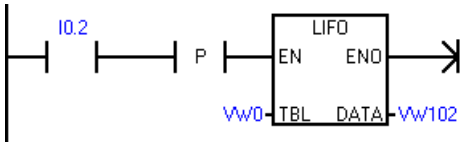
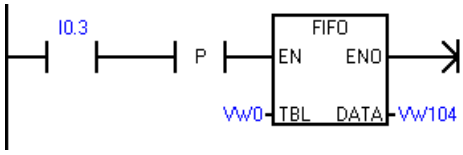
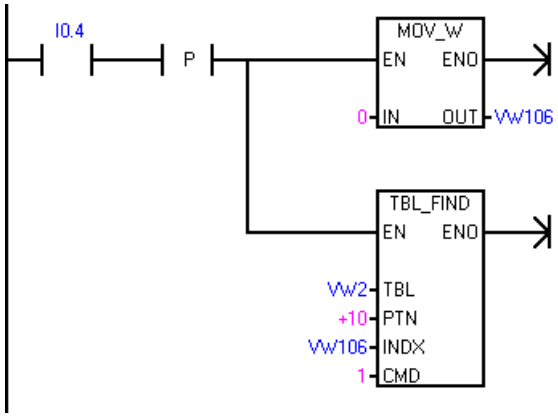
AC1 = 0 — Before the table can be searched again, the INDX value must be reset to 0.

Example: Table

The following program creates a table with 20 entries. The first memory location of the table contains the length of the table (in this case 20 entries). The second memory location shows the current number of table entries. The other locations contain the entries. A table can have up to 100 entries. It does not include the parameters defining the maximum length of the table or the actual number of entries (here VW0 and VW2). The actual number of entries in the table (here VW2) is automatically incremented or decremented by the CPU with every command.

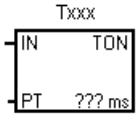
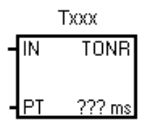
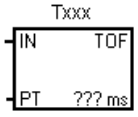
Before you work with a table, assign the maximum number of table entries. Otherwise, you cannot make entries in the table. Also, be sure that all read and write commands are activated with edge instructions.

To search the table, the index (VW106) must set to 0 before doing the find. If a match is found, the index will have the table entry number, but if no match is found, the index will match the current entry count for the table (VW2).

LAD		STL
	Create table with 20 entries starting with memory location 4. • On the first scan, define the maximum length of the table.	**Network 1** **LD SM0.1** **MOVW +20, VW0**
	Reset table with input I0.0. • On the rising edge of I0.0, fill memory locations from VW2 with "+0".	**Network 2** **LD I0.0** **EU** **FILL +0, VW2, 21**
	Write value to table with input I0.1. • On the rising edge of I0.1, copy value of memory location VW100 to table.	**Network 3** **LD I0.1** **EU** **ATT VW100, VW0**
	Read last table value with input I0.2. • Move the last table value to location VW102. This reduces the number of entries. On the rising edge of I0.2, move last table value to VW102.	**Network 4** **LD I0.2** **EU** **LIFO VW0, VW102**
	Read first table value with input I0.3. • Move the first table value to location VW104. This reduces the number of entries. On the rising edge of I0.3, move first table value to VW104.	**Network 5** **LD I0.3** **EU** **FIFO VW0, VW104**
	Search table for the first location that has a value of 10. • On the rising edge of I0.4, reset index pointer. • Find a table entry that equals 10.	**Network 6** **LD I0.4** **EU** **MOVW +0, VW106** **FND= VW2, +10, VW106**

7.17 Timer

7.17.1 Timer instructions

LAD / FBD	STL	Description
	`TON Txxx, PT`	Use **TON** On-delay timers for a timing a single time interval.
	`TONR Txxx, PT`	Use **TONR** On-delay retentive timers for accumulating the time value of many timed intervals.
	`TOF Txxx, PT`	Use the **TOF** Off-delay timer for extending a time interval past an OFF (or FALSE) condition, such as a delay time for cooling a motor.

Input / output	Data type	Operand
Txxx	WORD	Timer number (T0 to T255)
IN	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
PT	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant

Timer resolution

TON, TONR, and TOF timers are available in three resolutions. The resolution is determined by the timer number, as shown below. Each unit of the current value is a multiple of the time base. For example, a count of 50 on a 10 ms timer represents 500 ms of elapsed time.

Your Txxx timer number assignment determines the resolution of the timer. When a valid timer number is assigned, the resolution is displayed in LAD or FBD timer boxes.

Timer number and resolution options

Timer type	Resolution	Maximum value	Timer number
TON, TOF	1 ms	32.767 s	T32, T96
	10 ms	327.67 s	T33-T36, T97-T100
	100 ms	3276.7 s	T37-T63, T101-T255
TONR	1 ms	32.767 s	T0, T64
	10 ms	327.67 s	T1-T4, T65-T68
	100 ms	3276.7 s	T5-T31, T69-T95

Note

Avoid timer number conflicts

You cannot use the same timer number for both a TON and TOF timer. For example, you cannot have both a TON T32 and a TOF T32.

Note

To guarantee a minimum time interval, increase the preset value (PV) by 1.

For example: To ensure a minimum timed interval of at least 2100 ms for a 100-ms timer, set the PV to 22.

TON and TONR timer operation

The TON and TONR instructions begin timing when the enabling input IN is ON. When the current value is equal to or greater than the preset time, the timer bit is set ON.

- The current value of a TON timer is cleared when the enabling input is OFF.
- The current value of the TONR timer is maintained when the enabling input is OFF. You can use the TONR timer to accumulate time when the input IN is ON. Use the Reset instruction (R) to clear the current value of the TONR.
- Both the TON and the TONR timers continue timing after the preset time is reached, and they stop timing at the maximum value of 32,767.

TOF timer operation

The TOF instruction is used to delay turning an output OFF for a fixed period of time after the input turns OFF. When the enabling input turns ON, the timer bit turns ON immediately, and the current value is set to 0. When the input turns OFF, timing begins and continues until the current time equals the preset time.

- When the preset is reached, the timer bit turns OFF and the current value stops incrementing; however, if the enabling input turns ON again before the TOF reaches the preset value, the timer bit remains ON.
- The enabling input must make an ON-OFF transition for a TOF timer to begin timing the OFF-delay time interval.
- If a TOF timer is inside an SCR region and the SCR region is inactive, then the current value is set to 0, the timer bit is turned OFF, and the current value does not increment.

Type	Current >= Preset	State of IN, the enabling input	Power cycle / first scan
TON	Timer bit ON Current value continues timing to 32,767	ON: Current value = timing value OFF: Timer bit OFF, current value = 0	Timer bit = OFF Current value = 0
TONR[1]	Timer bit ON Current value continues timing to 32,767	ON: Current value = timing value OFF: Timer bit and current value maintain last state and value	Timer bit = OFF Current value can be maintained[1]
TOF	Timer bit OFF Current = Preset, stops timing	ON: Timer bit ON, current value = 0 OFF: Timer begins timing after ON-to-OFF transition	Timer bit = OFF Current value = 0
[1] The retentive timer current value can be assigned for retention through a power cycle. See Configuring the retentive ranges for details (Page 137).			

Note

Using the Reset instruction with timer instructions

The TONR timer can only be reset with the Reset (R) instruction.

The TON and TOF timers can be reset by the timer's enable input and also the Reset (R) instruction.

The Reset instruction performs the following actions:

- Timer bit = OFF
- Timer current value = 0
- After a reset, TOF timers require the enable input to make the transition from ON-to-OFF in order restart the OFF-delay timer.

7.17.2 Timer programming tips and examples

Timer types

You can use timers to implement time-based counting functions. The S7-200 instruction set provides three different types of timers.

- On-Delay Timer (TON) for timing a single interval
- Retentive On-Delay Timer (TONR) for accumulating a number of timed intervals
- Off-Delay Timer (TOF) for extending time past an off (or false condition), such as for cooling a motor after it is turned off

Addressing timer values

The meaning of the T number depends on the context in your program.

- "T37" assigned to a timer box identifies which timer is to use.
- "T37" assigned to normally open contacts addresses the Boolean T37 timer bit.
- "T37" assigned to integer operations addresses the T37 current time value, as a data word.

1-millisecond resolution

The 1-ms timers count the number of 1-ms timer intervals that have elapsed since the active 1-ms timer was enabled. The execution of the timer instruction starts the timing; however, the 1-ms timers are updated (timer bit and timer current) every millisecond asynchronous to the scan cycle. In other words, the timer bit and timer current are updated multiple times throughout any scan that is greater than 1 ms.

The timer instruction is used to turn the timer on, reset the timer, or, in the TONR timer, to turn the timer off.

Since the timer can be started anywhere within a millisecond, the preset must be set to one time interval greater than the minimum desired timer interval. For example, to guarantee a timed interval of at least 56 ms using a 1-ms timer, the preset time value should be set to 57.

10-millisecond resolution

The 10-ms timers count the number of 10-ms timer intervals that have elapsed since the active 10-ms timer was enabled. The execution of the timer instruction starts the timing; however the 10-ms timers are updated at the beginning of each scan cycle (in other words, the timer current and timer bit remain constant throughout the scan), by adding the accumulated number of 10-ms intervals (since the beginning of the previous scan) to the current value for the active timer.

Since the timer can be started anywhere within a 10-ms interval, the preset must be set to one time interval greater than the minimum desired timer interval. For example, to guarantee a timed interval of at least 140 ms using a 10-ms timer, the preset time value should be set to 15.

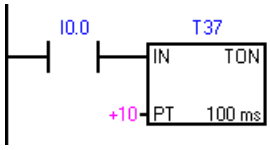
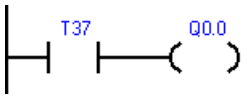
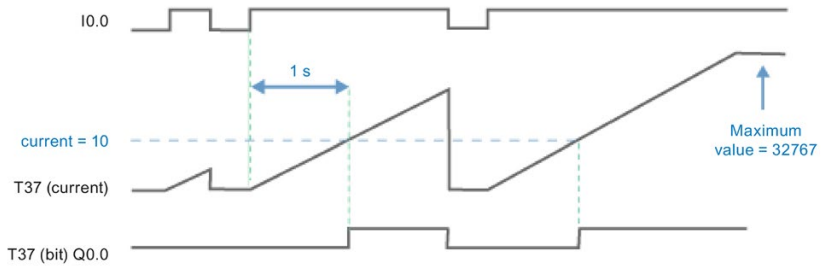
100-millisecond resolution

The 100-ms timers count the number of 100-ms timer intervals that have elapsed since the active 100-ms timer was last updated. These timers are updated by adding the accumulated number of 100-ms intervals (since the previous scan cycle) to the timer's current value when the timer instruction is executed.

The current value of a 100-ms timer is updated only if the timer instruction is executed. Consequently, if a 100-ms timer is enabled but the timer instruction is not executed each scan cycle, the current value for that timer is not updated and it loses time. Likewise, if the same 100-ms timer instruction is executed multiple times in a single scan cycle, the number of 100-ms intervals is added to the timer's current value multiple times, and it gains time. 100-ms timers should only be used where the timer instruction is executed exactly once per scan cycle.

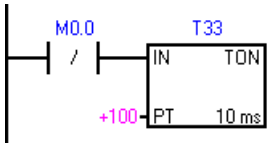
Since the timer can be started anywhere within a 100-ms interval, the preset must be set to one time interval greater than the minimum desired timer interval. For example, to guarantee a timed interval of at least 2100 ms using a 100-ms timer, the preset time value should be set to 22.

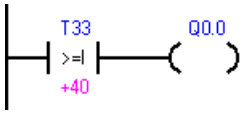
TON On-delay timer example

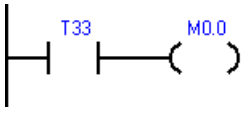
<table>
<tr><th colspan="2">LAD</th><th>STL</th></tr>
<tr><td> 
</td><td>100 ms timer T37 times out after 1 s (10 x 100 ms)
• I0.0 ON = T37 enabled,
• I0.0 OFF = disable and reset T37</td><td>Network 1
LD I0.0
TON T37, +10</td></tr>
<tr><td> 
</td><td>T37 bit is controlled by timer T37</td><td>Network 2
LD T37
= Q0.0</td></tr>
<tr><td colspan="3">Timing Diagram
 
</td></tr>
</table>

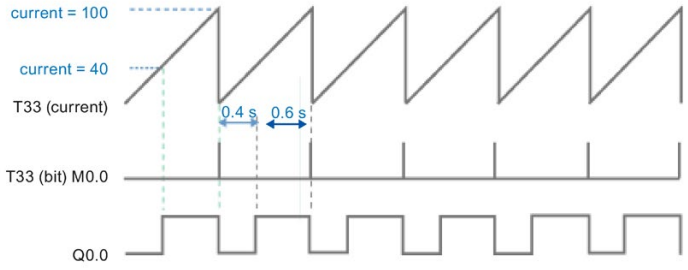
TON self-resetting On-delay timer example

LAD		STL

	10 ms timer T33 times out after 1 s (100 x 10 ms).
	M0.0 pulse is too fast to monitor with Status view.
		`Network 1`
		`LDN M0.0`
		`TON T33, +100`

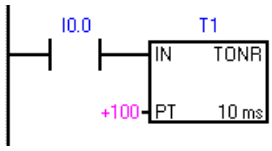
	The Compare contact becomes TRUE at a rate that is visible in Status view.
	Turn ON Q0.0 after (40 x 10 ms) for 40% OFF / 60% ON.
		`Network 2`
		`LDW>= T33, +40`
		`= Q0.0`

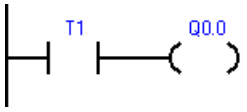
	T33 (bit) pulse is too fast to monitor with Status view.
	Reset the timer with M0.0 after the (100 x 10 ms) period.
		`Network 3`
		`LD T33`
		`= M0.0`

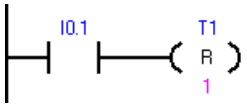
Timing Diagram 		

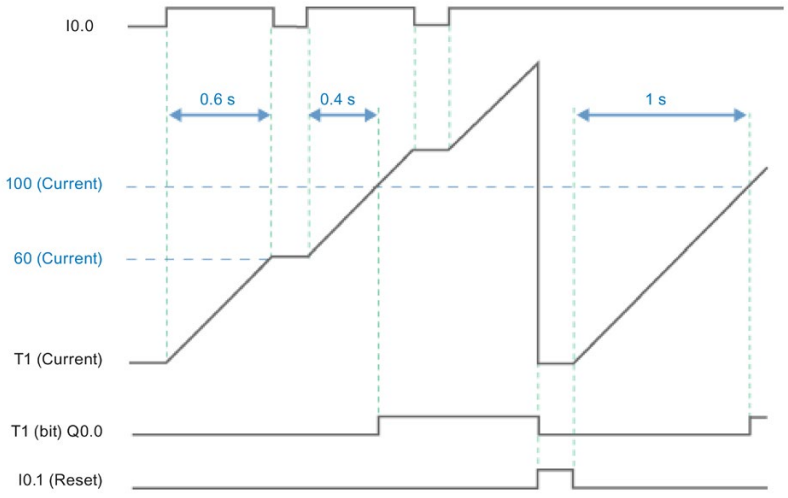
TONR retentive On-delay timer example

LAD		STL

	10 ms TONR timer T1 times out at PT = 1 s (100 x 10 ms).	`Network 1`
		`LD I0.0`
		`TONR T1, +100`

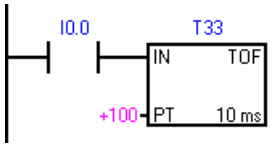
	T1 bit is controlled by timer T1.
	Q0.0 is ON after the timer accumulates a total of 1 second.
		`Network 2`
		`LD T1`
		`= Q0.0`

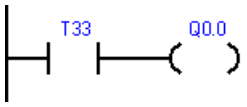
	TONR timers must be reset by a Reset instruction with a T address.
	Reset timer T1 (current value and bit) when I0.1 is on.
		`Network 3`
		`LD I0.1`
		`R T1, 1`

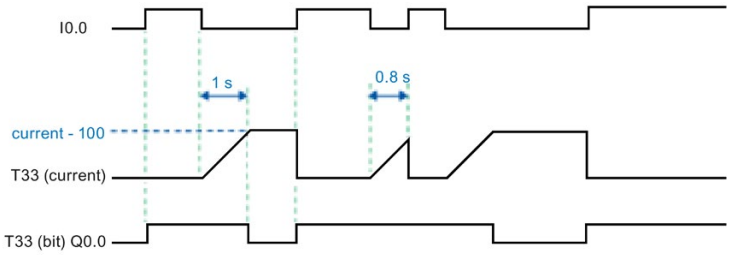
Timing Diagram 		

TOF Off-delay timer example

LAD		STL

	10-ms timer T33 times out after 1 s (100 x 10 ms).
	• I0.0 ON-to-OFF = T33 enabled
	• I0.0 OFF-to-ON=disable and reset T33
		`Network 1`
		`LD I0.0`
		`TOF T33, +100`

	Timer T33 controls Q0.0 through timer contact T33.	`Network 2`
		`LD T33`
		`= Q0.0`

Timing Diagram 		

Effect of timer resolution on when timer bits and current time values are updated

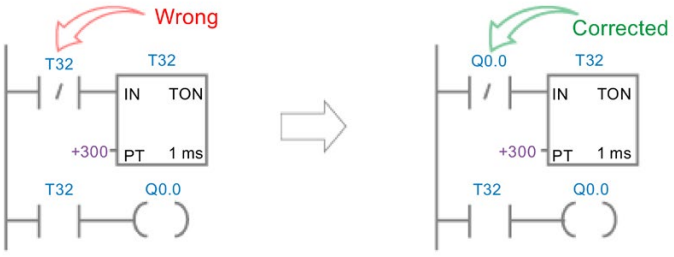
- **1 ms** timer: The timer bit and the current value are updated asynchronous to the scan cycle. For scans greater than 1 ms, the timer bit and the current value are updated multiple times throughout the scan.
- **10 ms** timer: The timer bit and the current value are updated at the beginning of each scan cycle. The timer bit and current value remain constant throughout the scan. Time intervals that accumulate during the scan are added to the current value at the start of each scan.
- **100 ms** timer: The timer bit and current value are updated when the instruction is executed; therefore, ensure that your program executes the instruction for a 100-ms timer only once per scan cycle in order for the timer to maintain the correct timing.

Example: automatically retriggered One-shot timers

The corrected examples use the normally closed contact Q0.0 instead of the timer bit as the timer enabling input. This ensures that output Q0.0 is turned ON for one scan cycle, each time a timer reaches the preset value.

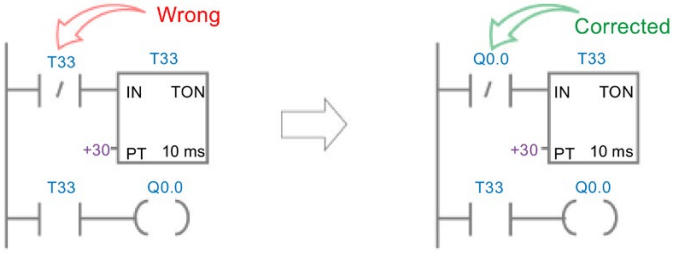
1 ms timer

Q0.0 is turned ON for one scan whenever the timer's current value is updated after the normally closed contact T32 is executed and before the normally open contact T32 is executed.



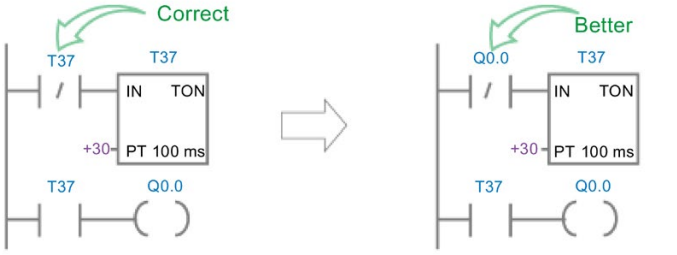
10 ms timer

Q0.0 is never turned ON, because the timer bit T33 is turned ON from the top of the scan to the point where the timer box is executed. Once the timer box has been executed, the timer's current value and its T–bit are set to zero. When the normally open contact T33 is executed, T33 is OFF and Q0.0 is turned OFF.



100 ms timer

Q0.0 is always turned ON for one scan whenever the timer's current value reaches the preset value.



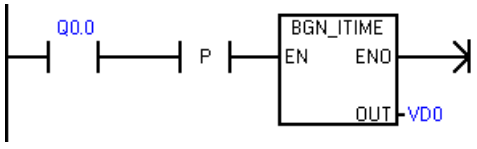
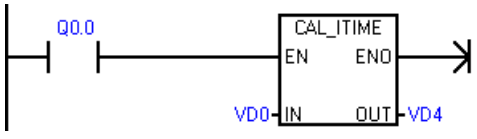
7.17.3 Interval timers

LAD / FBD	STL	Description
	BITIM OUT	The Begin interval time instruction reads the current value of the built-in 1 millisecond counter and stores the value in OUT. The maximum timed interval for a DWORD millisecond value is 2 raised to the 32 power or 49.7 days.
	CITIM IN, OUT	The Calculate interval time instruction calculates the time difference between the current time and the time provided at IN. The difference is stored in OUT. The maximum timed interval for a DWORD millisecond value is 2 raised to the 32 power or 49.7 days. CITIM automatically handles the one millisecond timer rollover that occurs within the maximum interval, depending on when the BITIM instruction was executed.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
IN	DWORD	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
OUT	DWORD	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: Begin and Calculate interval timers

LAD		STL
 Ex1_Interval_time_net1	Capture the time that Q0.0 turned ON.	**Network 1** **LD Q0.0** **EU** **BITIM VD0**
	Calculate the time Q0.0 has been ON.	**Network 2** **LD Q0.0** **CITIM VD0, VD4**

7.18 Subroutine

7.18.1 CALL (subroutine) and RET (conditional return)

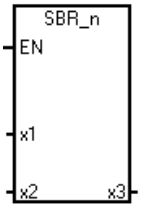
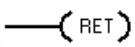
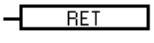
To add a new subroutine, select the **Edit** ribbon strip then **Insert Object** and **Subroutine** command. STEP 7-Micro/WIN SMART automatically adds an unconditional return from each subroutine. You can also add conditional return CRET instructions within the subroutine.

From the main program, you can nest subroutines (place a subroutine call within a subroutine) to a depth of eight.

From an interrupt routine, you can nest subroutines to a depth of four.

Note

Recursion (a subroutine that calls itself) is not prohibited, but you should use caution when using recursion with subroutines.

LAD / FBD	STL	Description
	`CALL SBR_n, x1, x2, x3`	The Call subroutine instruction transfers control to subroutine SBR_n. You can use a Call subroutine instruction with or without parameters. After the subroutine completes its execution, control returns to the instruction that follows the Call subroutine. The call parameters x1 (IN), x2 (IN_OUT), and x3 (OUT) represent three call parameters passed in, in and out, or out of the subroutine. The call parameters are optional. You may use from 0 to 16 call parameters. When a subroutine is called, the entire logic stack is saved, the top of stack is set to one, all other stack locations are set to zero, and control is transferred to the called subroutine. When this subroutine is completed, the stack is restored with the values saved at the point of call, and control is returned to the calling routine. Accumulators are common to subroutines and the calling routine. No save or restore operation is performed on accumulators due to subroutine use. When a subroutine is called more than once in the same cycle, the edge up, edge down, timer and counter instructions should not be used.
LAD:  FBD: 	`CRET`	The Conditional Return from Subroutine instruction (CRET) terminates the sub-routine based upon the preceding logic.

Error conditions with ENO = 0	SM bits affected
• 0006H Indirect address • 008H Maximum subroutine nesting exceeded	None

Input / output	Data type	Operand
SBR_n	WORD	Constant: 0-127
IN	BOOL	V, I, Q, M, SM, S, T, C, L, Power Flow (LAD), Logic flow (FBD)
	BYTE	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *LD, *AC[1], Constant
	WORD, INT	VW, T, C, IW, QW, MW, SMW, SW, LW, AC, AIW, *VD, *LD, *AC[1], Constant
	DWORD, DINT	VD, ID, QD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC[1], &VB, &IB, &QB, &MB, &T, &C, &SB, &AI, &AQ, &SMB, Constant
	STRING	*VD, *LD, *AC[1], Constant
IN_OUT	BOOL	V, I, Q, M, SM[2], S, T, C, L
	BYTE	VB, IB, QB, MB, SMB[2], SB, LB, AC, *VD, *LD, *AC[1]
	WORD, INT	VW, T, C, IW, QW, MW, SMW[2], SW, LW, AC, *VD, *LD, *AC[1]
	DWORD, DINT	VD, ID, QD, MD, SMD[2], SD, LD, AC, *VD, *LD, *AC[1]
OUT	BOOL	V, I, Q, M, SM[2], S, T, C, L
	BYTE	VB, IB, QB, MB, SMB[2], SB, LB, AC, *VD, *LD, *AC[1]
	WORD, INT	VW, T, C, IW, QW, MW, SMW[2], SW, LW, AC, AQW, *VD, *LD, *AC[1]
	DWORD, DINT	VD, ID, QD, MD, SMD[2], SD, LD, AC, *VD, *LD, *AC[1]

[1] Only AC1, AC2 or AC3 (AC0 not allowed)

[2] Must be from byte offset 30 to byte offset 999 for read/write access

Calling a subroutine with call parameters

Subroutines have the option of using passed parameters. The parameters are defined in the variable table of the subroutine. Each parameter must be assigned a local symbol name (a maximum of 23 characters), a variable type, and a data type. A maximum of sixteen parameters can be passed to or from a subroutine. The VAR_Type type field in the variable table defines whether the variable is passed into the subroutine (IN), passed into and out of the subroutine (IN_OUT), or passed out of the subroutine (OUT).

To add a new parameter row, place the cursor on the Var_Type field of the type (IN, IN_OUT, OUT, or TEMP) that you want to add. Click the right mouse button to get a menu of options. Select the Insert option and then the Row Below option. Another parameter row of the selected type will appear below the current entry.

Temporary (TEMP) parameters can be assigned in the variable table to store data that is valid only within the scope of the subroutine execution. Local TEMP data is not passed as a call parameter. You can also assign TEMP parameters in the main routine and interrupt routines, but only subroutines can use IN, IN_OUT, and OUT call parameters.

Variable table parameter types for a subroutine

Parameter	Description
IN	Parameters are passed into the subroutine. If the parameter is a direct address (such as VB10), the value at the specified location is passed into the subroutine. If the parameter is an indirect address (such as *AC1), the value at the location pointed to is passed into the subroutine. If the parameter is a data constant (16#1234) or an address (&VB100), the constant or address value is passed into the subroutine.
IN_OUT	The value at the specified parameter location is passed into the subroutine, and the result value from the subroutine is returned to the same location. Constants (such as 16#1234) and addresses (such as &VB100) are not allowed for input/output parameters.
OUT	The result value from the subroutine is returned to the specified parameter location. Constants (such as 16#1234) and addresses (such as &VB100) are not allowed as output parameters. Since output parameters do not retain the value assigned by the last execution of the subroutine, you must assign values to outputs each time the subroutine is called.
TEMP	Any local memory that is not used for passed parameters can be used for temporary storage within the subroutine.

Data types allowed for call parameters

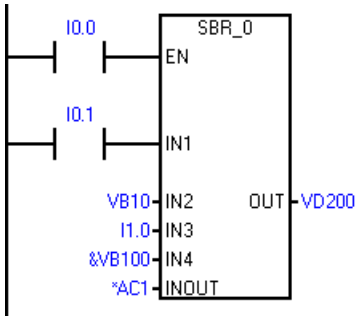
- Power Flow: Boolean power flow is allowed only for bit (Boolean) inputs. This declaration assigns an input parameter to the result of power flow based on a combination of bit logic instructions. Power flow inputs are similar to the EN input in that they connect to bit logic (for ex. LAD contacts) and not to a direct/indirect address assignment. Boolean power flow input(s) must be assigned in the top row(s) of the variable table before any non-BOOL data type assignment. Only input parameters are allowed to be used this way. The enable input (EN) and the IN1 inputs in the following example use power flow logic.
- BOOL: This data type is used for single bit inputs and outputs. IN3 in the following example is a Boolean input assigned to a direct address.
- BYTE, WORD, DWORD: These data types identify an unsigned input or output parameter of 1, 2, or 4 bytes, respectively.
- INT, DINT: These data types identify signed input or output parameters of 2 or 4 bytes, respectively.
- REAL: This data type identifies a single precision (4 byte) IEEE floating-point value.
- STRING: This data type is used as a four-byte pointer to a string.

Example variable table

Variable Table

	Address	Symbol	Var Type	Data Type	Comment
1		EN	IN	BOOL	
2	LW0	Lvl	IN	INT	Tank transmitter
3	LW2	HSet	IN	INT	High setpoint
4	LW4	Offset	IN	INT	Offset
5			IN		
6			IN_OUT		
7	LW6	Result	OUT	INT	Result value
8	L8.0	Valve	OUT	BOOL	Control valve
9			OUT		

Example: Subroutine call with call parameters

LAD	STL
	STL only: **Network 1** **LD I0.0** **CALL SBR_0, I0.1, VB10, I1.0, &VB100, *AC1, VD200** To display correctly in LAD and FBD: **Network 1** **LD I0.0** **= L60.0** **LD I0.1** **= L63.7** **LD L60.0** **CALL SBR_0, L63.7, VB10, I1.0, &VB100, *AC1, VD200**

Note

There are two STL examples provided above. STL programmers can use the first simplified STL instructions, which can only be displayed in the STL editor. This is because the BOOL parameters used as LAD/FBD power flow inputs are not saved to L memory.

The second set of compiler generated STL instructions can be displayed in the LAD, FBD, and STL editors, because L memory is used by the program compiler to save the state of the BOOL input parameters that are assigned as power flow inputs in LAD/FBD.

Address parameters such as IN4 (&VB100) are passed into a subroutine as a DWORD (unsigned double word) value. The type of a constant parameter must be specified for the parameter in the calling routine with a constant descriptor in front of the constant value. For example, to pass an unsigned double word constant with a value of 12,345 as a parameter, the constant parameter must be specified as DW#12345. If the constant describer is omitted from the parameter, the constant can be assumed to be a different type.

There are no automatic data type conversions performed on the input or output parameters. For example, if the variable table specifies that a parameter has the data type REAL, and in the calling routine a double word (DWORD) is specified for that parameter, the value in the subroutine will be a double word.

When values are passed to a subroutine, they are placed into the local memory of the subroutine. The left-most column of the variable table shows the local memory address for each passed parameter. Input parameter values are copied to the subroutine's local memory when the subroutine is called. Output parameter values are copied from the subroutine's local memory to the specified output parameter addresses when the subroutine execution is complete.

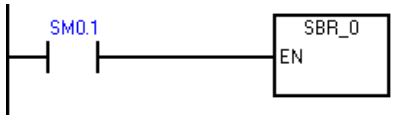
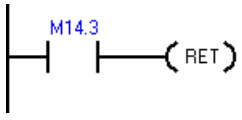
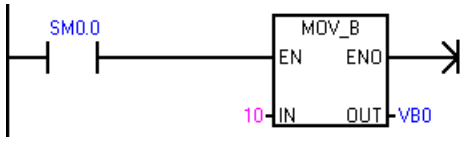
The data element size and type are represented in the coding of the parameters. Assignment of parameter values to local memory in the subroutine is as follows:

- Parameter values are assigned to local memory in the order specified by the call subroutine instruction with parameters starting at L 0.0.
- One to eight consecutive bit parameter values are assigned to a single byte starting with Lx.0 and continuing to Lx.7.
- Byte, word, and double word values are assigned to local memory on byte boundaries (LBx, LWx, or LDx).

In the Call Subroutine instruction with parameters, parameters must be arranged in order with input parameters first, followed by input/output parameters, and then followed by output parameters.

If you are programming in STL, the format of the CALL instruction is:
`CALL subroutine number, parameter 1, parameter 2, ... , parameter 16`

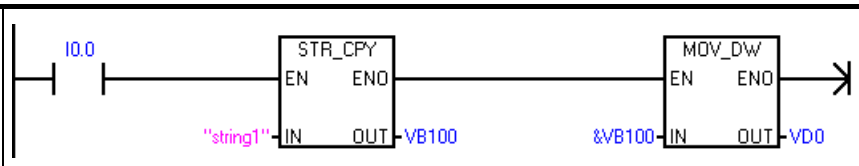
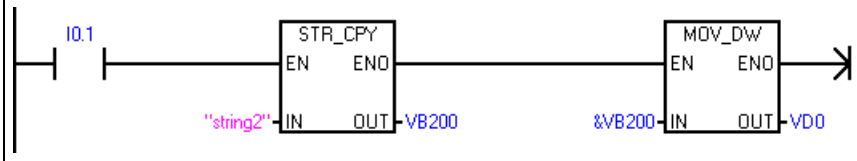
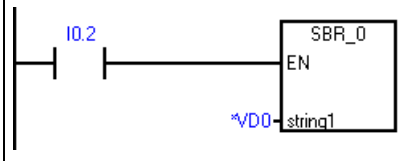
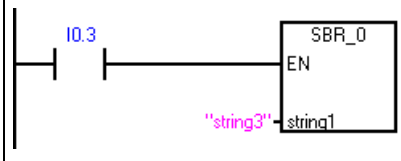
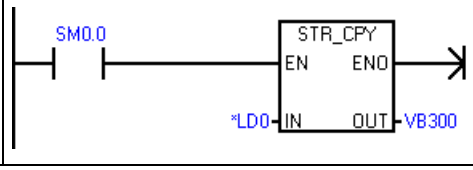
Example: Subroutine and return from subroutine instructions

LAD			STL
MAIN		On the first scan, call subroutine 0 for initialization.	**`Network 1`** **`LD SM0.1`** **`CALL SBR_0`**
SBR0		You can use a conditional return to leave the subroutine before the last network.	**`Network 1`** **`LD M14.3`** **`CRET`**
SBR0		This network will be skipped if M14.3 is ON.	**`Network 2`** **`LD SM0.0`** **`MOVB 10, VB0`**

Example: Subroutine call using string parameter

This example copies a different string literal to a unique address depending upon the given input. The unique address of this string is saved. The string address is then passed to the subroutine by using an indirect address. The data type of the subroutine input parameter is string. The subroutine then moves the string to a different location.

A string literal can also be passed to the subroutine. The string reference inside the subroutine is always the same.

LAD		STL
MAIN		**Network 1** **LD I0.0** **SCPY "string1", VB100** **AENO** **MOVD &VB100, VD0**
MAIN		**Network2** **LD I0.1** **SCPY "string2", VB200** **AENO** **MOVD &VB200, VD0**
MAIN		**Network3** **LD I0.2** **CALL SBR_0, *VD0**
MAIN		**Network4** **LD I0.3** **CALL SBR_0, "string3"**
SBR0		**Network 1** **LD SM0.0** **SSCPY *LD0, VB300**

7.19 PROFINET

7.19.1 Features of the programming instruction "PROFINET"

There are two groups of programming instructions under the folder "PROFINET" :

- RDREC and WRREC (Page 374): Read a data record from any connected PROFINET device or write a data record to any connected PROFINET device.
- BLKMOV_BIR and BLKMOV_BIW (Page 378): Read multiple bytes immediately from PROFINET device or write multiple bytes immediately to PROFINET device.

Note

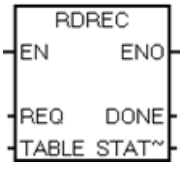
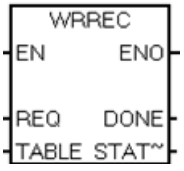
For any legacy instruction that can access the I or Q memory area, it accesses to the I or Q memory area of a PROFINET network.

7.19.2 Read and Write data record

7.19.2.1 Input and output interface of RDREC and WRREC instruction

The RDREC and WRREC instructions are as follows:

Table 7- 21 RDREC and WRREC

LAD/FBD	STL	Description
	RDREC Req, Table, Done, Status	Use the RDREC instruction to read a data record from PROFINET device.
	WRREC Req, Table, Done, Status	Use the WRREC instruction to write a data record to PROFINET device.

The parameters of the RDREC and WRREC instructions are as follows:

Table 7- 22 Parameters of RDREC and WRREC instruction

Parameter and type		Data type	Operand	Description
REQ	IN	BOOL	I, Q, V, M, T, C, SM, S, L	REQ=1: Transfer data record
TABLE	IN/OUT	BYTE	Q, V, M, SM, S, L, *AC,*VD,*LD, Constant	A table of parameters you set for the data read/ write record. For detailed information, refer to Definition of "TABLE" parameters (Page 376).
DONE	OUT	BOOL	I, Q, V, M, SM, S, L	The instruction is completed.
STATUS	OUT	BYTE	I, Q, V, M, SM, S, L	The status of the current operation. For detailed information, refer to Definition of "STATUS" parameters (Page 377).

Note

The supported maximum data record length is 1024 bytes.

Definition of "TABLE" parameters

The following table lists the parameter information of "TABLE":

Table 7- 23 TABLE

Byte Offset	Parameter and Type		Comment
0	Device Number	Input **Note:** The value range is from 1 to 8.	Device Number, API Number, Slot Number and SubSlot Number are used to address a submodule. You can find the Device Number, API Number, Slot Number and SubSlot Number in the PROFINET wizard.
1			
2	API Number	Input	
3			
4			
5			
6	Slot Number	Input	
7			
8	SubSlot Number	Input	
9			
10	Record Index	Input	The Record Index includes the record index from protocol or the user-defined record index. For detailed information of the index from the protocol, refer to *Technical Specification for PROFINET IO (Version 2.3).*
11			
12	Buffer Length	Input	This parameter refers to the number of bytes of the buffer. The buffer stores the data record read from or written to the device. The value range: from 1 to 1024.
13			
14	Data Address	Input	Address of the buffer read from or written to the device. **Note:** If the buffer length is greater than the actual record data length, the buffer contains all the record data. If the buffer length is smaller than actual record data length, the buffer contains partial record data and an error occurs.
15			
16			
17			
18	Actual record data length	Output	This parameter is valid for the instruction RDREC and returns the actual data length specified by the device.
19			
20	PROFINET Error Code	Output	The error code defined by the PROFINET protocol. 0 = no error If the value is not 0, check the specific error code in *Technical Specification for PROFINET IO (Version 2.3).*
21			
22			
23			

Definition of "STATUS" parameters

The following table lists the parameter information of "STATUS":

Table 7- 24 STATUS

Byte Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A [1]	E [2]	Error code [3]					

[1] A : 1 = a request is in process

[2] E : 1= an error occurs

[3] Error code: The system error code. For detailed information, refer to System-defined error code of the instructions RDREC and WRREC (Page 377).

7.19.2.2 System-defined error code of the instructions RDREC and WRREC

The error codes are as follows:

Error code	Description
0	No error.
1	The data length parameter is 0 or is greater than the supported maximum length (1024 bytes).
2	The data buffer is not in I, Q, M, or V memory areas.
3	The data buffer does not fit in the memory area.
4	The table doesn't match with the memory.
5	The device number is invalid and not within the range: from 1 to 8.
6	An instance mismatch: The connection is busy with another instance, whose device number, API number, slot number and subslot number are same as the requested instance, but with a different buffer size and data address.
7	The PROFINET device is not connected.
8	The size of the received buffer exceeds 1024 bytes.
9	Call sequence is invalid.
10	Parameters are invalid (for example, out-of-range).
11	The AR is created afresh in the meanwhile.
12	The RPC reports a timeout error.
13	The RPC reports a communication error.
14	The RPC Server of the IOD signaled “busy” (for example, the call can be repeated later).
15	CLRPC reports an error or the PDU cannot be parsed.
16	CM response is OK, but has a PROFINET protocol defined error.
17	The instruction parameter is invalid.
24	REQ is not enabled.
25	The buffer length is smaller than the actual data record length.
63	Unknown error.

7.19.3 Read and Write multiple bytes between physical PROFINET and memory address

7.19.3.1 Input and output interface of BLKMOV_BIR and BLKMOV_BIW

The BLKMOV_BIR and BLKMOV_BIW instruction is as follows:

Table 7- 25 BLKMOV_BIR and BLKMOV_BIW

LAD/FBD	STL	Description
	BMIR In, Out, N	The "BLKMOV_BIR Immediate Read" instruction reads multiple bytes of physical PROFINET input IN and writes the result to the memory address OUT, but the process image register is not updated. The input signal N defines the number of bytes. **Note**: • N ≤ 128 • N bytes cannot exceed the submodule boundary.
	BMIW In, Out, N	The "BLKMOV_BIW Immediate Write" instruction reads multiple bytes from the memory address IN and writes to physical PROFINET output OUT, and the corresponding process image location is updated. The input signal N defines the number of bytes. **Note**: • N ≤ 128 • N bytes cannot exceed the submodule boundary.

The parameters of BLKMOV_BIR and BLKMOV_BIW are as follows:

Parameter and type		Data type	Operand
IN (BLKMOV_BIR)	IN	BYTE	IB, *VD, *LD, *AC
IN (BLKMOV_BIW)	IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT (BLKMOV_BIR)	OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
OUT (BLKMOV_BIW)	OUT	BYTE	QB, *VD, *LD, *AC
N	IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, Constant, *VD, *LD, *AC

7.19.3.2 Error code of the instructions BLKMOV_BIR and BLKMOV_BIW

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address • Unable to access expansion module	None

Communication

8

The S7-200 SMART offers several types of communication between CPUs, programming devices, and HMIs:

- Ethernet:
  - Exchange of data from the programming device to the CPU
  - Exchange of data between HMIs and the CPU
  - S7 peer-to-peer communication with other S7-200 SMART CPUs
  - Open User Communication (OUC) with other Ethernet-capable devices
  - PROFINET communication with PROFINET devices

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

- PROFIBUS:
  - High speed communications for distributed I/O (up to 12 Mbps)
  - One bus master connects to many I/O devices (supports 126 addressable devices).
  - Exchange of data between the master and I/O devices
  - EM DP01 module is a PROFIBUS I/O device.
- RS485:
  - Provides a STEP 7-Micro/WIN SMART connection for programming when using a USB-PPI cable
  - Supports a total of 126 addressable devices (32 devices per network segment)
  - Supports PPI (point-to-point interface) protocol
  - Exchange of data between HMIs and the CPU
  - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)
- RS232:
  - Supports a point-to-point connection to one device
  - Supports PPI protocol
  - Exchange of data between HMIs and the CPU
  - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)

8.1 CPU communication connections

The CPU supports the following maximum number of simultaneous, asynchronous communication connections:

- Ethernet programming port:
  - Open User Communication (OUC) connections: Eight active (client) connections and eight passive (server) connections to support S7-200 SMART CPUs or other Ethernet devices.
  - HMI/OPC connections: Eight dedicated HMI/OPC server connections.
  - PG connections: One programming device (PG) connection.
  - Peer-to-peer (GET/PUT) connections: Eight active (client) connections and eight passive (server) connections to support S7-200 SMART CPUs or network devices.
  - PROFINET connections: Each PROFINET controller can support eight connections (IO device or drive).

Note

You can program an S7-200 SMART CPU through the Ethernet port if your CPU model supports it. Only one PG can monitor one CPU at a time.

Note

The S7-200 SMART CPU uses the GET and PUT instructions for CPU-to-CPU communications.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

- Integrated RS485 port (Port 0): Four connections to support HMI devices and one connection reserved for programming with STEP 7-Micro/WIN SMART.

Note

You can make the following RS485 communication connections:

- Use a USB-PPI cable to program all CPU models through any serial port, including the RS485 port, the signal board port, and the DP01 PROFIBUS port.
- Use the RS485 and RS232 ports for HMI access (Data read/write) and Freeport communications.

- PROFIBUS port: Each EM DP01 PROFIBUS DP module can support six connections.
- CM01 Signal Board (SB) RS232/RS485 port (Port 1): Four connections to support HMI devices.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

8.2 CPU communication ports

There are four communication interfaces on the S7-200 SMART CPU that provide the following communication types:

- Ethernet port (if your CPU model supports it):
  - STEP 7-Micro/WIN SMART programming
  - GET/PUT communication
  - HMI: Ethernet type
  - Open User Communication (OUC) over UDP, TCP, or ISO-on-TCP
  - PROFINET Communication
- RS485 port (Port 0):
  - STEP 7-Micro/WIN SMART programming when using a USB-PPI cable
  - TDs/HMI: RS485 type
  - Freeport (XMT/RCV) including Siemens-provided USS and Modbus RTU libraries
- PROFIBUS port: The S7-200 SMART CPUs can support two EM DP01 modules for PROFIBUS DP and HMI communication if your CPU model supports expansion modules.
- RS485/RS232 signal board (SB) (if present, Port 1):
  - TDs/HMIs: RS485 or RS232 type
  - Freeport (XMT/RCV) including Siemens-provided USS (RS485 only) and Modbus RTU (RS485 or RS432) libraries

8.3 HMIs and communication drivers

HMIs

The S7-200 SMART supports the HMIs from the following Siemens HMI families:

- COMFORT HMIs (PROFINET and PROFIBUS):
  - SIMATIC HMI TP700 COMFORT
  - SIMATIC HMI TP900 COMFORT
  - SIMATIC HMI TP1200 COMFORT
  - SIMATIC HMI KP400 COMFORT
  - SIMATIC HMI KP700 COMFORT
  - SIMATIC HMI KP900 COMFORT
  - SIMATIC HMI KP1200 COMFORT
  - SIMATIC HMI KTP400 COMFORT
- SMART HMIs (Ethernet, PPI and MPI):
  - SMART 700 IE V3
  - SMART 1000 IE V3
- BASIC HMIs (Industry Ethernet Interface & RS485 Interface):
  - SIMATIC HMI KTP400 BASIC MONO PN
  - SIMATIC HMI KTP600 BASIC MONO PN
  - SIMATIC HMI KTP600 BASIC COLOR PN
  - SIMATIC HMI KTP1000 BASIC COLOR PN
  - SIMATIC HMI TP1500 BASIC COLOR PN
  - SIMATIC HMI KP300 BASIC MONO PN
- BASIC HMIs (PROFIBUS):
  - SIMATIC HMI KTP600 BASIC COLOR DP
  - SIMATIC HMI KTP1000 BASIC COLOR DP
- Micro HMIs (PROFIBUS):
  - TD 400C TEXT DISPLAY, 4 LINES

Communication drivers

Communication drivers for your S7-200 SMART HMIs can be selected in two locations:

- WinCC Flexible SMART
- TIA portal

WinCC Flexible

In the WinCC Flexible SMART package, you can select the required "Communication driver" with the following menu selections:

- Communications
- Connections table

In the "Connections table", select the "SIMATIC S7 200 SMART" driver. If the SMART driver is not in the list, select the "SIMATIC S7 200" driver.

TIA portal

In the TIA portal, you can select the required "Communication driver" with the following menu selections:

- HMI tags
- Connections

In "Connections", select the "SIMATIC S7 200" driver.

Note

If the HMI panel is using a PROFIBUS DP connection (RS485), then also set the "Network Profile" to PPI.

8.4 Ethernet

8.4.1 Overview

An Ethernet network is a differential (multi-point) network that can have up to 32 segments and 1,024 nodes. Ethernet allows for data transfer at a high speed (up to 100 Mbit/s) and long distances (Copper: Maximum approximately 1.5km; Optical: Maximum approximately 4.3km).

Possible Ethernet connections include connections for the following:

- Programming devices
- CPU-to-CPU GET/PUT communication
- HMI displays
- Open User Communication (OUC)
- PROFINET Communication

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and no functions related to the use of Ethernet communications.

8.4.2 Local/partner connection

A Local / Partner (remote) connection defines a logical assignment of two communication partners to establish a communication connection. A connection is defined by the following:

- Communication partners involved (one active, one passive)
- Type of connection (programming device, HMI, CPU, or other device)
- Connection path (network, IP address, subnet mask, gateway)

The communication partners set up and establish the communication connection. The active device establishes the connection, and the passive device either accepts or rejects the connection request from the active device. After a connection is established, it is automatically maintained by the active device and monitored by both the devices.

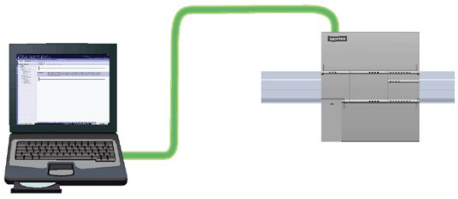
If the connection is terminated (for example, due to a line break or one of the partners breaks the connection), the active partner attempts to re-establish the connection. The passive device will also note the termination of the connection and take actions (for example, revoking the password privileges of the now disconnected active partner).

The S7-200 SMART CPUs are both active and passive devices. When an active device (for example, a computer running STEP 7-MicroWIN SMART or an HMI) establishes a connection, the CPU decides whether to accept or reject the connection request, based upon the type of the connection and how many connections of a given type are allowed.

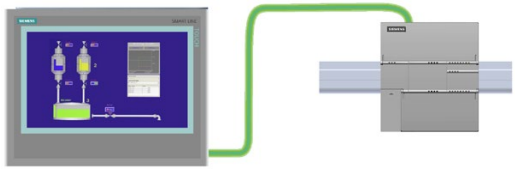
8.4.3 Sample Ethernet network configurations

You have three different types of communication options when using the S7-200 SMART CPU Ethernet network:

- Connecting a CPU to a programming device
- Connecting a CPU to an HMI
- Connecting a CPU to another S7-200 SMART CPU



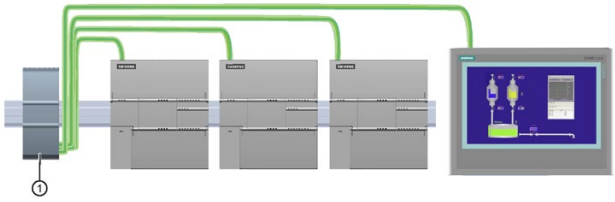
Programming device connected to the CPU



HMI connected to the CPU

A CPU connected to another CPU

The Ethernet port on the CPU does not contain an Ethernet switching device. A direct connection between a programming device or HMI and a CPU does not require an Ethernet switch. However, a network with more than two CPUs or HMI devices requires an Ethernet switch.



① CSM1277 Ethernet switch

You can use the rack-mounted CSM1277 4-port Ethernet switch for connecting multiple CPUs and HMI devices.

8.4.4 Assigning Internet Protocol (IP) addresses

8.4.4.1 Assigning IP addresses to programming and network devices

If your programming device is using a network adapter card connected to your plant LAN (and possibly the World Wide Web), both the programming device and the CPU must exist on the same subnet. The subnet is specified as a combination of the IP Address and subnet mask for the device. Please see your local network administrator for help.

The Network ID is the first part of the IP address (first three octets) (for example, **211.154.184**.16) that determines what IP network you are on. The subnet mask normally has a value of **255.255.255.0**; however, since your computer is on a plant LAN, the subnet mask may have various values (for example, **255.255.254.0**) in order to set up unique subnets.

The subnet mask, when combined with the device IP address in a logical AND operation, defines the boundaries of an IP subnet.

Note

In a World Wide Web scenario, where your programming devices, network devices, and IP routers will communicate with the world, unique IP addresses must be assigned to avoid conflict with other network users. Contact your company IT department personnel, who are familiar with your plant networks, for assignment of your IP addresses.

Note

A secondary network adapter card is useful when you do not want your CPU on your company LAN. During initial testing or commissioning tests, this arrangement is particularly useful.

Assigning or checking the IP address of your programming device using "My Network Places" (on your desktop)

If you are using Windows 7, you can assign or check your programming device's IP address with the following menu selections:

- "Start"
- "Control Panel"
- "Network and Sharing Center"
- "Local Area Connection" for the network adapter connected to your CPU
- "Properties"

- In the "Local Area Connection Properties" dialog, in the "This connection uses the following items:" field:
  - Scroll down to "Internet Protocol Version 4 (TCP/IPv4)".
  - Click "Internet Protocol Version 4 (TCP/IPv4)".
  - Click the "Properties" button.
  - Select "Obtain an IP address automatically" or "Use the following IP address" (to enter a static IP address).
- If you have selected "Obtain an IP address automatically" you might want to change the selection to "Use the following IP address" to connect to the S7-200 SMART CPU:
  - Select an IP address on the same subnet as the CPU (**192.168.2**.1).
  - Set the IP address to an address with the same Network ID (for example, **192.168.2**.200).
  - Select a subnet mask of **255.255.255**.0.
  - Leave the default gateway blank.
  - This will allow you to connect to the CPU.

Note

The Communication Interface (for Ethernet, a network interface card (NIC)) and the CPU must be on the same subnet to allow STEP 7-MicroWIN SMART to find and communicate to the CPU.

Consult your IT personnel to help you set up a network configuration to allow you to connect to the S7-200 SMART CPU.

8.4.4.2 Configuring or changing an IP address for a CPU or device in your project

You must enter the following IP information for each S7-200 SMART CPU that is attached to your Ethernet network:

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

- IP address: Each CPU or device must have an Internet Protocol (IP) address. The CPU or device uses this address to deliver data on a more complex, routed network. Each IP address is divided into four 8-bit segments and is expressed in a dotted, decimal format (for example, 211.154.184.16). The first part of the IP address is used for the Network ID (What network are you on?), and the second part of the address is for the Host ID (unique for each device on the network). An IP address of 192.168.x.y is a standard designation recognized as part of a private network that is not routed on the Internet.

  Note

  All S7-200 SMART CPUs have a default IP address of: 192.168.2.1

  Note

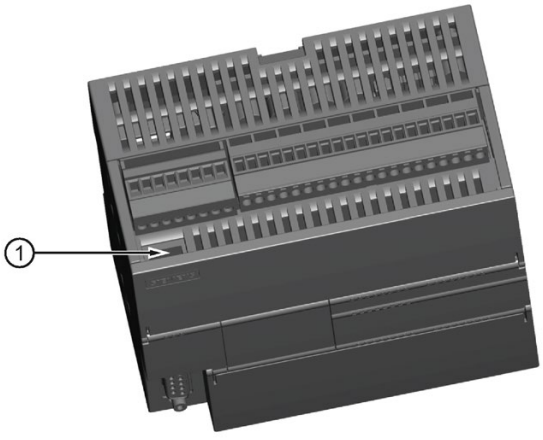
  You must have a unique IP address for each device on your network.

- Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet are usually located in close physical proximity to each other on a Local Area Network (LAN). The subnet mask defines the boundaries of an IP subnet.

  Note

  A subnet mask of 255.255.255.0 is generally suitable for a local network.

- Default gateway IP address: Gateways (or IP routers) are the link between LANs. Using a gateway, a computer in a LAN can send messages to other networks, which might have other LANs behind them. If the data destination is not within the LAN, the gateway forwards the data to another network or group of networks where it can be delivered to its destination. Gateways rely on IP addresses to deliver and receive data packets.



① PROFINET (LAN) port

There are three ways to configure or change the IP information for the onboard Ethernet port of a CPU or device:

- Configuring the IP information in the "Communications" dialog (dynamic IP information)
- Configuring the IP information in the "System Block" dialog (static IP information)
- Configuring the IP information in the user program (dynamic IP information)

Note

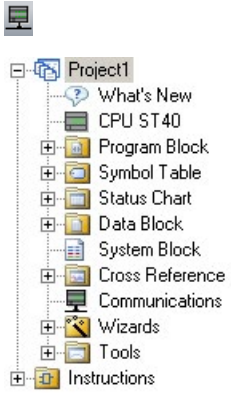
You can have static or dynamic IP information in the CPU:

- Static IP information: If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block is checked, then the Ethernet network information that you enter is static. Static IP information must be downloaded to the CPU before it is active in the CPU, and, if you want to change the IP information, this IP information can only be changed in the system block dialog and once again downloaded to the CPU.
- Dynamic IP information: If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block is not checked, then you change the IP address of the CPU through other means and this IP address information is considered to be dynamic. You can change the IP address information in the Communications dialog or with the SIP_ADDR instruction in the user program.
- For both static and dynamic IP, the information is stored in persistent memory.

Configuring the IP information in the Communications dialog (dynamic IP information)

IP information changes done through the Communications dialog are immediate and do not require a download of the project.

To access this dialog, perform one of the following:



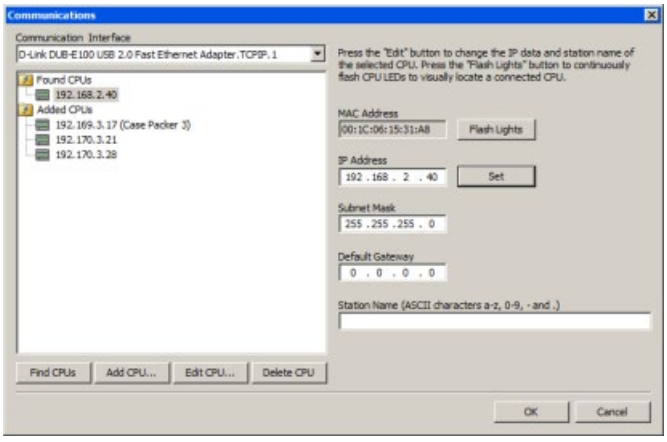
- In the Navigation bar, click the "Communications" button.
- In the Project tree, select the "Communications" node, then press Enter; or double-click the "Communications" node.

You can access CPUs in one of two ways:

- "Found CPUs": CPUs located on your local network
- "Added CPUs": CPUs on the local or remote networks (for example, CPUs accessed on another network through a router)

For "Found CPUs" (CPUs located on your local network), use the "Communications dialog" to connect with your CPU:

- Click the "Communication Interface" dropdown list, and select the "TCP/IP" Network Interface Card (NIC) for your programming device.
- Click the "Find CPUs" button to display all operational CPUs ("Found CPUs") on the local Ethernet network. All CPUs have a default IP address.
- Highlight a CPU, and then click "OK".

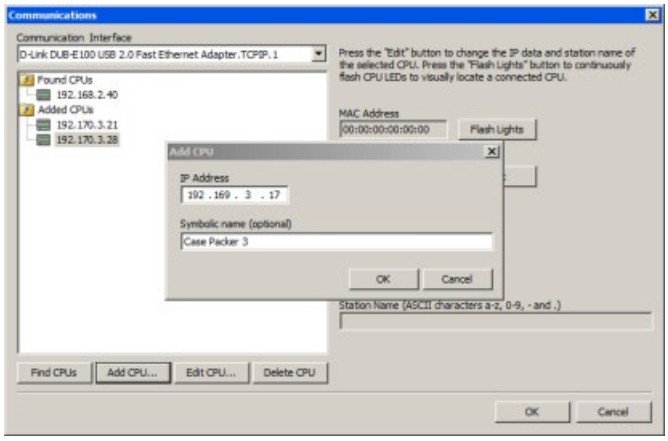


For "Added CPUs" (CPUs on the local or remote networks), use the "Communications dialog" to connect with your CPU:

- Click the "Communication Interface" dropdown list, and select the "TCP/IP" Network Interface Card (NIC) for your programming device.
- Click the "Add CPU" button to do one of the following:
  - Enter the IP address of a CPU that is accessible from the programming device, but is not on the local network.

    You can add these CPUs, select them as the communication partner in STEP 7-Micro/WIN SMART, and program and operate these CPUs in the same way you would a CPU on the local network. As long as there is a valid network path through routers, STEP 7-Micro/WIN SMART can communicate with any S7-200 SMART CPU.
  - Enter the IP address of a CPU directly that is on the local network.

  You can add multiple CPUs, on the local network and/or remote network. As always, STEP 7-Micro/WIN SMART communicates with one CPU at a time. All CPUs have a default IP address.
- Highlight a CPU, and then click "OK".



To enter or change IP information, perform the following:

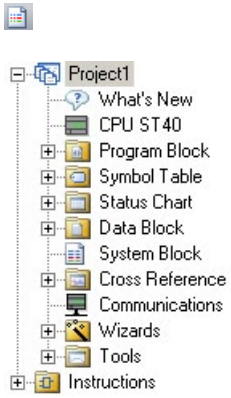
- Click the required CPU.
- If you need to identify which CPU to configure or change, click the "Flash Lights" button. This button flashes the STOP, RUN, and FAULT lights for the highlighted CPU in the list.
- Click the "Edit" button to make changes in the IP information.
- Change the following IP information:
  - IP address
  - Subnet mask
  - Default gateway
  - Station name
- Press the "Set" button. When the "Set" button is pressed, these values are updated within the CPU.
- When finished, click "OK".

When you configure IP information for the onboard Ethernet port in the Communications dialog, this information is "dynamic". If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block dialog is not checked, then you must enter IP information in the Communications dialog. You can enter new IP address information and update this information in the CPU by clicking the "Set" button.

Configuring the IP information in the System Block dialog (static IP information)

IP information configuration or changes done in the system block are part of the project and do not become active until you download your project to the CPU.

To access this dialog, perform one of the following:



- In the Navigation bar, click the "System Block" button.
- In the Project tree, select the "System Block" node, then press Enter; or double-click the "System Block" node.

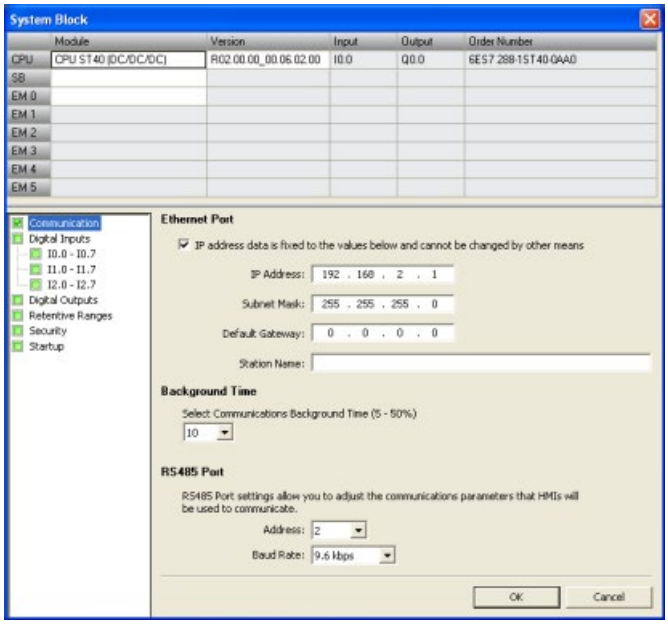
To enter or change IP information, perform the following:

- If not already checked, click the "IP address data is fixed to the values below and cannot be changed by other means" checkbox. The Ethernet port IP information fields are enabled.
- Enter or change the following IP information:
  - IP address
  - Subnet mask
  - Default gateway
  - Station name

Note

The Station Name follows the standard DNS (Domain Name System) naming conventions. The S7-200 SMART CPUs limit the Station Name to a maximum of 63 characters. The Station Name can consist of the lower case letters a through z, the digits 0 through 9, the hyphen character (minus sign), and the period character.

Certain names are not allowed and this can depend on the tool used to set the Station Name. The Station Name must not have the format "n.n.n.n" where n is a value of 0 through 999. The Station Name cannot begin with the string "port-nnn" or the string "port-nnn-nnnnn" where "n" is a digit 0 through 9 (for example, "port-123" and "port-123-45678" are illegal). The Station Name cannot start or end with the hyphen or period.



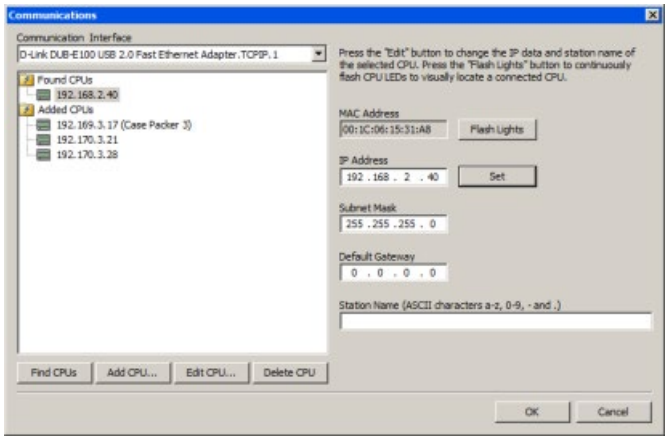
When you check the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block dialog, the IP information that you enter for the onboard Ethernet port is static. Static IP information must be downloaded to the CPU before it is active in the CPU. If you want to change the IP information, this IP information can only be changed in the system block dialog and once again downloaded to the CPU.

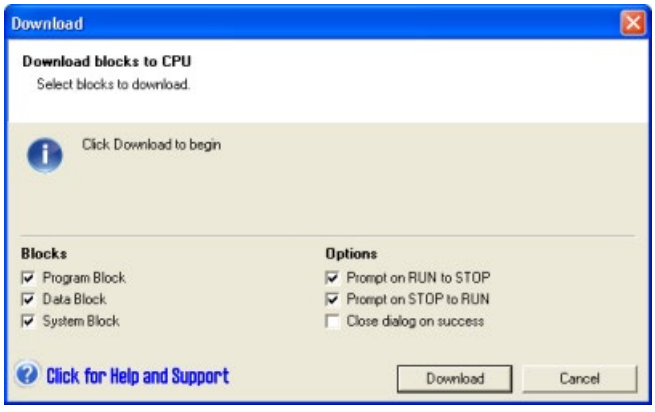
Note

If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox is checked, then the IP information cannot be set in the Communications dialog.

In order to use the SIP_ADDR instruction, the "IP address data is fixed to the values below and cannot be changed by other means" checkbox must be unchecked.

After completing the IP information configuration, download the project to the CPU. All CPUs and devices that have valid IP addresses are displayed in the Communications dialog.





Configuring the IP information in the user program (dynamic IP information)

The SIP_ADDR instruction sets the CPU's IP address to the value found in its "ADDR" input, the CPU's subnet mask to the value found in its "MASK" input, and the CPU's gateway to the value found in its "GATE" input.

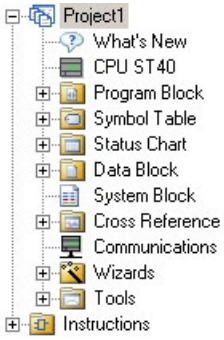
IP information or changes done through the SIP_ADDR instruction are immediate and do not require a download of the project. The IP address information set with the SIP_ADDR instruction is stored in permanent memory in the CPU.

Refer to the "Get IP address and set IP address (Ethernet)" (Page 205) instructions for more information.

8.4.4.3 Searching for CPUs and devices on your Ethernet network

You can search for and identify the S7-200 SMART CPUs that are attached to your Ethernet network in the "Communications" dialog. To access this dialog, click one of the following:

Communications button in the navigation bar



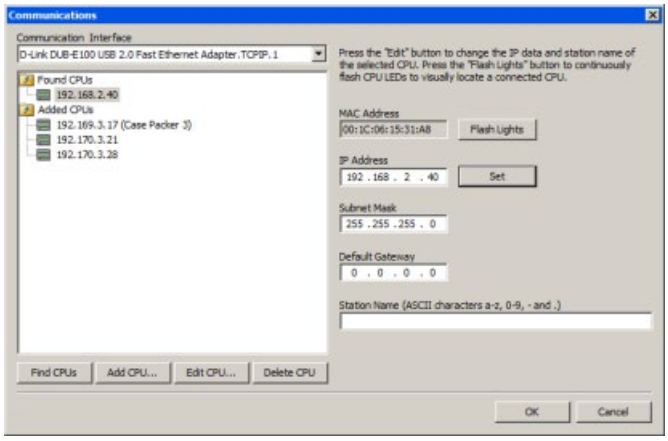
Communications in the project tree

Communications from the Component drop-down list in the Windows area of the View menu ribbon strip

The "Communications" dialog will autodetect all connected and available S7-200 SMART CPUs on a given Ethernet network by creating a lifelist. (See the figure below.) After selecting a CPU, the following detailed information about the CPU is listed:

- MAC address
- IP information
- Station name

The IP address of a CPU is not associated with a STEP 7-Micro/WIN SMART project. Opening a STEP 7-Micro/WIN SMART project does not automatically select an IP address or establish a connection to a CPU. Every time you create a new or open an existing STEP 7-Micro/WIN SMART project, you must go to the Communications dialog to establish a connection to a CPU. The Communications dialog will show the last selected CPU.



8.4.5 Locating the Ethernet (MAC) address on the CPU

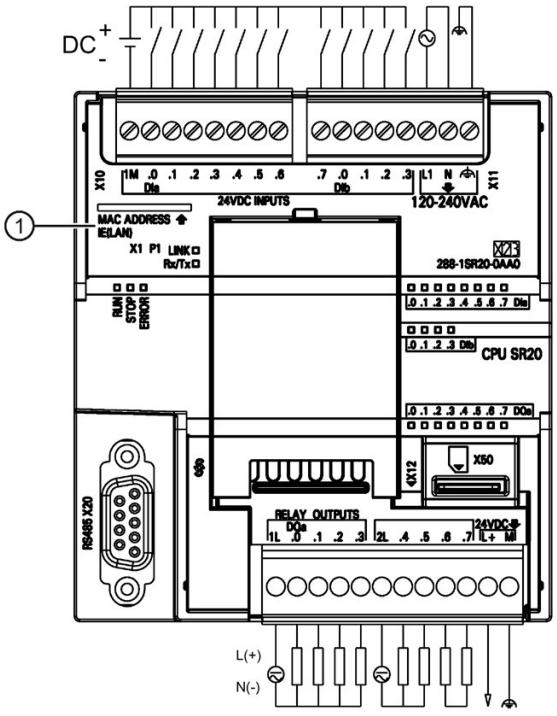
In Ethernet networking, a Media Access Control address (MAC address) is an identifier assigned to the network interface by the manufacturer for identification. A MAC address usually encodes the manufacturer's registered identification number.

The standard (IEEE 802.3) format for printing MAC addresses in human-friendly form is six groups of two hexadecimal digits separated by hyphens (-) or colons (:), for example, 01-23-45-67-89-ab or 01:23:45:67:89:ab.

Note

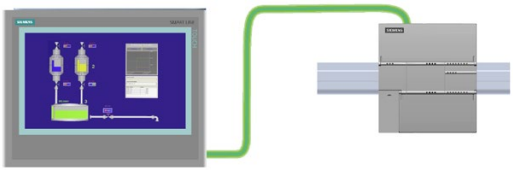
Each CPU is loaded at the factory with a permanent, unique MAC address. You cannot change the MAC address of a CPU.

The MAC address is printed on the front, upper-left corner of the CPU. Note that you have to open the upper door to see the MAC address information.



① MAC address

8.4.6 HMI-to-CPU communication



The CPU supports Ethernet communication connections to HMIs.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

The following requirements must be considered when setting up communications between CPUs and HMIs:

- Configuration/Setup:
  - The CPU must be configured with an IP address.
  - The HMI must be setup and configured to connect with the IP address of the CPU.
  - No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

Note

The rack-mounted CSM1277 4-port Ethernet switch can be used to connect your CPUs and HMI devices. The Ethernet port on the CPU does not contain an Ethernet switching device.

- Supported functions:
  - The HMI can read/write data to the CPU.
  - Messages can be triggered, based upon information retrieved from the CPU.
  - System diagnostics

To ensure that your CPU and HMI are communicating properly, follow the sequence of steps in the table below:

Table 8- 1 Required steps in configuring communications between an HMI and a CPU

Step	Task
1	Establishing the hardware communications connection An Ethernet interface establishes the physical connection between an HMI and a CPU. Since Auto-Cross-Over functionality is built into the CPU, you can use either a standard or crossover Ethernet cable for the interface. An Ethernet switch is not required to connect an HMI and a CPU. Refer to "Establishing the hardware communications connection" (Page 29) for more information.
2	If you have already created a project with a CPU, open your project in STEP 7-Micro/WIN SMART. If not, create a project and insert a CPU In the project.
3	Configuring an IP address in your project Use the same configuration process; however, you must configure IP addresses for the HMI and the CPU. You must download the configuration for each CPU and HMI device. Refer to "Configuring or changing an IP address for a CPU or device in your project" (Page 389) for more information.

Note

You can restrict communication writes to a specific range of V memory; this can affect HMI communications. See "Configuring system security" (Page 138) for more information.

8.4.7 Open user communication

8.4.7.1 Protocols

Open User Communication (OUC) provides a mechanism for your program to transmit and receive messages over an Ethernet network. You can select the Ethernet protocol used as the transport mechanism: UDP, TCP, or ISO-on-TCP

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

UDP (User Datagram Protocol)

User Datagram Protocol (UDP) uses a simple connectionless transmission model with a minimum of protocol overhead. There is no handshake mechanism in UDP protocol so the protocol is only as reliable as the underlying network. There is no guarantee of delivery, ordering, or duplicate message protection. UDP does provide checksums for data integrity and generally uses different port numbers to address different functions.

TCP (Transmission Control Protocol)

Transmission Control Protocol (TCP) is a core internet protocol. TCP provides a reliable, ordered, and error-checked delivery of messages between applications running on hosts communicating over an Ethernet network. TCP guarantees that all bytes received are identical with the bytes sent, and in the original order. TCP protocol creates a connection between an active device (the device that initiates the connection) and a passive device (the device that accepts the connection). Once a connection is established, either device can initiate a transfer of data.

TCP protocol is a "streaming" protocol. This means that there is no end sign included in a message. All the messages that are received are considered to be part of a "stream" of data. For example, the client device sends three messages of 20 bytes each to a server. The server sees only a "stream" of 60 bytes received (assuming the server executes a receive operation after the three messages are received.)

ISO-on-TCP

ISO-on-TCP is a protocol add-on using RFC 1006. The main advantage of ISO-on-TCP is that you have a clear data end sign in the data so that you know when the entire message has been received. SPS7 protocol (Put/Get) utilizes ISO-on-TCP protocol. ISO-on-TCP only uses port 102 and utilizes TSAPs (Transport Services Access Point) to route the message to the proper receiver instead of a port as in TCP.

The ISO-on-TCP protocol delineates each message received. For example, a client sends three messages to a server using the ISO-on-TCP protocol. Even if the server waits for all messages to accumulate before checking for a received message, the server will get each

message as it was sent and see three distinct messages. This is the difference between TCP protocol and the ISO-on-TCP protocol.

8.4.7.2 Connections

The S7-200 SMART CPU has two OUC instructions to perform connection management:

- The TCON instruction to establish an active connection (client) or open a passive connection (server)
- The TDCON instruction to force a disconnection (for example, close a connection). A RUN-to-STOP transition forces the closure of all open connections by the CPU.

The CPU supports two types of connections for OUC:

- Active: A connection established and maintained by the local CPU. The local CPU is responsible for issuing the connection request to another device and maintaining the connection so that the connection does not time out due to inactivity.
- Passive: A passive connection is one in which the local CPU opens a port and/or TSAP to accept the connection request from another device.

The CPU supports eight active and eight passive connections.

The CPU creates a passive or active connection based upon the connection table passed to the TCON instruction. UDP connections are always passive. TCP and ISO-on-TCP protocol connections use a configuration parameter to determine the type of connection.

8.4.7.3 Ports and TSAPs

Ports and Transport Service Access Points (TSAPs) provide for the routing of messages to the proper receiver within the CPU or other device.

Ports

The UDP and TCP protocols allow you to select the local and remote port numbers. The port number is fixed at port 102 when you select the ISO-on-TCP protocol.

Port numbers must be in the range of 1 to 49151. It is recommended that port numbers be in the range of 2000 to 5000. The S7-200 SMART CPU range and restriction rules for port numbers are shown in the tables below:

Port number	Description
1 to 1999	• You can use these numbers; however, they are outside the recommended range. • Some ports are excluded (see below).
2000 to 5000	Recommended range

Port number	Description
5001 to 49151	• You can use these numbers; however, they are outside the recommended range. • Some ports are excluded (see below).
49152 to 65535	• These are dynamic or private ports. • The use of these port numbers is restricted.

You cannot use the port numbers shown in the following table for local port numbers in the S7-200 SMART CPU. You have no restrictions for the remote port number:

Port number	Description
20	FTP data transmission
21	FTP control
25	SMTP
80	Web server
102	ISO-on-TCP
135	DCE for PROFINET
161	SNMP
162	SNMP Trap
443	HTTPS
34962 to 34964	PROFINET

You can have multiple active connections use the same port number for the local and remote port numbers. For example, a TCP client can connect to multiple servers on port 2500. Both the local and remote ports would normally be port 2500 for the active connection.

You cannot have multiple passive connections use the same port number as the local port number. For example, the CPU does not allow multiple TCP servers (multiple passive connections) to be on local port 2500. The CPU does not know to which of the multiple 2500 ports to route the messages.

TSAPs

Using Transport Service Access Points (TSAPs), ISO-on-TCP protocol allows multiple connections to a single IP address. TSAPs uniquely identify these communication end point connections to an IP address.

The ISO-on-TCP protocol utilizes port 102 exclusively. You cannot set the port number for this protocol; however, you can set the TSAP for the local and remote partner.

The TSAP rules are shown below:

- You always enter TSAPs as an S7-200 SMART string data type (a length byte followed by the string characters).
- TSAPs must be at least 2 characters and no more than 16 ASCII characters in length.
- The local TSAP cannot start with string "SIMATIC-"
- The local TSAP must begin with the hexadecimal character "0xE0" if the TSAP is exactly 2 characters long. For example, the TSAP "$E0$01" is legal but the TSAP "$01$01" is not legal. (The "$" character denotes that the following value is a hexadecimal character.)

8.4.8 PROFINET

8.4.8.1 Introduction

What is PROFINET IO?

PROFINET IO is the Ethernet-based automation standard of PROFIBUS International. It defines a cross-vendor communication, automation, and engineering model.

With PROFINET IO, a switching technology allows all stations to access the network at any time. In this way, the network can be used much more efficiently through the simultaneous data transfer of several nodes. Simultaneous sending and receiving is enabled through the full-duplex operation of Switched Ethernet, with a bandwidth of 100 Mbps.

A PROFINET IO system consists of the following devices:

- PROFINET Controller: Device that addresses the connected IO devices and exchanges input and output signals with field devices.
- PROFINET IO Device: A field device that is monitored and controlled by a PROFINET Controller. A PROFINET IO Device consists of several modules and submodules.

Note

You can have a maximum of 8 PROFINET IO devices and 64 modules on your S7-200 SMART PROFINET network.

Objectives of PROFINET

The objectives of PROFINET are:

- Industrial networking, based on Industrial Ethernet (open Ethernet standard)
- Compatibility of Industrial Ethernet and standard Ethernet components
- High robustness due to Industrial Ethernet devices. Industrial Ethernet devices are suited to the industrial environment (such as temperature and noise immunity).
- Real-time capability
- Seamless integration of other fieldbus systems

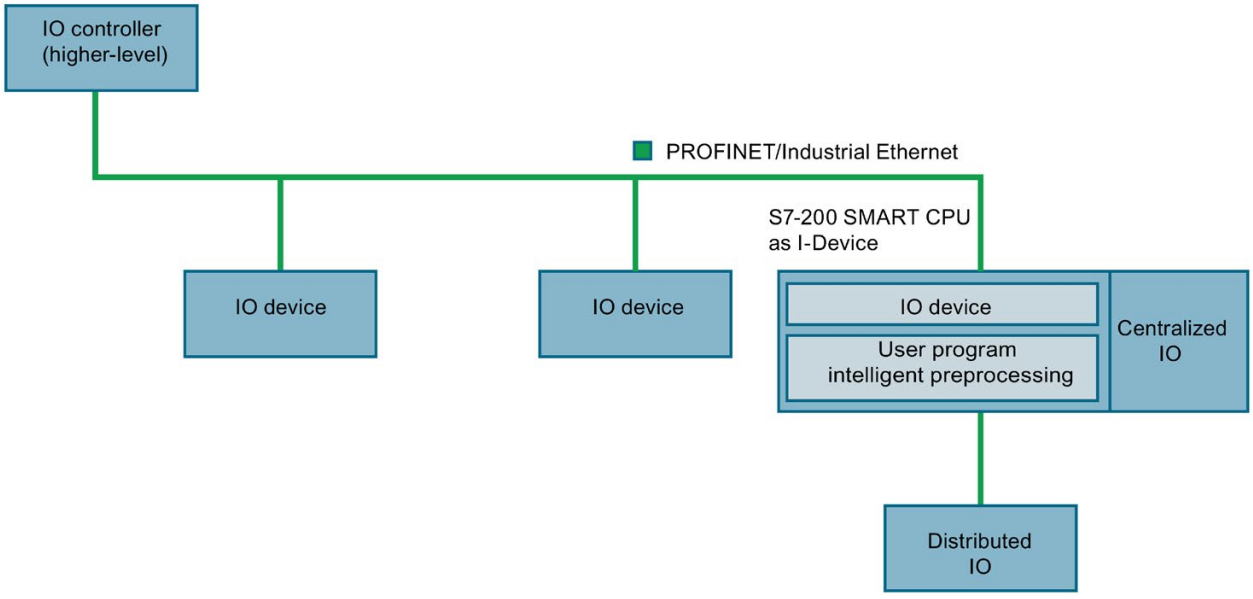
8.4.8.2 I-Device

I-Device functionality

I-Device is a CPU with the "Intelligent IO device" configuration.

The I-Device functionality of a CPU enables data exchange between a CPU and an IO controller, and CPU can work as intelligent preprocessing unit of sub processes.

The preprocessing is handled by the user program on the CPU. The process values acquired in the centralized or distributed (PROFINET IO) I/O are preprocessed by the user program and made available through a PROFINET IO interface to the higher-level IO controller.

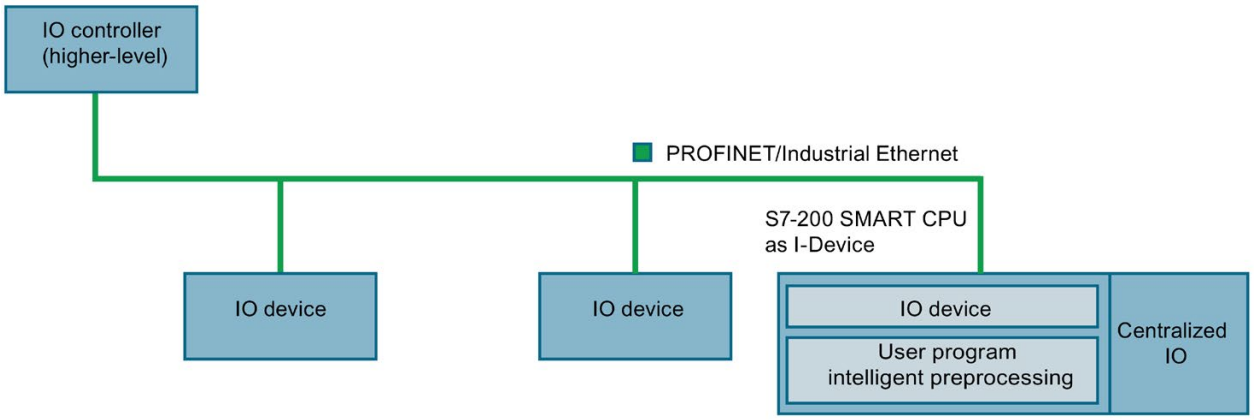


Two types of I-Device are supported for S7-200 SMART V2.5:

- I-Device without lower-level PROFINET IO system
- I-Device with lower-level PROFINET IO system

I-Device without lower-level PROFINET IO system

The I-Device does not have its own distributed IO. The configuration and parameter assignment of I-Device in the role of an IO device is the same as for a distributed I/O system.



I-Device with lower-level PROFINET IO system

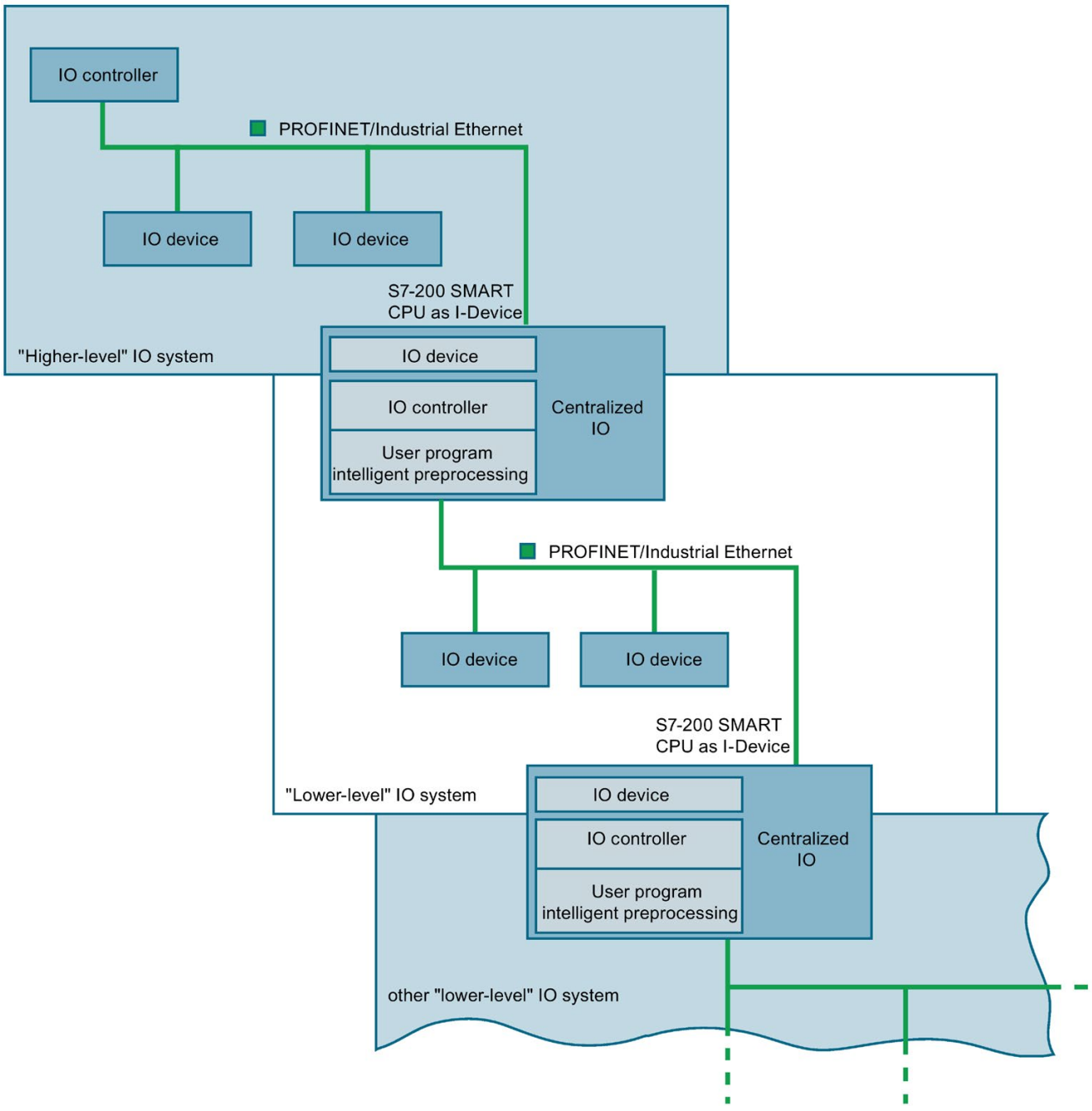
In addition to the role as an IO device, an I-Device can also work as an IO controller on a PROFINET interface.

This means that the I-Device can be part of a higher-level IO system through its PROFINET interface and work as an IO controller that supports its own lower-level IO system.

The lower-level IO system can contain I-Devices. This makes hierarchically structured IO systems possible.

Note

If S7-200 SMART CPU works as I-Device with lower-level PROFINET IO system, the maximum number of supported lower-level IO devices is eight, and only one higher-level controller is supported.



8.4.8.3 Device database file in XML: GSDML

The properties of PROFINET devices are stored in an XML file whose structure and rules are determined by a GSDML schema.

The language used to describe the GSD files is GSDML (Generic Station Description Markup Language). You can get the GSDML schemas (as schema files) from PROFINET device manufacturers.

STEP 7-Micro/WIN SMART supports GSDML Specification 2.34 or earlier versions. When you import a GSDML file, STEP 7-Micro/WIN SMART validates the GSDML file. If the objects pass validation, STEP 7-Micro/WIN SMART puts them into the PROFINET device catalog tree (Page 414).

Before adding an IO device to the PROFINET network, you can import its GSDML files (Page 408) to STEP 7-Micro/WIN SMART.

8.4.8.4 GSDML Management

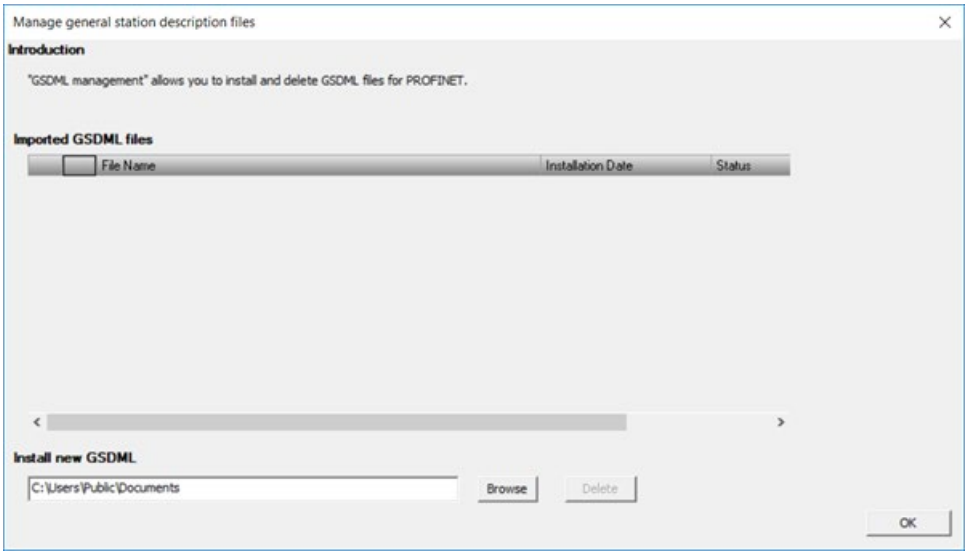
"GSDML Management" allows you to import or delete GSDML files. GSDML files for PROFINET describe the features of PROFINET device and related modules.

Importing GSDML files

To import GSDML files, follow these steps:

1. Click the "GSDML Management" button from the GSDML section of the File menu ribbon strip.

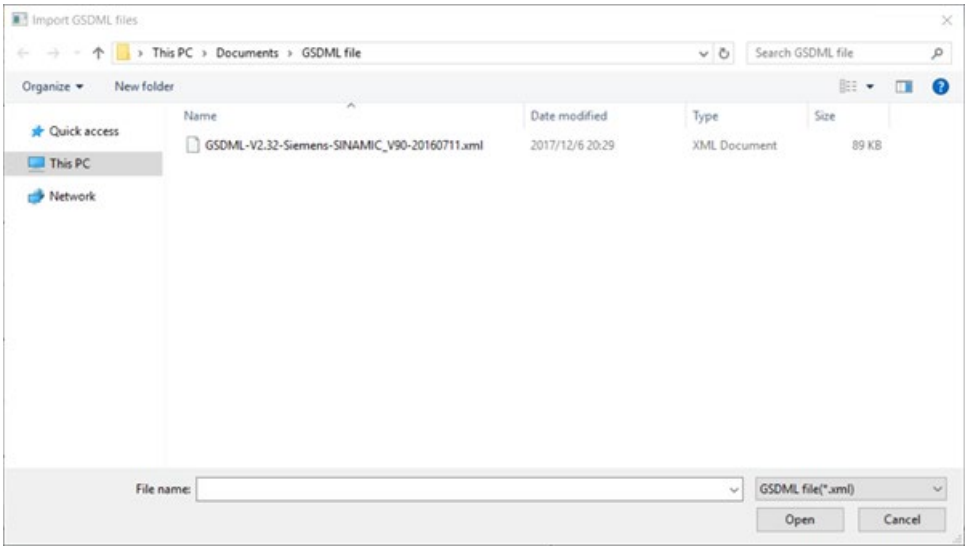
2. Click the "Browse" button in the "Manage general station description files" dialog.



Note

The next time you import GSDML files, the dialog displays the file path you navigate to last time.

3. Navigate to the folder where you save GSDML files.



4. Select the GSDML file you want to import. You can also import multiple GSDML files.

Note

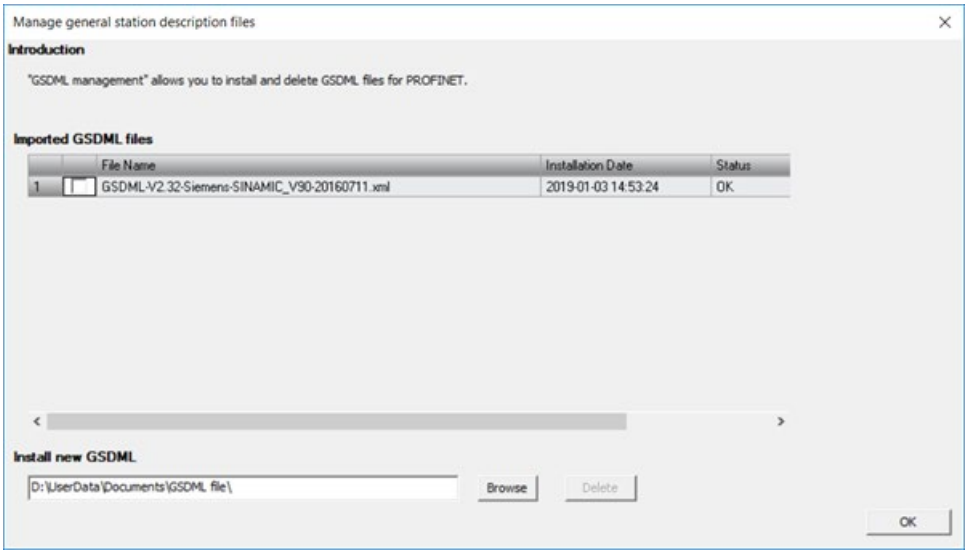
The GSDML file must conform to the GSDML Specification.

For any issue about GSDML files, contact the manufacture.

Note

The maximum number of imported GSDML file is 20.

5. Click the "Open" button. The GSDML file and the installation date displays in the "Imported GSDML files" field.

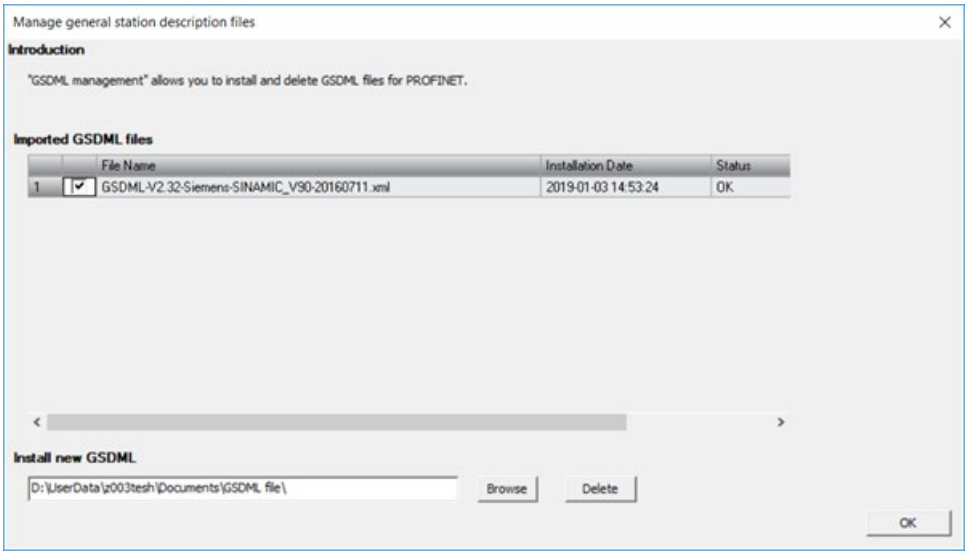


6. Click the "OK" button to close the dialog.

Deleting GSDML files

To delete GSDML files, follow these steps:

1. Click the "GSDML Management" button from the GSDML section of the File menu ribbon strip.
2. Select the GSDML file(s) you want to delete in the "Manage general station description files" dialog.
3. Click the checkbox for your desired GSDML file, and click the "Delete" button. You can also delete multiple GSDML files.



4. Confirm to delete the GSDML file in the pop-up reminder window.
5. Click the "OK" button to close the dialog.

The deleted GSDML file is removed from the "Imported GSDML files" field.

8.4.8.5 PROFINET device naming

All PROFINET devices **must** have a device name and an IP Address. Use STEP 7-Micro/WIN SMART to define the device names. Device names are assigned to the devices through PROFINET DCP (Discovery and Configuration Protocol).

Make sure that the PROFINET devices and the PC are in the same subnet.

Open the "Find the PROFINET Devices" dialog

To open the "Find PROFINET Devices", use one of the following methods:

- Click the "Find PROFINET Devices" button from the Tools area of the Tools menu ribbon strip.

- Open the Tools folder in the project tree; select the "Find PROFINET Devices" node and press Enter, or double-click the "Find PROFINET Devices" node.

Configuring the PROFINET device name in the "Find PROFINET Devices" dialog

To assign a name to a device, proceed with the following steps:

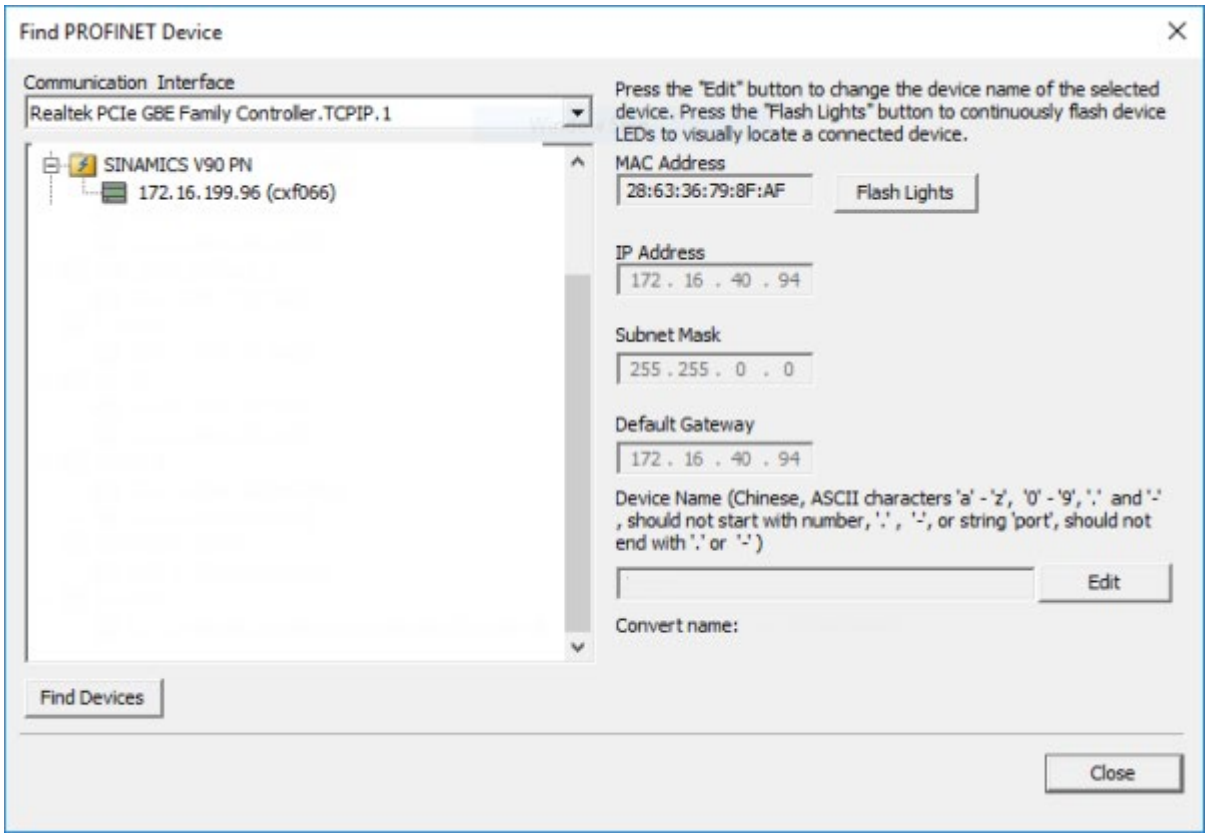
1. Open the "Find PROFINET Devices" tool.

   The "Find PROFINET Devices" dialog detects all connected and available PROFINET devices on the local Ethernet network. After selecting a PROFINET device, STEP 7-Micro/WIN SMART displays the following detailed information about the device:

   – MAC Address
   – IP Address
   – Subnet Mask
   – Default Gateway
   – Device name

2. Click the "Communication Interface" dropdown list, and select the "TCP/IP" Network Interface Card (NIC) for your programming device.

3. Click the "Find Devices" button to display all available PROFINET devices on the local Ethernet network.



Note: To modity the IP address for the PN devices, go to Configuring CPU as a controller (Page 414).

4. Click the required device.

   If you need to identify which device to configure or change, click the "Flash Lights" button.

   The "Flash Lights" function is work for the devices that accord with DCP standard.

5. Click the "Edit" button to change the device name.

Note

The device name follows the standard DNS (Domain Name System) naming conventions. The naming rule is as follows:

- The maximum supported characters is 63.
- The device name can consist of the lower case letters a through z, the digits 0 through 9, the hyphen character "-", and the period character ".".
- The device name can consist of the Chinese characters (Simplified or Traditional).
- The device name must not have the format n.n.n.n where n is a value of 0 through 999.
- You cannot begin the device name with the string port-nnn or the string port-nnn-nnnnn where n is a digit 0 through 9. For example, port-123 and port-123-45678 are illegal device names.
- The device name cannot start or end with the hyphen character "-" or the period character ".".
- The device name cannot start with the digit.
- The device name can contain name segments. Name segments are separated by the period character ".". Each name segment follows the rules above. For example, plc.1 and smart-.device are illegal device names.

6. Click the "Set" button. The device name is assigned to the device.
7. When finished, click "OK".

8.4.8.6 LED status indicators for PROFINET network

The CPU status LED "Error" on the front panel indicates the operational state of PROFINET:

- For the IO device, the LED blinks red at 1 Hz rate when either of the following conditions occurs:
  - Any IO device loses connection to IO controller
  - Any IO device has an alarm
- For the I-Device, the LED blinks red at 1 Hz rate when either of the following conditions occurs:
  - Any I-Device loses connection to higher-level controller
  - Any I-Device is connected with higher-level controller, but the configuration of I-Device doesn't match with the configuration of higher-level controller.

8.4.8.7 Configuring CPU as a controller

Requirements

Ensure that the following conditions are met before you start to configure CPU as a controller:

- GSDML files are imported for the PROFINET devices. (Page 408)

Note

You can also configure a PROFINET controller through catalog configuration (Page 435).

- The CPU module is selected in the system block dialog (Page 129).

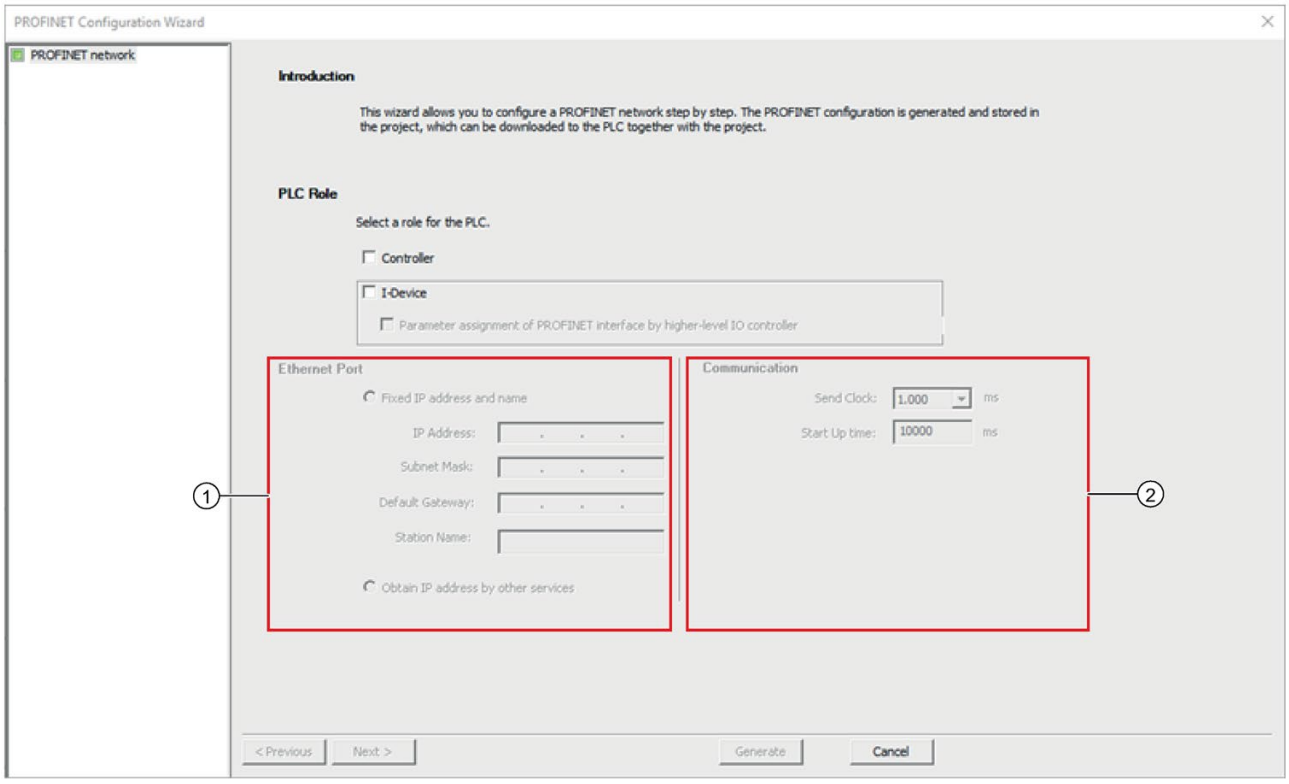
Note

You can only use ST/SR 20, ST/SR 30, ST/SR 40 or ST/SR 60 as a PROFINET controller. The firmware version must be V2.4 or later.

Procedure

To configure a CPU as a controller, follow these steps:

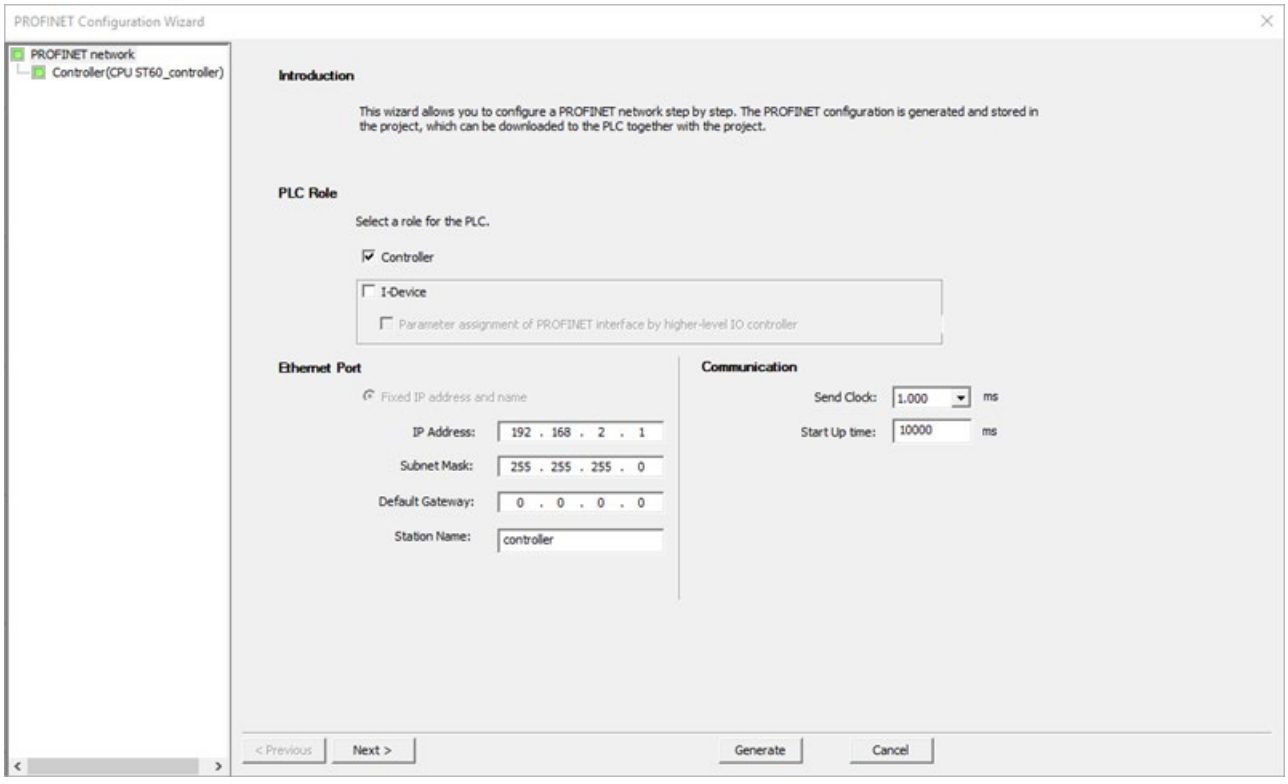
1. Open the PROFINET Wizard.



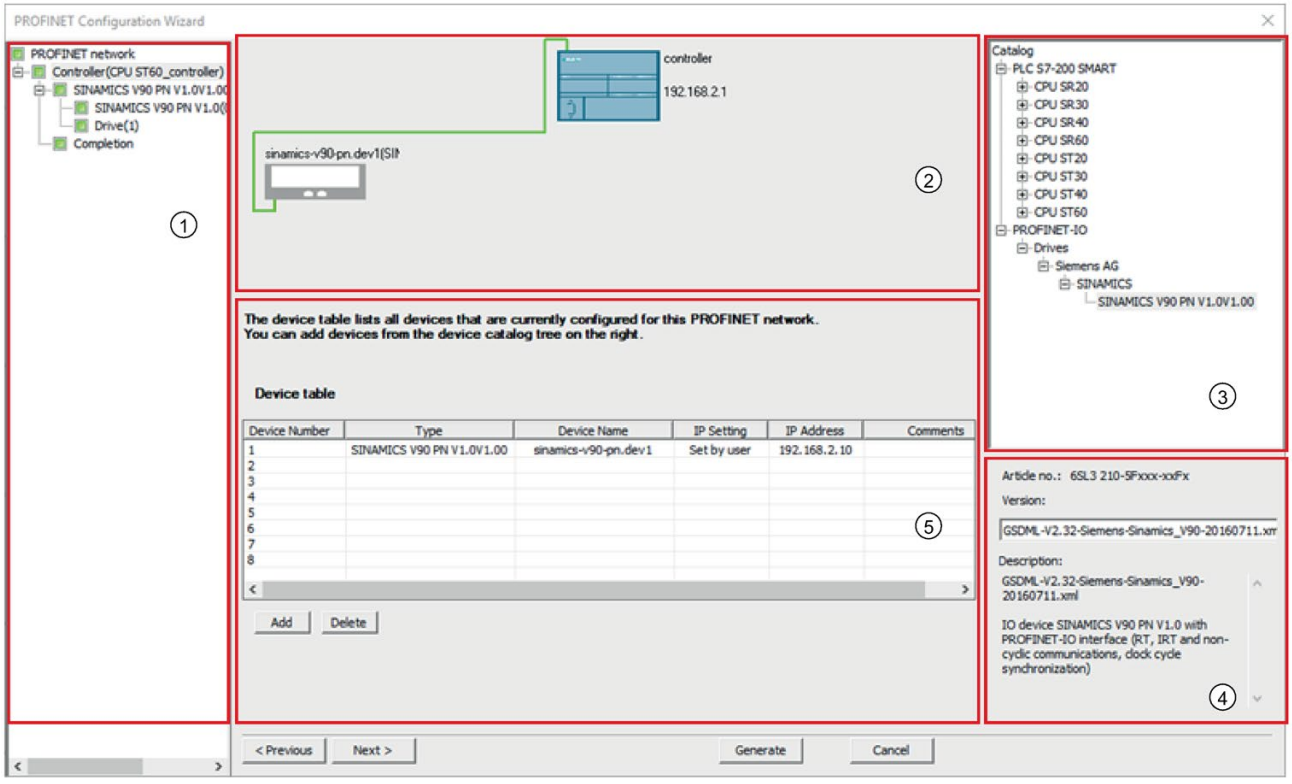
①	Ethernet Port	There are two methods to configure IP address: • Assign a fixed IP address • Obtain IP address by other services, such as DCP and programming instruction. **Note:** The IP address setting varies as follows: • If you select the CPU as an I-Device, you can select any method above. • If you select the CPU as a controller, you only need to assign a fixed IP address and station name.

②	Communica-tion	You can assign "Send Clock" and "Start up time" for the controller. **Note:** Your setting for the "start up time" impacts the amount of time the CPU takes to switch to RUN mode after the CPU is powered on. The default value for "start up time" is 10 s, and the maximum value is one minute. The situation varies as follows: • If "start up time" is set as zero, the CPU switches to RUN mode immediately. • If the full connection is established within the "start up time", the CPU switches to RUN mode after the connection is established. • If the connection is not established after the "start up time", the CPU switches to RUN mode after the "start up time".

2. Select the "Controller" checkbox.
3. Enter a fixed IP address and station name, set send clock and start up time.



4. Click "Next" in the bottom of the window or click the CPU name in the PROFINET network tree to enter the configuration page.



①	PROFINET network tree	The structure tree of PROFINET network.
②	PROFINET network view	The PROFINET network view.
③	PROFINET device catalog tree	The PROFINET device catalog. There are two nodes in the PROFINET device catalog tree: • PLC S7-200 SMART: S7-200 SMART CPUs are displayed. • PROFINET-IO: Devices defined by the imported GSDML files are displayed. **Note:** If you add S7-200 SMART CPU as the device, you can import GSDML file or just add a specific CPU from the "PLC S7-200 SMART" node (Page 435) in the PROFINET device catalog tree. **Note:** For the unicode character, STEP 7-Micro/WIN SMART supports importing the GSDML file that contains MBCS (Multi-Bytes Character Set) character. If you import the GSDML file that contains special unicode characters, for example, "Ä" and "È", STEP 7-Micro/WIN SMART cannot display GSDML file description and GSDML parameters normally and the question mark "?" shows in the PROFINET wizard.

④	PROFINET device description	The description of the device you select from the PROFINET device catalog tree.
⑤	Device table	List of devices that are configured in the PROFINET network. You can enter the device name or add comments (optional) for the devices in the device table. The address assignment attributes in the "IP setting" drop-down list are as follows: • Fixed: The PROFINET IO Device has a fixed IP for the interface and you do not need to set the IP address here. • Set by user: The IP address of PROFINET IO Device is assigned by DCP protocol through controller, and you must set the IP Address.

5. Select a PROFINET device in the PROFINET device catalog tree and click the "Add" button below the device table. You can also drag and drop the device to the device table.

   Result: The PROFINET device is added in the device table, network view, and PROFINET network tree.

6. Enter the device name, set IP address and add optional comments.

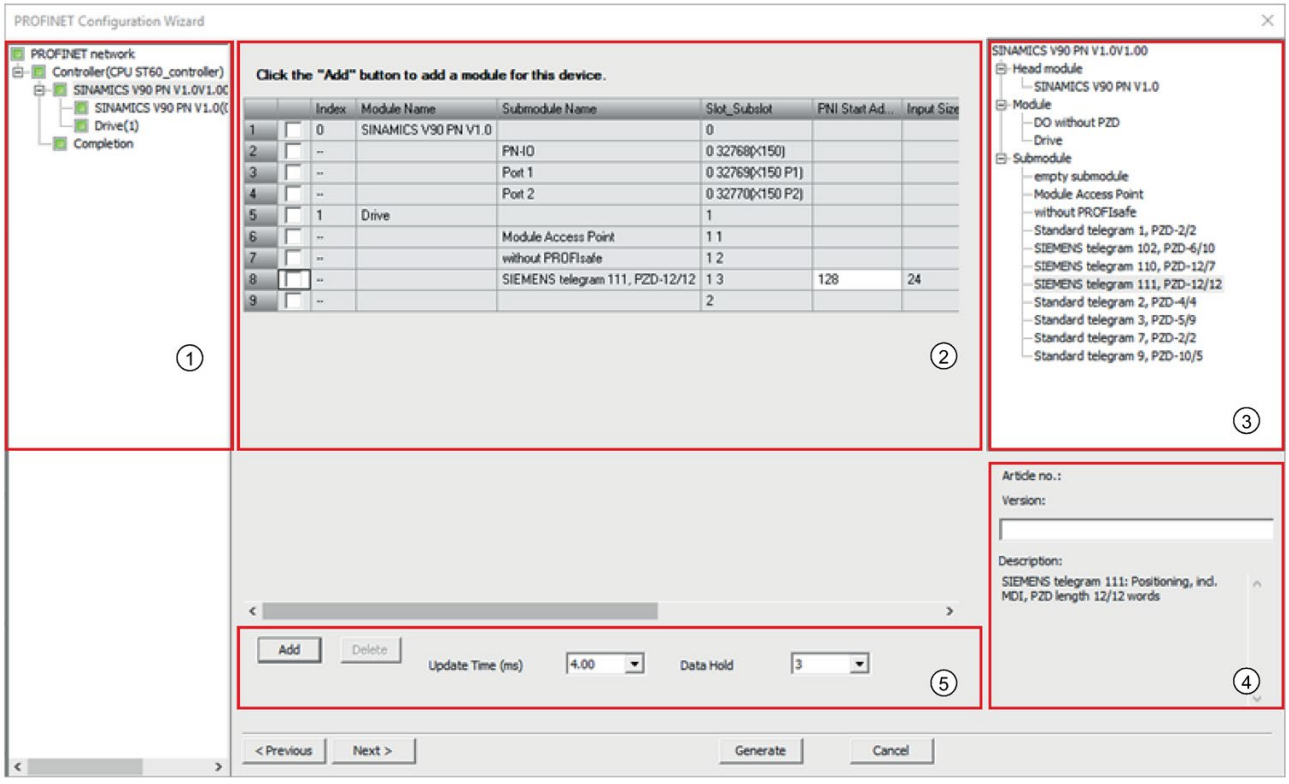
Note

For any PROFINET device, you need to keep the device name in the device table consistent with the actual device name.

Note

The comments that you enter in the PROFINET wizard are not downloaded to PLC. The comments are only saved in the project file.

7. Click the "Next" button or modules in the PROFINET network tree to enter the module configuration dialog.



①	PROFINET network tree	The structure tree of PROFINET network.
②	Submodule table for device	The table lists the module and submodules for the selected devices in the PROFINET network. You can assign the PNI/PNQ start address for the device here. For the range of the PNI/PNQ start address, refer toMemory ranges and features (Page 904).
③	Module catalog tree	The list for the selected devices.
④	Device parameters	The parameters for the currently selected device.
⑤	Submodule description	The description of the submodule you select from the PROFINET device catalog tree.

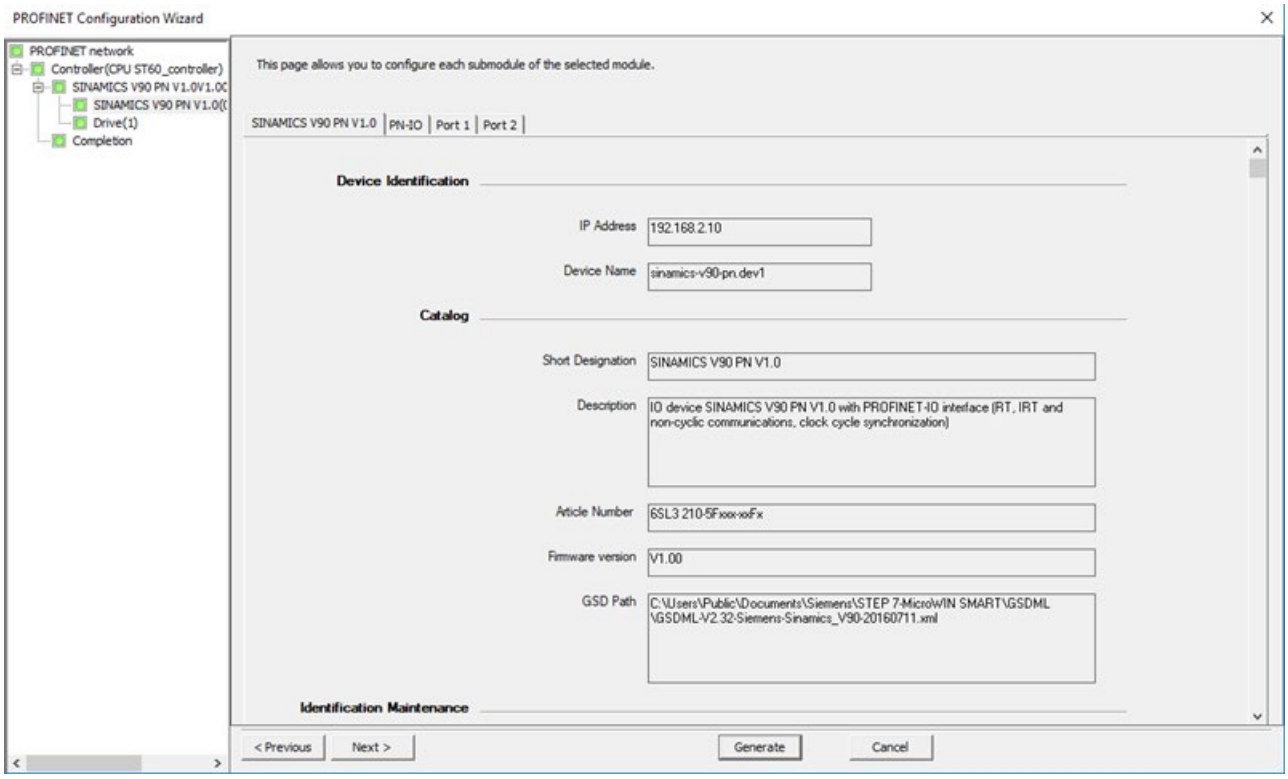
8. Click the module or submodule from the module catalog tree. The slots where you can add a module or submodule turn green.

Note

Make sure the module type you select exists in your real network.

9. Click the "Add" button or drag and drop the module to the slot.

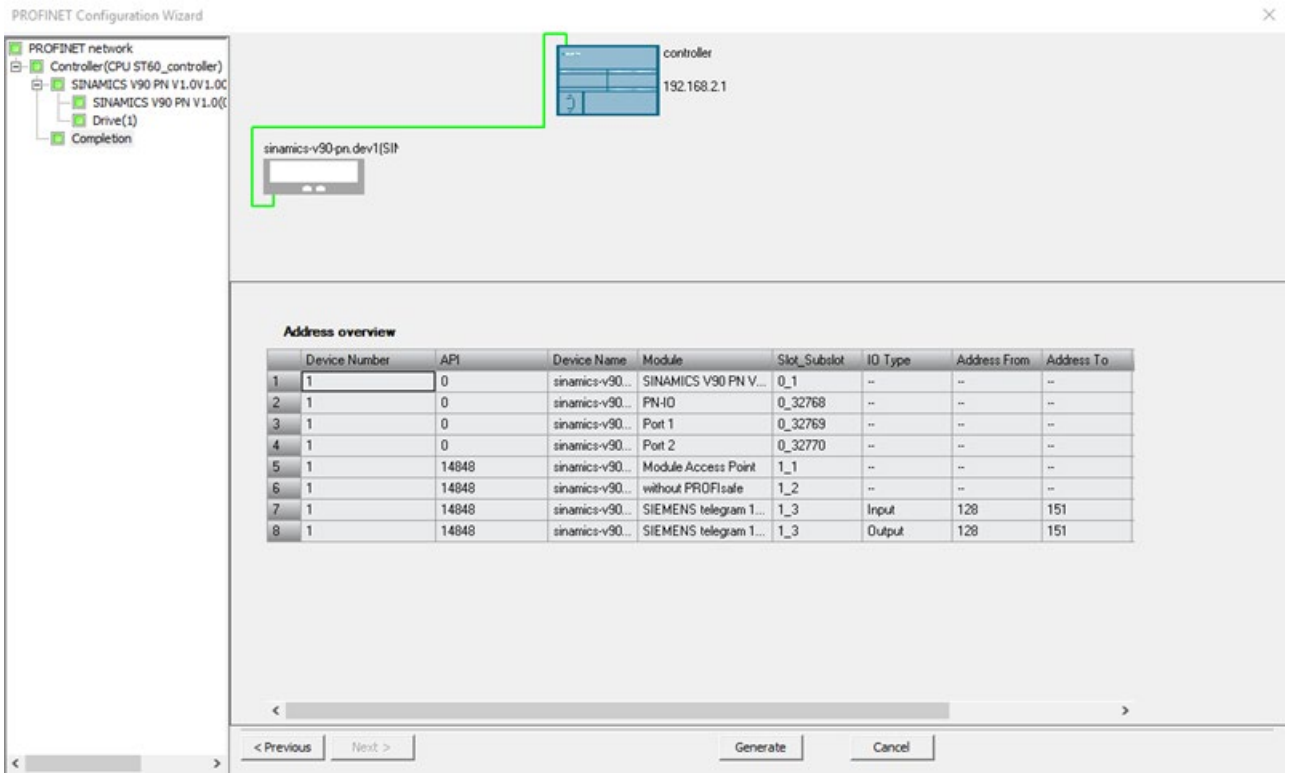
10. Click the "Next" button or the module name in the PROFINET network tree to configure the module or to check the detailed information of the module.



Note

For the "Enable autonegotiation" option in the "Port" tab, you must keep the setting of this option consistent with that of the partner port. Otherwise, the operating parameters of the connected network are not detected, the data transmission rate and transmission mode cannot be optimally set.

11. Verify your PROFINET configuration.



12. Click the "Generate" button to save the configuration.

Result

The PROFINET devices and modules are added to the network and are ready for downloading.

8.4.8.8 Configuring CPU as an I-Device without lower-level PROFINET IO system

Requirement

Ensure that the CPU module is selected in the system block dialog.

Note

You can only use ST/SR 20, ST/SR 30, ST/SR 40 or ST/SR 60 as an I-Device and the firmware version must be V2.5 or later.

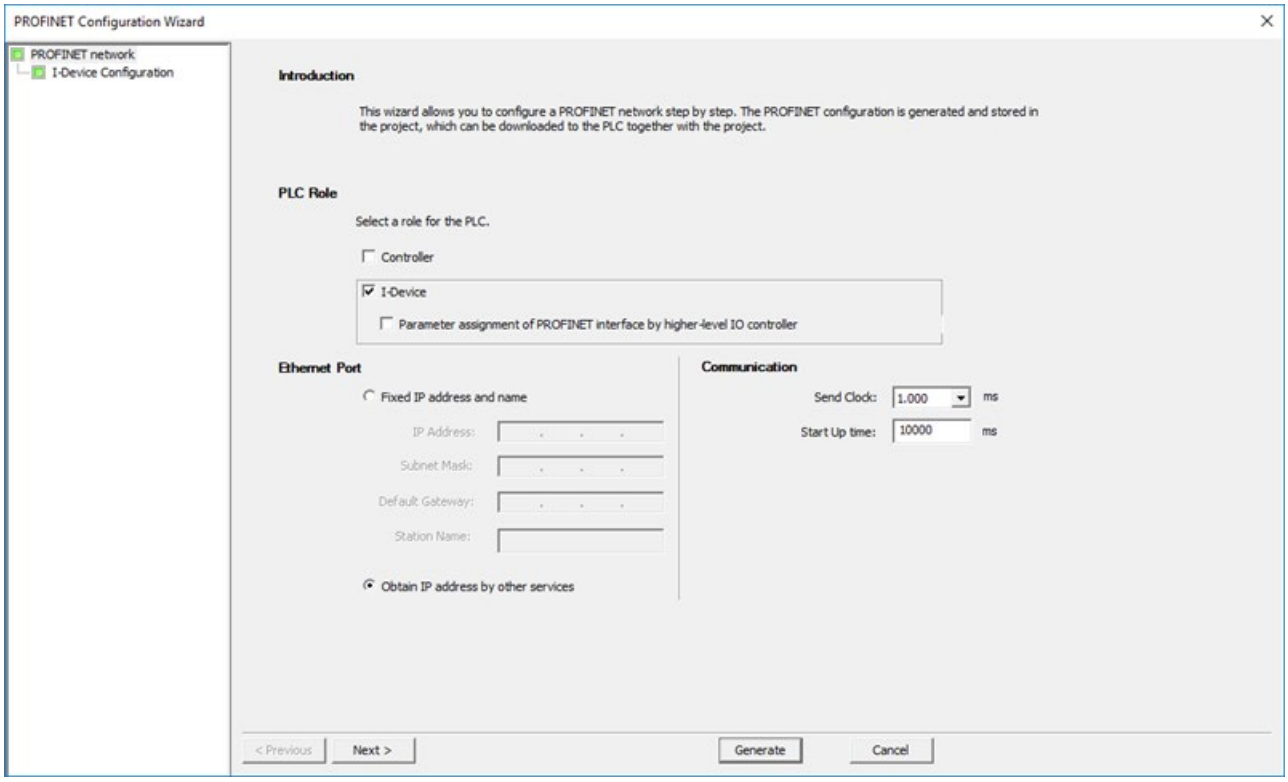
Procedure

To configure an I-Device project in PROFINET wizard, follow these steps:

1. Open the PROFINET wizard.
2. Select the "I-Device" checkbox.

   The I-Device PROFINET interface parameters and ports can be assigned by the I-Device itself or a higher-level IO controller:

   If you select the "Parameter assignment of PROFINET interface by higher-level IO controller" checkbox, the I-Device PROFINET interface parameters are set by the higher-level controller.

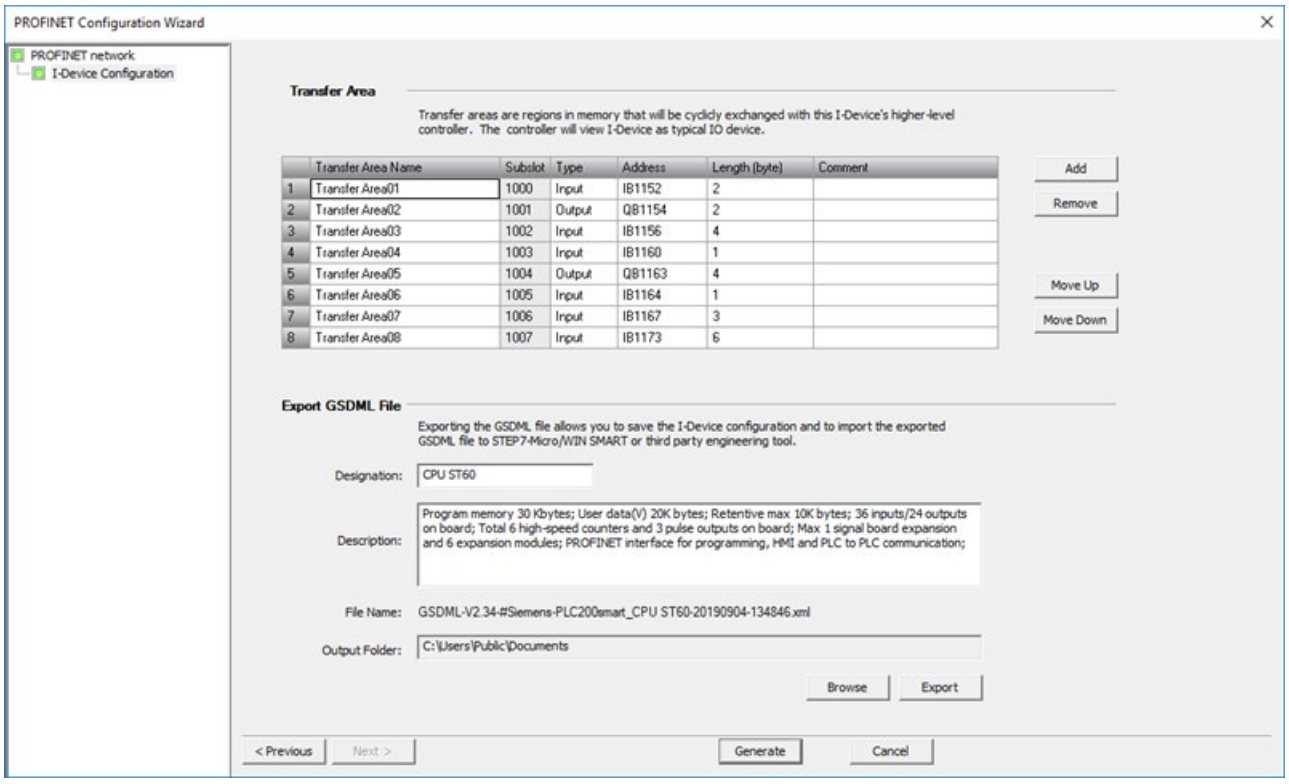


3. Configure an IP address.

   For IP address, you can assign a fixed IP address or obtain IP address by other services.

4. Set "Send Clock" and "Start up time".

5. Click the "Next" button to enter the configuration page. "Transfer area" table and "Export GSDML" field are displayed.



Transfer area work as an interface to the user program of the I-Device CPU and exchange the communication data between higher-level controller and I-Device. Inputs are processed in the user program and outputs are the result of the processing in the user program.

Note

- The maximum number of supported transfer area is 8.
- At least one transfer area must be configured.

The "Transfer Area" table lists the following items:

Items	Description
Transfer area name	Submodule name of one subslot. You can define the transfer area name or use the default transfer area name. Transfer area name is not downloaded to PLC. **Note:** The maximum number of supported characters is 64.
Subslot	Subslot number. The subslot number starts from 1000, and the system automatically generates a sequential digit for the subslot number.

Items	Description
Type	Transfer area type. You can select one transfer area as input or output.
Address	Transfer area IO address. You can enter the input or output address. • The input range is from IB1152 to IB1279. • The output range is from QB1152 to QB1279.
Length	Transfer area IO data length. • The maximum size of transfer area total inputs is 128 bytes. • The maximum size of transfer area total outputs is 128 bytes.
Comment	You can add optional comments for one transfer area.

The "Export GSDML" field lists the following items:

Items	Description
Designation	The device family name is displayed by default. For example, CPU ST60. You can also enter any information as you need: • The Chinese, English and digit character are supported. • The character '|', '\\', '/', ':', '\', '', '<', '>', '*' and '?' are not supported. • The maximum number of supported characters is 20.
Description	The CPU information is displayed by default or you can enter any information as you need.
File name	GSDML file name is automatically generated, and is of an assigned format.
Output folder	The folder where you want to save the GSDML file. You can click the "Browse" button to select one folder or use the default folder.

6. Configure the transfer area.
7. Click the "Export" button to export the GSDML file.
8. Click the "Generate" button to save the configuration.
9. Download this I-Device project to a CPU.

8.4.8.9 Configuring an I-Device project with lower-level PROFINET IO system

Requirement

Ensure that the CPU module is selected in the system block dialog.

Note

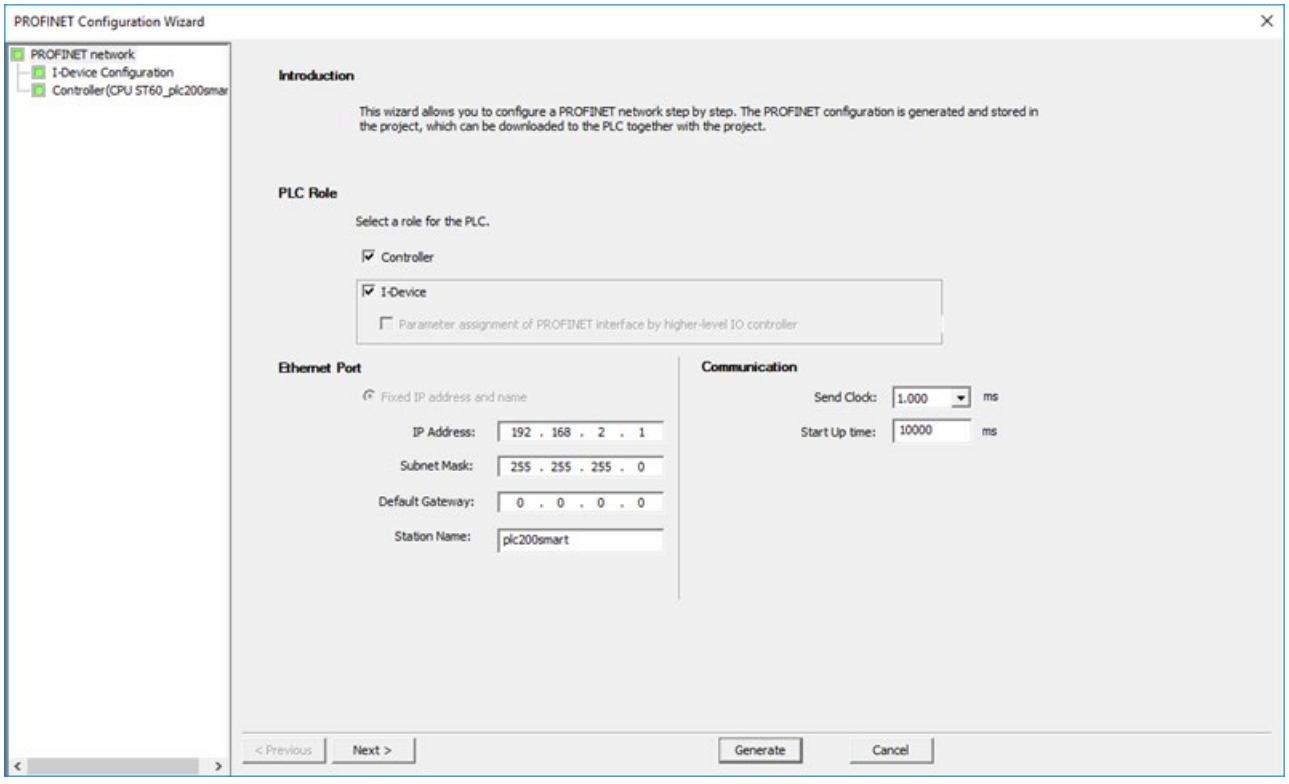
You can only use ST/SR 20, ST/SR 30, ST/SR 40 or ST/SR 60 as an I-Device with lower-level PROFINET IO system and the firmware version must be V2.5 or later.

Procedure

To configure an I-Device project with lower-level PROFINET network, follow these steps:

1. Open the PROFINET wizard.
2. Select the "Controller" and "I-Device" checkbox. Then two separate "I-Device" and "Controller" configuration pages are displayed in the wizard.

   If you configure the I-Device with lower-level PROFINET IO system, the I-Device PROFINET interface parameters cannot be set by the higher-level controller.



3. Assign a fixed IP address, enter the station name, and set "Send Clock" and "Start up time".
4. Configure the CPU as an I-Device (Page 421) in the I-Device configuration page.

   In this step, you export one I-Device GSDML file.
5. Configure the CPU as a controller (Page 414)and add an IO Device in the controller configuration page.
6. Click the "Generate" button to save the configuration.
7. Download this I-Device project to a CPU.

Note

For an I-Device with lower-level PROFINET IO system, if there is an error with I-Device's lower-level IO device, this I-Device sends the following error to the higher-level controller: Device is running with error.

8.4.8.10 Example: Configuring a PROFINET network

This example describes how to configure a PROFINET network project where one S7-200 SMART CPU works as an I-Device without lower-level PROFINET IO system and one S7-200 SMART CPU works as a higher-level controller.

There are two methods to configure this PROFINET network:

- Method 1: Configuring a PROFINET network by importing GSDML file.
- Method 2: Configuring a PROFINET network through catalog configuration.

  Catalog configuration means that there are built-in S7-200 SMART CPUs under the "PLC S7-200 SMART" node of PROFINET device catalog tree. You can select a CPU model in the "PLC S7-200 SMART" node to add it as IO device of one controller, and you don't need to import the GSDML file.

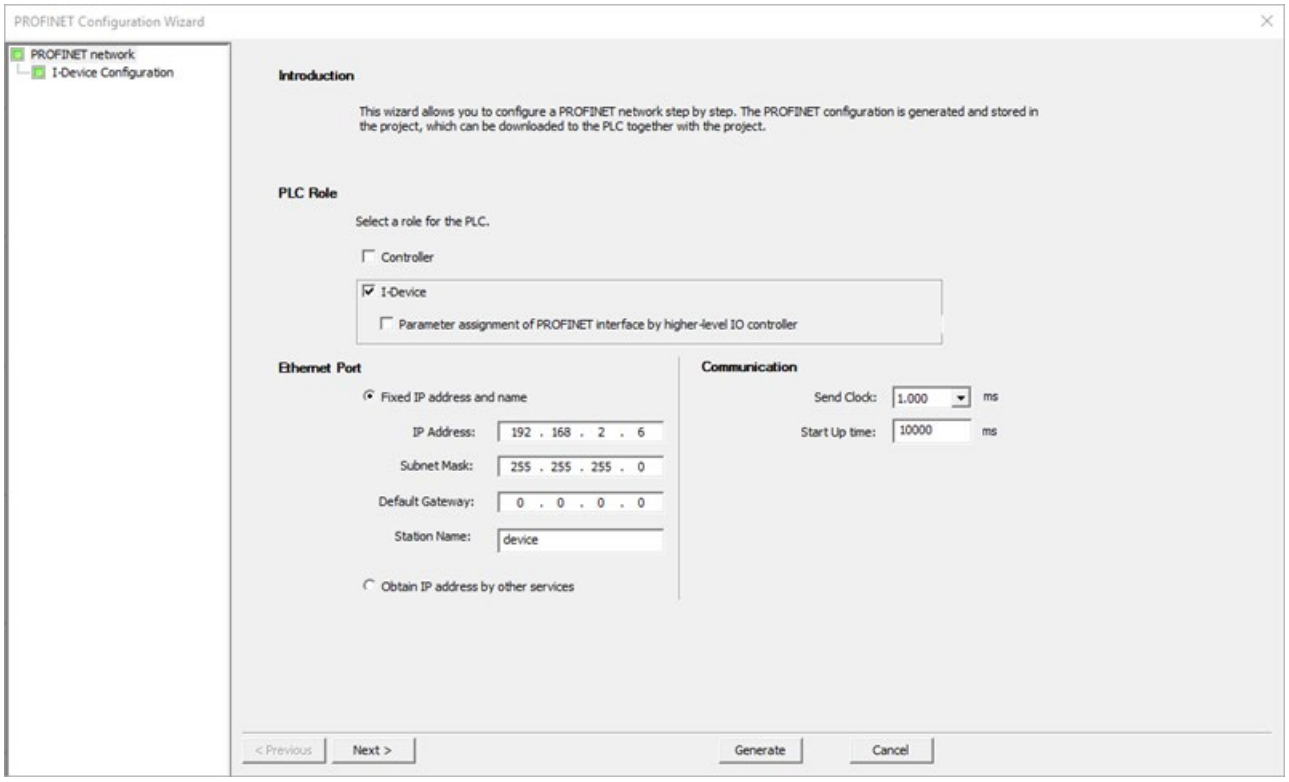
Preparation

1. Select one ST-60 S7-200 SMART CPU as an I-Device, name it as "device" and assign the following IP address:
   - IP address: 192.168.2.6
   - Subnet Mask: 255.255.255.0
   - Default Gateway: 0.0.0.0
2. Select one ST-60 S7-200 SMART CPU as a controller, name it as "controller" and assign the following IP address:
   - IP address: 192.168.2.1
   - Subnet Mask: 255.255.255.0
   - Default Gateway: 0.0.0.0

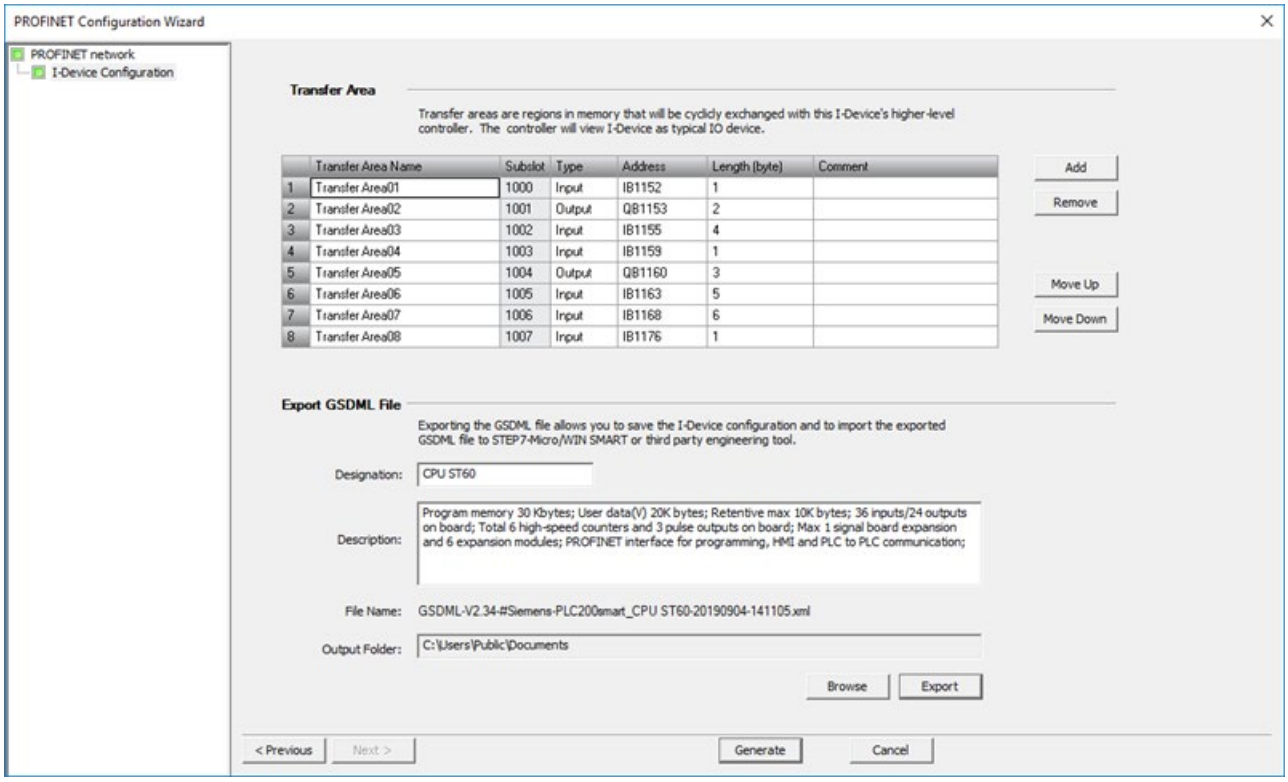
Method 1: Configuring a PROFINET network by importing GSDML file

You can configure a PROFINET network by importing GSDML file as follows:

1. Open the PROFINET wizard, select the "I-Device" checkbox and assign a fixed IP address and name as follows:

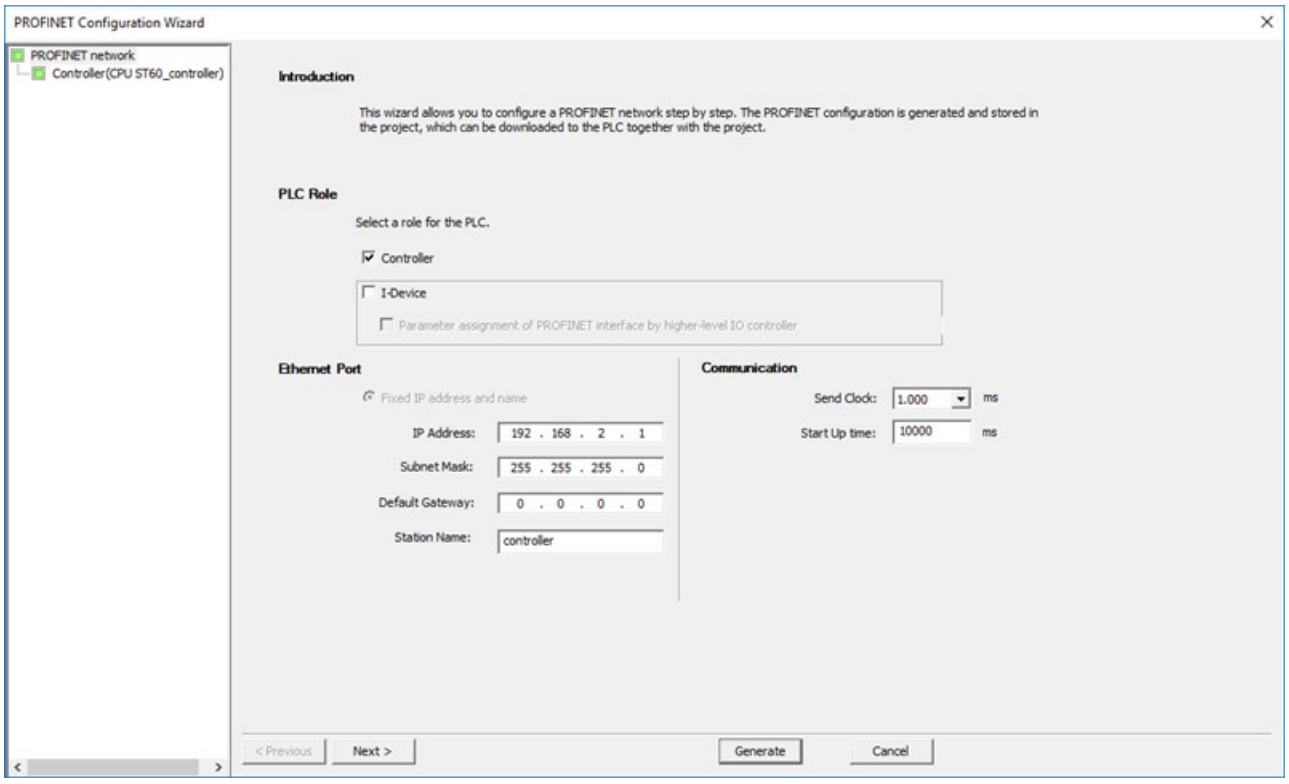


2. Configure the following 8 transfer areas:

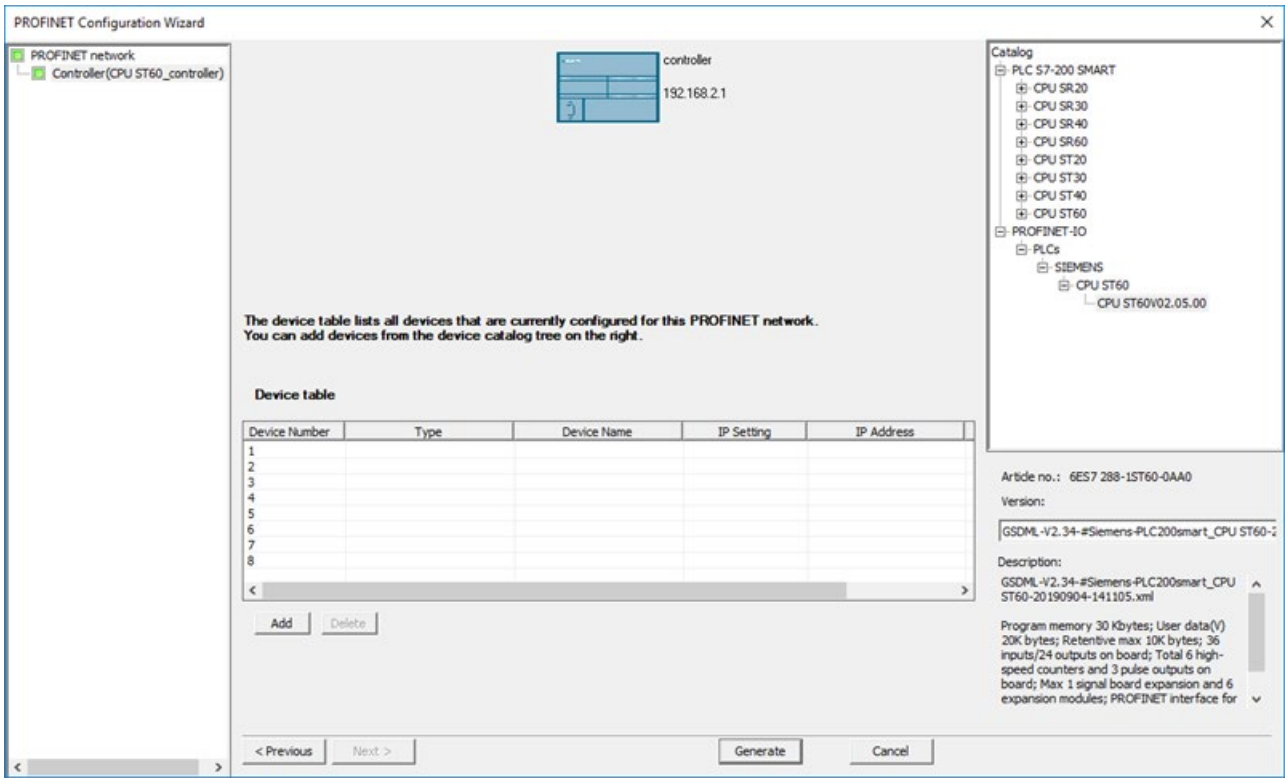


3. Export the GSDML file. An I-Device GSDML file is created.
4. Generate the configuration. Then an I-Device project is created.
5. Download this project to I-Device CPU.
6. Open a new project and import the I-Device GSDML file.

7. Navigate to the PROFINET wizard. Select the "Controller" checkbox, assign a fixed IP address and name as follows:

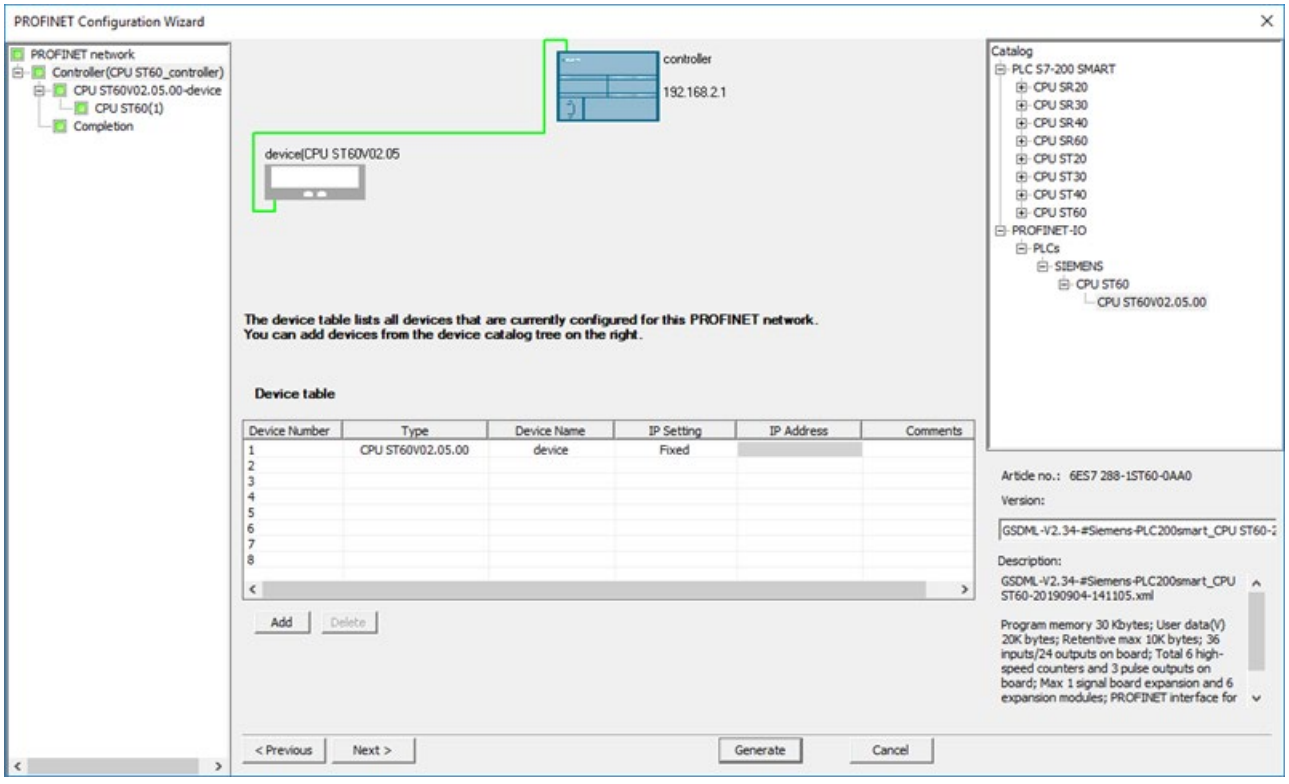


8. Navigate to the next page, and the CPU ST60 defined by the imported GSDML file is displayed in the "PROFINET-IO" node under "PROFINET device catalog tree".

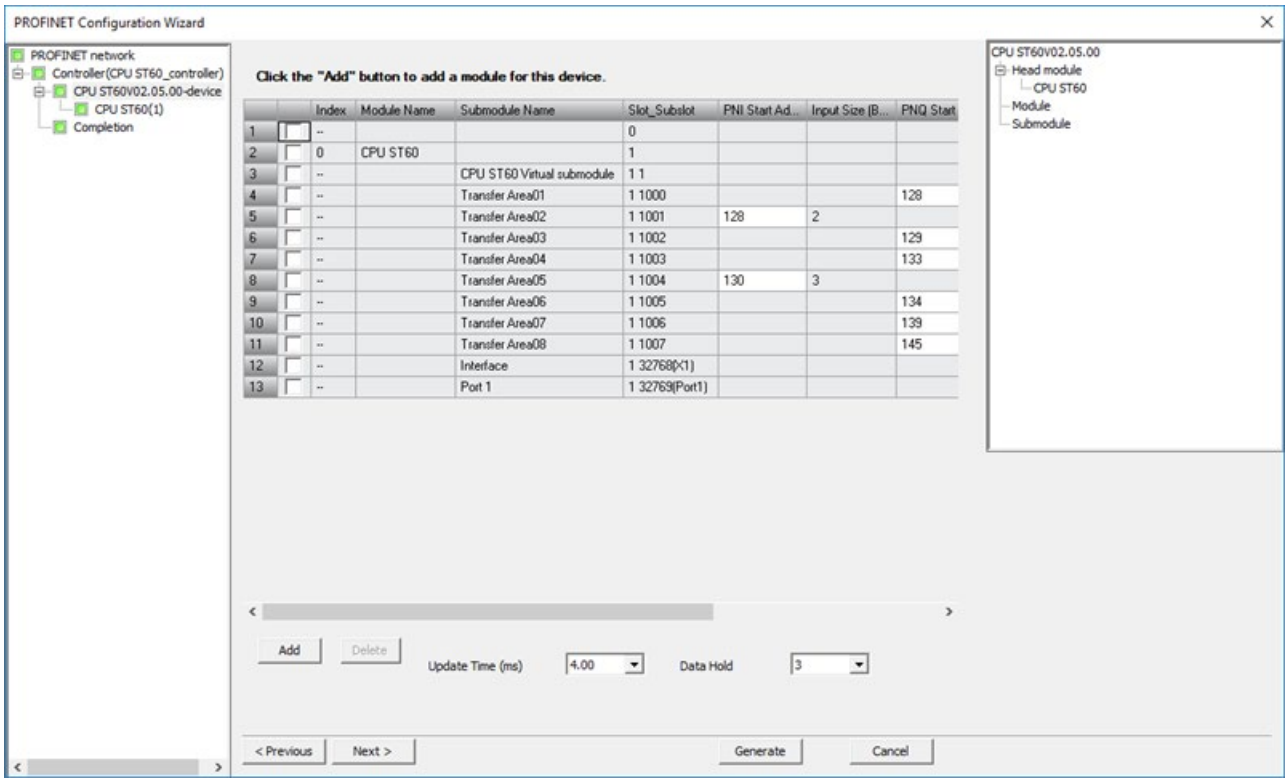


9. Add "CPU ST60 V02.05.00" to device table:

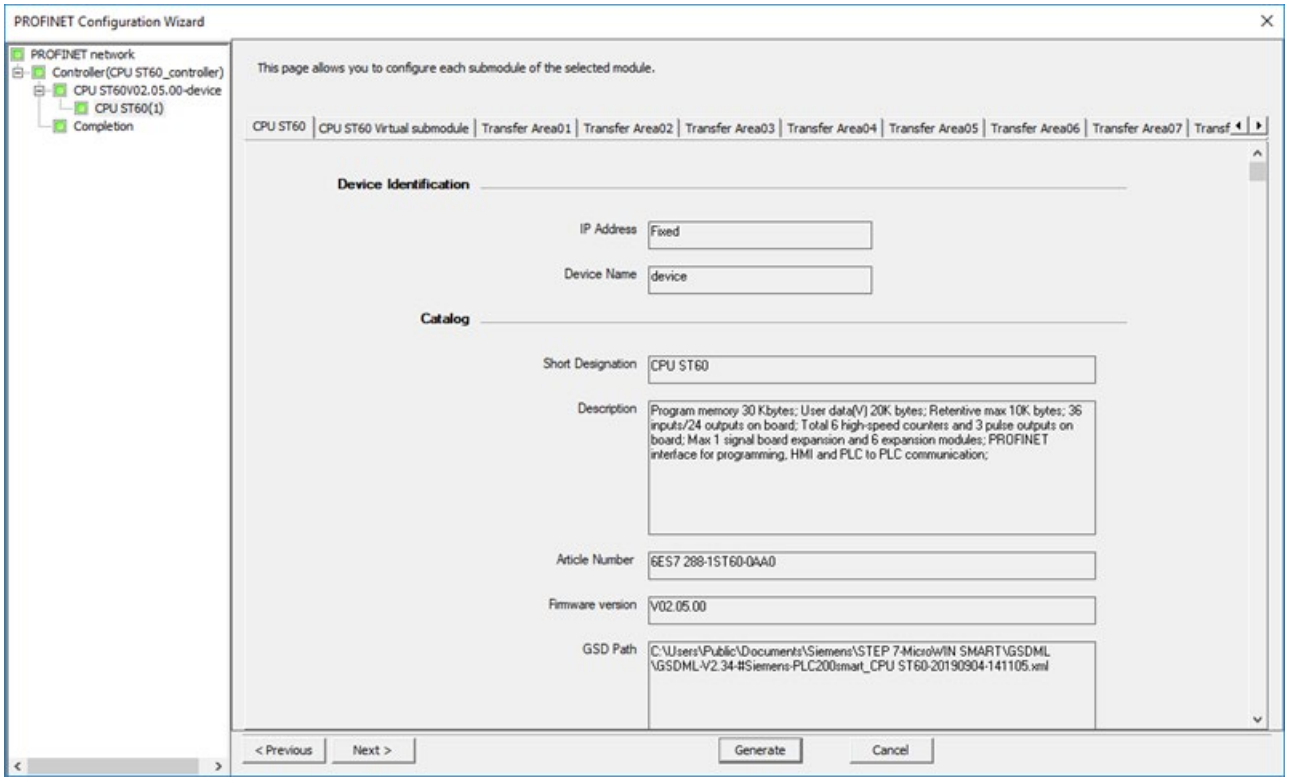
   Since "Fixed IP address and name" is selected for I-Device configuration in this example, and you add this I-Device in device table by importing the I-Device GSDML file, you do not need to change the device name and IP address here.



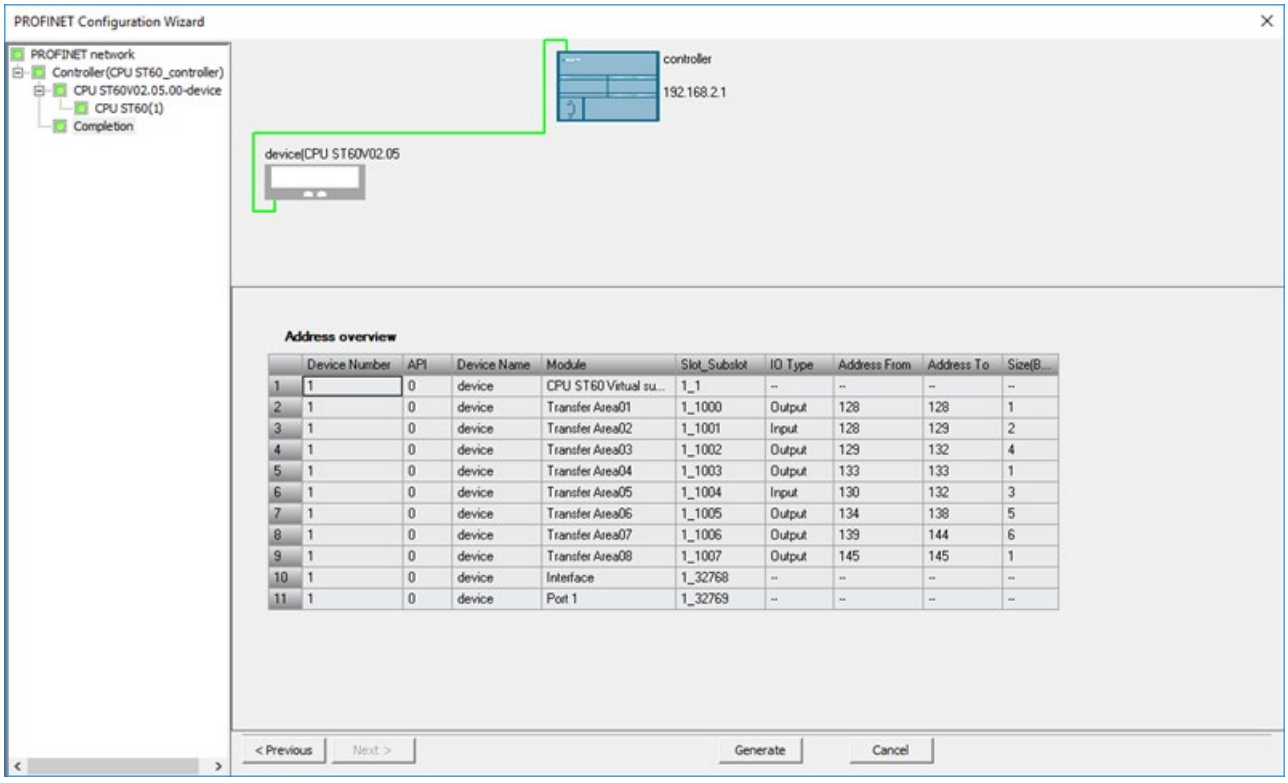
10. Navigate to the next page and the transfer area configuration information is automatically displayed in the module configuration page.



11.Navigate to the next page to check the submodule information.



12.Verify the configuration in the completion page.



13.Click the "Generate" button to save the configuration.

14.Download this project to controller CPU.

Note

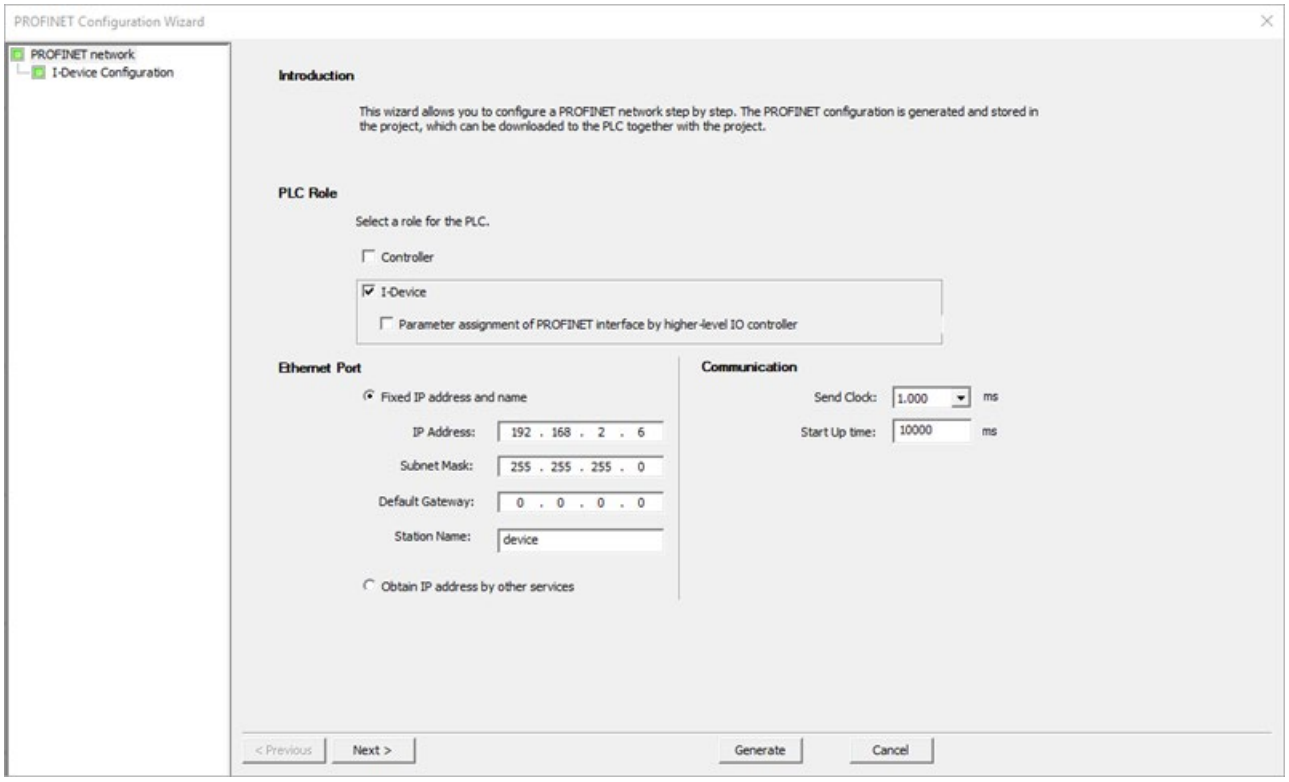
For a S7-200 SMART CPU that works as a controller, if this controller configures I&M data record of its lower-level device and you download the project to the controller, after this controller connects with its lower-level device, the I&M data record will be written to the lower-level device.

For a S7-200 SMART CPU that works as an I-Device, the I&M data record will be cleared if you download a project to the I-Device.

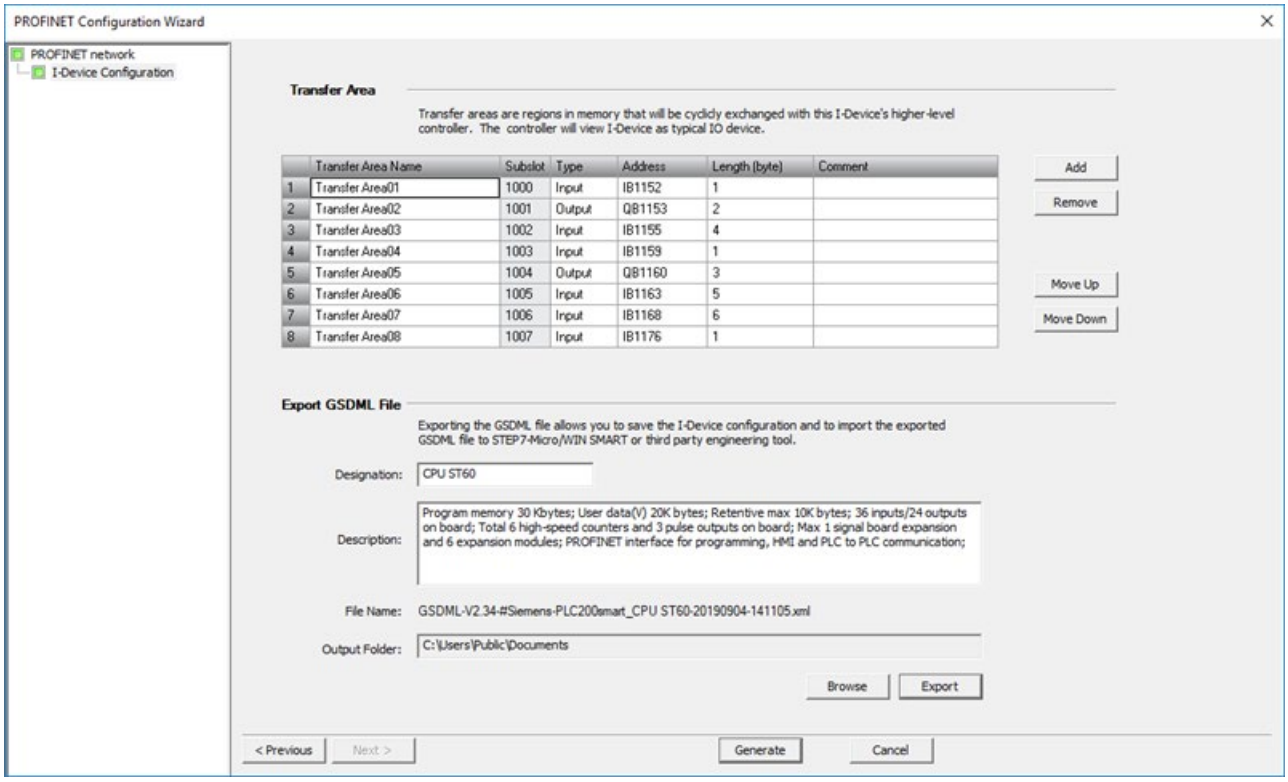
Method 2: Configuring a PROFINET network through catalog configuration

You can configure a PROFINET network through catalog configuration as follows:

1. Open the PROFINET wizard, select the "I-Device" checkbox and assign a fixed IP address and name as follows:

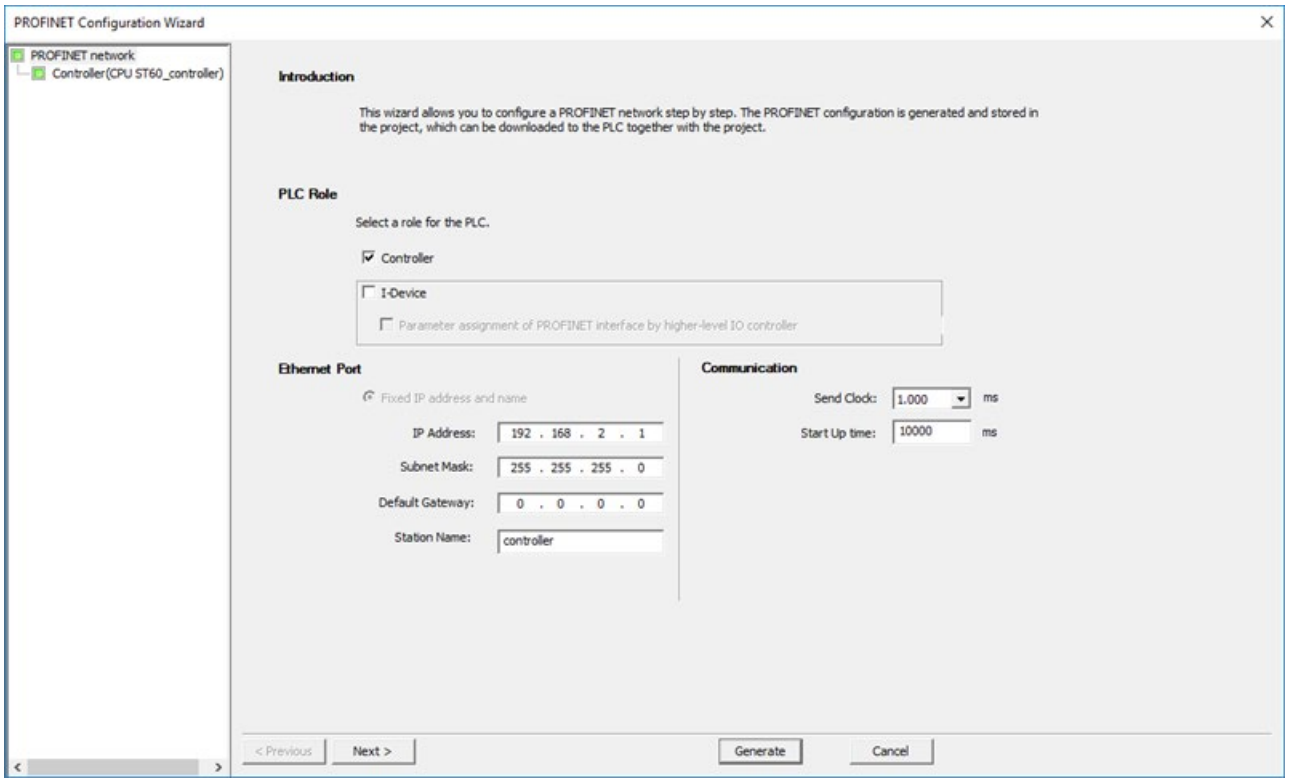


2. Configure the following 8 transfer areas:

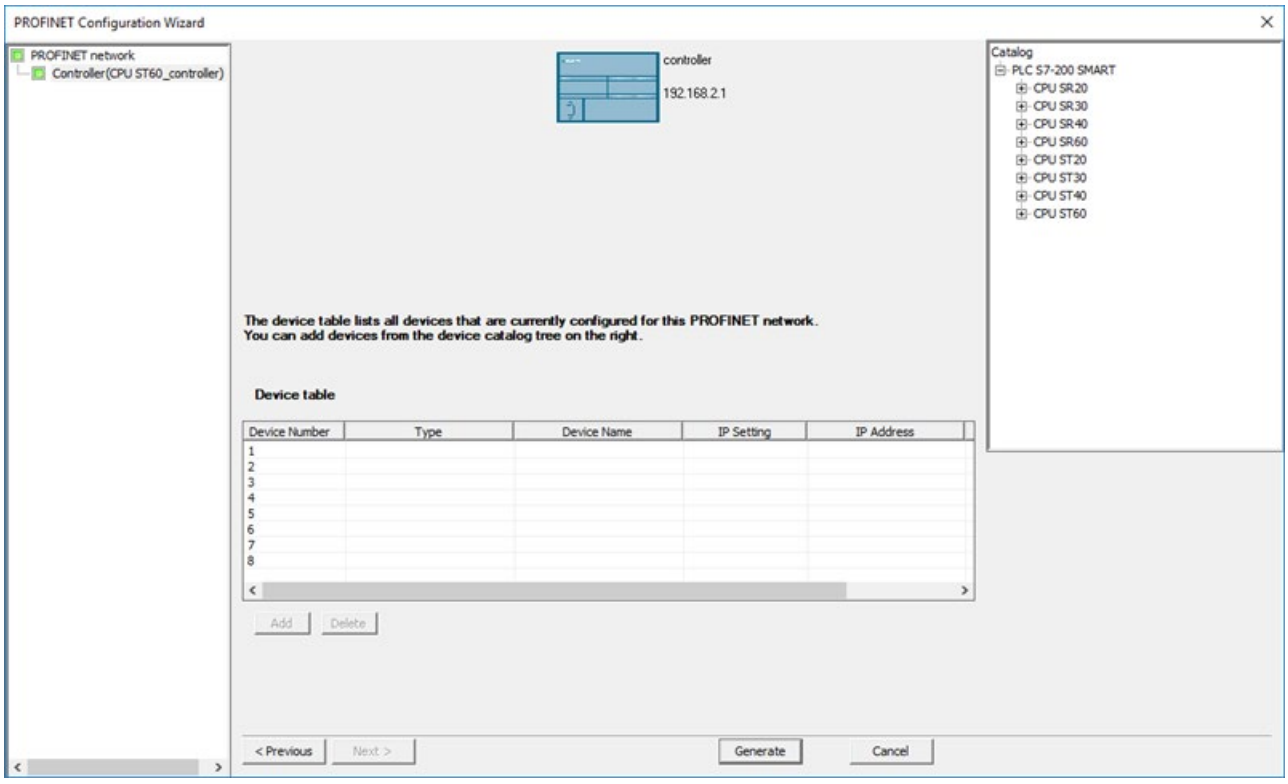


3. Click the "Generate" button to save the configuration.
4. Download this I-Device project to I-Device CPU.

5. Open a new project and navigate to the PROFINET wizard. Select the "Controller" checkbox, and assign a fixed IP address and name as follows:



6. Navigate to the device configuration page. S7-200 SMART CPUs are displayed in the "PLC S7-200 SMART" node of "PROFINET device catalog" tree.



7. Add CPU ST60V2.5.0 in the device table, and configure it as follows:
   - Device name: device

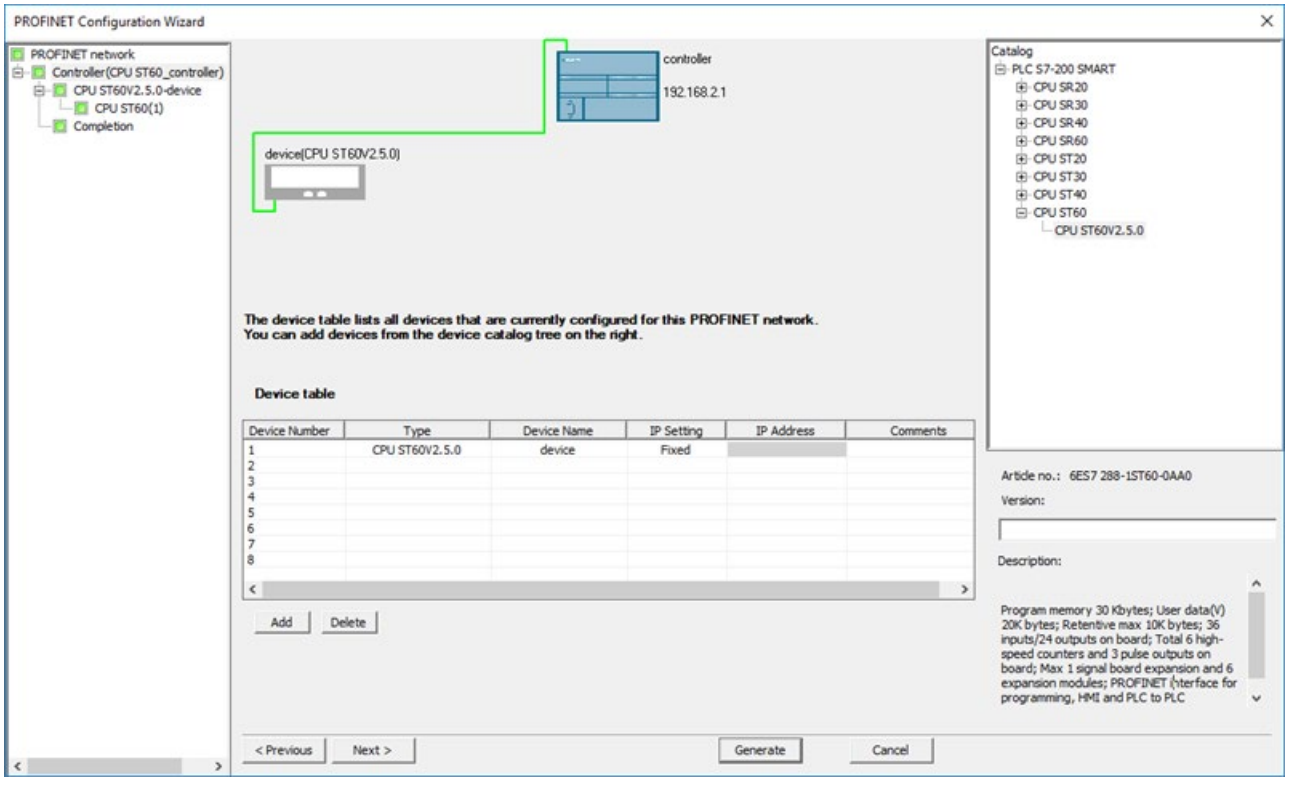
     Note

     If you select "Fixed IP address and name" for I-Device configuration and add this I-Device as IO device of a higher-level controller, keep the "Device Name" here consistent with the "Station Name" of I-Device.

   - IP Setting: Fixed.

     Note

     You need to select the "IP Setting" option as "Fixed" for IP address of I-Device is fixed in the I-Device project.

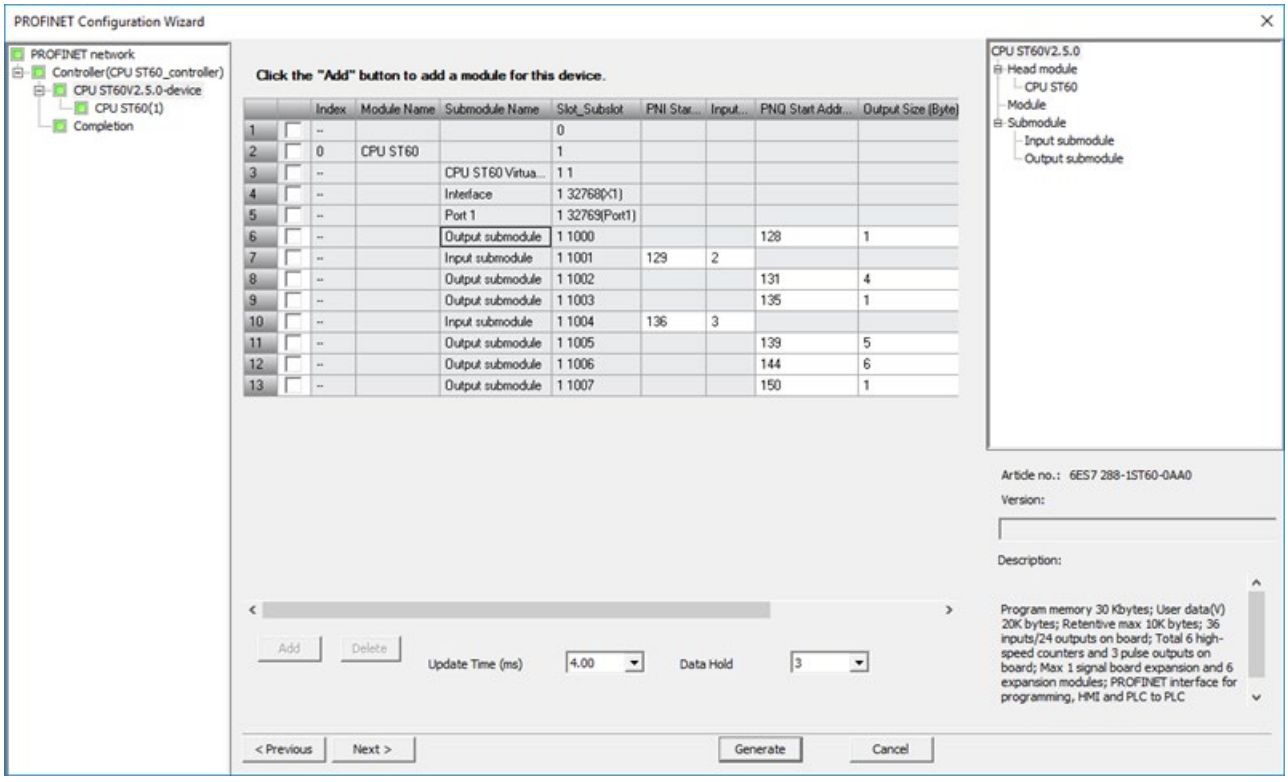


8. Click the "Next" button and configure modules.

Note

For subslots in the "Slot_Subslot" column, select the "Input submodule" and click the "Add" button to add the submodule if you set this slot as "output transfer area" in the I-Device configuration.

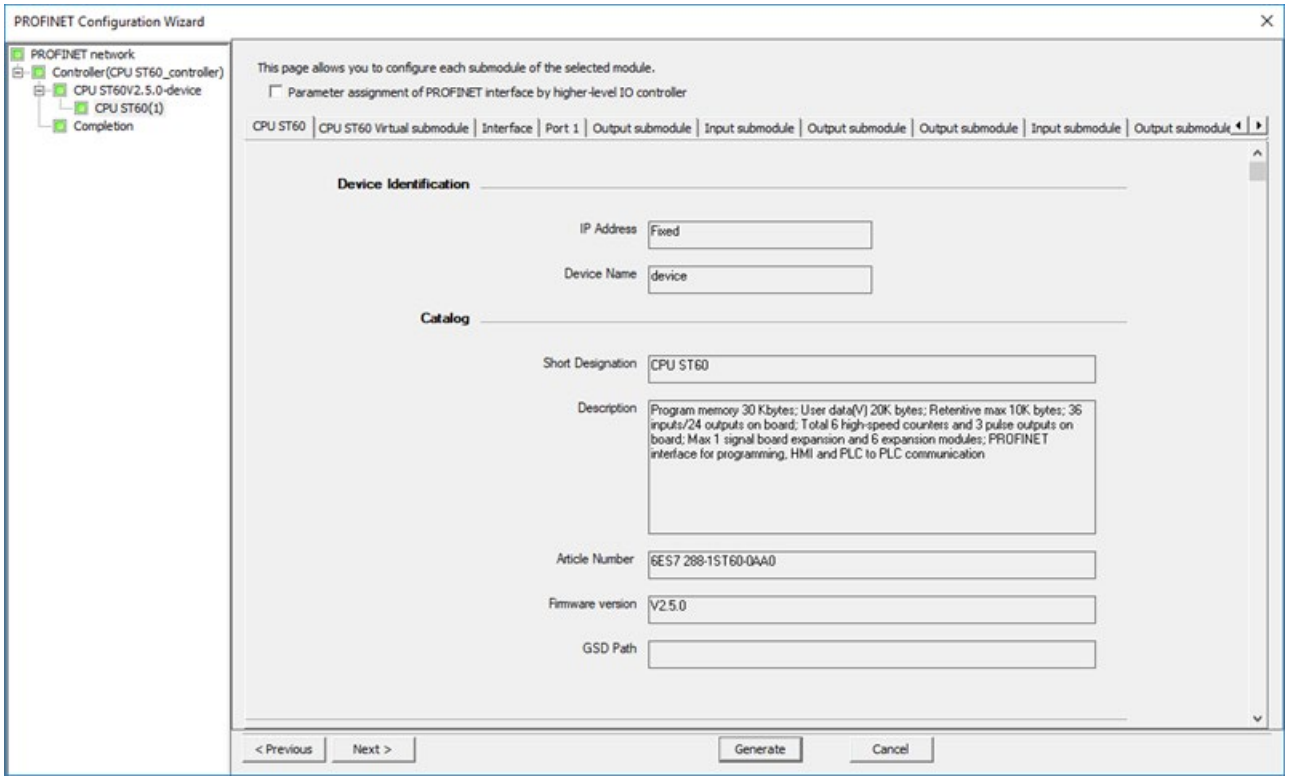
Select the "Output submodule" and click the "Add" button to add the submodule if you set this slot as "input transfer area" in the I-Device configuration. For example, the subslot 1000 is set as "Input" in the transfer area of I-Device project, here you need to set this subslot as "output submodule".



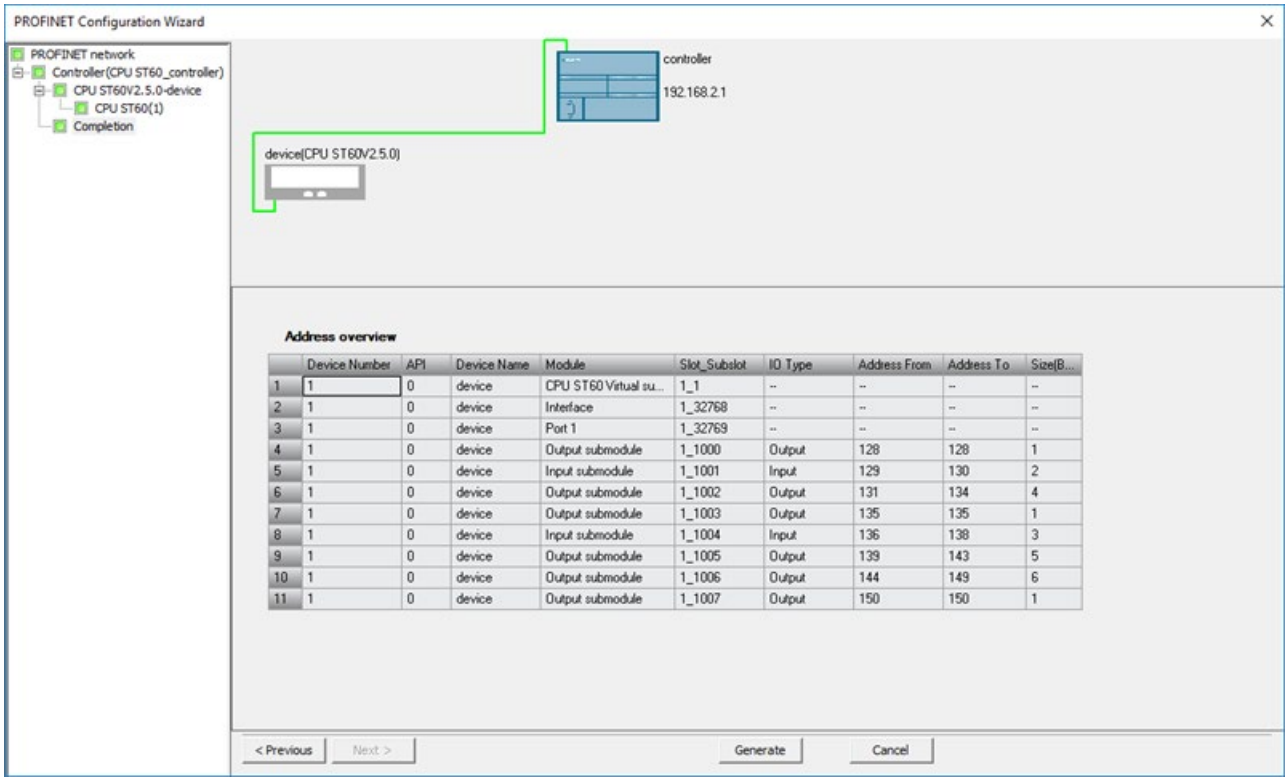
9. Click the "Next" button and configure submodules or check the submodule configuration page.

Note

If you select the checkbox "Parameter assignment of PROFINET interface by higher-level IO controller" when you configure I-Device project, you need to select the checkbox in this page. Then you can configure parameters under "Port 1" tab.



10.Verify the configuration in the completion page.



11.Click the "Generate" button to save the configuration.

12.Download this project to controller CPU.

8.5 PROFIBUS

The PROFIBUS protocol is designed for high-speed communications with distributed I/O devices (remote I/O). A PROFIBUS system uses a bus master to poll DP I/O devices distributed in a multi-drop fashion on an RS485 serial bus.

There are many PROFIBUS devices available from a variety of manufacturers. These devices range from simple input or output modules to motor controllers and PLCs. A PROFIBUS DP device is any peripheral device which processes information and sends its output to the master. The DP device forms a passive station on the network (since it does not have bus access rights) and can only acknowledge received messages or send response messages to the master upon request. All PROFIBUS DP devices have the same priority, and all network communication originates from the master.

A PROFIBUS master forms an "active station" on the network. PROFIBUS DP defines two classes of masters. A class 1 master (normally a central programmable controller (PLC) or a PC running special software) handles the normal communication or exchange of data with the DP devices assigned to it. A class 2 master (usually a configuration device, such as a laptop or programming console used for commissioning, maintenance, or diagnostics purposes) is a special device primarily used for commissioning DP devices and for diagnostic purposes.

PROFIBUS networks typically have one master and several DP I/O devices. (Refer to the figure below.) You configure the master device to know what types of DP devices are connected and at what addresses. The master initializes the network and verifies that the DP devices on the network match the configuration. The master continuously writes output data to the DP devices and reads input data from them.

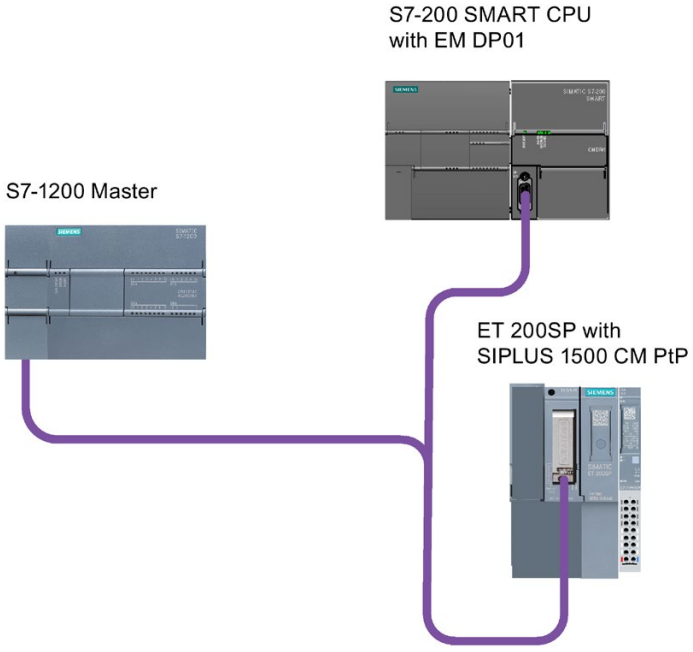
When a PROFIBUS DP master configures a DP device successfully, it then owns that DP device. If there is a second master device on the network, it has very limited access to the DP devices owned by the first master.

The EM DP01 PROFIBUS DP module connects the S7-200 SMART CPU to a PROFIBUS network as a DP device. The EM DP01 can be the communications partner of DP V0/V1 masters. You can access the EM DP01 GSD file at Siemens Customer Support.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

In the figure below, the S7-200 SMART CPU is a DP device for an S7-1200 controller:



You can configure two PROFIBUS EMs per S7-200 SMART CPU (ST and SR models only).

The local CPU stores the PROFIBUS EM configuration data, and you set the PROFIBUS addresses with switches on each module. This allows simple replacement of these communications modules when necessary.

8.5.1 EM DP01 PROFIBUS DP module

8.5.1.1 Distributed Peripheral (DP) standard communications

PROFIBUS DP (or DP Standard) is a remote I/O communications protocol defined by the European Standard EN 50170. Devices that adhere to this standard are compatible even though they are manufactured by different companies. DP stands for distributed peripherals, that is, remote I/O. PROFIBUS stands for Process Field Bus.

The EM DP01 PROFIBUS DP module has implemented the DP Standard protocol as defined for DP devices in the following communications protocol standards:

- EN 50 170 (PROFIBUS) describes the bus access and transfer protocol and specifies the properties of the data transfer medium.
- EN 50 170 (DP Standard) describes the high-speed cyclic exchange of data between DP masters and DP devices. This standard defines the procedures for configuration and parameter assignment, explains how cyclic data exchange with distributed I/O functions, and lists the diagnostic options which are supported.

A DP master is configured to know the addresses, DP device types, and any parameter assignment information that the DP devices require. The DP master is also told where to place data that is read from the DP devices (inputs) and where to get the data to write to the DP devices (outputs). The DP master establishes the network and then initializes its DP devices. The DP master writes the parameter assignment information and I/O configuration to the DP device. The DP master then reads the diagnostics from the DP device to verify that the DP device accepted the parameters and the I/O configuration. The DP master then begins to exchange I/O data with the DP device. Each transaction with the DP device writes outputs and reads inputs. The data exchange mode continues indefinitely. The DP devices can notify the DP master if there is an exception condition, and the DP master then reads the diagnostic information from the DP device.

Once a DP master has written the parameters and I/O configuration to a DP device, and the DP device has accepted the parameters and configuration from the DP master, the DP master owns that DP device. The DP device only accepts write requests from the DP master that owns it. Other DP masters on the network can read the DP device's inputs and outputs, but they cannot write anything to the DP device.

8.5.1.2 Using the EM DP01 to connect an S7-200 SMART as a DP device

You can connect the S7-200 SMART CPU to a PROFIBUS DP network through the EM DP01 PROFIBUS DP module. You connect the EM DP01 to the S7-200 SMART CPU as an expansion module, and you connect the PROFIBUS network to the EM DP01 PROFIBUS DP module through its DP communications port. This port operates at any PROFIBUS baud rate between 9.6 Kbps and 12 Mbps. Refer to the EM DP01 PROFIBUS DP module Technical Specifications for the baud rates supported.

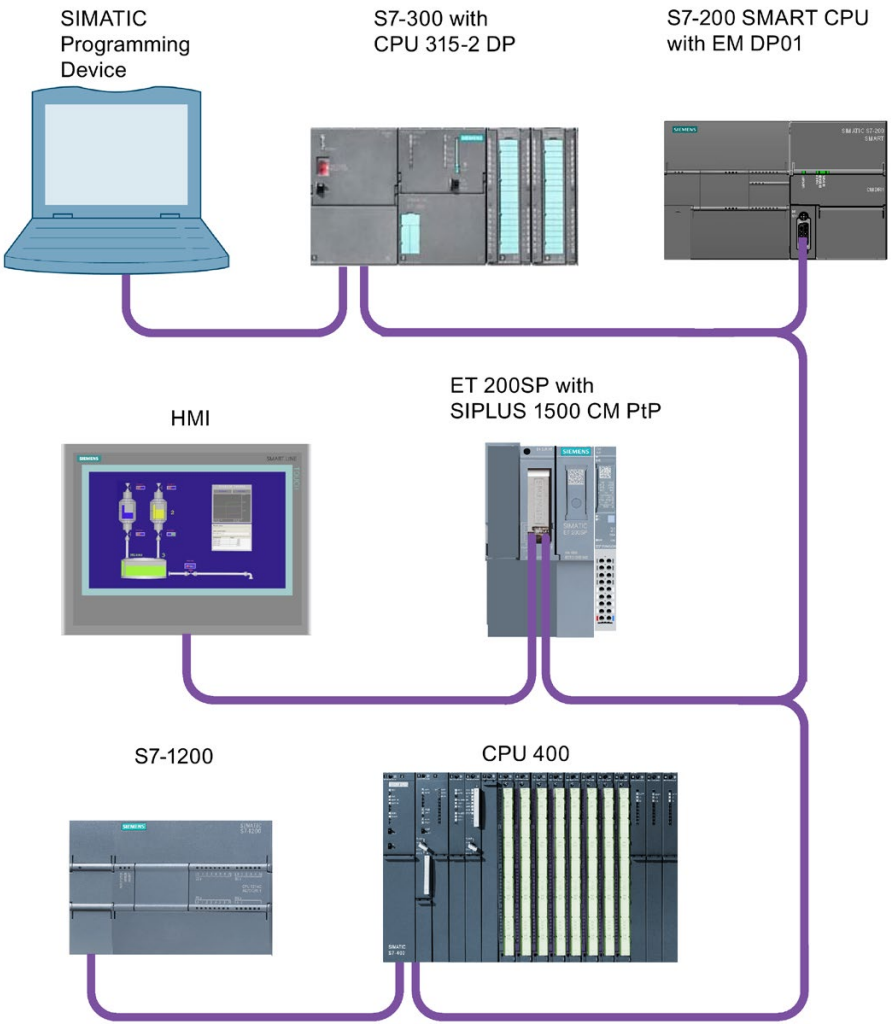
Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

As a PROFIBUS DP device, the EM DP01 accepts several different I/O configurations from the DP master, allowing you to tailor the amount of data transferred to meet the requirements of the application. Unlike many DP devices, the EM DP01 does not transfer only I/O data. The EM DP01 also transfers inputs, counter values, timer values, or any other values that you move to the variable memory in the S7-200 SMART CPU. Likewise, the EM DP01 transfers data from the DP master to the variable memory in the S7-200 SMART CPU. You can then move this data from variable memory to other data areas.

You can attach the DP port of the EM DP01 PROFIBUS DP module to a DP master on the network and still communicate as an MPI device with other master devices such as SIMATIC HMI devices or S7-300 / S7-400 CPUs on the same network. The following figure shows a PROFIBUS network with an S7-200 SMART CPU SR20 and an EM DP01 PROFIBUS DP module:

- The S7-300 with CPU 315-2 is the DP master and has been configured by a SIMATIC programming device with STEP 7 programming software. The S7-315-2 DP can read data from or write data to the EM DP01, from 1 byte up to 244 bytes.
- The S7-200 SMART CPU SR20 is a DP device owned by the CPU 315-2. The ET 200 I/O module is also a DP device owned by the CPU 315-2.
- The S7-400 CPU is attached to the PROFIBUS network and is reading data from the CPU SR20 by means of X_GET instructions in the S7-400 CPU user program. (Other SIMATIC CPUs can use DB1 to access V memory in the S7-200 SMART CPU.)



8.5.1.3 Configuring the EM DP01

Procedure

1. To use the S7-200 SMART EM DP01 PROFIBUS DP module as a DP device, you must set the station address of the DP port to match the address in the configuration of the DP master. The station address is set with the rotary switches on the EM DP01.
2. You must power cycle the S7-200 SMART CPU after you have made a switch change in order for the new DP device address to take effect.

Result

The DP master device exchanges data with each of its DP devices by sending information from its output area to the DP device's output buffer. The DP device responds to the message from the DP master by returning an input buffer which the DP master stores in an input area.

Configuration steps

The S7-200 SMART EM DP01 PROFIBUS DP module can be configured by the DP master to accept output data from the DP master and return input data to the DP master. The output and input data buffers reside in the variable memory (V memory) of the S7-200 SMART CPU. When you configure the DP master, you define the byte location in V memory where the output data buffer starts as part of the parameter assignment information for the EM DP01. You also define the I/O configuration as the amount of output data to be written to the S7-200 SMART CPU and amount of input data to be returned from the S7-200 SMART CPU. The EM DP01 determines the size of the input and output buffers from the I/O configuration. The DP master writes the parameter assignment and I/O configuration information to the EM DP01. The EM DP01 then transfers the V memory address and input and output data lengths to the S7-200 SMART CPU. These values are available in the special memory (SM) of the S7-200 SMART CPU for use in the user program. Refer to the SM status information in "User program considerations" (Page 454) for further details.

8.5.1.4 Data consistency

PROFIBUS supports three types of data consistency:

- Byte: Ensures that bytes are transferred as whole units
- Word: Ensures that word transfers cannot be interrupted by other processes in the CPU.
- Buffer: Ensures that the entire buffer of data is transferred as a single unit, uninterrupted by any other process in the CPU.

The EM DP01 always utilizes buffer consistency in its data handling.

S7-200 SMART CPU and EM DP01 data buffer consistency

The EM DP01 and S7-200 SMART CPU offer buffer consistency for the entire transfer:

- The EM DP01 receives the outputs from the DP master in one message.
- The EM DP01 transfers all outputs to the S7-200 SMART CPU in one message that cannot be interrupted.
- The S7-200 SMART CPU transfers all outputs to the V memory area at one time. The transfer cannot be interrupted by a user interrupt.

The same consistency is true for the inputs to the DP master:

- The S7-200 SMART CPU transfers all inputs from the V memory at one time. The transfer cannot be interrupted by a user interrupt.
- The S7-200 SMART CPU transfers all the inputs to the EM DP01 in one message. This transfer cannot be interrupted.
- The EM DP01 sends the inputs to the DP master in one message.

DP master consistency

Consistency in the DP master CPU is not always buffer consistent. The DP master CPUs do not handle the entire DP message as one indivisible object unless it is very small. The DP master CPUs usually move the PROFIBUS data in smaller pieces. They can either move the data to the I/O area or the user can control the movement with DPRD_DAT (Read consistent data for DP devices) and DPWR_DAT (Write consistent data for DP devices) instructions. Using the DPRD_DAT and DPWR_DAT instructions, you obtain the information for one configuration "slot" at a time. Since we allow two configuration slots, it can take two DPRD_DAT instructions to obtain all of the data. Consistency is only guaranteed for each DPRD_DAT instruction.

8.5.1.5 Supported configurations

The following table lists the configurations that are supported by the S7-200 SMART EM DP01 PROFIBUS DP module:

Table 8- 2 EM DP01 PROFIBUS DP configuration options

Configuration	Inputs to master	Outputs from master	Data consistency
1	Universal module		Buffer consistency [1]
2	4 bytes	4 bytes	
3	8 bytes	8 bytes	
4	16 bytes	16 bytes	
5	32 bytes	32 bytes	
6	64 bytes	64 bytes	
7	122 bytes	122 bytes	
8	128 bytes	128 bytes	

[1] All EM DP01 configurations are buffer consistent.

We can mix and match any two of these configurations in an EM DP01 configuration. Here are two examples:

- A configuration of 32 bytes input and output plus a configuration of 8 bytes input and output yields a total of 40 input bytes and 40 output bytes.
- A configuration of 122 bytes input and output plus a configuration of 122 bytes input and output yields a total of 244 input bytes and 244 output bytes.

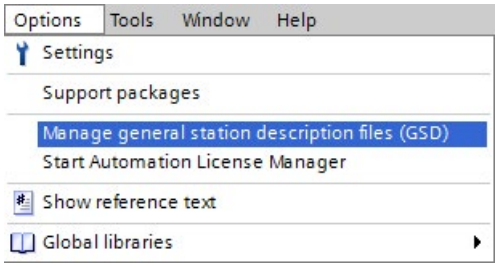
The EM DP01 allows a maximum of 244 input bytes and 244 output bytes. If you use two configurations for the EM DP01, all of the input data is contiguous, and all of the output data is contiguous. Refer to "Example of V memory and I/O address area" (Page 453) for further information.

8.5.1.6 Installing the EM DP01 GSD file

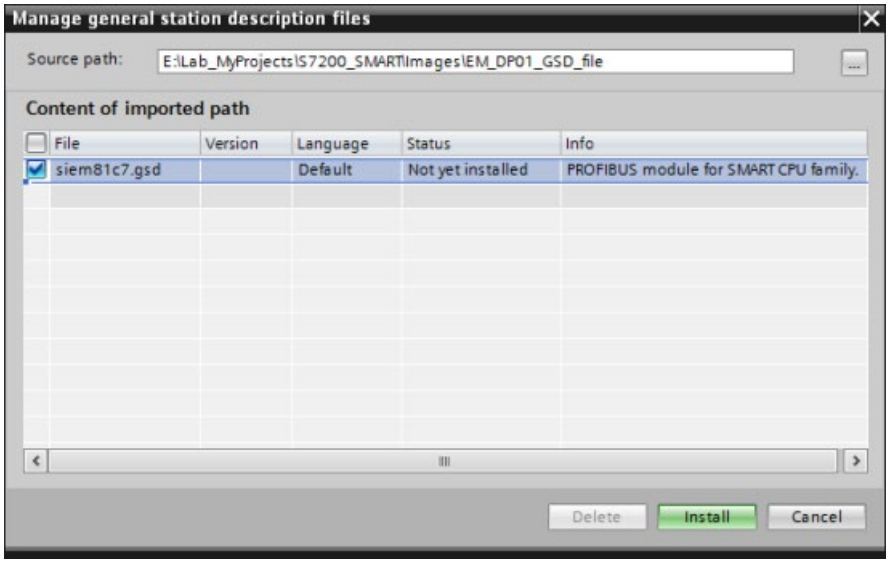
A PROFIBUS GSD file describes the DP device and its capabilities. The programmer uses the GSD file to configure the DP master.

To install the EM DP01 GSD file, follow these steps:

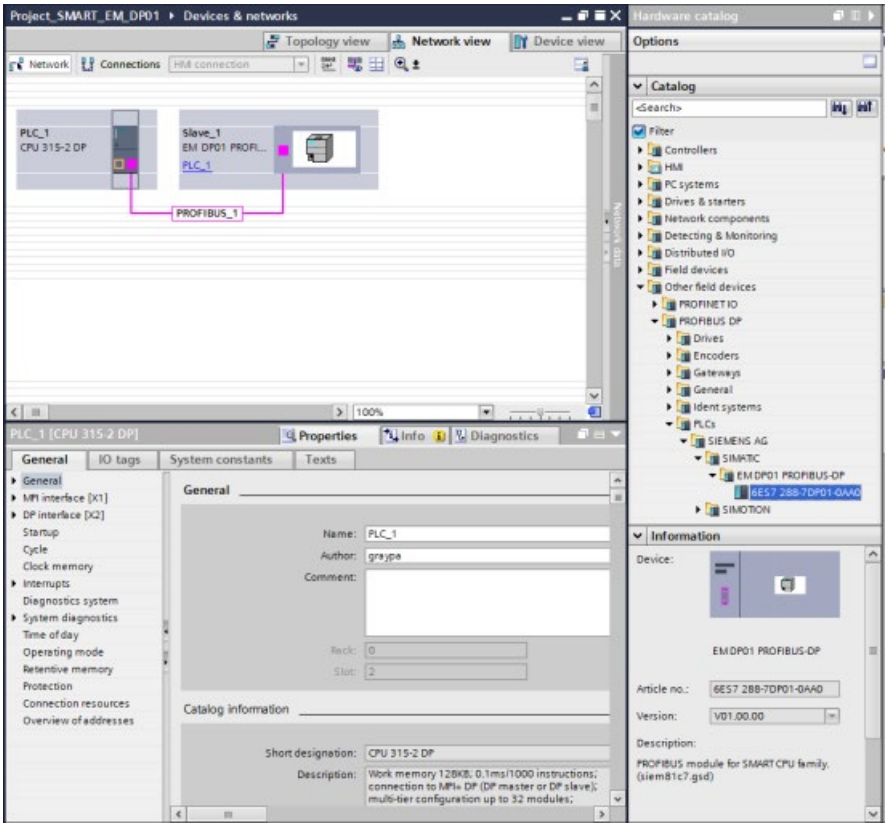
1. Start the TIA Portal software.
2. Create a new project.
3. In the project view, locate the menu bar and select: Options > Manage general station description files (GSD)



4. In the Source path, using the dropdown button, locate the EM DP01 GSD file that you have previously loaded on your computer.
5. Select the check box for the GSD file line.
6. Click the Install button:



7. This action installs the EM DP01 GSD file in the Hardware catalog as shown in the following figure:



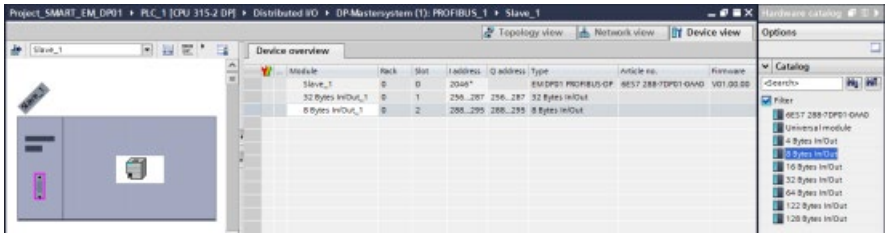
8. Insert a CPU 315-2 DP, for example, as the DP master.
9. Insert the EM DP01 PROFIBUS DP module.
10. Create a PROFIBUS network between the DP master and device as shown in the figure above.

8.5.1.7 Configuring the EM DP01 I/O

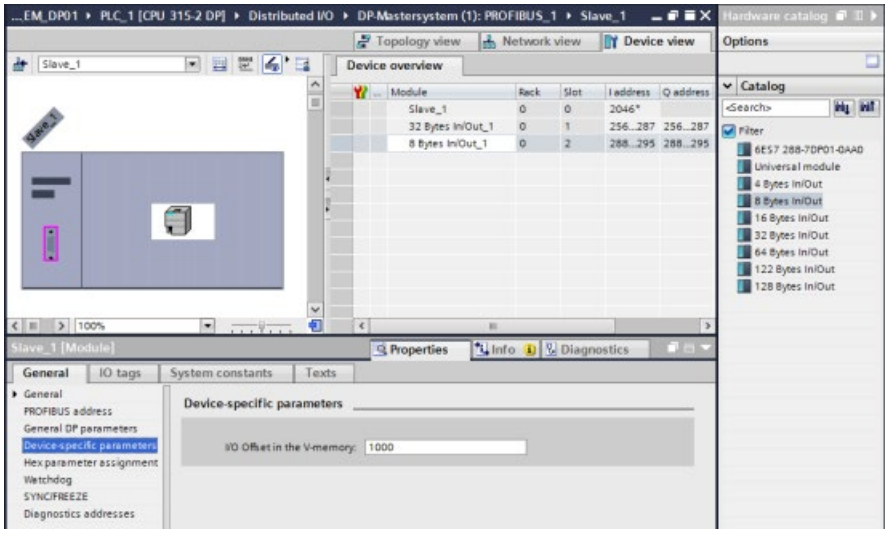
You can configure the EM DP01 I/O by using pre-configured or universal module I/O configuration selections. The EM DP01 configuration allows for two slots so that you can have more than 128 bytes of data transferred between the DP master and the S7-200 SMART CPU. This makes it possible for you to configure the maximum of 244 bytes that PROFIBUS allows. Two possible I/O configuration combinations are shown in the following examples.

32 Bytes In/Out and 8 Bytes In/Out configuration

In this example, slot one contains the "32 Bytes In/Out" pre-configured I/O selection, and slot two contains the "8 Bytes In/Out" pre-configured I/O selection.

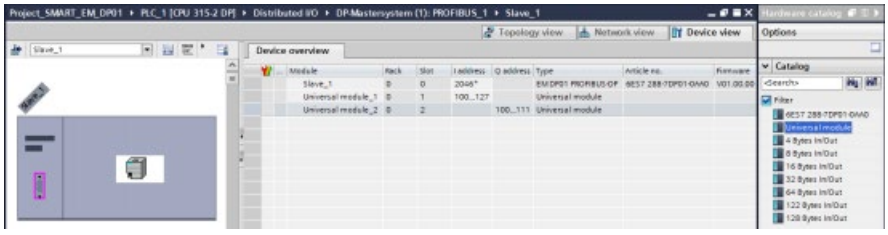


In the "Properties"", "General" tab area navigation, click on "Device-specific parameters" to display the "I/O Offset in the V memory" field. Here, you can assign the starting address of the section of V memory reserved for this operation.



Universal module configuration

In this example, slots one and two contain the "Universal module" I/O selection, and you can configure these two slots with the number of inputs and outputs your application requires (up to a maximum of 244 input bytes and 244 output bytes).



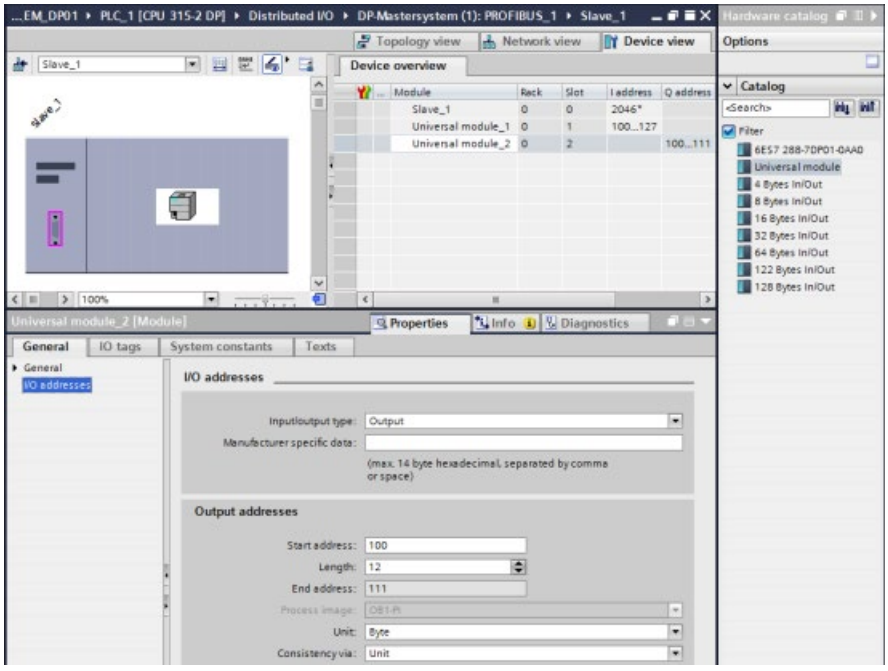
In the "Properties", "General" tab area navigation, click on "I/O addresses" to display the input/output address configuration fields. In the "Input/output type:" field, you must make one of the following selections for the universal module in this slot:

- Input
- Output
- Input/output

Then, you can configure the Input and/or output address ranges for your application.

Note

"Empty slot" is the default selection for the "Input/output type:" field. You must change "Empty slot" to 'Input", "Output", or "Input/output" in order to configure your I/O addresses.

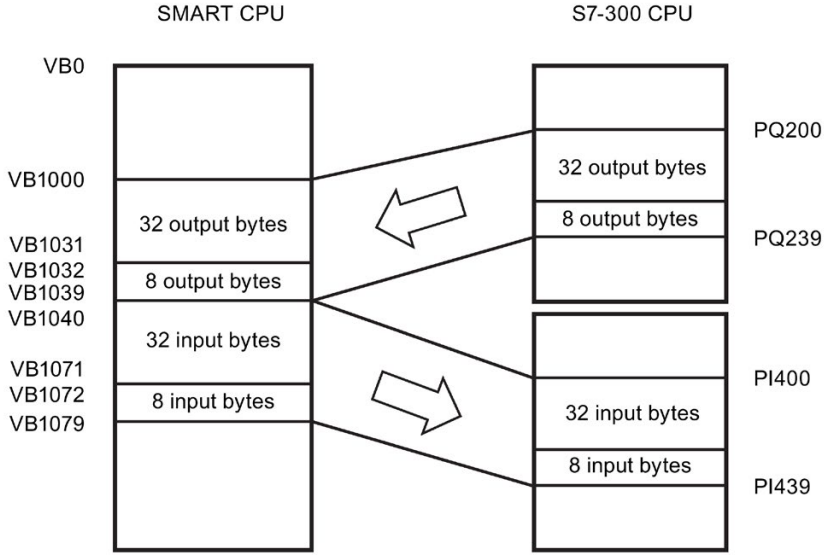


Note

In the examples above, the CPU 315-2 DP is the configured DP master. Depending on the master CPU type, the EM DP01 "Properties" can appear slightly different than those shown here.

8.5.1.8 Example of V memory and I/O address area

The following figure shows an example of the V memory in the S7-200 SMART CPU and the I/O address area of an S7-300 PROFIBUS DP master:



In this example, the DP master has defined an I/O configuration consisting of two slots and a V memory offset of 1000. The example configures the first slot as 32 bytes in and out and the second slot as 8 bytes in and out. The output and input buffers in the S7-200 SMART CPU are both 40 bytes (32 + 8). The output data (from the DP master) buffer starts at V1000; the input data (to the DP master) buffer immediately follows the output buffer and begins at V1040.

All of the output data (all 40 bytes) is treated as one buffer consistent block of data in the EM DP01 and SMART CPU. The output data in the S7-300 is treated with different consistencies depending on whether the user utilizes the I and Q areas or whether they use the DPRD_DAT (Read consistent data for DP devices) and DPWR_DAT (Write consistent data for DP devices) instructions. Even using the DPRD_DAT and DPWR_DAT instructions, the data is only consistent within the 32 byte and 8 bytes blocks. The entire 40 bytes is consistent only if the user manages this by not reading or writing the data in user interrupt blocks.

Note

If you are working with a data unit (consistent data) greater than four bytes, you can use the DPRD_DAT instruction to read the inputs of the DP device and the DPWR_DAT instruction to address the outputs of the DP device. For further information, refer to "Data consistency" and the *System Software for S7-300 and S7-400 System and Standard Functions Reference Manual*.

You can configure the location of the input and output buffers to be anywhere in the V memory of the S7-200 SMART CPU. The default address for the input and output buffers is VB0. The location of the input and output buffers is part of the parameter assignment information that the DP master writes to the S7-200 SMART CPU. You configure the DP master to recognize its DP devices and to write the required parameters and I/O configuration to each of its DP devices.

You configure SIMATIC S7 DP masters using STEP 7 programming software. For detailed information about using this configuration and programming software package, refer to the manuals for these devices. For detailed information about the PROFIBUS network and its components, refer to the *ET 200 Distributed I/O System Manual*.

See also

Data consistency (Page 447)

8.5.1.9 User program considerations

After a DP master successfully configures the EM DP01 PROFIBUS DP module, the EM DP01 and the DP master enter data exchange mode. In data exchange mode, the DP master writes output data to the EM DP01, and the EM DP01 then responds with the most recent S7-200 SMART CPU input data. The EM DP01 continuously updates its inputs from the S7-200 SMART CPU in order to provide the most recent input data to the DP master. The EM DP01 then transfers the output data to the S7-200 SMART CPU. The S7-200 SMART CPU places the output data from the DP master into V memory (the output buffer) starting at the address that the DP master supplied during initialization. The S7-200 SMART CPU takes the input data to the DP master from the V memory locations (the input buffer) immediately following the output data.

The user program in the S7-200 SMART CPU must move the output data from the DP master from the output buffer to the data areas where the program uses it. Likewise, the user program must move the input data to the DP master from the various data areas to the input buffer for transfer to the master.

The S7-200 SMART CPU places the output data from the DP master into V memory immediately prior to the user program portion of the scan. The S7-200 SMART CPU copies the input data (to the DP master) from V memory to the EM DP01 for transfer to the DP master after the user program portion of the scan.

The S7-200 SMART CPU transmits the input data to the DP master on the EM DP01's next data exchange with the DP master.

Status information

There are 50 bytes of special memory (SM) allocated to each expansion module based upon its physical position. The module updates the SM locations corresponding to the modules' relative position to the CPU (with respect to other modules). If it is the first module, it updates SMB1400 through SMB1449. If it is the second module, it updates SMB1450 through SMB1499, and so on. Refer to the table below:

Table 8- 3 Special memory bytes SMB1400 to SMB1699

Special memory bytes SMB1400 to SMB1699					
Intelligent module in slot 0	**Intelligent module in slot 1**	**Intelligent module in slot 2**	**Intelligent module in slot 3**	**Intelligent module in slot 4**	**Intelligent module in slot 5**
SMB1400 to SMB1449	SMB1450 to SMB1499	SMB1500 to SMB1549	SMB1550 to SMB1599	SMB1600 to SMB1649	SMB1650 to SMB1699

These SM locations show default values if DP communications have not been established with a DP master. After a DP master has written parameters and I/O configuration to the EM DP01 PROFIBUS DP module, these SM locations show the configuration set by the DP master. You should check the protocol status byte (for example SMB1424 for slot 0) to be sure that the EM DP01 is currently in data exchange mode with the DP master before using the information in the SM locations shown in the following table or data in the V memory buffer.

Note

You cannot configure the EM DP01 PROFIBUS DP I/O buffer sizes or buffer location by writing to SM memory locations. Only the DP master can configure the EM DP01 PROFIBUS DP module for DP operation.

Table 8- 4 Special memory bytes for the EM DP01 PROFIBUS DP

Intelligent module in slot 0	...	Intelligent module in slot 5	Description	
SMB1400	...	SMB1650	DP device's station address as set by address switches (0 - 99 decimal)	
SMB1401	...	SMB1651	Address of the DP device's master (0 to 126) (displays 255 if no DP master is attached)	
SMW1402	...	SMW1652	V memory address of the output buffer as an offset from VB (for example, 1000 means VB1000).	
SMB1404	...	SMB1654	Number of bytes of output data	
SMB1405	...	SMB1655	Number of bytes of input data	
SMB1406	...	SMB1656	DP standard protocol status byte	
			Num-ber	Description
			0	DP communications not initiated since power on
			1	Configuration/parameterization error detected
			2	Currently in data exchange mode
			3	Dropped out of data exchange mode
SMB1407 to SMB1449	...	SMB1657 to SMB1699	Reserved - cleared on power up	
Note: SM locations are updated each time the DP device accepts configuration / parameterization information. These locations are updated even if a configuration/parameterization error is detected. The locations are cleared on each power up. Note: This information is also available in the STEP 7-Micro/WIN SMART "PLC information" for the EM DP01. Note: The user program can access this information and use it to process the EM DP01 data.				

8.5.1.10 LED status indicators for the EM DP01 PROFIBUS DP

The EM DP01 PROFIBUS DP module has four status LEDs on the front panel to indicate the operational state of the DP port:

- DIAG LED:
  - Dual color (red / green) LED indicates the operating state and fault status of the EM DP01
  - Flashing red: Upon startup, until the EM DP01 is logged in by the CPU, or if there is a fault in the EM DP01
  - Flashing green: While the EM DP01 is waiting on configuration and parameterization from the S7-200 SMART CPU (immediately after login), or during a firmware update
  - ON green: No fault is present and the EM DP01 is configured
- POWER LED:
  - ON green: If user 24 V DC is applied
  - OFF: No user 24 V DC
- DP ERROR LED:
  - Flashing red: If there is an error in the I/O configuration or parameter information that the DP master writes to the EM DP01
  - ON red: If DP communications are interrupted
  - OFF: No error or data exchange has never been established
- DX MODE LED:
  - OFF: After the S7-200 SMART CPU is turned ON as long as DP communications are not attempted, or if DP communications are interrupted
  - ON green: Once DP communications have been successfully initiated (the EM DP01 has entered Data Exchange Mode with the DP master); remains on until the EM DP01 exits Data Exchange Mode

Note

If DP communications are lost, which forces the EM DP01 to exit Data Exchange Mode, the DX MODE LED turns OFF and the DP ERROR LED turns red. This condition persists until the S7-200 SMART CPU is powered off or Data Exchange Mode is resumed.

The following table summarizes the status indications signified by the EM DP01 status LEDs:

Table 8- 5 EM DP01 PROFIBUS DP module status LEDs

Description	POWER LED (Green)	DIAG LED (Dual Red / Green)	DP ERROR LED (Red)	DX MODE LED (Green)
24 V DC user power good	Green			
No 24 V DC user power	Off			
Internal module failure		Red		

Description	POWER LED (Green)	DIAG LED (Dual Red / Green)	DP ERROR LED (Red)	DX MODE LED (Green)
Upon startup, until the EM DP01 is logged in by the CPU, or if there is a fault in the EM DP01		Flashing red		
While the EM DP01 is waiting on configuration and parameterization from the S7-200 SMART CPU, or during a firmware update		Flashing green		
No fault is present; EM DP01 is configured		Green		
No DP error			Off	
DP communications interrupted; Data Exchange Mode stopped			Red	
Parameterization / configuration error (from the DP Master)			Flashing red	
Data Exchange Mode inactive, or DP communications interrupted				Off
Data Exchange Mode active				Green

8.5.1.11 Using HMIs and S7-CPUs with the EM DP01

The EM DP01 PROFIBUS DP module can be used as a communications interface to MPI masters, whether or not it is being used as a PROFIBUS DP device. The EM DP01 can provide a connection from an S7-300/400 to the S7-200 SMART using the X_GET/X_PUT functions of the S7-300/400. HMI devices such as the SMART HMI or the TD 400 can be used to communicate with the S7-200 SMART through the EM DP01.

Some devices allow you to select V memory as the memory area in the S7-200 SMART CPU. If V memory is not an option, you should configure the client (CPU or HMI device) to read and write to DB1 to access the V memory in the S7-200 SMART CPU. For example, a X_GET would need the remote address set to P#DB1.DBX100.0 BYTE 20 to read 20 bytes of V memory starting at VB100.

Note

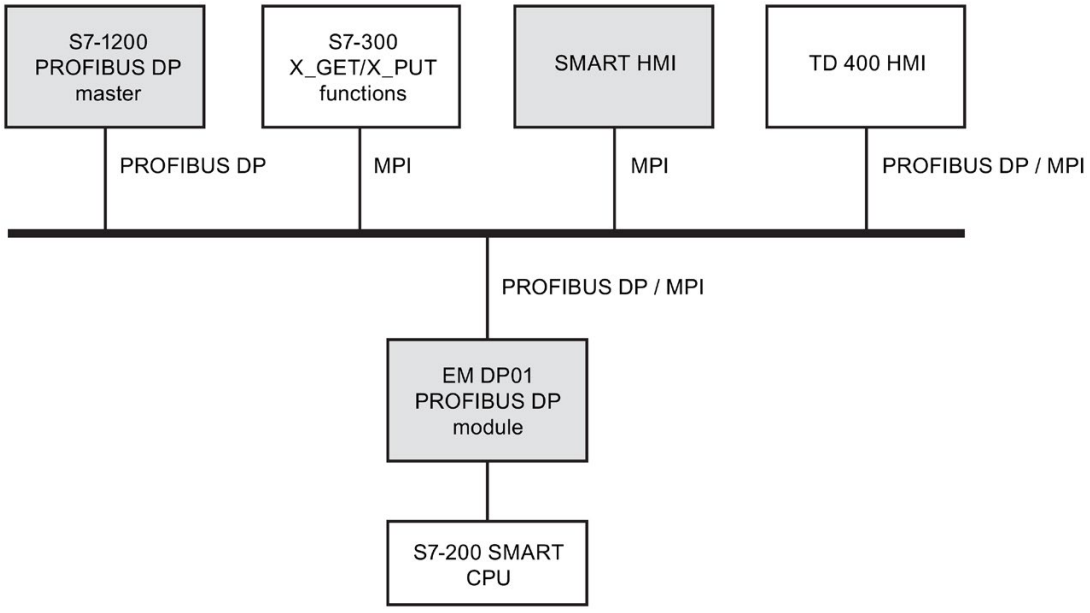
An S7-1200 PROFIBUS DP master cannot access an S7-200 SMART CPU using GET/PUT functions. The S7-1200 DP master can still access the S7-200 SMART CPU using PROFIBUS Data Exchange Mode.

When the EM DP01 PROFIBUS DP module is used for MPI communications, the address parameter of the XGET/XPUT functions must be set to the address of the EM DP01

(address switches). MPI messages sent to the EM DP01 are passed on to the S7-200 SMART CPU.

Messages from MPI and HMI devices are (serviced during)/(subject to) the communication background time in the S7-200 SMART CPU. The communication background time can be increased to provide faster responses to the MPI and HMI requests.

A maximum of six connections (six devices) in addition to the DP master can be connected to the EM DP01. In order for the EM DP01 to communicate with multiple masters, all masters must be operating at the same baud rate. Refer to the figure below for one possible network configuration:



8.5.1.12 Device database file: GSD

Different PROFIBUS devices have different performance characteristics. These characteristics differ with respect to functionality (for example, the number of I/O signals and diagnostic messages) or bus parameters, such as transmission speed and time monitoring. These parameters vary for each device type and vendor and are usually documented in a technical manual. To help you achieve a simple configuration of PROFIBUS, the performance characteristics of a particular device are specified in an electronic data sheet called a device database file, or GSD file. Configuration tools based upon GSD files allow simple integration of devices from different vendors in a single network.

The GSD device database file provides a comprehensive description of the characteristics of a device in a precisely defined format. These GSD files are prepared by the vendor for each type of device and made available to the PROFIBUS user. The GSD file allows the configuration system to read in the characteristics of a PROFIBUS device and use this information when configuring the network.

If your version of software does not include a configuration file for the EM DP01, you can access the latest GSD file (SIEM81C7.GSD) from Siemens Customer Support.

If you are using a non-Siemens master device, refer to the documentation provided by the manufacturer on how to configure the master device by using the GSD file.

GSD file for the EM DP01 PROFIBUS-DP 6ES7288-7DP01-0AA0

Table 8- 6 General parameters

Parameters	Values
#Profibus_DP	
GSD_Revision	= 5
Vendor_Name	= "Siemens"
Model_Name	= "EM DP01 PROFIBUS-DP"
Revision	= "V01.00.00"
Ident_Number	= 0x81C7
Protocol_Ident	= 0
Station_Type	= 0
FMS_supp	= 0
Hardware_Release	= 1
Software_Release	= "V01.00.00"
;	
9.6_supp	= 1
19.2_supp	= 1
45.45_supp	= 1
93.75_supp	= 1
187.5_supp	= 1
500_supp	= 1
1.5M_supp	= 1
3M_supp	= 1
6M_supp	= 1
12M_supp	= 1
;	
MaxTsdr_9.6	= 40
MaxTsdr_19.2	= 40
MaxTsdr_45.45	= 40
MaxTsdr_93.75	= 40
MaxTsdr_187.5	= 40
MaxTsdr_500	= 40
MaxTsdr_1.5M	= 40
MaxTsdr_3M	= 50
MaxTsdr_6M	= 100
MaxTsdr_12M	= 200
;	
Redundancy	= 0
Repeater_Ctrl_Sig	= 2
24V_Pins	= 2
Implementation_Type	= "DPC31"
Bitmap_Device	= "EM_DP01N"

Table 8- 7 Slave-Specification

Parameters	Values
OrderNumber	= "6ES7 288-7DP01-0AA0"
Periphery	= "SIMATIC S7"
Info_Text	= "PROFIBUS module for SMART CPU family."
Slave_Family	= 10@TdF@SIMATIC
Freeze_Mode_supp	= 1
Sync_Mode_supp	= 1
Set_Slave_Add_Supp	= 0
Auto_Baud_supp	= 1
Min_Slave_Intervall	= 1
Fail_Safe	= 0
;	
Modular_Station	= 1
Max_Module	= 2
Modul_Offset	= 0
;	
Max_Input_len	= 244
Max_Output_len	= 244
Max_Data_len	= 488
Max_Diag_Data_Len	= 6

Table 8- 8 DPV1 support

Parameters	Values
DPV1_Slave	= 1
C1_Read_Write_supp	= 1
C2_Read_Write_supp	= 1
C1_Max_Data_Len	= 240
C2_Max_Data_Len	= 240
C1_Response_Timeout	= 100
C2_Response_Timeout	= 100
C1_Read_Write_required	= 0
C2_Read_Write_required	= 0
C2_Max_Count_Channels	= 6
Max_Initiate_PDU_Length	= 64
Ident_Maintenance_supp	= 1
DPV1_Data_Types	= 0
WD_Base_1ms_supp	= 0

Parameters	Values
Check_Cfg_Mode	= 0
Publisher_supp	= 0

Table 8- 9 UserPrmData-Definition

Parameters	Values
ExtUserPrmData	= 1 "I/O Offset in the V-memory"
Unsigned16 0 0-20479	
EndExtUserPrmData	

Table 8- 10 UserPrmData: Length and Preset

Parameters	Values
Max_User_Prm_Data_Len	= 5
Ext_User_Prm_Data_Const (0)	= 0x00,0x00,0x00,0x00,0x00
Ext_User_Prm_Data_Ref (3)	= 1

Table 8- 11 Module Definition List

Parameters	Values
Module 20 EndModule	= " 4 Bytes In/Out" 0xF1
Module 21 EndModule	= " 8 Bytes In/Out" 0xF3
Module 22 EndModule	= " 16 Bytes In/Out" 0xF7
Module 23 EndModule	= " 32 Bytes In/Out" 0xFF
Module 24 EndModule	= " 64 Bytes In/Out" 0xC0, 0xDF, 0xDF
Module 25 EndModule	= "122 Bytes In/Out" 0xC0, 0xFC, 0xFC
Module 26 EndModule	= "128 Bytes In/Out" 0xC0, 0xFF, 0xFF

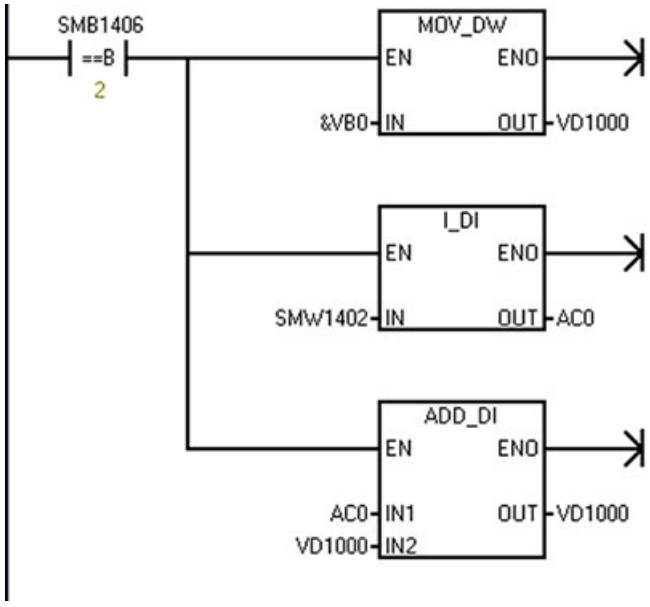
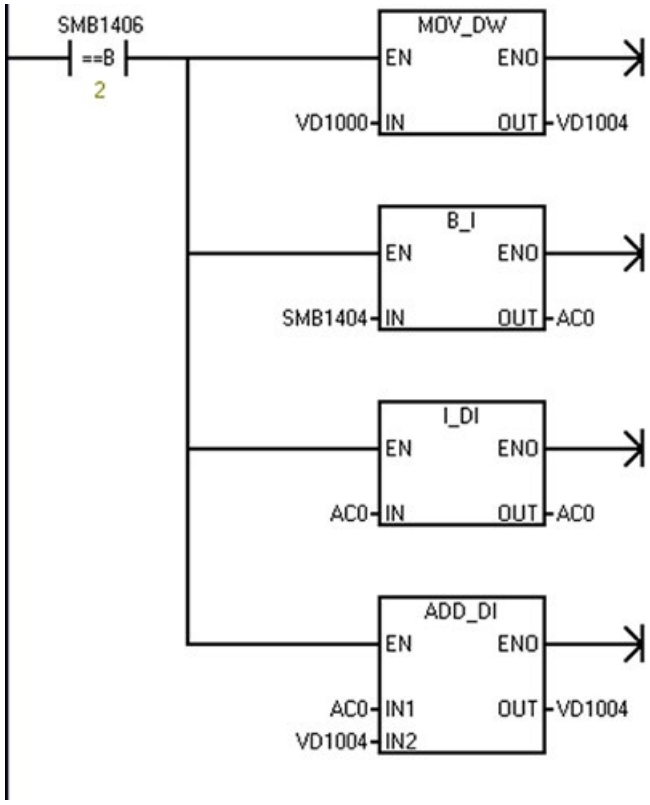
8.5.1.13 PROFIBUS DP communications to a CPU example program

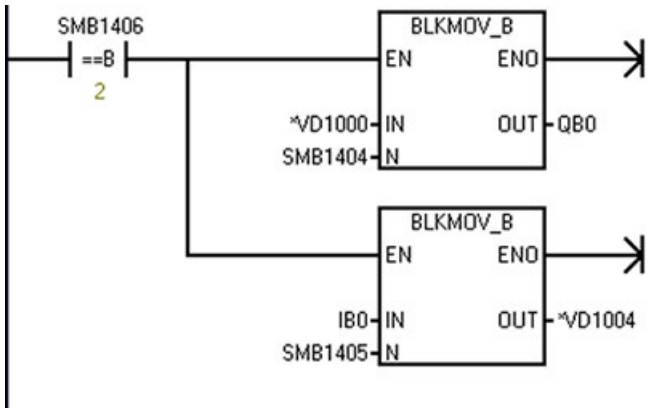
An example program for the PROFIBUS DP module in slot 0 for a CPU that uses the DP port information in SM memory is shown below. The program determines the location of the DP buffers from SMW1402 and the sizes of the buffers from SMB1404 and SMB1405. This information is used to copy the data in the DP output buffer to the process image output register of the CPU. Similarly, the data in the process image input register of the CPU are copied into the V memory input buffer.

In the following example program for a DP module in position 0, the DP configuration data in the SM memory area provides the configuration of the DP device. The program uses the following data:

SMB1406	DP Status
SMB1401	Master Address
SMW1402	V memory offset of outputs
SMB1404	Number of bytes of output data
SMB1405	Number of bytes of input data
VD1000	Output Data Pointer
VD1004	Input Data Pointer

Table 8- 12 Example: Configuring DP communications to an S7-200 SMART CPU

LAD/FBD	Description	STL
Network 1: 	Calculate the Output data pointer. If in data exchange mode: 1. Output buffer is an offset from VB0. 2. Convert V memory offset to double integer. 3. Add to VB0 address to get output data pointer.	**LDB= SMB224, 2** **MOVD &VB0, VD1000** **ITD SMW226, AC0** **+D AC0, VD1000**
Network 2: 	Calculate the Input data pointer. If in data exchange mode: 1. Copy the output data pointer. 2. Get the number of output bytes. 3. Add to output data pointer to get starting input data pointer.	**LDB= SMB224, 2** **MOVD VD1000, VD1004** **BTI SMB228, AC0** **ITD AC0, AC0** **+D AC0, VD1004**

LAD/FBD	Description	STL
Network 3: 	Transfer Master outputs to CPU outputs. Copy CPU inputs to the Master inputs. If in data exchange mode: 1. Copy Master outputs to CPU outputs. 2. Copy CPU inputs to Master inputs.	**LDB= SMB224, 2** **BMB *VD1000, QB0, VB1008** **BMB IB0, *VD1004, VB1009**

8.5.1.14 Reference to the EM DP01 PROFIBUS DP module technical specifications

For further information on the EM DP01 PROFIBUS DP module, refer to the "EM DP01 PROFIBUS DP module" (Page 845) technical specifications.

8.6 RS485

An RS485 network is a differential (multi-point) network and can have up to 126 addressable nodes per network and up to 32 devices per segment. Repeaters are used to segment the network. Repeaters are not addressable nodes; therefore, they are not included in the count of addressable nodes, but are counted in the devices per segment.

RS485 allows for data transfer at a high speed (from 100 m at 12 Mbit/s to 1 km at 187.5 Kbit/s).

RS485 can operate with the PPI protocol and Freeport:

- PPI protocol: Can operate on RS485 or RS232 (half-duplex). Possible connections include:
  - PPI protocol devices
  - RS485 HMI displays
- Freeport: Can operate on RS485 or RS232 (half-duplex). Possible connections include:
  - RS485-compatible devices (for example, a bar code scanner)
  - Devices that have RS485 interfaces (for example, a control system)
  - Third-party devices using Freeport
  - Modems

8.6.1 PPI protocol

Definition

PPI is a master-slave protocol: the master devices send requests to the slave devices, and the slave devices respond. See the following figure. Slave devices do not initiate messages, but wait until a master sends them a request or polls them for a response.

Masters communicate to slaves by means of a shared connection which is managed by the PPI protocol. PPI does not limit the number of masters that can communicate with any one slave; however, you cannot install more than 32 masters on the network.

PPI protocol and S7-200 SMART CPUs

PPI Advanced allows network devices to establish a logical connection between the devices. With PPI Advanced, there are a limited number of connections supplied by each device. See the following table for the number of connections supported by the S7-200 SMART CPU.

All S7-200 SMART CPUs support both PPI and PPI Advanced protocols.

Table 8- 13 Number of connections for the S7-200 SMART CPU

Module	Baud rate	Connections
RS485 port	9.6 Kbps, 19.2 Kbps, or 187.5 Kbps	5
RS485/RS232 signal board	9.6 Kbps, 19.2 Kbps, or 187.5 Kbps	4

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

8.6.2 Baud rate and network address

8.6.2.1 Definition of baud rate and network address

Baud rate

The speed that data is transmitted across the network is the baud rate, which is typically measured in units of kilobits (Kbps) or megabits (Mbps). The baud rate measures how much data can be transmitted within a given amount of time. For example, a baud rate of 19.2 Kbps describes a transmission rate of 19,200 bits per second.

Every device that communicates over a given network must be configured to transmit data at the same baud rate. Therefore, the fastest baud rate for the network is determined by the slowest device connected to the network.

The following table lists the baud rates supported by the S7-200 SMART CPU.

Table 8- 14 Baud rate supported by the S7-200 SMART CPU

Network	Baud rate
PPI protocol	9.6 Kbps, 19.2 Kbps, and 187.5 Kbps only
Freeport Mode	1.2 Kbps to 115.2 Kbps

Network address

The network address is a unique number that you assign to each device on the network. The unique network address ensures that the data is transferred to or retrieved from the correct device. The S7-200 SMART CPU supports network addresses from 0 to 126. The following table lists the default (factory) settings for the S7-200 SMART devices.

Table 8- 15 Default addresses for S7-200 SMART devices

S7-200 SMART device	Default address
STEP 7-Micro/WIN SMART	0
HMI	1
S7-200 SMART CPU	2

8.6.2.2 Setting the baud rate and network address for the S7-200 SMART CPU

Introduction

To communicate over the RS485 network with STEP 7-Micro/WIN SMART or SIMATIC HMIs (Page 384), you must configure the RS485 network address and baud rate for the S7-200 SMART CPU.

The RS485 port network address must be unique for all devices on the RS485 network, and the RS485 port baud rate must be the same as the other devices on the RS485 network.

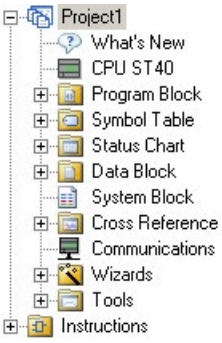
The default RS485 port network address is 2, and the default RS485 port baud rate for each CPU port is 9.6 Kbps.

The system block of the CPU stores the RS485 port network address and baud rate. After you select the parameters for the CPU, you must download the system block to the S7-200 SMART CPU.

Procedure

To access the "System Block" dialog, click one of the following:

System block button in the navigation bar

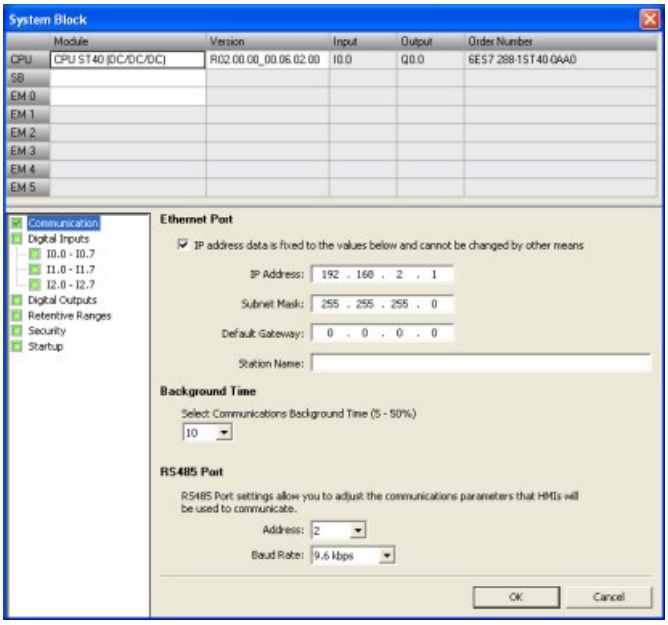


System block in the project tree

System block in the Component drop-down list in the Windows area of the View menu ribbon strip

After you select the "System Block" dialog, you perform the following steps:

1. Select the network address and baud rate for the RS485 port.
2. Download the system block to the CPU.



Note

Freeport protocol baud rates are set using SM memory.

8.6.3 Sample RS485 network configurations

8.6.3.1 Single-master PPI networks

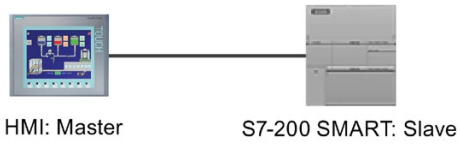
Introduction

The following network configurations are possible using only S7-200 SMART devices:

- Single-master PPI networks
- Multi-master and multi-slave PPI networks
- Complex PPI networks

Single-master PPI networks

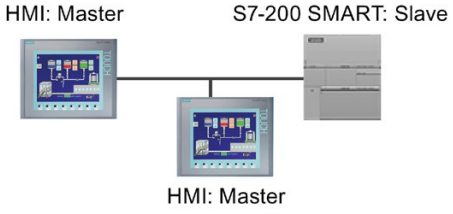
In the sample network in the figure below, a human-machine interface (HMI) device (for example, a TD400C, TP, or KP) is the network master:



In the sample network, the CPU is a slave that responds to requests from the master.

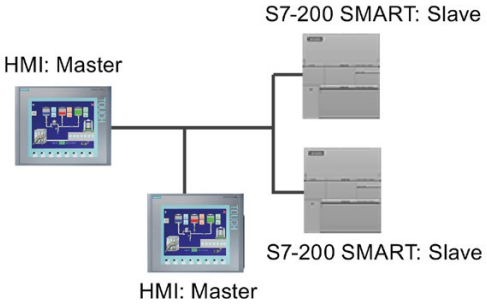
8.6.3.2 Multi-master and multi-slave PPI networks

The following figure shows a sample network of multiple masters with one slave. The HMI devices share the network.



The HMI devices are masters and must have separate network addresses. The S7-200 SMART CPU is a slave.

The following figure shows a PPI network with multiple masters communicating with multiple slaves. In this example, the HMI can request data from any CPU slave.



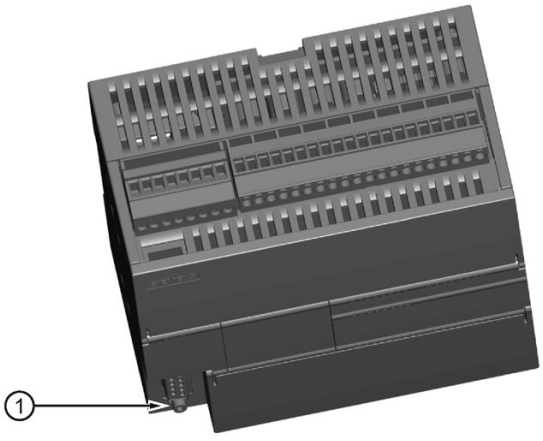
All devices (masters and slaves) have different network addresses. The S7-200 SMART CPUs are slaves.

8.6.4 Assigning RS485 addresses

8.6.4.1 Configuring or changing an RS485 address for a CPU or device in your project

You must enter the following information for each S7-200 SMART CPU that is attached to your RS485 network:

- RS485 address: Each CPU or device must have an RS485 address. The CPU or device uses this address to deliver data over the network.
- Baud rate: The speed that data is transmitted across the network is the baud rate, which is typically measured in units of Kbps or Mbps. The baud rate measures how much data can be transmitted within a given amount of time (for example, a transmission rate of 19.2 Kbps).



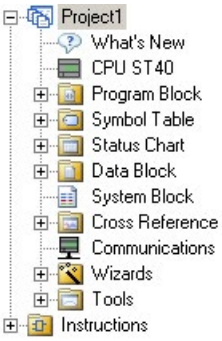
① RS485 port

You must configure or change RS485 network information for the onboard RS485 port of a CPU or device in the "System Block" dialog and download the configuration to the CPU.

Configuring RS485 network information in the System Block dialog

RS485 network information configuration or changes done in the system block are part of the project and do not become active until you download your project to the CPU.

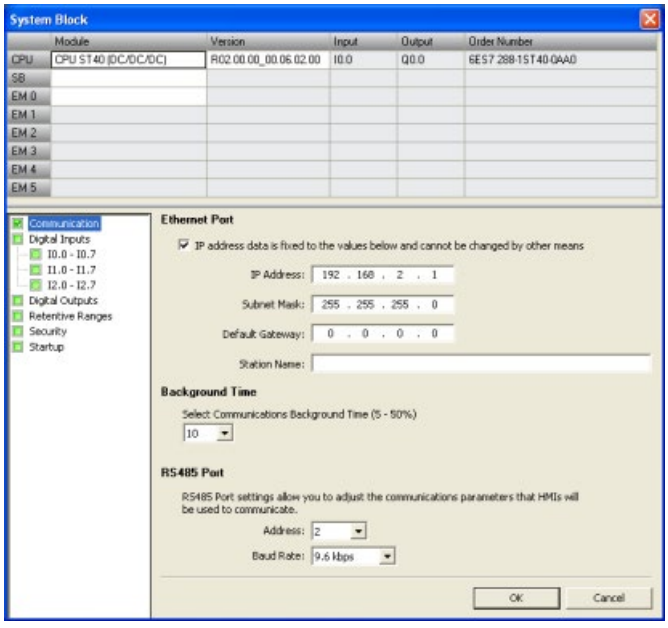
To access this dialog, perform one of the following:



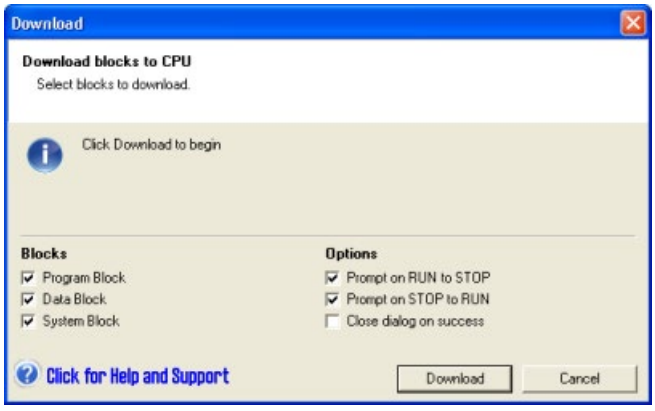
- In the navigation bar, click the "System Block" button.
- In the Project tree, select the "System Block" node, then press Enter; or double-click the "System Block" node.

Enter or change the following access information:

- RS485 port address
- RS485 port baud rate



After completing the RS485 network configuration, download the project to the CPU.



All CPUs and devices that have valid RS485 port addresses are displayed in the "Communications" dialog.

You can access CPUs in one of two ways:

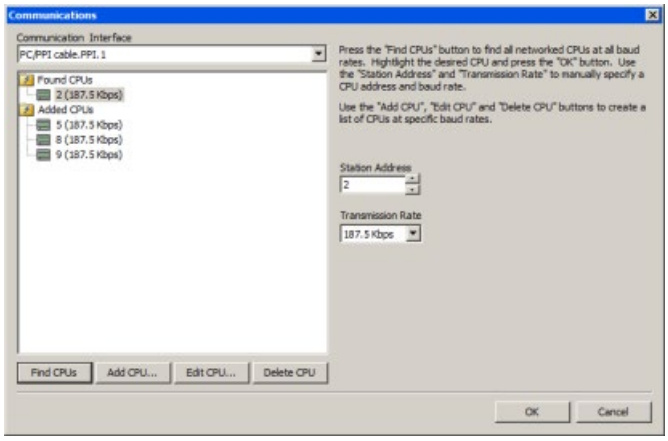
- "Found CPUs": CPUs located on the RS485 network
- "Added CPUs": CPUs on the RS485 network (for example, enter the RS485 network address of a CPU directly that is on the RS485 network)

For "Found CPUs" (CPUs located on your local network), use the "Communications" dialog to connect with your CPU:

- Click the "Communication Interface" dropdown list, and select "PC/PPI cable.PPI.1" for your RS485 network.
- Click the "Find CPUs" button to display all operational CPUs ("Found CPUs") on the RS485 network. All CPUs default their RS485 network settings to address 2 and 9.6 Kbps.
- Highlight a CPU, and then click "OK".

Note

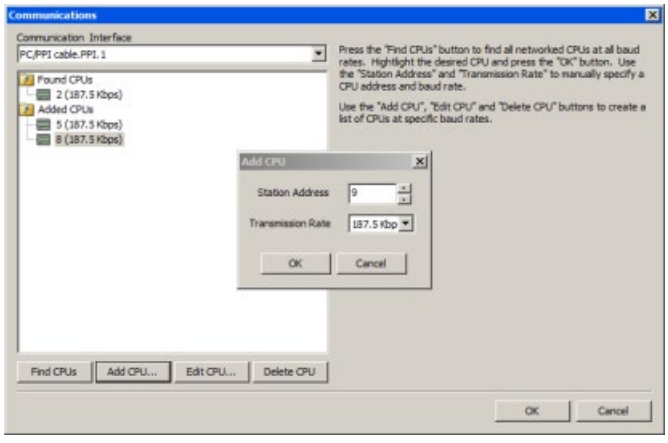
You can open multiple copies of STEP 7-Micro/WIN SMART on a computer. Be aware that when you open a second copy of STEP 7-Micro/WIN SMART or use the "Find CPUs" button in either copy, the communication connection to the CPU in your first/other copy of STEP 7-Micro/WIN SMART might be disconnected.



For "Added CPUs" (CPUs on the RS485 network), use the "Communications" dialog to connect with your CPU:

- Click the "Communication Interface" dropdown list, and select "PC/PPI cable.PPI.1" for your RS485 network.
- Click the "Add CPU" button, and enter the following access information for a CPU that you wish to access directly on the RS485 network:
  - RS485 network address
  - RS485 network baud rate

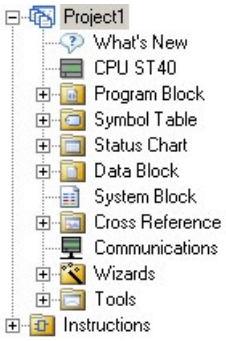
  You can add multiple CPUs on the RS485 network. As always, STEP 7-Micro/WIN SMART communicates with one CPU at a time. All CPUs default their RS485 network settings to address 2 and 9.6 Kbps.
- Highlight a CPU, and then click "OK".



8.6.4.2 Searching for CPUs and devices on your RS485 network

You can search for and identify the S7-200 SMART CPUs that are attached to your RS485 network in the "Communications" dialog. To access this dialog, click one of the following:

"Communications" button in the navigation bar



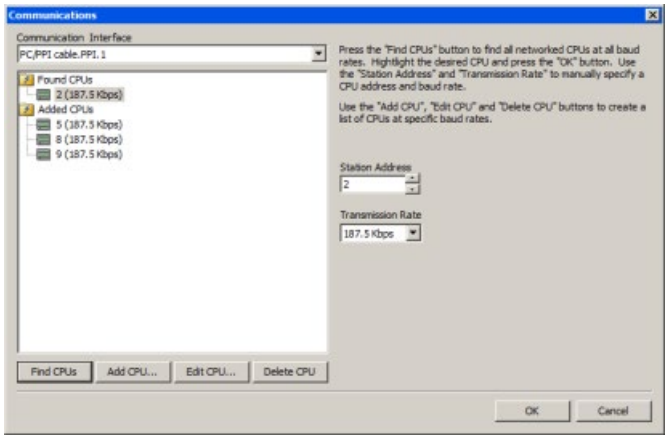
"Communications" in the project tree

"Communications" from the "Component" drop-down list in the Windows area of the "View" menu ribbon strip

The "Communications" dialog autodetects all connected and available S7-200 SMART CPUs on a given RS485 network by creating a lifelist. (See the figure below.) After selecting a CPU, the dialog lists the following detailed information about the CPU:

- RS485 port address
- RS485 network baud rate

The STEP 7-Micro/WIN SMART project includes all added CPUs. However, opening a STEP 7-Micro/WIN SMART project does not automatically select an RS485 network address or establish a connection to a CPU. Every time you create a new or open an existing STEP 7-Micro/WIN SMART project, you must go to the Communications dialog to establish a connection to a CPU. The Communications dialog will show the last selected CPU.



Note

When you press the "Find CPUs" button, the EM DP01 PROFIBUS DP module autobauds and displays in the "Found CPUs " folder at 9.6 Kbps. If you want to communicate through a DP01 modules at a higher baud rate, you must press the "Add CPU" button and add the EM DP01 module using the module's network address and specify a baud rate such as 187.5 Kbps.

You must power cycle the DP01 module to connect at the new baud rate.

8.6.5 Building your network

8.6.5.1 General guidelines

Always install appropriate surge suppression devices for any wiring that could be subject to lightning surges.

Avoid placing low-voltage signal wires and communication cables in the same wire tray with AC wires and high-energy, rapidly switched DC wires. Always route wires in pairs, with the neutral or common wire paired with the hot or signal-carrying wire.

The communication port of the S7-200 SMART CPU is not isolated. Consider using an RS485 repeater to provide isolation for your network.

NOTICE
Avoiding unwanted current flow Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable. These unwanted currents can cause communications errors or can damage equipment. Be sure all equipment that you are about to connect with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows.

8.6.5.2 Determining the distances, transmission rates, and cable lengths for your network

As shown in the following table, the maximum length of a network segment is determined by two factors: isolation (using an RS485 repeater) and baud rate.

Isolation is required when you connect devices at different ground potentials. Different ground potentials can exist when grounds are physically separated by a long distance. Even over short distances, load currents of heavy machinery can cause a difference in ground potential.

Table 8- 16 Maximum length for a network cable

Baud rate	Non-isolated CPU port[1]	CPU port with repeater
9.6 Kbps to 187.5 Kbps	50 m	1,000 m
500 Kbps	Not supported	400 m
1 Mbps to 1.5 Mbps	Not supported	200 m
3 Mbps to 12 Mbps	Not supported	100 m
[1] The maximum distance allowed without using an isolator or repeater is 50 m. You measure this distance from the first node to the last node in the segment.		

8.6.5.3 Repeaters on the network

An RS485 repeater provides bias and termination for the network segment. You can use a repeater for the following purposes:

- **To increase the length of a network**

  Adding a repeater to your network allows you to extend the network another 50 m. If you connect two repeaters with no other nodes in between, (as shown in the figure below), you can extend the network to the maximum cable length for the baud rate. You can use up to 9 repeaters in series on a network, but the total length of the network must not exceed 9600 m.

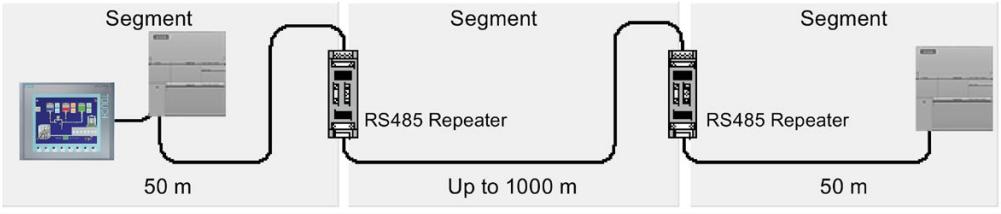
- **To add devices to a network**

  Each segment can have a maximum of 32 devices connected up to 50 m at 9.6 Kbps. Using a repeater allows you to add another segment (32 devices) to the network.

- **To electrically isolate different network segments**

  Isolating the network improves the quality of the transmission by separating the network segments which may be at different ground potentials.

A repeater on your network counts as one of the nodes on a segment, even though it is not assigned a network address. The following is a sample network with repeaters.



8.6.5.4 Specifications for RS485 cable

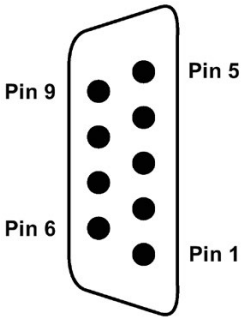
S7-200 SMART CPU networks use the RS485 standard on twisted pair cables. The following table lists the specifications for the network cable. You can connect up to 32 devices on a network segment.

Specifications	Description
Cable type	Shielded, twisted pair
Loop resistance	≤115 Ω/km
Effective capacitance	30 pF/m
Nominal impedance	Approximately 135 Ω to 160 Ω (frequency = 3MHz to 20 MHz)
Attenuation	0.9 dB/100 m (frequency=200 kHz)
Cross-sectional core area	0.3 mm^2 to 0.5 mm^2
Cable diameter	8 mm +0.5 mm

8.6.5.5 Connector pin assignments

The RS485 communication port on the S7-200 SMART CPUs is RS485-compatible on a nine-pin subminiature D connector, in accordance with the PROFIBUS standard as defined in the European Standard EN 50170. The following table shows the connector that provides the physical connection for the communication port and describes the communication port pin assignments.

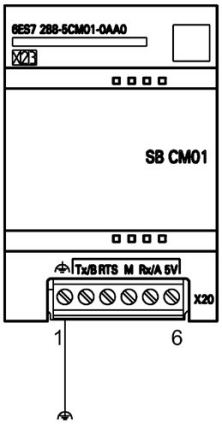
Table 8- 17 Pin assignments for the S7-200 SMART CPU integrated RS485 port (Port 0)

Pin number	Connector	Signal	Integrated RS485 port (Port 0)
1		Shield	Chassis ground
2		24 V Return	Logic common
3		RS485 Signal B	RS485 Signal B
4		Request-to-Send	RTS (TTL)
5		5 V Return	Logic common
6		+5 V	+5 V output, 100 Ω series resistor
7		+24 V	+24 V output
8		RS485 Signal A	RS485 Signal A
9		Not applicable	Programmer detection (input) [1]
Connector shell		Shield	Chassis ground

[1] The CPU utilizes pin 9 of the RS485 connector to detect when a USB-PPI cable is connected. The check for USB-PPI cables is only done on the CRs models. The ST and SR models ignore the state of pin 9. Ensure that any cables used for Freeport do not connect to pin 9 for the CRs models.

The CM01 signal board is RS485-compatible. The following table shows the connector that provides the physical connection for the signal board and describes the pin assignments.

Table 8- 18 Pin assignments for the S7-200 SMART CM01 Signal Board (SB) port (Port 1)

Pin number	Connector	Signal	CM01 Signal Board (SB) port (Port 1)
1		Ground	Chassis ground
2		Tx/B	RS232-Tx/RS485-B
3		Request-to-Send	RTS (TTL)
4		M-ground	Logic common
5		Rx/A	RS232-Rx/RS485-A
6		+5 V DC	+5 V, 100 Ω series resistor

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support the use of expansion modules or signal boards.

8.6.5.6 Biasing and terminating the network cable

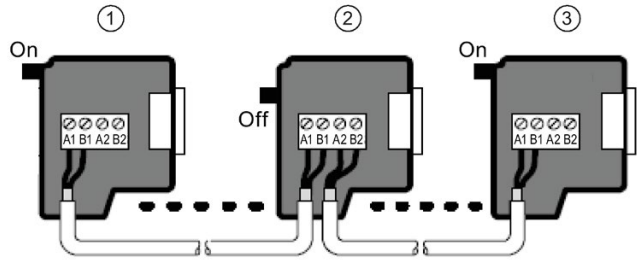
Siemens provides two types of network connectors that you can use to easily connect multiple devices to a network:

- Standard network connector
- Connector that includes a port which allows you to connect an HMI device to the network without disturbing any existing network connections

The programming port connector passes all signals (including the power pins) from the S7-200 SMART CPU through to the programming port, which is especially useful for connecting devices that draw power from the S7-200 SMART CPU (such as a TD 400C).

Both connectors have two sets of terminal screws to allow you to attach the incoming and outgoing network cables. Both connectors also have switches to bias and terminate the network selectively. The following shows typical biasing and termination for the cable connectors.

Table 8- 19 Biasing and termination for cable connectors

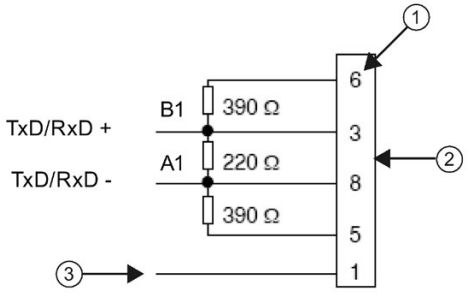
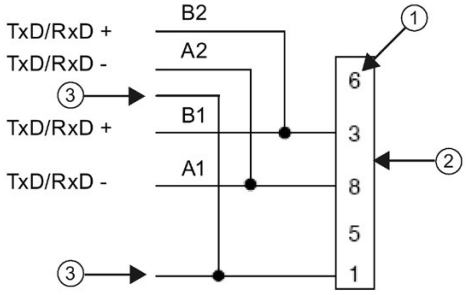
Cable must be terminated and biased at both ends. Bare shielding: Approximately 12 mm (1/2 in) must contact the metal guides of all locations.


① Switch position = On: Terminated and biased

② Switch position = Off: No termination or bias

③ Switch position = On: Terminated and biased

Table 8- 20 Termination and bias switch positions

Switch position = On: Termination and biased	Switch position = Off: No termination or bias
	

① Pin number

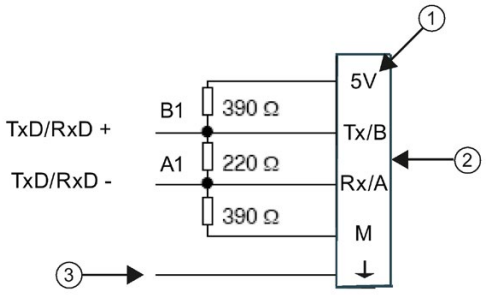
② Network connector

③ Cable shield

8.6.5.7 Biasing and terminating the CM01 signal board

You can use the CM01 signal board to easily connect multiple devices to a network.

The signal board passes all signals (including the power pins) from the S7-200 SMART CPU through to the programming port, which is especially useful for connecting devices that draw power from the S7-200 SMART CPU (such as a TD 400C).



① Terminal name
② Terminal block
③ Cable shield

8.6.5.8 Using HMI devices on your RS485 network

Introduction

The S7-200 SMART CPU supports many types of RS485 HMI devices from Siemens and also from other manufacturers. While some of these HMI devices (such as the TD400C) do not allow you to select the communication protocol used by the device, other devices (such as the KP and TP product lines) allow you to select the communication protocol for that device.

Guidelines

If your HMI device allows you to select the communication protocol, consider the following guideline. For an HMI device connected to the communication port of the CPU, with no other devices on the network, select the PPI protocol for the HMI device.

For more information about how to configure the HMI device, refer to the specific manual for your device (see the following table). These manuals are included in the STEP 7-Micro/WIN SMART documentation CD.

Table 8- 21 RS485 HMI devices supported by the S7-200 SMART CPU

HMI	Configuration software
TD400C	Text Display wizard (part of STEP 7-Micro/WIN SMART)
KTP600 DP	WinCC flexible
KTP1000 DP	WinCC flexible

8.6.6 Freeport mode

8.6.6.1 Creating user-defined protocols with Freeport mode

Introduction

Freeport mode allows your program to control the communication port of the S7-200 SMART CPU. You can use Freeport mode to implement user-defined communication protocols to communicate with many types of intelligent devices. Freeport mode supports both ASCII and binary protocols.

Using Freeport mode

To enable Freeport mode, you use special memory bytes SMB30 (for the Integrated RS485 port (Port 0)) and, if your CPU model supports it, SMB130 (for the CM01 Signal Board (SB) port (Port 1)). Your program uses the following to control the operation of the communication port:

- Transmit instruction (XMT) and the transmit interrupt:

  The Transmit instruction allows the S7-200 SMART CPU to transmit up to 255 characters from the COM port. The transmit interrupt notifies your program in the CPU when the transmission has been completed.

- Receive character interrupt:

  The receive character interrupt notifies the user program that a character has been received on the COM port. Your program can then act on that character, based on the protocol being implemented.

- Receive instruction (RCV):

  The Receive instruction receives the entire message from the COM port and then generates an interrupt for your program when the message has been completely received. You use the SM memory of the CPU to configure the Receive instruction for starting and stopping the receiving of messages, based on defined conditions. The Receive instruction allows your program to start or stop a message based on specific characters or time intervals. Most protocols can be implemented with the Receive instruction, instead of using the more cumbersome receive-character interrupt method.

Freeport mode is active only when the CPU is in RUN mode. Setting the CPU to STOP mode halts all Freeport communication, and the communication port then reverts to the PPI protocol with the settings which were configured in the system block of the CPU.

Note

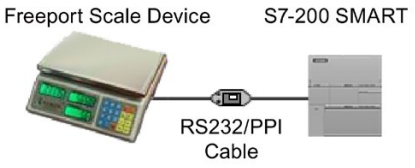
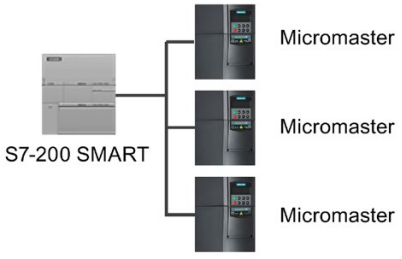
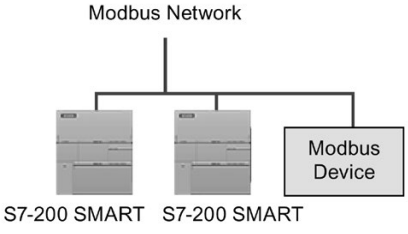
Since the compact CRs models (CR20s, CR30s, CR40s and CR60s) have no Ethernet port, the RS485 port is the programming port. This creates a conflict if the user program is using the RS485 port for Freeport. While the user program is using the RS485 port for Freeport, STEP 7-Micro/WIN SMART cannot communicate to the the CPU.

Attaching a USB-PPI cable to the CPU's RS485 port forces the CPU to exit Freeport mode and enable PPI mode. This allows STEP 7-Micro/WIN SMART to regain control of the the CPU.

If you have attached a USB-PPI cable to the CPU's RS485 port, the CPU cannot enable Freeport. The CPU will not automatically restart Freeport when you remove the USB-PPI cable.

The CPU utilizes pin 9 of the RS485 connector to detect when you connect a USB-PPI cable. Only the CRs models perform the check for USB-PPI cables. The ST and SR models ignore the state of pin 9. Ensure that any cables used for Freeport do not connect to pin 9 for the CRs models.

Table 8- 22 Using Freeport mode

Network configuration		Description
Using Freeport over an RS232 connection		Example: Using an S7-200 SMART CPU with an electronic scale that has an RS232 port. • Connect the two devices using one of the following methods: – RS232/PPI Multi-Master cable connects the RS232 port on the scale to the RS485 port on the CPU. (Set the cable to PPI/Freeport mode, switch 5 = 0.) – Using the CM01 signal board (SB) (S CPUs only) which supports RS232 and RS485, you can connect the RS232 device directly to the CPU SB RS232 without using the RS232/PPI cable. • CPU uses Freeport to communicate with the scale. • Baud rate can be from 1.200 Kbps to 115.2 Kbps. • User program defines the protocol.
Using USS protocol		Example: Using an S7-200 SMART CPU with SIMODRIVE MicroMaster drives. • STEP 7-Micro/WIN SMART provides a USS library. • The CPU is a master, and the drives are slaves.
Creating a user program that emulates a slave device on another network		Example: Connecting S7-200 SMART CPUs to a Modbus network. • User program in the CPU emulates a Modbus slave. • STEP 7-Micro/WIN SMART provides a Modbus library.

8.6.6.2 Using the RS232/PPI Multi-Master cable and Freeport mode with RS232 devices

Purpose

You can use the RS232/PPI Multi-Master cable and the Freeport communication functions to connect the S7-200 SMART CPU to many devices that are compatible with the RS232 standard. The cable must be set to PPI/Freeport mode (switch 5 = 0) for Freeport operation. Switch 6 selects either Local mode (DCE) (switch 6 = 0), or Remote mode (DTE) (switch 6 = 1). In CRs models only, set switch 7 = 1 to allow Freeport mode.

The RS232/PPI Multi-Master cable is in Transmit mode when data is transmitted from the RS232 port to the RS485 port. The cable is in Receive mode when it is idle or is transmitting data from the RS485 port to the RS232 port. The cable changes from Receive to Transmit mode immediately when it detects characters on the RS232 transmit line.

The CM01 signal board (SB) (S CPUs only) supports both RS232 half-duplex and RS485. With the CM01 signal board, you can connect an RS232 device directly to the CPU SB RS232 port without using a RS232/PPI cable.

Baud Rates and turnaround time

The RS232/PPI Multi-Master cable supports baud rates between 1.2 Kbps and 115.2 Kbps. Use the DIP switches on the housing of the RS232/PPI Multi-Master cable to configure the cable for the correct baud rate. The following table shows the baud rates (bits per second) and switch positions.

Table 8- 23 Turnaround time and settings

Baud rate	Turnaround time	Settings (1 = Up)
115200	0.15 ms	110
57600	0.3 ms	111
38400	0.5 ms	000
19200	1.0 ms	001
9600	2.0 ms	010
4800	4.0 ms	011
2400	7.0 ms	100
1200	14.0 ms	101

The cable switches back to Receive mode when the RS232 transmit line is in the idle state for a period of time defined as the turnaround time of the cable. The baud rate selection of the cable determines the turnaround time, as shown in the table.

If you use the RS232/PPI Multi-Master cable in a system which uses Freeport communications, the program in the S7-200 SMART CPU must comprehend the turnaround time for the following situations:

- The CPU responds to messages transmitted by the RS232 device.

  After the CPU receives a request message from the RS232 device, the CPU must delay the transmission of a response message for a period of time greater than or equal to the turnaround time of the cable.

- The RS232 device responds to messages transmitted from the CPU.

  After the CPU receives a response message from the RS232 device, the CPU must delay the transmission of the next request message for a period of time greater than or equal to the turnaround time of the cable.

In both situations, the delay allows the RS232/PPI Multi-Master cable sufficient time to switch from Transmit mode to Receive mode so that data can be transmitted from the RS485 port to the RS232 port.

8.7 RS232

An RS232 network is a point-to-point connection between two devices. RS232 allows for data transfer at relatively slow speeds (up to 115.2 Kbps) and short distances (up to 50 feet).

Possible RS232 connections include the following:

- Freeport
- Modems
- RS232-compatible devices (for example, a bar code scanner)
- Devices that have RS232 interfaces (for example, a control system)
- RS232 displays

Libraries

9

9.1 Library types (Siemens and user-defined)

Library types

Siemens provides two types of libraries with the installation of STEP 7-Micro/WIN SMART:

- Siemens-supplied (Modbus RTU (Page 492), Modbus TCP (Page 511), Open User Communication (Page 529),PN Read Write Record library (Page 571), SINAMICS Library (Page 591) and USS protocol (Page 574))
- User-defined (Page 644) (libraries that you create from project POUs or obtain from other sources)

Note

You must start STEP 7-Micro/WIN SMART with a "Run as administrator" command to create a user-defined library.

Note

You cannot give a user-defined library the same name as a Siemens-supplied library.

Note

Only call the library functions from either the main program or from interrupt routines, but not both.

Note

The program space varies according to the start address of V memory area. The program space of each library is calculated based on the start address VB0.

Modbus RTU

STEP 7-Micro/WIN SMART makes communicating to Modbus devices easier by including pre-configured subroutines and interrupt routines for Modbus communication through the serial ports of the CPUs. With the Modbus RTU instructions, you can configure the S7-200 SMART to act as a Modbus RTU master or slave device.

You can find these instructions in the Libraries folder of the Instructions folder in the project tree (Page 102). When you put a Modbus RTU library instruction in your program, STEP 7-Micro/WIN SMART automatically puts one or more associated subroutines and interrupt routines in your project.

Modbus TCP

STEP 7-Micro/WIN SMART makes communicating to Modbus devices easier by including preconfigured subroutines that are specifically designed for Modbus communication over Industrial Ethernet. With the Modbus TCP protocol instructions, you can configure the S7-200 SMART to act as a Modbus TCP client or server device.

You can find these instructions in the Libraries folder of the Instructions folder in the project tree (Page 102). When you put a Modbus TCP library instruction in your program, STEP 7-Micro/WIN SMART automatically puts one or more associated subroutines in your project.

Open user communication

Open User Communication (OUC) provides a mechanism for your program to transmit and receive messages over an Ethernet network. You can select the Ethernet protocol used as the transport mechanism: UDP, TCP, or ISO-on-TCP

You can find these instructions in the Libraries folder of the Instructions folder in the project tree (Page 102). When you put an OUC library instruction in your program, STEP 7-Micro/WIN SMART automatically puts one or more associated subroutines in your project.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

PN Read Write Record library

STEP 7-Micro/WIN SMART makes reading a data record from any connected PROFINET device or writing a data record to any connected PROFINET device easier by including pre-configured subroutines that are specifically designed for the PN Read Write Record library.

SINAMICS library

STEP 7-Micro/WIN SMART makes controlling the drive easier by including pre-configured subroutines that are specifically designed for the SINAMICS library. With the SINAMICS instructions, you can control the position and speed of a physical drive, and read or modify the drive parameters.

You can find these instructions in the Libraries folder of the Instructions folder in the project tree (Page 102). When you put a SINAMICS library instruction in your program, STEP 7-Micro/WIN SMART automatically puts one or more associated subroutines in your project.

USS protocol

The USS protocol library supports Siemens drives. The STEP 7-Micro/WIN SMART USS instruction libraries make controlling drives easier by including preconfigured subroutines and interrupt routines that are specifically designed for using the USS protocol to communicate with the drive. With the USS instructions, you can control the physical drive and the read/write drive parameters.

You can find these instructions in the Libraries folder of the Instructions folder in the project tree (Page 102). When you put a USS library instruction in your program, STEP 7-Micro/WIN SMART automatically puts one or more associated subroutines in your project

⚠ WARNING

Security: Protecting networks against physical access and possible reads and writes of PLC data

Communication through Modbus RTU, Open User Communication, and USS Protocol library instructions have no security features. If an attacker can physically access your networks through one of these forms of communication, the attacker can possibly read and write PLC data. Unauthorized access to PLC data can result in death or severe personal injury.

You must protect these forms of communication by limiting physical access. For security information and recommendations, refer to the following document: Operational Guidelines for Industrial Security (http://www.industry.siemens.com/topics/global/en/industrial-security/Documents/operational_guidelines_industrial_security_en.pdf)

9.2 Overview of Modbus communication

STEP 7-Micro/WIN SMART and the S7-200 SMART CPUs make communicating to Modbus devices easier by including pre-configured subroutines and interrupt routines for the following types of communication:

- Modbus RTU communication through the serial ports of the CPUs
- Modbus TCP communication over Industrial Ethernet

Some characteristics of Modbus communication are common to both Modbus RTU and Modbus TCP.

9.2.1 Modbus addressing

Modbus addresses are five-to-six digit numbers that indicate the data type as well as the address value.

Modbus RTU master/ Modbus TCP client addressing

Modbus RTU master and Modbus TCP client instructions map the address to the correct functions to send to the slave device or client device. The Modbus address definitions are as follows:

- 00001 to 09999 are discrete outputs (coils)
- 10001 to 19999 are discrete inputs (contacts)
- 30001 to 39999 are input registers (generally analog inputs)
- 40001 to 49999 and 400001 to 465535 are holding registers

All Modbus addresses are one-based, meaning that the first data value starts at address one. The actual range of valid addresses depends on the slave device. Different devices support different data types and address ranges.

Modbus RTU slave/ Modbus TCP server addressing

The Modbus RTU slave instructions and Modbus TCP server instructions support the following addresses:

- 00001 to 09216 are discrete outputs mapped to Q0.0 - Q1151.7.
- 10001 to 19216 are discrete inputs mapped to I0.0 - I1151.7 .
- 30001 to 30056 are analog input registers mapped to AIW0 - AIW110.
- 40001 to 49999 and 400001 to 465535 are holding registers mapped to V memory.

Mapping Modbus addresses to CPU addresses

All Modbus addresses are one-based.

Table 9- 1 Mapping Modbus addresses to CPU addresses

Modbus address	CPU address
00001	Q0.0
00002	Q0.1
00003	Q0.2
...	...
01025	Q128.0 [1]
01026	Q128.1[1]
01027	Q128.2[1]
...	...
09215	Q1151.6[1]
09216	Q1151.7[1]
10001	I0.0
10002	I0.1
10003	I0.2
...	...
11025	I128.0[1]

Modbus address		CPU address
11026		I128.1[1]
11027		I128.2[1]
...		...
19215		I1151.6[1]
19216		I1151.7[1]
30001		AIW0
30002		AIW2
30003		AIW4
...		...
30056		AIW110
40001	400001	Vx (Holding reg. start)
40002	400002	Vx+2 =(Hold reg. start+2)
40003	400003	Vx+4 =(Hold reg. start+4)
...		...
4yyyy	4zzzzz	Vx+2(yyyy-1) or Vx+2(zzzzz-1)

Note

[1] CPU V2.4 and later versions support the updated memory address: Q128.0 - Q1151.7, and I128.0 - I1151.7. For the detailed information, refer to Memory ranges and features (Page 904).

MBUS parameters that limit slave accessibility

The Modbus slave/protocol allows you to limit the number of inputs, outputs, analog inputs, and holding registers (V memory) that are accessible to a Modbus master.

- **MaxIQ** assigns the maximum number of discrete inputs or outputs (I or Q addresses) a Modbus master is allowed to access.
- **MaxAI** assigns the maximum number of input registers (A or W addresses) a Modbus master is allowed to access.
- **MaxHold** assigns the maximum number of holding registers (V memory words) a Modbus master is allowed to access.

See the description of the MBUS_INIT (Page 504) instruction for more information on setting up the memory restrictions for the Modbus RTU slave.

See the description of the MBUS SERVER (Page 517) instruction for more information on setting up the memory restrictions for the Modbus TCP server.

See also

MBUS_MSG / MB_MSG2 instruction (Page 498)

9.2.2 Modbus read and write functions

The Modbus RTU master instructions utilize the Modbus functions shown below to read or write a specific Modbus address. The Modbus RTU slave device must support the appropriate Modbus function to read or write a particular Modbus address.

Table 9- 2 Required Modbus slave function support

Modbus address	Read or write	Modbus slave function required
00001 – 09999 discrete outputs	Read	Function 1
	Write	Function 5 for a single output point Function 15 for multiple output points
10001 – 19999 discrete inputs	Read	Function 2
	Write	not possible
30001 – 39999 input registers	Read	Function 4
	Write	not possible
40001 – 49999 holding registers 400001 - 465535	Read	Function 3
	Write	Function 6 for a single register Function 16 for multiple registers

Modbus message length

The S7-200 SMART CPU supports Modbus messages with up to 240 bytes (1920 bits or 120 registers) of data per message. Some slave devices might support fewer than 240 bytes of data.

9.3 Modbus RTU library

9.3.1 Modbus communication overview

9.3.1.1 Modbus RTU library features

STEP 7-Micro/WIN SMART includes Siemens Modbus RTU libraries. The Modbus RTU libraries includes pre-configured subroutines and interrupt routines that make communicating to Modbus RTU master and slave devices easier.

STEP 7-Micro/WIN SMART supports Modbus communication over RS-485 (integrated port 0 and optional signal board port 1) and RS-232 (optional signal board port 1 only) for both master and slave devices.

Modbus RTU master instructions can configure the S7-200 SMART to act as a Modbus RTU master device and communicate to one or more Modbus RTU slave devices. You can configure up to two Modbus RTU masters.

Modbus RTU slave instructions can configure the S7-200 SMART to act as a Modbus RTU slave device and communicate with Modbus RTU master devices.

Open the Libraries folder in the Instruction folder of the project tree for access to the Modbus instructions. When you place a Modbus instruction in your program, STEP 7-Micro/WIN SMART places one or more associated POUs in your project.

Note

Only call the library functions from either the main program or from interrupt routines, but not both.

Note

For the compact CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s, do not connect pin 9 of the RS485 cable used for Modbus RTU communication. The CRs CPU uses pin 9 to disable Freeport mode.

9.3.1.2 Requirements for using Modbus instructions

Modbus RTU master protocol

Modbus master instructions use the following resources from the CPU:

- MBUS_CTRL / MB_CTRL2 (Page 496) execution initializes the Modbus master protocol and dedicates the assigned CPU port (0 or 1) for Modbus master communication.

  When you use a CPU port for Modbus communications, you cannot use it for any other purpose, including communication with an HMI.
- Modbus master instructions affect all of the SM locations associated with Freeport communications on the port assigned by the MBUS_CTRL / MB_CTRL2 instruction.
- Modbus master instructions use interrupts for some functions. These interrupts must not be disabled by the user program.
- Modbus master instructions program size
  - 3 subroutines and 1 interrupt routine
  - 1942 bytes of program space for two master instructions and support routines
  - Variables for Modbus master instructions require a 286 byte block of V memory. You must assign the starting address for this block using the Library Memory command in STEP 7-Micro/WIN SMART. This command is available from the shortcut memory of the Library node under the Program Block node in the project tree, or from the Libraries section of the File menu ribbon strip.

    Note

    To change the CPU communication port from Modbus back to PPI so that you can communicate with an HMI device, set the mode parameter of the MBUS_CTRL / MB_CTRL2 instruction to a zero (0).

Modbus RTU slave protocol

Modbus slave protocol instructions use the following resources from the CPU:

- The MBUS_INIT instruction (Page 504) initializes the Modbus slave protocol and dedicates the assigned CPU port (0 or 1) for Modbus slave communication.

  When you use a CPU port for Modbus communication, you cannot use it for any other purpose, including communications with an HMI.

- Modbus slave instructions affect all of the SM locations associated with a Freeport communications on the port assigned by the MBUS_INIT instruction.
- Modbus slave instructions program size:
  - 3 subroutines and 2 interrupts.
  - 2113 bytes of program space for the two slave instructions and support routines.
  - The variables for the Modbus slave instructions require a 786 byte block of V memory. You must assign the starting address for this block using the Library Memory command in STEP 7-Micro/WIN SMART. This command is available from the shortcut memory of the Library node under the Program Block node in the project tree, or from the Libraries section of the File menu ribbon strip.

Note

To change the CPU communication port from Modbus back to PPI so that you can communicate with an HMI device, set the mode parameter of the MBUS_INIT instruction to a zero (0).

9.3.1.3 Initialization and execution time for Modbus protocol

- **Modbus RTU master protocol:** The master protocol requires a small amount of time every scan to execute the MBUS_CTRL and MB_CTRL2 instruction, if present. The time is about 0.2 milliseconds when MBUS_CTRL / MB_CTRL2 is initializing the Modbus master (first scan), and about 0.1 milliseconds on subsequent scans.

  Execution of the MBUS_MSG / MB_MSG2 instruction extends the scan time, especially in calculating the Modbus CRC for the request and response. The CRC (Cyclic Redundancy Check) ensures the integrity of the communications message. Each word in request and in the response extends the PLC scan time by about 86 microseconds. A maximum request/response (read or write of 120 words) extends the scan time to approximately 10.3 milliseconds. A read request extends the scan mainly when the program is receiving a response from a slave, and to a lesser extent when sending the request. A write request extends the scan mainly when sending data to a slave, and to a lesser extent when receiving a response.

- **Modbus RTU slave protocol:** Modbus communication uses a CRC (cyclic redundancy check) to ensure the integrity of the communications messages. The Modbus slave protocol uses a table of pre-calculated values to decrease the time required to process a message. The initialization of this CRC table requires about 11.3 milliseconds. The MBUS_INIT instruction performs this initialization, which normally happens during the first scan after entering RUN mode. You are responsible for resetting the watchdog timer if the time required by the MBUS_INIT instruction and any other user initialization exceeds

the 500 millisecond scan watchdog time. Writing to the outputs of the module resets the output module watchdog timer.

MBUS_SLAVE extends the scan time when it services a request. Calculating the Modbus CRC extends the scan time by about 40 microseconds for every byte in the request and in the response. A maximum request/response (read or write of 120 words) extends the scan time by approximately 4.8 milliseconds.

9.3.2 Modbus RTU master

9.3.2.1 Using the Modbus RTU master instructions

STEP 7-Micro/WIN SMART and the S7-200 SMART CPU support two Modbus RTU Masters. For a single Modbus RTU Master, use the instructions MBUS_CTRL (Page 496)and MBUS_MSG (Page 498). For a second Modbus RTU Master, use the instructions MB_CTRL2 (Page 496) and MB_MSG2 (Page 498).

If you use two Modbus masters in your project, make sure to use different port numbers for MBUS_CTRL and MB_CTRL2.

Procedure

To use the Modbus RTU master instructions in your S7-200 SMART program, follow these steps:

1. Insert the MBUS_CTRL / MB_CTRL2 instruction in your program and execute it on every scan. You can use the MBUS_CTRL / MB_CTRL2 instruction either to initiate or to change the Modbus communications parameters. When you insert the MBUS_CTRL / MB_CTRL2 instruction, STEP 7-Micro/WIN SMART adds several protected subroutines and interrupt routines to your program.
2. Click the Memory button Memory from the Libraries area of the File menu ribbon strip to assign a starting address for the V-Memory that the Modbus library requires. Alternatively, you can right-click the Program Block node in the project tree and select "Library Memory" from the context menu.

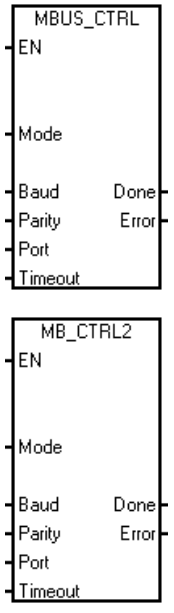
3. Place one or more MBUS_MSG / MB_MSG2 instructions in your program. You can add as many MBUS_MSG / MB_MSG2 instructions to your program as you require, but only one of these instructions can be active at a time.
4. Connect a communications cable between the S7-200 SMART CPU port you assigned with the MBUS_CTRL / MB_CTRL2 port parameter and the Modbus slave device.

NOTICE
Avoiding unwanted current flow Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable. These unwanted currents can cause communications errors or damage equipment. Ensure that you connect all equipment that with a communications cable that either shares a common circuit reference or is isolated to prevent unwanted current flows.

9.3.2.2 MBUS_CTRL / MB_CTRL2 instruction (initialize master)

MBUS_CTRL and MB_CTRL2 have identical behavior and parameters. MBUS_CTRL is for a single Modbus RTU master. MB_CTRL2 is for a second Modbus RTU master. Correspondingly, MBUS_MSG is for use with MBUS_CTRL and a single Modbus RTU master. MB_MSG2 is for use with MB_CTRL2 and a second Modbus RTU master.

Table 9- 3 MBUS_CTRL and MB_CTRL2 instruction

LAD / FBD	STL	Description
	**CALL MBUS_CTRL, Mode, Baud, Parity, Port, Timeout, Done, Error** **CALL MB_CTRL2, Mode, Baud, Parity, Port, Timeout, Done, Error**	The program calls the MBUS_CTRL / MB_CTRL2 instruction to initialize, monitor, or to disable Modbus communications. Before executing the MBUS_MSG / MB_MSG2 instruction, the program must execute the MBUS_CTRL / MB_CTRL2 without errors. The instruction completes and sets the Done bit ON before continuing to the next instruction. This instruction executes on each scan when the EN input is on.

The program must call the MBUS_CTRL / MB_CTRL2 instruction every scan (including the first scan) to allow it to monitor the progress of any outstanding messages initiated with the

MBUS_MSG / MB_MSG2 instruction. The Modbus master protocol does not operate correctly unless the program executes MBUS_CTRL / MB_CTRL2 every scan.

Table 9- 4 Parameters for the MBUS_CTRL / MB_CTRL2 instruction

Parameter	Data type	Operands
Mode	BOOL	I, Q, M, S, SM, T, C, V, L
Baud	DWORD	VD, ID, QD, MD, SD, SMD, LD, AC, Constant, *VD, *AC, *LD
Parity, Port	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Timeout	WORD	VW, IW, QW, MW, SW, SMW, LW, AC, Constant, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

The value for the **Mode** input selects the communications protocol. An input value of 1 assigns the CPU port to Modbus protocol and enables the protocol. An input value of 0 assigns the CPU port to PPI system protocol and disables Modbus protocol.

Parameter **Parity** is set to match the parity of the Modbus slave device. All settings use one start bit and one stop bit. The allowed values are: 0 (no parity). 1 (odd parity), and 2 (even parity).

Parameter **Port** sets the physical communication port (0 = RS-485 integrated in CPU, 1 = RS-485 or RS-232 located on the optional CM01 signal board).

Parameter **Timeout** is set to the number of milliseconds to wait for the response from the slave. The Timeout value can be set anywhere in the range of 1 millisecond to 32767 milliseconds. A typical value would be 1000 milliseconds (1 second). The Timeout parameter should be set to a value large enough so that the slave device has time to respond at the selected baud rate.

The Timeout parameter is used to determine if the Modbus slave device is responding to a request. The Timeout value determines how long the Modbus Master will wait for the first character of the response after the last character of the request has been sent. The Modbus master will receive the entire response from the Modbus slave device if at least one character of the response is received within the Timeout time.

When the MBUS_CTRL / MB_CTRL2 instruction completes, the instruction returns TRUE for the Done output.

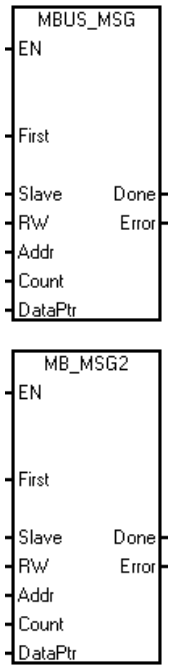
The Error output contains the result of executing the instruction.

See also Modbus RTU master execution error codes (Page 501)

9.3.2.3 MBUS_MSG / MB_MSG2 instruction

MBUS_MSG and MB_MSG2 have identical behavior and parameters. MBUS_MSG is for a single Modbus RTU master. MB_MSG2 is for a second Modbus RTU master.

Table 9- 5 MBUS_MSG / MB_MSG2 instruction

LAD / FBD	STL	Description
	**CALL MBUS_MSG, First, Slave, RW, Addr, Count, DataPtr, Done, Error** **CALL MB_MSG2, First, Slave, RW, Addr, Count, DataPtr, Done, Error**	The program calls the MBUS_MSG / MB_MSG2 instruction to initiate a request to a Modbus slave and process the response.

The MBUS_MSG / MB_MSG2 instruction initiates a master request to a Modbus slave when both the EN input and the First inputs are ON. Sending the request, waiting for the response, and processing the response usually requires several PLC scan times. The EN input must be ON to enable the send request, and must remain ON until the instruction returns ON (True) for the Done bit.

Only one each of MBUS_MSG or MB_MSG2 instruction can be active at a time. If the program enables more than one MBUS_MSG instruction or more than one MB_MSG2 instruction, then the CPU processes the first MBUS_MSG instruction or MB_MSG2 instruction and all subsequent MBUS_MSG or MB_MSG2 instructions will abort with an error code 6.

Table 9- 6 Parameters for the MBUS_MSG / MB_MSG2 instruction

Parameter	Data type	Operands
First	BOOL	I, Q, M, S, SM, T, C, V, L (Power flow conditioned by a positive edge detection element)
Slave	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
RW	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Addr	DWORD	VD, ID, QD, MD, SD, SMD, LD, AC, Constant, *VD, *AC, *LD
Count	INT	VW, IW, QW, MW, SW, SMW, LW, AC, Constant, *VD, *AC, *LD

Parameter	Data type	Operands
DataPtr	DWORD	&VB
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

Set parameter **First** ON for only one scan when there is a new request to send. Pulse the First input through an edge detection element (for example, Positive Edge), which causes the program to transmit the request one time. See the example program (Page 507) for details.

Parameter **Slave** is the address of the Modbus slave device. The allowed range is 0 through 247. Address 0 is the broadcast address. Use address 0 only for write requests. There is no response to a broadcast request to address 0. Not all slave devices support the broadcast address. The S7-200 SMART Modbus slave library does not support the broadcast address.

Use parameter **RW** to indicate whether this message is to be a read or a write: 0 (Read) and 1 (Write).

Discrete outputs (coils) and holding registers support both read and write requests. Discrete inputs (contacts) and input registers only support read requests.

Parameter **Addr** is the starting Modbus address. S7-200 SMART supports the following ranges of addresses:

- 00001 to 09999 for discrete outputs (coils)
- 10001 to 19999 for discrete inputs (contacts)
- 30001 to 39999 for input registers
- 40001 to 49999 and 400001 to 465535 for holding registers

The addresses that the Modbus slave device supports determine the actual range of values for Addr.

Parameter **Count** assigns the number of data elements to read or write in this request. The Count is the number of bits for the bit data types, and the number of words for the word data types.

- Address 0xxxx Count is the number of bits to read or write
- Address 1xxxx Count is the number of bits to read
- Address 3xxxx Count is the number of input register words to read
- Address 4xxxx or 4yyyyy Count is the number of holding register words to read or write

The MBUS_MSG / MB_MSG2 instruction reads or writes a maximum of 120 words or 1920 bits (240 bytes of data). The actual limit on the value of Count depends on the limits in the Modbus slave device.

The parameter **DataPtr** is an indirect address pointer that points to the V memory in the CPU for the data associated with the read or write request. For a read request, set the DataPtr to the first CPU memory location used to store the data read from the Modbus slave. For a write request, set DataPtr to point to the first CPU memory location of the data to be sent to the Modbus slave.

The program passes the DataPtr value to MBUS_MSG / MB_MSG2 as an indirect address pointer. For example, if the data to be written to a Modbus slave device starts at address

VW200 in the CPU, the value for the DataPtr would be &VB200 (address of VB200). Pointers must always be a type VB even if they point to word data.

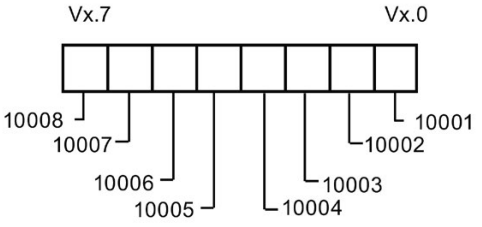
Memory layout

Holding registers (address 4xxxx or 4yyyyy) and input registers (address 3xxxx) are word values (2 bytes or 16 bits). CPU words are formatted the same as Modbus registers. The lower numbered V memory address is the most significant byte of the register. The higher numbered V memory address is the least significant byte of the register. The table below shows how the CPU byte and word addressing corresponds to the Modbus register format.

Table 9- 7 Modbus Holding Register

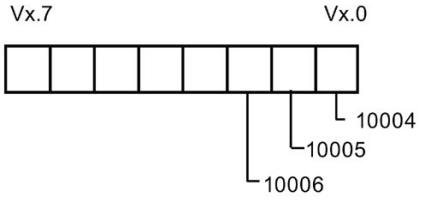
CPU memory byte address		CPU memory word address		Modbus holding register address	
Address	Hex data	Address	Hex data	Address	Hex data
VB200	12	VW200	12 34	40001	12 34
VB201	34				
VB202	56	VW202	56 78	40002	56 78
VB203	78				
VB204	9A	VW204	9A BC	40003	9A BC
VB205	BC				

The CPU reads and writes the bit data (addresses 0xxxx and 1xxxx) areas as packed bytes; that is, each byte consists of 8 bits of data. The least significant bit of the first data byte is the addressed bit number (the parameter Addr). If you intend to write only a single bit then you must set the bit in the least significant bit (Vx.0) of the byte pointed to by DataPtr.



Format for Packed Bytes (Discrete Input Addresses)

For bit data addresses that do not start on a byte boundary, you must set the bit corresponding to the starting address in the least significant bit of the byte. See the example of the packed byte format for 3 bits starting at Modbus address 10004.



Format for Packed Bytes (Discrete input starting at address 10004)

When writing to the discrete output data type (coils), you must place the bits in the correct bit positions within the packed byte before passing the data to the MBUS_MSG / MB_MSG2 instruction by means of the DataPtr.

Outputs

The Done output is FALSE after the program has sent a request and is receiving a response. The Done output is TRUE when the response is complete or when the MBUS_MSG / MB_MSG2 instruction aborts because of an error.

The Error output (Page 501) is valid only when the Done output is TRUE.

9.3.2.4 Modbus RTU master execution error codes

The high numbered error codes (starting with 101) are errors that are returned by the Modbus slave device. These errors indicate that the slave does not support the requested function or that the requested address (either data type or range of addresses) is not supported by the Modbus slave device.

The low numbered error codes (1 through 12) are errors that are detected by the MBUS_MSG instruction. These error codes generally indicate a problem with the input parameters of the MBUS_MSG instruction, or a problem receiving the response from the slave. Parity and CRC errors indicate that there was a response but that the data was not received correctly. This is usually caused by an electrical problem such as a bad connection or electrical noise.

MBUS_CTRL error code	Description
0	No error
1	Invalid parity type
2	Invalid baud rate
3	Invalid timeout
4	Invalid mode
9	Invalid port number
10	Signal board port 1 missing or not configured

MBUS_MSG error code	Description
0	No error
1	Parity error in response: This is only possible if even or odd parity is used. The transmission was disturbed and possibly incorrect data was received. This error is usually caused by an electrical problem such as incorrect wiring or electrical noise affecting the communication.
2	Not used
3	Receive timeout: There was no response from the slave within the Timeout time. Some possible causes are bad electrical connections to the slave device, master and slave are set to a different baud rate / parity setting, and incorrect slave address.
4	Error in request parameter: One or more of the input parameters (Slave, RW, Addr, or Count) is set to an illegal value. Check the documentation for allowed values for the input parameters.
5	Modbus master not enabled: Call MBUS_CTRL on every scan prior to calling MBUS_MSG.

MBUS_MSG error code	Description
6	Modbus is busy with another request: Only one MBUS_MSG instruction can be active at a time.
7	Error in response: The response received does not correspond to the request. This indicates some problem in the slave device or that the wrong slave device answered the request.
8	CRC error in response: The transmission was disturbed and possibly incorrect data was received. This error is usually caused by an electrical problem such as incorrect wiring or electrical noise affecting the communication.
11	Invalid port number
12	Signal board port 1 missing or not configured
101	Slave does not support the requested function at this address: See the required Modbus slave function support table in the "Using the Modbus master Instructions" help topic.
102	Slave does not support the data address: The requested address range of Addr plus Count is outside the allowed address range of the slave.
103	Slave does not support the data type: The Addr type is not supported by the slave device.
104	Slave device failure
105	Slave accepted the message but the response is delayed: This is an error for MBUS_MSG and the user program should resend the request at a later time.
106	Slave is busy and rejected the message: You can try the same request again to get a response.
107	Slave rejected the message for an unknown reason.
108	Slave memory parity error: There is an error in the slave device.

9.3.3 Modbus RTU slave

9.3.3.1 Using the Modbus RTU slave instructions

Procedure

To use the Modbus slave instructions in your S7-200 SMART program, follow these steps:

1. Insert the MBUS_INIT instruction in your program and execute the MBUS_INIT instruction for one scan only. You can use the MBUS_INIT instruction either to initiate or to change the communications parameters. When you insert the MBUS_INIT instruction, several hidden subroutines and interrupt routines are automatically added to your program.
2. Click the Memory button Memory from the Libraries area of the File menu ribbon strip to assign a starting address for the V memory that the Modbus library requires. Alternatively, you can right-click the Program Block node in the project tree and select "Library Memory" from the context menu. In addition to this V memory block, you define another V memory block with the HoldStart and MaxHold parameters of MBUS_INIT. Be careful that your program assignments in V memory do not overlap. If there is any overlap of the memory areas, the MBUS_INIT instruction returns an error.

3. Place only one MBUS_SLAVE instruction in your program. This instruction should be called every scan to service any requests that have been received.
4. Connect a communications cable between the S7-200 SMART CPU port you assigned with the MBUS_INIT port parameter and the Modbus master device.

NOTICE
Avoiding unwanted current flow Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable. These unwanted currents can cause communications errors or damage equipment. Ensure that all equipment that is connected with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows.

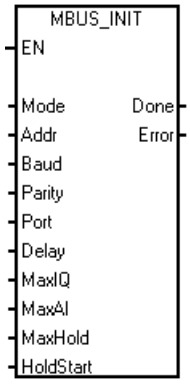
The accumulators (AC0, AC1, AC2, AC3) are used by the Modbus slave instructions and appear in the Cross Reference listing. Prior to execution, the values in the accumulators of a Modbus slave instruction are saved and restored to the accumulators before the Modbus slave instruction is complete, ensuring that all user data in the accumulators is preserved while executing a Modbus slave instruction.

The Modbus slave instructions support the Modbus RTU protocol. These instructions use the Freeport feature of the S7-200 SMART CPU to support the most common Modbus functions. The following Modbus functions are supported:

Function	Description
1	Read single/multiple coil (discrete output) status. Function 1 returns the on/off status of any number of output points (Qs).
2	Read single/multiple contact (discrete input) status. Function 2 returns the on/off status of any number of input points (Is).
3	Read single/multiple holding registers. Function 3 returns the contents of V memory. Holding registers are word values under Modbus and allow you to read up to 120 words in one request.
4	Read single/multiple input registers. Function 4 returns analog Input values.
5	Write single coil (discrete output). Function 5 sets a discrete output point to the specified value. The point is not forced and the program can overwrite the value written by the Modbus request.
6	Write single holding register. Function 6 writes a single holding register value to the V memory of the S7-200 SMART.
15	Write multiple coils (discrete outputs). Function 15 writes the discrete output values to the Q image register of the S7-200 SMART. The starting output point must begin on a byte boundary (for example, Q0.0 or Q2.0) and the number of outputs written must be a multiple of eight. This is a restriction for the Modbus slave protocol instructions. The points are not forced and the program can overwrite the values written by the Modbus request.
16	Write multiple holding registers. Function 16 writes multiple holding registers to the V memory of the S7-200 SMART. There can be up to 120 words written in one request.

9.3.3.2 MBUS_INIT instruction (initialize slave)

Table 9- 8 MBUS_INIT instruction

LAD/ FBD	STL	Description
	**CALL MBUS_INIT, Mode, Addr, Baud, Parity, Port, Delay, MaxIQ, MaxAI, MaxHold, HoldStart, Done, Error**	The MBUS_INIT instruction enables, initializes, or disables Modbus communications. Before an MBUS_SLAVE instruction can be used, MBUS_INIT must be executed without errors. The instruction completes and the Done bit is set immediately, before continuing to the next instruction. The instruction is executed on each scan when the EN input is ON.

The program must execute MBUS_INIT instruction exactly once for each change in communications state. Therefore, pulse the EN input through an edge detection element, or execute MBUS_INIT only on the first scan.

Table 9- 9 MBUS_INIT parameters

Inputs/outputs	Data type	Operands
Mode, Addr, Parity, Port	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Baud, HoldStart	DWORD	VD, ID, QD, MD, SD, SMD, LD, AC, Constant, *VD, *AC, *LD
Delay, MaxIQ, MaxAI, MaxHold	WORD	VW, IW, QW, MW, SW, SMW, LW, AC, Constant, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

The value for the **Mode** input selects the communications protocol: an input value of 1 assigns Modbus protocol and enables the protocol, and an input value of 0 PPI protocol and disables Modbus protocol.

Parameter **Addr** sets the address at inclusive values between 1 and 247.

Parameter **Baud** sets the baud rate at 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Parameter **Parity** is set to match the parity of the Modbus master. All settings use one stop bit. The accepted values are: 0 (no parity), 1 (odd parity), and 2 (even parity).

Parameter **Port** sets the physical communication port (0 = RS-485 integrated in CPU, 1 = RS-485 or RS-232 located on an optional signal board).

Parameter **Delay** extends the standard Modbus end-of-message timeout condition by adding the assigned number of milliseconds to the standard Modbus message timeout. The typical value for this parameter should be 0 when operating on a wired network. If you are using modems with error correction, set the delay to a value of 50 to 100 milliseconds. If you are using spread spectrum radios, set the delay to a value of 10 to 100 milliseconds. The Delay value can be 0 to 32767 milliseconds.

Parameter **MaxIQ** sets the number of I and Q points available to Modbus addresses 0xxxx and 1xxxx at values of 0 to 256. A value of 0 disables all reads and writes to the inputs and outputs. The suggested value for MaxIQ is 256.

Parameter **MaxAI** sets the number of word input (AI) registers available to Modbus address 3xxxx at values of 0 to 56. A value of 0 disables reads of the analog inputs. The suggested value for MaxAI to allow access to all of the CPU analog inputs, is as follows:

- 0 for CPUs CR20s, CR30s, CR40s, and CR60s
- 56 for all other CPU models

Parameter **MaxHold** sets the number of word holding registers in V memory available to Modbus address 4xxxx or 4yyyyy. For example, if you want to allow Modbus master access for 2000 bytes of V memory, set MaxHold to a value of 1000 words (holding registers).

Parameter **HoldStart** is the address of the start of the holding registers in V memory. This value is generally set to VB0, so the parameter HoldStart is set to &VB0 (address of VB0). Other V memory addresses can be specified as the starting address for the holding registers to allow VB0 to be used elsewhere in the project. The Modbus master has access to MaxHold number of words of V memory starting at HoldStart.

When the MBUS_INIT instruction completes, the Done output is turned ON.

The Error output (Page 506) byte contains the result of executing the instruction. This output is only valid if Done is ON. If Done is OFF, the error parameter is not changed.

9.3.3.3 MBUS_SLAVE instruction

Table 9- 10 MBUS_SLAVE instruction

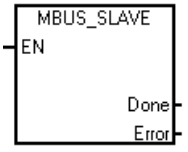
LAD / FBD	STL	Description
	`CALL MBUS_SLAVE, Done, Error`	The MBUS_SLAVE instruction is used to service a request from the Modbus master and must be executed every scan to allow it to check for and respond to Modbus requests. The instruction is executed on each scan when the EN input is ON. The MBUS_SLAVE instruction has no input parameters.

Table 9- 11 Parameters for the MBUS_SLAVE Instruction

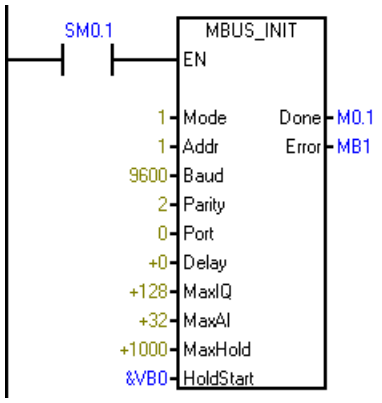
Parameter	Data Type	Operands
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

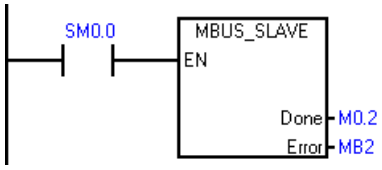
The Done output is ON when the MBUS_SLAVE instruction responds to a Modbus request. The Done output is OFF, if there was no request serviced.

The Error output (Page 506) contains the result of executing the instruction. This output is only valid if Done is ON. If Done is OFF, the error parameter is not changed.

Table 9- 12 Example program of S7-200 SMART CPU operating as a Modbus slave

LAD		STL

	Initialize the Modbus Slave protocol on the first scan. Set the slave address to 1, set port 0 to 9600 baud with even parity, all access to all I, Q and AI values, allow access to 1000 holding registers (2000 bytes) starting at VB0.	`Network 1`
		`LD SM0.1`
		`CALL MBUS_INIT, 1, 1, 9600,`
		`2, 0, 128, 32, 1000, &VB0,`
		`M0.1, MB1`

	Execute the Modbus Slave protocol on every scan.	`Network 2`
		`LD SM0.0`
		`CALL MBUS_SLAVE, M0.2, MB2`

9.3.3.4 Modbus RTU slave execution error codes

Error code	Description
0	No error
1	Memory range error
2	Illegal baud rate or parity
3	Illegal slave address
4	Illegal value for Modbus parameter
5	Holding registers overlap Modbus Slave symbols
6	Receive parity error
7	Receive CRC error
8	Illegal function request/function not supported
9	Illegal memory address in request
10	Slave function not enabled
11	Invalid port number
12	Signal board port 1 missing or not configured

9.3.4 Modbus RTU master example program

This example program shows how to use the Modbus Master instructions to write and read four holding registers to and from a Modbus slave each time input I0.0 turns on.

The CPU writes four words starting at VW100 to the holding registers of the Modbus slave starting at address 40001.

The CPU then reads four holding registers from 40010 to 40013 from the Modbus slave and places the data into the V memory of the CPU starting at VW200.

This example uses a single master and the MBUS_CTRL and MBUS_MSG instructions. The same concepts apply to examples with a second master and the MB_CTRL2 and MB_MSG2 instructions.

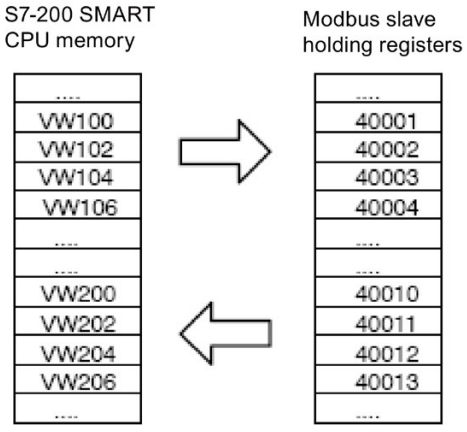
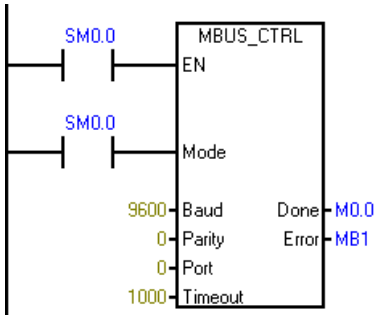
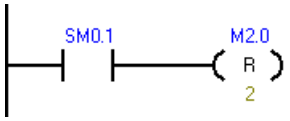
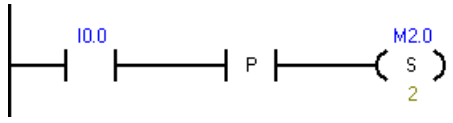
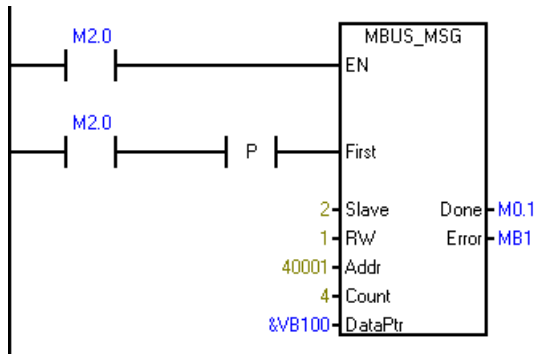
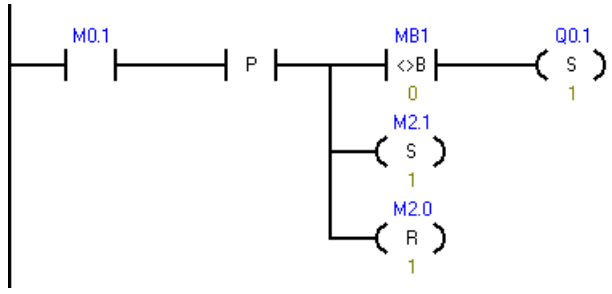
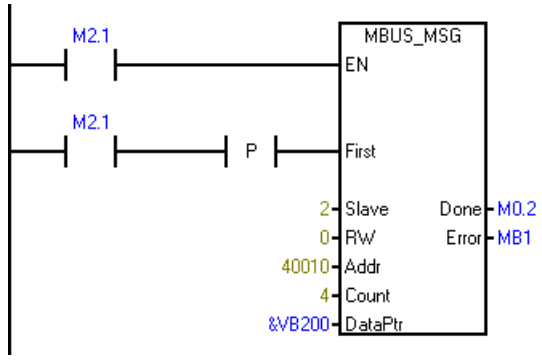
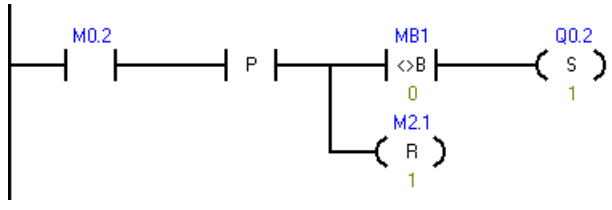


Figure 9-1 Example Program Data Transfers

The following program turns on outputs Q0.1 and Q0.2 if the MBUS_MSG instruction returns an error.

Table 9- 13 Example Modbus master program

LAD	Description
	Network 1 Initialize and monitor the Modbus master by calling MBUS_CTRL on every scan. The Modbus master is set for 9.6 Kbps and no parity. The slave device is allowed 1000 milliseconds (1 second) to respond.
	Network 2 On the first scan, reset the enable flags (M2.0 and M2.1) used for the two MBUS_MSG instructions.

LAD	Description
	Network 3 When I0.0 changes from OFF to ON, set the enable flag for the first MBUS_MSG instruction (M2.0).
	Network 4 Call the MBUS_MSG instruction when the first enable flag (M2.0) is ON. The First parameter must be set for only the first scan that the instruction is enabled. This instruction writes (RW = 1) 4 holding registers to slave 2. The write data is taken from VB100-VB107 (4 words) in the CPU and written to address 40001 - 40004 in the Modbus slave.
	Network 5 When the first MBUS_MSG instruction is complete (Done goes from 0 to 1), clear the enable for the first MBUS_MSG and set the enable for the second MBUS_MSG instruction. If Error (MB1) is not zero, then set Q0.1 to show the error.
	Network 6 Call the second MBUS_MSG instruction when the second enable flag (M2.1) is ON. The First parameter must be set for only the first scan that the instruction is enabled. This instruction reads (RW = 0) 4 holding registers from slave 2. The data is read from address 40010 - 40013 in the Modbus slave and copied to VB200 - VB207 (4 words) in the CPU.
	Network 7 When the second MBUS_MSG instruction is complete (Done goes from 0 to 1), clear the enable for the second MBUS_MSG instruction. If Error (MB1) is not zero, then set Q0.2 to show the error.

9.3.5 Modbus RTU advanced user information

Overview

This topic contains information for advanced users of the Modbus RTU master library. Most users do not need this information and will not need to modify the default operation of the Modbus RTU master library.

Retries

The Modbus master instructions automatically resend the request to the slave device if one of the following errors is detected:

- There is no response within the response timeout time (parameter Timeout on the MBUS_CTRL / MB_CTRL2) instruction (Error code 3).
- The time between characters of the response exceeded the allowed value (Error code 3).
- There is a parity error in the response from the slave (Error code 1).
- There is a CRC error in the response from the slave (Error code 8).
- The returned function did not match the request (Error code 7).

The Modbus Master resends the request two additional times before setting the Done and Error output parameters.

You can change the number of retries by finding the symbol mModbusRetries in the Modbus master symbol table and changing this value after the program executes MBUS_CTRL / MB_CTRL2. The mModbusRetries value is a BYTE with a range of 0 to 255 retries.

Inter-character timeout

Modbus master execution aborts a response from a slave device if the time between characters in the response exceeds an assigned time limit. The default time is set to 100 milliseconds which should allow the Modbus master instructions to work with most slave devices over wire or telephone modems. If the CPU detects this error, the MBUS CTRL / MB_CTRL2 instruction returns error code 3 in the Error parameter.

Communication might possibly require a longer time between characters, either because of the transmission medium (for example, telephone modem) or because the slave device itself requires more time. You can lengthen this timeout by finding the symbol *mModbusCharTimeout* in the Modbus master symbol table and changing this value after MBUS_CTRL / MB_CTRL2 has been executed. The mModbusCharTimeout value is an INT with a range of 1 to 30000 milliseconds.

Single vs. multiple bit / word write functions

Some Modbus slave devices do not support the Modbus functions to write a single discrete output bit (Modbus function 5) or to write a single holding register (Modbus function 6). These devices only support the multiple bit write (Modbus function 15) or multiple register write (Modbus function 16) instead. The MBUS_MSG / MB_MSG2 instruction returns an error code 101 if the slave device does not support the single bit/word Modbus functions.

The Modbus master protocol allows you to force the MBUS_MSG / MB_MSG2 instruction to use the multiple bit/word Modbus functions instead of the single bit/word Modbus functions. You can force the multiple bit/word instructions by finding the symbol *mModbusForceMulti* in the Modbus master symbol table and changing this value after the program executes MBUS_CTRL / MB_CTRL2. Set mModbusForceMulti to TRUE to force the use of the multiple bit/word functions when writing a single bit or register.

Accumulator usage

The Modbus master instructions use accumulators (AC0, AC1, AC2, AC3), which appear in the Cross Reference listing. The Modbus master instructions save and restore the values in the accumulators. All CPU preserves all user data in the accumulators while executing the instructions.

Holding register addresses greater than 49999

Modbus holding addresses are within the range of 40001 to 49999. This range is adequate for most applications but there are some Modbus slave devices with data mapped into holding registers at a higher address range.

The MBUS_MSG / MB_MSG2 instruction allows an additional range for the parameter **Addr** to support an extended range of holding register addresses at addresses 400001 to 465536.

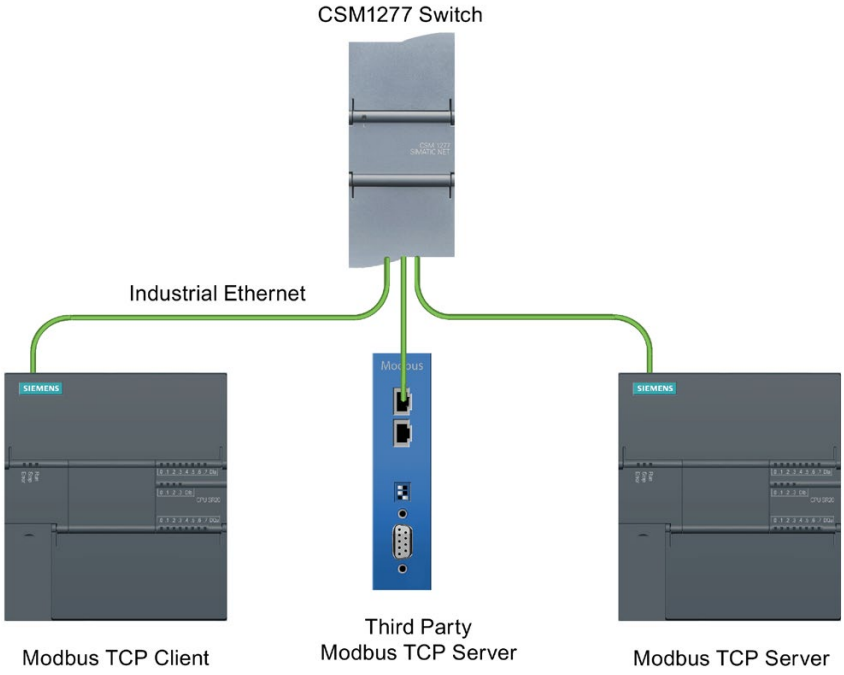
For example: to access holding register 16768, the **Addr** parameter of MBUS_MSG / MB_MSG2 should be set to 416768.

The extended addressing allows access to the full range of 65536 possible addresses supported by the Modbus protocol. This extended addressing is only applicable for holding registers.

9.4 Modbus TCP library

9.4.1 Modbus TCP library features

Modbus TCP is Modbus communication transmitted over an Industrial Ethernet TCP/IP network. The S7-200 SMART uses a client-server approach, in which a Modbus client device initiates a TCP/IP connection with a Modbus server device. With a connection established, a client makes a request to a server, which responds to the client's requests. The client can request to read a section of memory from the server device or to write a quantity of data to the memory of the server device. The server replies to the request with a response if the request is valid or an error message if the request is invalid.



STEP 7-Micro/WIN SMART provides two Modbus TCP library instructions, which are available from the Libraries folder of the Instructions folder in the STEP 7-Micro/WIN SMART project tree:

- MBUS_CLIENT (Page 513)
- MBUS_SERVER (Page 517)

Modbus TCP client protocol

The Modbus client instruction (MBUS_CLIENT) uses the following resources from the CPU:

- One active connection resource for every connection to a Modbus server. MBUS_CLIENT automatically generates the connection ID.
- The Modbus client uses the following program entities:
  - 1 subroutine
  - 2849 bytes of program space
  - A 638-byte block of V memory for the instruction symbols
    You must assign the starting address for this block from the Library Memory command in STEP 7-Micro/WIN SMART. The Library Memory command is accessible from the Program Block or Program Block > Library folder in the project tree after you have placed an MBUS_CLIENT instruction in the program.

Modbus TCP server protocol

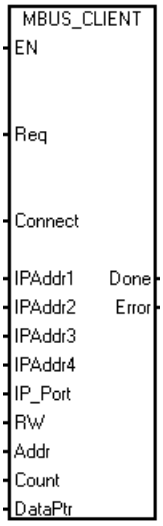
The Modbus server instruction (MBUS_SERVER) uses the following resources from the CPU:

- One passive connection resource for every connection to a Modbus server. MBUS_SERVER automatically generates the connection ID.
- The Modbus server uses the following program entities:
  - 1 subroutine
  - 2969 bytes of program space
  - A 445-byte block of V memory for the instruction symbols
    You must assign the starting address for this block from the Library Memory command in STEP 7-Micro/WIN SMART. The Library Memory command is accessible from the Program Block or Program Block > Library folder in the project tree after you have placed an MBUS_SERVER instruction in the program.

9.4.2 Modbus TCP client

9.4.2.1 MBUS_CLIENT instruction

Table 9- 14 MBUS_CLIENT instruction

LAD / FBD	STL	Description
	**Call MBUS_CLIENT Req, Connect, IPAddr1, IPAddr2, IPAddr3, IPAddr4, IP_Port, RW, Addr, Count, DataPtr, Done, Error**	MBUS_CLIENT communicates as a Modbus TCP client through the Ethernet port on the S7-200 SMART CPU. MBUS_CLIENT can make a client-server connection, send a Modbus function request, receive a client response, and connect to and disconnect from a Modbus TCP server.

The program execution cycle must call MBUS_CLIENT every scan until the Done output is TRUE. In each cycle, MBUS_CLIENT exits so that the program can continue to run. MBUS_CLIENT sets Done to TRUE when the client completes the request.

Table 9- 15 Data types for the parameters

Parameter and type		Data type	Description
Req	IN	BOOL	The Req parameter allows your program to send a Modbus request to the server. FALSE: No Modbus communication request TRUE: Request to communicate with a Modbus TCP server
Connect	IN	BOOL	The Connect parameter allows your program to connect to and disconnect from a Modbus server device. If Connect = TRUE and a connection does not exist, then MBUS_CLIENT attempts to make a connection to the assigned IP address and port number. If Connect = FALSE and a connection exists, then MBUS_CLIENT attempts a disconnect operation. When Connect = FALSE, the CPU ignores any further requests. This means if the program calls MBUS_CLIENT with Req = TRUE but Connect = FALSE, the CPU ignores the request.
IPAddr1	IN	BYTE	First octet of the IP address of the server to which the client attempts to connect and subsequently communication using the Modbus application protocol.

Parameter and type		Data type	Description
IPAddr2	IN	BYTE	Second octet of the IP address of the server to which the client attempts to connect and subsequently communicate using the Modbus application protocol.
IPAddr3	IN	BYTE	Third octet of the IP address of the server to which the client attempts to connect and subsequently communicate using the Modbus application protocol.
IPAddr4	IN	BYTE	Fourth octet of the IP address of the server to which the client attempts to connect and subsequently communicate using the Modbus application protocol.
IP_Port	IN	WORD	Port number of the server to which the client attempts to connect and subsequently communicate using Modbus TCP. Default: 502 Set the port to the actual port number of your device.
RW	IN	BYTE	Assigns the type of request (read or write) where 0 = Read and 1 = Write. See the Modbus functions table below for details.
Addr	IN	DWORD	Modbus starting Address: Assigns the starting address of the data to be accessed by MBUS_CLIENT. See the Modbus functions table below for details.
Count	IN	INT	Modbus data length: Number of bits or holding registers to be accessed in this request The ranges 10001 to 19999 and 30001 to 39999 are read-only addresses. For input and output bits, the maximum Count value is 1920 bits. For input and holding registers, the maximum Count value is 120 WORDS. See the Modbus functions table below for details.
DataPtr	IN_OUT	DWORD	Pointer to the Modbus data register: DataPtr points to the V memory location for the data associated with the read or write request. For read requests, this location is the first memory location at which to store the data read from the Modbus server. For write requests, this location is the first memory location of the data to be written to the Modbus server.
Done	OUT	BOOL	TRUE: One of the following conditions is true: • Client has established a connection with a server • Client is disconnected to a server • Client received a Modbus response • Error occurred FALSE: Client is busy establishing a connection or waiting for a Modbus response from the server.
Error	OUT	BOOL	Instruction execution result Valid for only a single cycle after the error occurred

RW and Addr parameters select the Modbus communication function

The MBUS_CLIENT instruction uses the RW input to indicate either a read or write function and the Addr input to define what type of date to read or write.

The following table shows the Modbus functions that the MBUS_CLIENT instruction provides based on the RW and Addr input parameters:

FC	Function	RW	Addr	Count	CPU Address
01	Read Bits	0	00001 to 09999	1 to 1920 bits	Q0.0 - Q1151.7
02	Read Bits	0	10001 to 19999	1 to 1920 bits	I0.0 - I1151.7
03	Read Words	0	40001 to 49999 400001 to 465535	1 to 120 words	V memory
04	Read Words	0	30001 to 39999	1 to 120 words	AIW0 - AIW110
05	Write Single Bit	1	00001 to 09999	1 bit	Q0.0 - Q1151.7
06	Write single Word	1	40001 to 49999 400001 to 465535	1 word	V memory
15	Write Multiple Bits	1	00001 to 09999	1 to 1920 bits	Q0.0 - Q1151.7
16	Write Multiple Words	1	40001 to 49999 400001 to 465535	1 to 120 words	V memory

Multiple client connections

A Modbus TCP client can support multiple connections up to the maximum number of Open User Communications connections that the PLC allows. The total number of connections for a PLC, including Modbus TCP clients and servers, must not exceed the maximum number of supported Open User Communications connections (Page 382). Multiple client connections must have different IPAddr or IP_Port input parameters.

Establishing a connection

When the Connect input is TRUE, the client attempts to establish a connection with the server device at the provided IP address and IP Port. If the server device is unreachable, the connection request eventually times out, which can take several seconds. While a connection request is in progress, no other operation can interrupt or abort it. If the server is unavailable, it immediately refuses the client's connection request. If the server is available, the client establishes a connection and can send a request to the server. The MBUS_CLIENT instruction returns an error if there are no connections resources available for the Modbus client.

Processing a request

The client only processes requests when Connect = TRUE. Once the client has established a connection with the server, the program makes a new request by calling MBUS_CLIENT with Req = TRUE when no Modbus request is active. When the Modbus client executes the request, it captures all of the input values. Pulse the Req input through an edge detection element (for example, Positive Edge), which causes the instruction to transmit the request one time. Any subsequent changes to the input values while the request is active result in MBUS_CLIENT returning an error code.

Once the client has sent a request to the server, the client waits for a response up to the mReceiveTimeout period of time. While the client is waiting for a response, it is not available for other Modbus operations. If the client does not receive a response within the mReceiveTimeout period of time, MBUS_CLIENT returns an error.

If the client receives a valid response from the server, it processes any further actions depending on the response. The client then returns to a ready state and is available for additional requests from the program.

Disconnecting an established connection

If the Connect input is FALSE, the client attempts to disconnect from the server, if there is an active connection between the client and server. If there is a connect or send operation in progress the disconnect operation returns an error. A disconnect request cannot interrupt an operation. If no operation is in progress, the CPU terminates the active connection and the client returns to an idle state. The connection resource is then available to other operations in the CPU.

9.4.2.2 Modbus TCP client execution error codes

The MBUS_CLIENT instruction (Page 513) can return the following error codes:

Error (decimal)	Description
0	No error
32	Unknown state Check network connections and check that the program is not modifying any library symbols that intefere with client/server communication.
33	Connection is busy with another request. A single connection can only service one Modbus request at a time.
34	Addr input is an illegal value.
35	Count input is an illegal value.
36	RW input is an illegal value.
37	Transaction ID of the request does not match the response from the server. This error indicates some problem in the server device or that the wrong server device answered the request. Received an invalid protocol ID from the server.
38	Received an invalid protocol ID from the server.
39	Number of bytes that the server sent does not match "Count" input value
40	Unit identifier of the request does not match the response from the server
41	Function code of the request does not match the response from the server

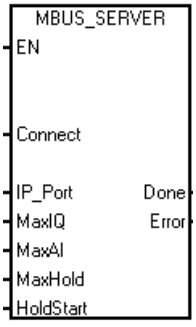
Error (decimal)	Description
42	The data that the server sent does not match the data that a Modbus TCP write function requested
43	Receive timeout: The server did not respond within the mReceiveTimeout time. Check the connection to the Modbus server device.
44	The input values do not match the values for the active request.
45	The Modbus data register range exceeds the V memory range.

In addition to the MBUS_CLIENT errors listed above, refer also to the Modbus TCP general exception codes (Page 528) and Open User Communication error codes (Page 552)

9.4.3 Modbus TCP server

9.4.3.1 MBUS_SERVER instruction

Table 9- 16 MBUS_SERVER instruction

LAD / FBD	STL	Description
	`Call MBUS_SERVER Connect, IP_Port, MaxIQ, MaxAI, MaxHold, HoldStart, Done, Error`	MBUS_SERVER communicates as a Modbus TCP server through the Ethernet port. MBUS_SERVER can accept a request to connect with Modbus TCP client, receive a Modbus function request, and send a response message.

Execute the MBUS_SERVER instruction in every scan in order for the Modbus server to respond to requests from Modbus clients in a reasonable amount of time. The MBUS_SERVER instruction handles establishing connections, receiving requests, and sending responses. The Modbus server cannot operate correctly unless the program calls MBUS_SERVER on every scan.

Table 9- 17 Data types for the parameters

Parameter and type		Data type	Description
Connect	IN	BOOL	You use the Connect parameter to connect to or disconnect from a client device. The Modbus server attempts to create a "passive" connection, which means that the server will accept a connection request from any requesting IP address. If Connect = TRUE, and the client has not established a connection to the server, the server will passively listen for a TCP connection request. If Connect = FALSE and a connection does exist, the server initiates a disconnect operation. The program can thus use the Connect parameter to control when the server can accept a connection. When Connect = FALSE, MBUS_SERVER performs no other operation. Note that MBUS_SERVER can automatically initiate a disconnect operation when specific TCP errors occur.
IP_Port	IN	WORD	Port number of the server to which the client will attempt to connect and communicate using the Modbus application protocol. Default: 502 Set the port to the actual port number of your device
MaxIQ	IN	WORD	Parameter MaxIQ sets the number of I and Q points available to Modbus addresses 0xxxx and 1xxxx at values of 0 to 256. A value of 0 disables all reads and writes to the inputs and outputs. The suggested value for MaxIQ is 256.
MaxAI	IN	WORD	Parameter MaxAI sets the number of word input (AI) registers available to Modbus address 3xxxx at values of 0 to 56. A value of 0 disables reads of the analog inputs. To allow access to all of the CPU analog inputs, the suggested value for MaxAI is as follows: • 0 for CPU CR40 and CR60 • 56 for all other CPU models
MaxHold	IN	WORD	Parameter MaxHold sets the number of word holding registers in V memory available to Modbus address 4xxxx or 4yyyyy. For example, if you want to allow Modbus client access for 2000 bytes of V memory, set MaxHold to a value of 1000 words (holding registers).
HoldStart	IN	DWORD	Parameter HoldStart is a pointer to the start of the holding registers in V memory. You typically set this value to &VB0 (address of VB0). You can set other V memory addresses as the starting address for the holding registers to allow VB0 to be used elsewhere in the project. The Modbus client has access to MaxHold number of words of V memory starting at HoldStart. If HoldStart points to a memory location that is outside the allowed range, the Modbus TCP library instruction returns an error. The CPU also generates the non-fatal error: Indirect addressing error (0x06).

Parameter and type		Data type	Description
Done	OUT	BOOL	TRUE: MBUS_SERVER performed one of the following actions: • Connected to a client device • Disconnected to a client • Responded to a Modbus request • Returned an error FALSE: No request serviced this program cycle
Error	OUT	BYTE	Instruction execution result Only valid for a single cycle after the error occurred

Opening a connection

When Connect = TRUE, the CPU uses a single passive connection resource from the Open User Communications available connections. Leave the Connect input TRUE while the program is expecting Modbus operations. You can set Connect to FALSE to free up a connection resource. The CPU captures the values of the input parameters when the Modbus server requests a connection. If the input values change while Connect = TRUE, MBUS_SERVER returns an error.

9.4.3.2 Modbus TCP server execution error codes

The MBUS_SERVER instruction (Page 517) can return the following error codes:

Error (decimal)	Description
0	No error
32	Unknown state Check network connections and check that the program is not modifying any library symbols that intefere with client/server communication.
33	Invalid value for input MaxIQ
34	Invalid value for input MaxAI
35	Invalid value for input MaxHold
36	HoldStart input is not in V memory or the range of holding registers exceeds the V memory range
37	Holding registers overlap Modbus server symbols
38	The input values do not match the values for the current connection. Reset the connection to update the input values.

In addition to the MBUS_SERVER errors listed above, refer also to the Modbus TCP general exception codes (Page 528) and Open User Communication error codes (Page 552)

9.4.4 Example: Modbus TCP application

The following example consists of a project with two Modbus TCP clients communication with two Modbus TCP servers. A unique IP Address identifies each server. The program logic monitors the Done outputs of the MBUS_CLIENT instructions to ensure that the program does not interrupt a communication request that is in progress. This example program performs the following functions:

- Write output bits
- Read output bits
- Write holding registers
- Read holding registers

The program, network, and symbol comments describe the functionality of the Modbus TCP example program in the following table.

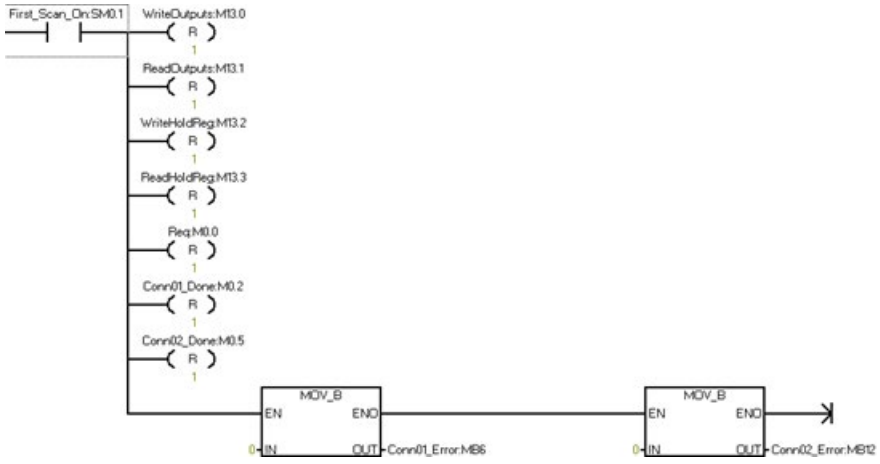
The basic description for this example:

Two Modbus clients establish connections with two Modbus server devices.

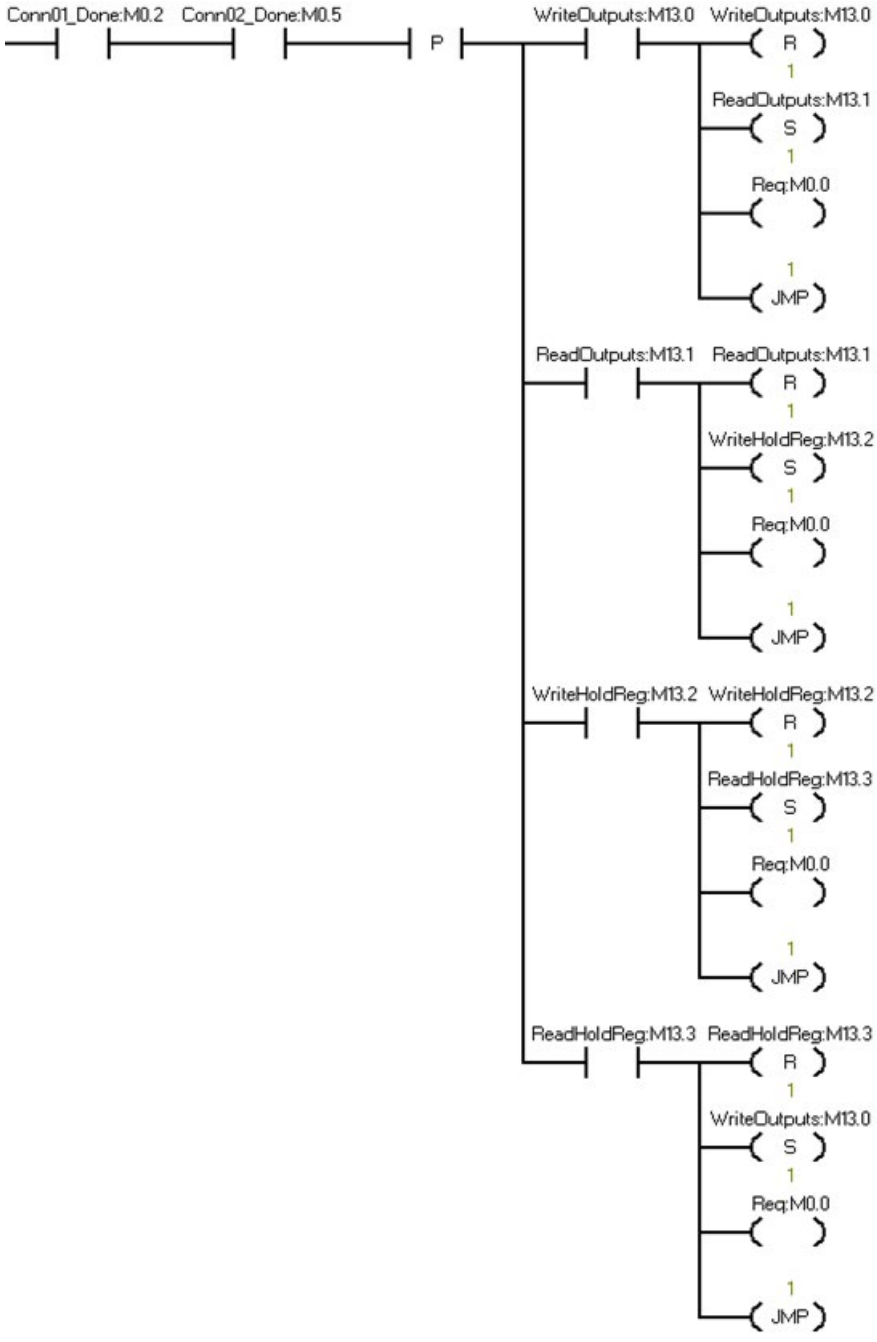
Modbus server 01: IP Address 192.168.2.10, Port 502

Modbus server 02: IP Address 192.168.2.66, Port 502

When CPU Input 0 transitions to TRUE, a sequence starts in which the Modbus client sends write and read requests to both Modbus servers. CPU Input 0 set False turns off the sequence.

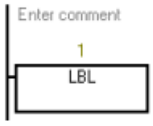
LAD	Description
	Network 1: On startup, clear all the flags and errors.

Symbol	Address	Comment
Conn01_Done	M0.2	Connection #1 has completed the current operation
Conn01_Error	MB6	Connection #1 Status
Conn02_Done	M0.5	Connection #2 has completed the current operation
Conn02_Error	MB12	Connection #2 Status
First_Scan_On	SM0.1	ON for the first scan cycle only
ReadHoldReg	M13.3	Initiate a Read Holding Register request
ReadOutputs	M13.1	Initiate a Read Output request
Req	M0.0	Initiate a Modbus request
WriteHoldReg	M13.2	Initiate a Write Holding Register request
WriteOutputs	M13.0	Initiate a Write Output request

LAD	Description
	Network 2: When both clients complete the Modbus requests, start the next Modbus request.

LAD	Description

Symbol	Address	Comment
Conn01_Done	M0.2	Connection #1 has completed the current operation
Conn02_Done	M0.5	Connection #2 has completed the current operation
ReadHoldReg	M13.3	Initiate a Read Holding Register request
ReadOutputs	M13.1	Initiate a Read Output request
Req	M0.0	Initiate a Modbus request
WriteHoldReg	M13.2	Initiate a Write Holding Register request
WriteOutputs	M13.0	Initiate a Write Output request



Network 3

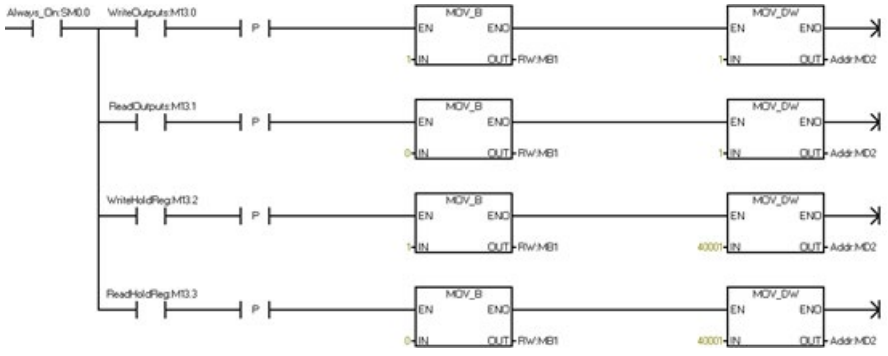
Network 4:

CPU_Input 0 rising edge triggers the Modbus request sequence to begin Write some data in V memory to send to the Modbus servers.

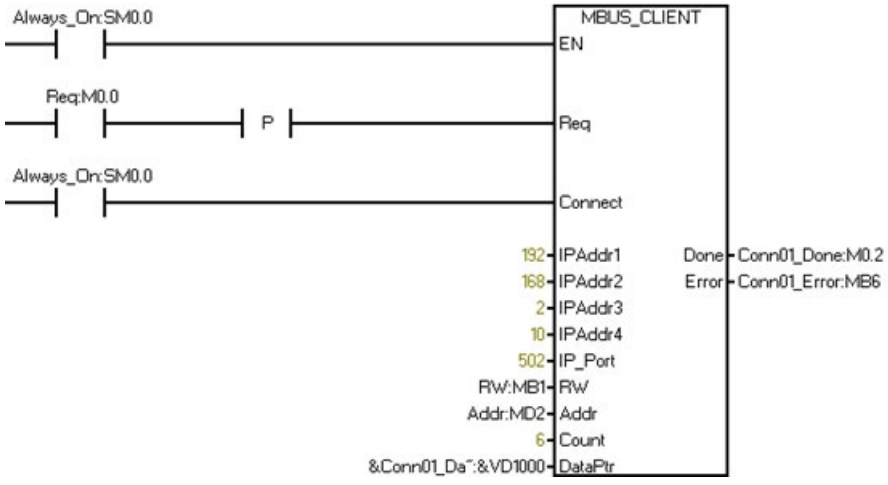
Set the Req input bit TRUE.

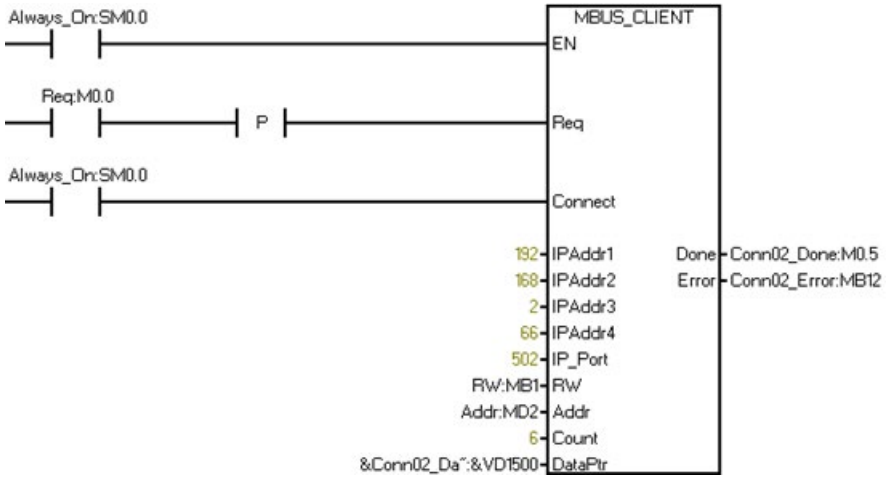
When the CPU_Input 0 is False, stop sending Modbus requests.

Symbol	Address	Comment
Conn01_Data	VD1000	Connection #1 Data Location
Conn02_Data	VD1500	Connection #2 Data Location
CPU_Input0	I0.0	
ReadHoldReg	M13.3	Initiate a Read Holding Register request
ReadOutputs	M13.1	Initiate a Read Output request
Req	M0.0	Initiate a Modbus request
WriteHoldReg	M13.2	Initiate a Write Holding Register request
WriteOutputs	M13.0	Initiate a Write Output request

LAD	Description
	Network 5: Set the Read/Write mode and the address for the selected Modbus function. Write Ouputs = Function Code 5 (single)/ 15 (multiple). Read Outputs = Function Code 1. Write Holding Registers = Function Code 6 (single) / 16 (multiple). Read Holding Registers = Function Code 3.

Symbol	Address	Comment
Addr	MD2	Modbus Address
Always_On	SM0.0	Always ON
ReadHoldReg	M13.3	Initiate a Read Holding Register request
ReadOutputs	M13.1	Initiate a Read Output request
RW	MB1	0 = Read Data; 1 = Write Data
WriteHoldReg	M13.2	Initiate a Write Holding Register request
WriteOutputs	M13.0	Initiate a Write Output request

	Network 6: Modbus client 01 establishes a connection with Modbus server 01. When Req is TRUE, send a Modbus Request to the server. Once the Modbus Client receives and processes the response from the server, the MBUS_CLIENT instruction sets the Done output to TRUE.

LAD	Description
<table><tr><th>Symbol</th><th>Address</th><th>Comment</th></tr><tr><td>Addr</td><td>MD2</td><td>Modbus Address</td></tr><tr><td>Always_On</td><td>SM0.0</td><td>Always ON</td></tr><tr><td>Conn01_Data</td><td>VD1000</td><td>Connection #1 Data Location</td></tr><tr><td>Conn01_Done</td><td>M0.2</td><td>Connection #1 has completed the current operation</td></tr><tr><td>Conn01_Error</td><td>MB6</td><td>Connection #1 Status</td></tr><tr><td>Req</td><td>M0.0</td><td>Initiate a Modbus request</td></tr><tr><td>RW</td><td>MB1</td><td>0 = Read Data; 1 = Write Data</td></tr></table>	
 <table><tr><th>Symbol</th><th>Address</th><th>Comment</th></tr><tr><td>Addr</td><td>MD2</td><td>Modbus Address</td></tr><tr><td>Always_On</td><td>SM0.0</td><td>Always ON</td></tr><tr><td>Conn02_Data</td><td>VD1500</td><td>Connection #2 Data Location</td></tr><tr><td>Conn02_Done</td><td>M0.5</td><td>Connection #2 has completed the current operation</td></tr><tr><td>Conn02_Error</td><td>MB12</td><td>Connection #2 Status</td></tr><tr><td>Req</td><td>M0.0</td><td>Initiate a Modbus request</td></tr><tr><td>RW</td><td>MB1</td><td>0 = Read Data; 1 = Write Data</td></tr></table>	Network 7: Modbus client 02 establishes a connection with Modbus server 02. When Req is TRUE, send a Modbus Request to the server. Once the Modbus Client receives and processes the response from the server, the MBUS_CLIENT instruction sets the Done output to TRUE.

9.4.5 Modbus TCP advanced user information

Overview

This topic contains information for advanced users of the Modbus TCP library. Most users do not need this information and will not need to modify the default operation of the Modbus TCP library.

MBUS_CLIENT variables

The table below shows some Modbus client variables that you can modify in your program to adjust the operation of the Modbus client if the default value is not suitable for your application:

Variable	Data type	Default	Description
mBlocked_Proc_Timeout	REAL	3000	Blocked process timeout: Amount of time (in milliseconds) to wait upon a blocked Modbus client instance before removing this instance as being ACTIVE. This can occur, for example, when the program has issued a client request and the application stops executing the client function before completely finishing the request.
mModbus_Unit_ID	WORD	255	Modbus unit identifier: The mModbus_Unit_ID parameter corresponds to the slave address in the Modbus RTU protocol. If a Modbus TCP server is used for a gateway to a Modbus RTU protocol, the MB_UNIT_ID can be used to identify the slave device connected on the serial network. The MB_UNIT_ID would be used to forward the request to the correct Modbus RTU slave address. Some Modbus TCP devices may require the MB_UNIT_ID parameter to be within a restricted range.
mReceiveTimeout	REAL	2000	Receive message timeout: Time in milliseconds that the MBUS_CLIENT waits for a server to respond to a request. Range: 500 - 65,535 milliseconds.
mConnected	BOOL	FALSE	Connection State: Indicates whether the connection to the assigned server is connected or disconnected: TRUE: Connected FALSE: Disconnected The program can check mConnected after processing an MBUS_CLIENT request.
mRetries	BYTE	3	Retries: Number of times a client attempts to disconnect and resend the request after an initial request returns with a connection error Range: 0 to 255

Retries

The Modbus client instruction automatically restarts the connection and resends the request to the server device if there is a connection-related error:

The Modbus client resends the request two additional times before setting the Done and Error output parameters.

You can change the number of retries by finding the symbol mModbusRetries in the Modbus client symbol table and changing the value before the program executes MBUS_CLIENT. The mRetries value is a BYTE with a range of 0 to 255 retries.

Single vs. multiple bit / word write functions

Some Modbus server devices do not support the Modbus functions to write a single discrete output bit (Modbus function 5) or to write a single holding register (Modbus function 6). These devices only support the multiple bit write (Modbus function 15) or multiple register write (Modbus function 16) instead. The MBUS_CLIENT instruction returns an error code 1 if the server device does not support the single bit/word Modbus functions.

The Modbus client protocol allows you to force the MBUS_CLIENT instruction to use the multiple bit/word Modbus functions instead of the single bit/word Modbus functions. You can force the multiple bit/word instructions by finding the symbol mModbusForceMulti in the Modbus client symbol table and changing this value before the program executes MBUS_CLIENT. Set mModbusForceMulti to TRUE to force the use of the multiple bit/word functions when writing a single bit or register.

Holding register addresses greater than 49999

Modbus holding addresses are within the range of 40001 to 49999. This range is adequate for most applications but there are some Modbus slave devices with data mapped into holding registers at a higher address range.

The MBUS_CLIENT instruction allows an additional range for the parameter **Addr** to support an extended range of holding register addresses at addresses 400001 to 465536.

For example, to access holding register 16768, set the **Addr** parameter of MBUS_CLIENT to 416768.

The extended addressing allows access to the full range of 65536 possible addresses supported by the Modbus protocol. This extended addressing is only applicable for holding registers.

MBUS_SERVER variables

The table below shows some Modbus server variables that you can modify in your program to adjust the operation of the Modbus server if the default value is not suitable for your application:

Variable	Data type	Default	Description
mConnected	BOOL	0	Connection state: Indicates whether the connection to the assigned client is connected or disconnected: TRUE: Connected FALSE: Disconnected The connection state is up to date after every MBUS_SERVER instruction execution.

9.4.6 Modbus TCP general exception codes

Error #	Description
1	Modbus Exception (code 0x01): Illegal Function - Server does not support requested function
2	Modbus Exception (code 0x02): Illegal Data Address - The requested address range of Addr plus Count is outside the allowed address range of the Server
3	Modbus Exception (code 0x03): Illegal Data Value - There is an error in the Modbus protocol received by the Server
4	Modbus Exception (code 0x04): Server Device Failure - An unrecoverable error occurred while the Server was attempting to perform the requested action
5	Modbus Exception (code 0x05): Acknowledge - Response from Server may be delayed; resend the request at a later time
6	Modbus Exception (code 0x06): Server Device Busy - Server rejected the message; resend request
7	Modbus Exception (code 0x07): Negative Acknowledgement - Server rejected the message for an unknown reason
10	Modbus Exception (code 0x0A): Gateway Path Unavailable - Usually means that the gateway is misconfigured or overloaded. (Modbus TCP only)
11	Modbus Exception (code 0x0B): Gateway Target Device Failed to Respond - Usually means that the device is not present on the network. (Modbus TCP only)

9.4.7 Modbus TCP general communication exception codes

The Modbus TCP communication exception codes are as follows:

Error code	Description
161	The data length parameter is greater than the maximum allowed (1024 bytes).
162	The data buffer is not in I, Q, M, or V memory areas.
163	The data buffer does not fit in the memory area.
164	The table parameter does not fit into the memory area.
165	The connection is locked in another context. You are attempting to access the same connection in both the background (the Main) and in an interrupt routine at the same time.
166	A UDP IP address or port error
167	An instance mismatch: The connection is busy with another instance or the input data does not match the data stored for the requested connection ID when the request was initiated.
168	The Connection ID does not exist because the connection has never been created, or the connection was terminated at your request (using the TDCON instruction).
169	A TCON operation is in progress with this Connection ID.
170	A TDCON operation is in progress with this Connection ID.
171	A TSEND instruction is in progress with this Connection ID.
172	A temporary communication error has occurred. The connection cannot be started at this time. Try again later.
173	The connection partner refused or actively dropped the connection (the partner issued a disconnect to this CPU).

Error code	Description
174	The connection partner cannot be reached (no answer to the connect request).
175	The connection aborted due to inconsistencies. Disconnect and reconnect to correct the situation.
176	The Connection ID is already in use with a different IP address, port, or TSAP combination.
177	No connection resource is available. All connections of the requested type (active/passive) are in use.
178	The local or remote port number is reserved or the port number is already in use for another server (passive) connection.
179	One of the following IP address errors have occurred: • The IP address is invalid (for example, address 0.0.0.0). • This IP address is the IP address of this CPU. • This CPU has IP address 0.0.0.0. • The IP address is a broadcast or multicast address.
180	A local or remote TSAP error (ISO-on-TCP only)
181	An invalid connection ID (65535 is reserved)
182	An active/passive error (UDP only allows passive)
183	The connection type is not one of the allowed types.
184	There is no operation pending so there is no status to report.
185	The receive buffer is too small: The CPU received more bytes than the buffer length supports. The CPU discards the extra bytes.
191	Unknown error

9.5 Open user communication library

The STEP 7-Micro/WIN SMART Open User Communication (OUC) library instructions create the tables required by the OUC instructions (Page 208) (TCON, TSEND, TRECV, and TDCON). The library instructions build the table as needed, call the OUC instruction, and then present you the status values on the outputs of the library instruction. The CPU uses library memory to create a table to pass to the OUC instructions. The Open User Communication library requires 50 bytes of V memory.

The library instructions are as follows:

- TCP_CONNECT: Create a TCP connection.
- ISO_CONNECT: Create an ISO-on-TCP connection.
- UDP_CONNECT: Create a UDP connection.
- TCP_SEND: Send data instruction for TCP and ISO-on-TCP connections.
- TCP_RECV: Receive data instruction for TCP and ISO-on-TCP connections.
- UDP_SEND: Send data instruction for UDP connections.
- UDP_RECV: Receive data instruction for UDP connections.
- DISCONNECT: Terminate the connection for all protocols.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

Note

Only call the library functions from either the main program or from interrupt routines, but not both.

9.5.1 Parameters common to the OUC library instructions

The following parameters are common to the OUC library instructions:

- EN: You set the EN input to TRUE to call the instruction. You must set the EN input to TRUE until the instruction completes (until either Done or Error is set). The CPU only updates the outputs when your program sets EN and calls the instruction.
- Req: You use the Req (Request) input to initiate the operation. The Req input bit is level-triggered. You should connect the Req input to the library instruction with a positive edge instruction so that the action is initiated only one time. The program ignores the Req input while the instruction is Busy.
- Active: The Active input commands the connect instructions whether to create an active client connection (Active = TRUE) or a passive server connection (Active = FALSE). An active connection is one in which the local CPU initiates communications to the remote device. A passive connection is one in which the local CPU waits for the remote device to initiate communications.

  The S7-200 SMART CPUs support eight active connections and eight passive connections for Open User Communication. UDP connections are counted as passive connections since there is no active communication establishment.
- Done: The OUC instruction sets the Done output when the operation is complete and there are no errors. If the instruction sets the Done output, Busy, Error, and Status outputs are zero. Other outputs (for example, the number of received bytes) are valid only when the Done output is set.
- Busy: The Busy output indicates that the operation is in progress. The OUC instruction sets the Busy output when you initiate an operation with Req set to TRUE. The Busy output remains set for all subsequent calls to the instruction until the operation is complete.
- Error: The Error output indicates that the operation completed with an error. If the OUC instruction sets the Error output, then the Done and Busy outputs are set to FALSE. If the OUC instruction sets the Error output, the Status output indicates the reason for the error. All other outputs are not valid if the Error output is set.
- ConnID: The ConnID number is the identifier for the connection. You establish the ConnID when you create the connection with TCP_CONNECT, ISO_CONNECT, or UDP_CONNECT. You can pick any value in the range 0 to 65534 for the ConnID. Each

connection must have a unique ConnID. The program uses the ConnID to specify the desired connection for subsequent send, receive, and disconnect operations.

- IPaddr1, IPaddr2, IPaddr3 and IPaddr4: These are the four IP address octets of the remote device. IPaddr1 is the most significant byte of the IP address, and IPaddr4 is the least significant byte of the IP address. For example: For the IP address 192.168.2.15, set these values:
  - IPaddr1 = 192
  - IPaddr2 = 168
  - IPaddr3 = 2
  - IPaddr4 = 15

  The IP address cannot be any of the following values:
  - 0.0.0.0 (for an active connection)
  - Any broadcast IP address (for example, 255.255.255.255)
  - Any multicast address
  - The IP address of the local CPU

  You can use the IP address of 0.0.0.0 for a passive connection. By selecting the IP address of 0.0.0.0, the S7-200 SMART CPU accepts a connection from any remote IP address. Selecting a non-zero IP address for a passive connection causes the CPU to only accept a connection from the specified address.
- RemPort: The RemPort is the port number on the remote device. You use port numbers for TCP and UDP protocols to route the message within the device.

  The rules for remote port numbers are as follows:
  - The valid port number range is 1 to 49151.
  - The suggested range for port numbers is 2000 to 5000.
  - The CPU ignores the remote port number for passive connections (You can set it to zero).
- LocPort: The LocPort parameter is the port number on the local CPU. You use port numbers for TCP and UDP protocols to route the message within the device. The local port number must be unique for all passive connections.

  The rules for local port numbers are as follows:
  - The valid port number range is 1 to 49151.
  - You cannot use port numbers 20, 21, 25, 80, 102, 135, 161, 162, 443, and 34962 to 34964. These ports have specific assignments.
  - The suggested range for port numbers is 2000 to 5000.
  - The local port numbers must be unique for passive connections (no duplicates).

- RemTsap: The RemTsap (remote Transport Service Access Point (TSAP)) parameter is a pointer to an S7-200 SMART string data type. You can only use the RemTsap parameter for the ISO-on-TCP protocol. The remote TSAP string serves the same purpose as a port number in routing the message to the proper connection.

  The rules for the RemTsap are as follows:

  – The TSAP is an S7-200 SMART string data type (a length byte followed by the characters).

  – The TSAP string must be at least 2 characters and no more than 16 characters.

- LocTsap: The LocTsap (local Transport Service Access Point (TSAP)) parameter is a pointer to an S7-200 SMART string data type. You can only use the local TSAP parameter for the ISO-on-TCP protocol. The local TSAP string serves the same purpose as a port number in routing the message to the proper connection.

  The rules for the LocTsap are as follows:

  – The TSAP is an S7-200 SMART string data type (a length byte followed by the characters).

  – The TSAP string must be at least 2 characters and no more than 16 characters.

  – If the TSAP is 2 characters, the first character must be a hexadecimal "E0".

  – The TSAP cannot start with the string "SIMATIC-".

9.5.2 Open user communication library instructions

9.5.2.1 TCP_CONNECT instruction

The TCP_CONNECT instruction creates a connection to another device using the TCP protocol.

LAD/FBD	STL	Description
	TCP_CONNECT Req, Active, ConnID, IPaddr1, IPaddr2, IPaddr3, IPaddr4, RemPort, LocPort, Done, Busy, Error, Status	The TCP_CONNECT creates a TCP communications connection from the CPU to a communication partner.

The connect operation is asynchronous and can take several scans to complete. The Busy output has a value of TRUE while the connect operation is pending. When the CPU completes the operation, the instruction sets either the Done or Error output. If there is an error, the Status output contains the error code.

You must not change the input parameters of the TCP_CONNECT while the instruction is busy. The CPU requires this so that it knows that this is a continuation of the call that started the connection process.

You assign the connection ID (ConnID) input to the connection and then use this ConnID to reference this connection when sending, receiving, or disconnecting.

The Active input bit determines whether this is an active connection (Active set to TRUE) or a passive connection (Active set to FALSE).

If this is an active connection (a client), then the S7-200 SMART CPU attempts to contact and create a connection to the specified IP address and remote port number (RemPort). The CPU opens a local port (LocPort) to receive messages from the remote device.

When the Active input is set to FALSE, the S7-200 SMART CPU creates a passive (server) connection. In this case, the CPU opens the requested local port (LocPort) and accepts connection requests from a remote device. You should set the IP address to 0.0.0.0 if you wish to accept a connection request from any remote IP address. If the IP address is non-zero, the CPU accepts a connection request only from the specified IP address. The CPU ignores the remote port number (RemPort) for a passive connection, and RemPort can be set to zero.

You can call the TCP_CONNECT instruction anytime to determine the current status of a connection. Set the Req input to FALSE and provide a valid connection ID (ConnID), and TCP_CONNECT returns the following:

- Busy if the connect process is still progressing.
- Done if the connection is active and ready to send or receive.
- Error if the connection is not usable. Status contains one of the error codes to specify the problem.

Note that active connections can require up to 30 seconds to determine if the remote device will allow the connection or not. Passive connections show a Busy status until a remote device attempts to connect to the CPU.

Note that the S7-200 SMART does not automatically attempt to reconnect to a device after the connection has been closed. If the remote device breaks the device connection, your program must execute another TCP_CONNECT instruction to reconnect the device. This is true for both active and passive connections.

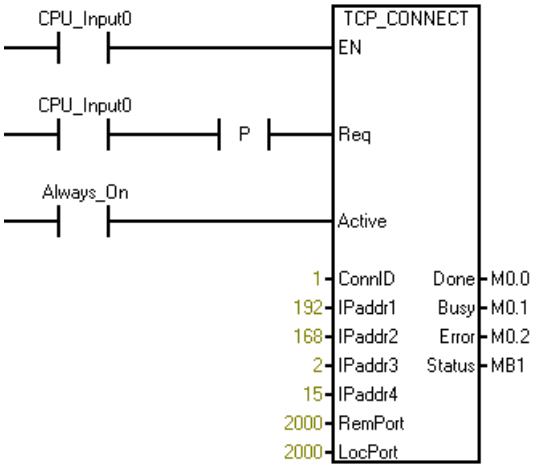
Table 9- 18 Parameters of the TCP_CONNECT instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
Req	IN	BOOL	The CPU starts the connect operation if Req = TRUE. If Req = FALSE, then the outputs show the current status of the connection.
Active	IN	BOOL	• TRUE = active connection • FALSE = passive connection

Parameter	Declaration	Data type	Description
ConnID	IN	WORD	The CPU uses the Connection ID (ConnID) number to identify this connection for the other instructions. The possible ConnID range is 0 to 65534.
IPaddr1 ... IPaddr4	IN	BYTE	These are the four IP address octets. IPaddr1 is the most significant byte and IPaddr4 is the least significant byte of the IP address.
RemPort	IN	WORD	The RemPort is the port number on the remote device. The remote port number range is 1 to 49151. Use zero for passive connections.
LocPort	IN	WORD	The LocPort is the port number on the local device. The local port number range is 1 to 49151 with some restrictions. Refer to the LocPort definition in "Parameters common to the OUC library instructions" (Page 530).
Done	OUT	BOOL	The instruction sets the Done output when the connect operation is complete with no errors.
Busy	OUT	BOOL	The instruction sets the Busy output while the connection operation is in progress.
Error	OUT	BOOL	The instruction sets the Error output when the connection operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.

Example

This is an example usage of the TCP_CONNECT instruction:



9.5.2.2 ISO_CONNECT instruction

The ISO_CONNECT instruction creates a connection to another device using the ISO-on-TCP protocol. This protocol uses RFC1006 in addition to the TCP protocol to better delineate messages. The advantage of ISO-on-TCP is that every sent message results in a different received message. The ISO-on-TCP protocol never combines multiple received messages into one message, as can happen with TCP protocol. The ISO-on-TCP protocol uses TSAPs (Transport Services Access Point) to route the message in the device instead of ports.

LAD/FBD	STL	Description
	ISO_CONNECT Req, Active, ConnID, IPaddr1, IPaddr2, IPaddr3, IPaddr4, RemTsap, LocTsap, Done, Busy, Error, Status	The ISO_CONNECT creates an ISO-on-TCP communications connection from the CPU to a communication partner.

The connect operation is asynchronous and can take several scans to complete. The instruction sets the Busy output while the connect operation is pending. When the CPU completes the operation, the instruction sets either the Done or Error output. If there is an error, the Status output contains the error code.

You must not change the input parameters of the ISO_CONNECT while the instruction is busy. The CPU requires this so that it knows that this is a continuation of the call that started the connection process.

You assign the connection ID (ConnID) input to the connection and then use this ConnID to reference this connection when sending, receiving, or disconnecting.

The Active input bit determines whether this is an active connection (Active set to TRUE) or a passive connection (Active set to FALSE).

If this is an active connection (a client), then the S7-200 SMART CPU attempts to contact and create a connection to the specified IP address and remote TSAP (RemTsap). The CPU opens a local TSAP (LocTsap) to receive messages from the remote device.

When the Active input is FALSE, the S7-200 SMART CPU creates a passive (server) connection. In this case, the CPU opens the requested local TSAP (LocTsap) and accepts connection requests from a remote device. You should set the IP address to 0.0.0.0 if you wish to accept a connection request from any remote IP address. If the IP address is non-zero, the CPU accepts a connection request only from the specified IP address. The CPU ignores the remote TSAP string (RemTsap) for passive connections, and the RemTsap can be set to an empty string (for example, "").

You can call the ISO_CONNECT instruction anytime to determine the current status of a connection. Set the Req input to FALSE and provide a valid connection ID (ConnID), and ISO_CONNECT returns the following:

- Busy if the connect process is still progressing.
- Done if the connection is active and ready to send or receive.
- Error if the connection is not usable. Status contains one of the error codes to specify the problem.

Note that active connections can require up to 30 seconds to determine if the remote device will allow the connection or not. Passive connections show a Busy status until a remote device attempts to connect to the CPU.

Note that the S7-200 SMART does not automatically attempt to reconnect to a device after the connection has been closed. If the remote device breaks the device connection, your program must execute another ISO_CONNECT instruction to reconnect the device. This is true for both active and passive connections.

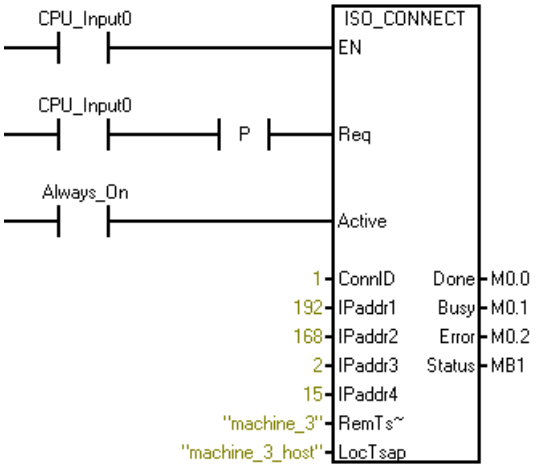
Table 9- 19 Parameters of the ISO_CONNECT instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
Req	IN	BOOL	The CPU starts the connect operation if Req = TRUE. If Req = FALSE, then the outputs show the current status of the connection.
Active	IN	BOOL	• TRUE = active connection • FALSE = passive connection
ConnID	IN	WORD	The CPU uses the Connection ID (ConnID) number to identify this connection for the other instructions. The possible ConnID range is 0 to 65534.
IPaddr1 ... IPaddr4	IN	BYTE	These are the four IP address octets. IPaddr1 is the most significant byte and IPaddr4 is the least significant byte of the IP address.
RemTsap	IN	DWORD	The RemPort is the remote TSAP string. The program uses a pointer to pass the string. (Refer to the example following this table for more information.)
LocTsap	IN	DWORD	The LocPort is the local TSAP string. The program uses a pointer to pass the string. (Refer to the example following this table for more information.)
Done	OUT	BOOL	The instruction sets the Done output when the connect operation is complete with no errors.
Busy	OUT	BOOL	The instruction sets the Busy output while the connection operation is in progress.

Parameter	Declaration	Data type	Description
Error	OUT	BOOL	The instruction sets the Error output when the connection operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.

Example

This is an example usage of the ISO_CONNECT instruction:

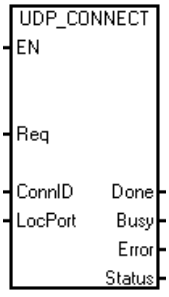


STEP7-Micro/WIN SMART always uses pointers to pass a string to the ISO_CONNECT instruction for the RemTsap and the LocTsap. If you use a constant string (as in the example above), STEP7-Micro/WIN SMART automatically creates the string and the pointer. If you wish to create strings in a Data Block and then pass a pointer to one of these strings, you would follow these steps:

1. In the Data Block, create the strings:
   - VB100 "machine_1"
   - VB120 "machine_2"
2. Use "&VB100" or "&VB120" (without the quote marks) for the TSAP parameters on the ISO_CONNECT instruction.

9.5.2.3 UDP_CONNECT instruction

The UDP_CONNECT instruction creates a passive connection using UDP protocol. UDP is a connectionless protocol so there is no actual connection created between this CPU and the remote device. The UDP connection opens the selected local port to use with UDP protocol.

LAD/FBD	STL	Description
	**UDP_CONNECT Req, ConnID, LocPort, Done, Busy, Error, Status**	The UDP_CONNECT creates a passive connection using UDP protocol to open a selected local port.

The UDP_CONNECT instruction only requires a connection ID and a local port number to create the connection. One UDP connection can send messages to any number of other devices since the IP address and remote port are supplied with each UDP_SEND instruction. You will need multiple UDP connections only if you require multiple local ports. You cannot use the same local port number for multiple UDP connections. All local port numbers must be unique.

The connect operation is asynchronous and can take several scans to complete. There is no active connection establishment with the remote device, nor are we waiting for another device to connect to this CPU. The Busy output is set while the connect operation is pending. When the connect operation is complete, the program sets the Done output. The program sets the Error output only if there is a problem with the input parameters or there is no passive connection available. The Status output byte contains the error code if the program sets the Error bit.

You must not change the parameters of the UDP_CONNECT while the instruction is busy so that the CPU knows this is a continuation of the call that started the connection process.

You can call the UDP_CONNECT instruction to determine the current status of a connection. Set the Req input FALSE and provide a valid connection ID (ConnID), and the UDP_CONNECT instruction returns the following:

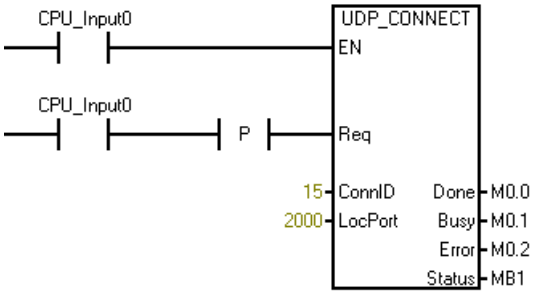
- The instruction sets the Done output if the connection is active and ready to send or receive. This only means that you can use the connection. This is not an indication that there is any remote device present.
- The instruction sets the Busy output if the connect is still processing.
- The instruction sets the Error output if the connection is not usable. The Status output byte contains one of the error codes to specify the problem.

Table 9- 20 Parameters of the UDP_CONNECT instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
Req	IN	BOOL	The CPU starts the connect operation if Req = TRUE. If Req = FALSE, then the outputs show the current status of the connection.
ConnID	IN	WORD	The CPU uses the Connection ID (ConnID) number to identify this connection for the other instructions. The possible ConnID range is 0 to 65534.
LocPort	IN	WORD	The LocPort is the port number on the local device. The local port number range is 1 to 49151 with some restrictions. Refer to the LocPort definition in "Parameters common to the OUC library instructions" (Page 530).
Done	OUT	BOOL	The instruction sets the Done output when the connect operation is complete with no errors.
Busy	OUT	BOOL	The instruction sets the Busy output while the connection operation is in progress.
Error	OUT	BOOL	The instruction sets the Error output when the connection operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.

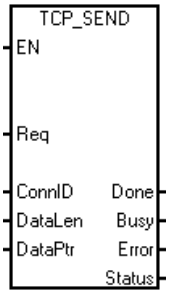
Example

This is an example usage of the UDP_CONNECT instruction:



9.5.2.4 TCP_SEND instruction

The TCP_SEND instruction transmits the requested number of bytes (DataLen) from the requested buffer location (DataPtr) over the existing connection (ConnID). You use the instruction for both TCP protocol and for ISO-on-TCP protocol.

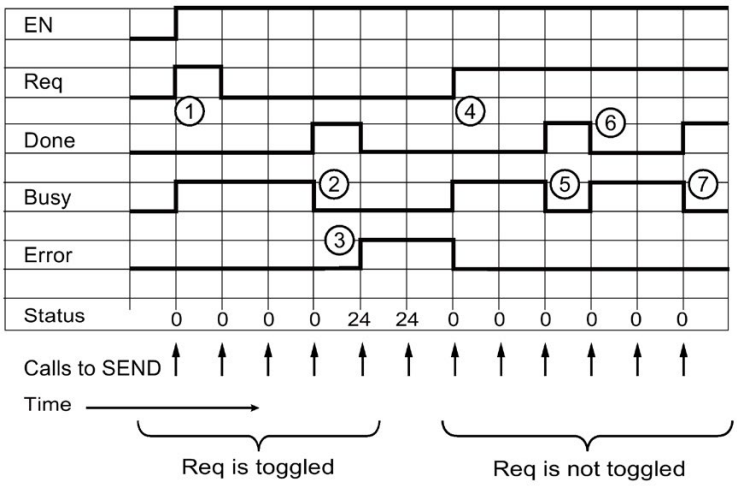
LAD/FBD	STL	Description
	**TCP_SEND Req, ConnID, DataLen, DataPtr, Done, Busy, Error, Status**	The TCP_SEND transmits the requested number of bytes from the requested buffer location over an existing connection.

The TCP_SEND instruction initiates sending the specified number of bytes when the following occur:

- The program calls the instruction with the Req input set to TRUE.
- The connection is not currently busy with another send operation.

The Req input is level-triggered. It is recommended that you put a positive edge trigger on the Req input so that the instruction does not initiate unintended send operations. The program ignores the Req input while the TCP_SEND is busy. The Done, Busy, and Error outputs and the Status output byte show the status of the TCP_SEND for each call.

The instruction displays the Done or Error status for one call of TCP_SEND after the send operation is complete. After that, the TCP_SEND responds with error code 24, which means no operation pending, if called with the Req input set to FALSE. If the Req input is left set to TRUE, the program initiates another send operation. The figure below shows the relationship of the input and output parameters.



① Req is set TRUE so that the message send begins. Busy is set TRUE.

② The message send is complete. Done is set, and Busy is cleared.

③ EN is TRUE, and Req is FALSE, but no message send is in progress. So, Error is set with error code 24.

④ Req is set TRUE again, so another message send begins. Busy is set TRUE.

⑤ The message send is complete. Done is set, and Busy is cleared for one scan.

⑥ Req remains TRUE, so another message send begins.

⑦ The message send is complete.

The maximum amount of data that you can send in one send operation is 1024 bytes. The program copies the data from the send buffer in user memory to an internal buffer when the TCP_SEND executes with the Req input set TRUE. You can change the program send buffer after the TCP_SEND executes and the instruction sets the Busy output.

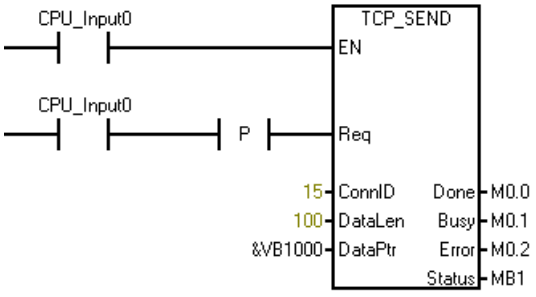
Table 9- 21 Parameters of the TCP_SEND instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
Req	IN	BOOL	The CPU starts the send operation if Req = TRUE. If Req = FALSE, then the outputs show the current status of the send operation.
ConnID	IN	WORD	The Connection ID (ConnID) is the number of the connection for this send operation. Use the ConnID that you selected for the TCP_CONNECT operation.
DataLen	IN	WORD	The DataLen is the number of bytes to transmit (1 to 1024).
DataPtr	IN	DWORD	The DataPtr is the pointer to the data to be sent. This is an S7-200 SMART pointer to I, Q, M, or V memory (for example, &VB100).
Done	OUT	BOOL	The instruction sets the Done output when the send operation is complete with no errors.
Busy	OUT	BOOL	The instruction sets the Busy output while the send operation is in progress.

Parameter	Declaration	Data type	Description
Error	OUT	BOOL	The instruction sets the Error output when the send operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.

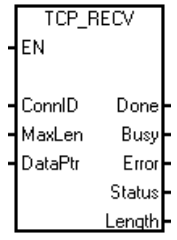
Example

This is an example usage of the TCP_SEND instruction:



9.5.2.5 TCP_RECV instruction

The TCP_RECV instruction retrieves data over an existing connection. You use this instruction for both TCP protocol and ISO-on-TCP protocol.

LAD/FBD	STL	Description
	`TCP_RECV ConnID, MaxLen, DataPtr, Done, Busy, Error, Status, Length`	The TCP_RECV retrieves data over an existing connection.

The TCP_RECV instruction only has an EN (Enable) input. The TCP_RECV instruction has no Req (Request) input. After the first execution of the TCP_RECV instruction, the status bits show the instruction is busy. Subsequent calls to TCP_RECV show a busy status until the CPU receives data on the specified connection.

After the CPU receives a message on the specified connection, the next execution of the TCP_RECV instruction performs the following tasks:

- Copies the message data to your program's data area (DataPtr)
- Sets the Length output to the number of bytes received
- Sets the Done output, clears the Busy and Error outputs, and sets the Status output byte value to zero (no error)

You should assign the receive area/buffer (DataPtr) and the maximum length of the receive buffer (MaxLen) so there is no possibility of a buffer overrun. If the CPU receives more bytes than can fit into your program's buffer (as specified by MaxLen), the TCP_RECV instruction copies MaxLen bytes to your program's data area and discards the rest of the received bytes. In this situation, the instruction sets the Error output and the Status output byte displays error code 25, which means the receive buffer is too small.

The maximum amount of data that you can receive in one message is 1024 bytes. The TCP_RECV instruction always operates in a mode that allows receipt of messages of varying lengths.

The TCP_RECV instruction operates differently depending on the protocol used. You selected the protocol for the connection when you called either TCP_CONNECT (TCP protocol) or ISO_CONNECT (ISO-on-TCP protocol) to create the connection.

Using TCP protocol, the TCP_RECV instruction returns all the bytes received by the S7-200 SMART CPU on the specified connection since the last time your program called the TCP_RECV instruction. Your program must call the TCP_RECV instruction often enough to delineate the messages properly because TCP acts as a "streaming" protocol. There is no delineation of the messages (no begin or end markers) in TCP protocol. Therefore, the CPU does not know when messages start or end.

For example, let us suppose that there is a TCP client that sends four 20-byte messages to the CPU in rapid succession, and your program does not call the TCP_RECV instruction during this time. When the program does call the TCP_RECV instruction after all four messages have been accepted by the CPU, the TCP_RECV instruction returns this as one receive message of 80 bytes. Your program is responsible for calling the TCP_RECV instruction often enough to receive each message as it is sent.

When you create a connection using the ISO-on-TCP protocol, the protocol itself delineates the messages. The TCP_RECV instruction receives and holds all messages sent from the remote device as separate messages in the CPU, no matter when or how often the program calls the TCP_RECV instruction.

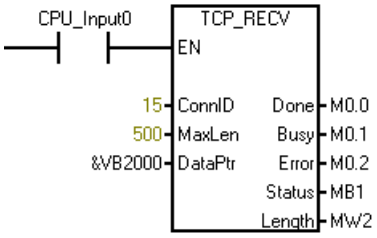
For example, let us suppose this time there is an ISO-on-TCP client that sends four 20-byte messages to the CPU in rapid succession. Assume also that your program does not call the TCP_RECV instruction during this time. The ISO-on-TCP protocol delivers the four messages during four subsequent calls to the TCP_RECV instruction (one message per call). This happens because ISO-on-TCP has start and end markers in the protocol to delineate the messages and separate them in the receiving device.

Table 9- 22 Parameters of the TCP_RECV instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
ConnID	IN	WORD	The Connection ID (ConnID) is the number of the connection for this receive operation (defined during the connect process).
MaxLen	IN	WORD	The MaxLen is the maximum number of bytes to accept (for example, the size of the buffer at DataPtr (1 to 1024)).
DataPtr	IN	WORD	The DataPtr is the pointer to where the receive data should be stored. This is an S7-200 SMART pointer to I, Q, M, or V memory (for example, &VB100).
Done	OUT	BOOL	The instruction sets the Done output when the receive operation is complete with no errors. When the instruction sets the Done output, the Length output is valid.
Busy	OUT	BOOL	The instruction sets the Busy output while the receive operation is in progress.
Error	OUT	BOOL	The instruction sets the Error output when the receive operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.
Length	OUT	WORD	The Length is the actual number of bytes received. Only when the instruction sets the Done or Error outputs is the Length valid. If the instruction sets the Done output, then the instruction received the entire message. If the instruction sets the Error output, the message was greater than the buffer size (MaxLen) and has been truncated.

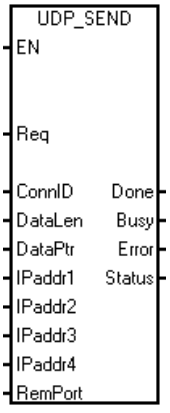
Example

This is an example usage of the TCP_RECV instruction:



9.5.2.6 UDP_SEND instruction

The UDP_SEND instruction transmits the requested number of bytes (DataLen) from the requested buffer location (DataPtr) to the device specified by the IP address (IPaddr1 – IPaddr4) and port (RemPort). This instruction is used only for UDP protocol and connections created with UDP_CONNECT.

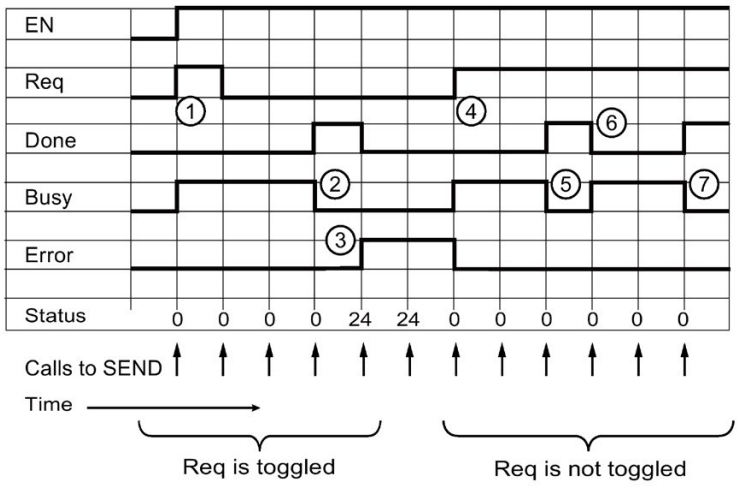
LAD/FBD	STL	Description
	**UDP_SEND Req, ConnID, DataLen, DataPtr, IPaddr1, IPaddr2, IPaddr3, IPaddr4, RemPort, Done, Busy, Error, Status**	The UDP_SEND instruction transmits the requested number of bytes from the requested buffer location to the device specified by the IP address and port.

The UDP_SEND instruction initiates sending the specified number of bytes when the following occur:

- The program calls the instruction with the Req input set to TRUE.
- The connection is not currently busy with another send operation.

The Req input is level-triggered. It is recommended that you put a positive edge trigger on the Req input so that the instruction does not initiate unintended send operations. The program ignores the Req input while the UDP_SEND is busy. The Done, Busy, and Error outputs and the Status output byte show the status of the UDP_SEND for each call.

The instruction displays the Done or Error status for one call of UDP_SEND after the send operation is complete. After that, the UDP_SEND responds with error code 24, which means no operation pending, if called with the Req input set FALSE. If the Req input is left set to TRUE, the program initiates another send operation. The figure below shows the relationship of the input and output parameters.



① Req is set TRUE so that the message send begins. Busy is set TRUE.

② The message send is complete. Done is set, and Busy is cleared.

③ EN is TRUE, and Req is FALSE, but no message send is in progress. So, Error is set with error code 24.

④ Req is set TRUE again, so another message send begins. Busy is set TRUE.

⑤ The message send is complete. Done is set, and Busy is cleared for one scan.

⑥ Req remains TRUE, so another message send begins.

⑦ The message send is complete.

The maximum amount of data that you can send in one send operation is 1024 bytes. The program copies the data from the send buffer in user memory to an internal buffer when the UDP_SEND executes with the Req input set to TRUE. You can change the program send buffer after the UDP_SEND executes and the instruction sets the Busy output.

The UDP_SEND instruction requires the IP address and port number on the remote device. When the UDP_CONNECT created the connection, the local port was set. The IP address (IPaddrx) and the remote port number (RemPort) are subject to the same rules and restrictions as described earlier. (Refer to "Parameters common to the OUC library instructions" (Page 530) for these rules.)

Note that UDP messages are not guaranteed to be delivered. There is no error returned if the remote device is not present.

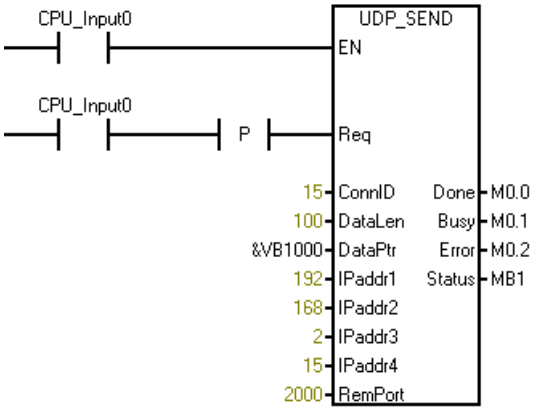
Table 9- 23 Parameters of the UDP_SEND instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
Req	IN	BOOL	The CPU starts the send operation if Req = TRUE. If Req = FALSE, then the outputs show the current status of the send operation.
ConnID	IN	WORD	The Connection ID (ConnID) is the number of the connection for this send operation (defined during the connect process with UDP_CONNECT).
DataLen	IN	WORD	The DataLen is the number of bytes to transmit (1 to 1024).

Parameter	Declaration	Data type	Description
DataPtr	IN	DWORD	The DataPtr is the pointer to the data to be sent. This is an S7-200 SMART pointer to I, Q, M, or V memory (for example, &VB100).
IPaddr1 ... IPaddr4	IN	BYTE	These are the four IP address octets. IPaddr1 is the most significant byte and IPaddr4 is the least significant byte of the IP address.
RemPort	IN	WORD	The RemPort is the port number on the remote device. The remote port number range is 1 to 49151.
Done	OUT	BOOL	The instruction sets the Done output when the connect operation is complete with no errors.
Busy	OUT	BOOL	The instruction sets the Busy output while the connection operation is in progress.
Error	OUT	BOOL	The instruction sets the Error output when the connection operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.

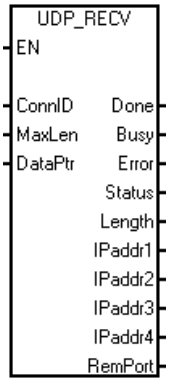
Example

This is an example usage of the UDP_SEND instruction:



9.5.2.7 UDP_RECV instruction

The UDP_RECV instruction retrieves data over an existing connection. This instruction is used only for UDP protocol on connections created with UDP_CONNECT.

LAD/FBD	STL	Description
	**UDP_RECV ConnID, MaxLen, DataPtr, Done, Busy, Error, Status, Length, IPaddr1, IPaddr2, IPaddr3, IPaddr4, RemPort**	The UDP_RECV retrieves data over an existing connection.

The UDP_RECV instruction only has an EN (Enable) input. The UDP_RECV instruction has no Req (Request) input. After the first execution of the UDP_RECV instruction, the status outputs show the instruction is busy. Subsequent calls to UDP_RECV show a busy status until the CPU receives data on the specified connection.

After the CPU receives a message on the specified connection, the next execution of the UDP_RECV instruction performs the following tasks:

- Copies the message data to your program's data area (DataPtr)
- Sets the returned Length to the number of bytes received
- Sets the IP address to the remote device that sent the message
- Set the remote port number (RemPort) to the remote device’s port
- Sets the Done output, clears the Busy and Error outputs, and sets the Status output byte value to zero (no error)

You should assign the receive area/buffer (DataPtr) and the maximum length of the receive buffer (MaxLen) so there is no possibility of a buffer overrun. If the CPU receives more bytes than can fit into your program's buffer (as specified by MaxLen), the UDP_RECV instruction copies MaxLen bytes to your program's data area and discards the rest of the received bytes. In this situation, the instruction sets the Error output and the Status output byte displays error code 25, which means the receive buffer is too small.

The maximum amount of data that you can receive in one message is 1024 bytes. The UDP_RECV instruction always operates in a mode that allows receipt of messages of varying lengths. Each message from a remote device is delineated as a separate message in the S7-200 SMART CPU. The UDP_RECV instruction only returns one received message for each call of the instruction.

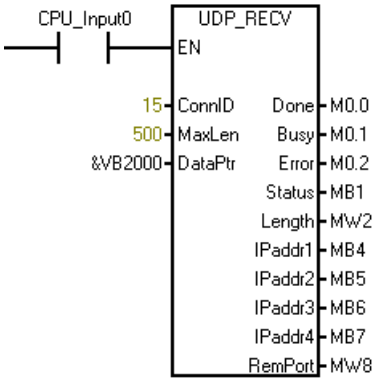
The UDP_RECV instruction also returns the IP address and port number of the remote device. This allows your program to respond to the remote device with a UDP_SEND where the remote IP address and port are required parameters.

Table 9- 24 Parameters of the UDP_RECV instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
ConnID	IN	WORD	The CPU uses the Connection ID (ConnID) number for this receive operation (defined during the connect process).
MaxLen	IN	WORD	The MaxLen is the maximum number of bytes to accept (for example, the size of the buffer at DataPt (1 to 1024)).
DataPtr	IN	DWORD	The DataPtr is the pointer to where the receive data should be stored. This is an S7-200 SMART pointer to I, Q, M, or V memory (for example, &VB100).
Done	OUT	BOOL	The instruction sets the Done output when the receive operation is complete with no errors. When the instruction sets the Done output, the Length output is valid.
Busy	OUT	BOOL	The instruction sets the Busy output while the receive operation is in progress.
Error	OUT	BOOL	The instruction sets the Error output when the receive operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.
Length	OUT	WORD	The Length is the actual number of bytes received. Only when the instruction sets the Done or Error outputs is the Length valid. If the instruction sets the Done output, then the instruction received the entire message. If the instruction sets the Error output, the message was greater than the buffer size (MaxLen) and has been truncated.
IPaddr1 ... IPaddr4	OUT	BYTE	These are the four IP address octets of the remote device that sent the message. IPaddr1 is the most significant byte and IPaddr4 is the least significant byte of the IP address.
RemPort	OUT	WORD	The RemPort is the port number of the remote device that sent the message.

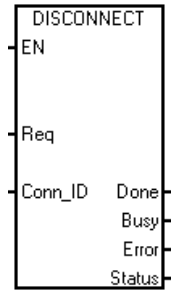
Example

This is an example usage of the UDP_RECV instruction:



9.5.2.8 DISCONNECT instruction

The DISCONNECT instruction terminates an existing communication connection. The DISCONNECT instruction is used for all protocols.

LAD/FBD	STL	Description
	`DISCONNECT ConnID, LocPort, Done, Busy, Error, Status`	The DISCONNECT terminates an existing communication connection for all protocols.

The DISCONNECT instruction initiates the connection termination when the program calls the DISCONNECT instruction with the Req input set TRUE. The Req input is level-triggered. It is recommended that you put a positive edge trigger on the Req input.

If the requested connection (ConnID) is currently busy connecting, disconnecting, or cannot be found because the connection has been reused, the DISCONNECT instruction returns an error.

The DISCONNECT instruction displays the Done or Error output status for at least one call of the instruction after the disconnect operation is complete. The instruction may return the status of the DISCONNECT instruction for the specified connection for subsequent calls until the CPU reuses the connection for another connect operation.

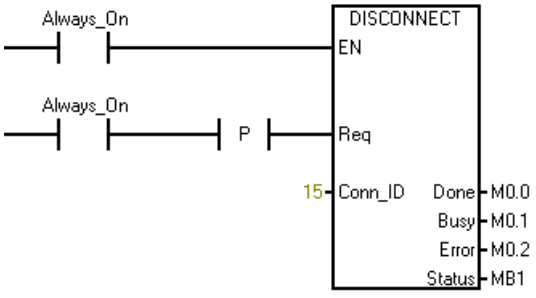
After the DISCONNECT instruction completes a disconnect, if the program calls the DISCONNECT instruction with Req set FALSE, the instruction returns error code 24, which means no operation pending.

Table 9- 25 Parameters of the DISCONNECT instruction

Parameter	Declaration	Data type	Description
EN	IN	BOOL	Enable input
Req	IN	BOOL	The CPU starts the disconnect operation if Req = TRUE.
ConnID	IN	WORD	The CPU uses the Connection ID (ConnID) number to identify the connection to be terminated (defined during the connect process).
Done	OUT	BOOL	The instruction sets the Done output when the disconnect operation is complete with no errors.
Busy	OUT	BOOL	The instruction sets the Busy output while the disconnect operation is in progress.
Error	OUT	BOOL	The instruction sets the Error output when the disconnect operation is complete with an error. Refer to "Open user communication library instruction error codes" (Page 552) for further information.
Status	OUT	BYTE	The Status output shows the error code if the instruction sets the Error output. Status is zero (no error) if the instruction sets the Busy or Done outputs.

Exampled

This is an example usage of the DISCONNECT instruction:



9.5.3 Open user communication library instruction error codes

The following table lists the Open User Communication (OUC) library (Page 529) instruction error codes:

Error code	Description	CONNECT	SEND	RECV	DISCONNECT
0	No error	X	X	X	X
1	The data length input parameter is greater than the maximum allowed (1024 bytes).		X	X	
2	The data buffer is not in I, Q, M, or V memory areas.		X	X	
3	The data buffer does not fit in the memory area.		X	X	
5	The connection is locked in another context. You are attempting to access the same connection in both the background (the Main) and in an interrupt program organizational unit (POU) at the same time.	X	X	X	X
6	A UDP IP address or port error		X		
7	An instance mismatch: The connection is busy with another instance or the input data does not match the data stored for the requested connection ID when the request was initiated.	X	X	X	X
8	The Connection ID does not exist because the connection has never been created, or the connection was terminated by your program using the DISCONNECT instruction.	X	X	X	X
9	A TCP_CONNECT, ISO_CONNECT, or UDP_CONNECT instruction is in progress with this Connection ID.		X	X	X
10	A DISCONNECT instruction is in progress with this Connection ID.	X	X	X	
11	A TCP_SEND or UDP_SEND instruction is in progress with this Connection ID.		X		X
12	A temporary communication error has occurred. The connection cannot be started at this time. Try again.	X	X	X	
13	The connection partner refused or actively dropped the connection. The partner issued a disconnect to this CPU.	X	X	X	
14	The CPU cannot reach the connection partner. There was no answer to the connect request.	X	X	X	
15	The CPU aborted the connection due to inconsistencies. Disconnect and reconnect to correct the situation.	X	X	X	X
16	The Connection ID is already in use with a different IP address, port, or TSAP combination.	X			

Error code	Description	CONNECT	SEND	RECV	DISCONNECT
17	No connection resource is available. All connections of the requested type (active/passive) are in use.	X			
18	The local or remote port number is reserved or the port number is already in use for another server (passive) connection.	X			
19	One of the following IP address errors have occurred: • The IP address is invalid (for example, address 0.0.0.0). • This IP address is the IP address of this CPU. • This CPU has IP address 0.0.0.0. • The IP address is a broadcast or multicast address.	X			
20	A local or remote TSAP error (ISO-on-TCP only)	X			
21	An invalid connection ID (65535 is reserved)	X			
24	There is no operation pending so there is no status to report.		X		X
25	The receive buffer is too small: The CPU received more bytes than the buffer length supports. The CPU discards the extra bytes.			X	
31	Unknown error. Disconnect and reconnect to fix the problem.	X	X	X	X

9.5.4 Open user communication library example

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

9.5.4.1 Active partner (client)

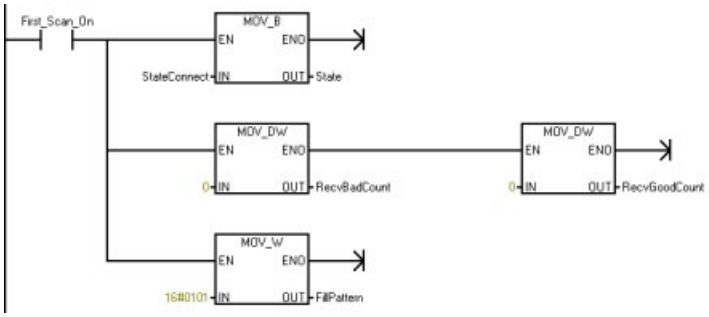
This program implements a simple state machine to manage the creating of a connection, cyclically sending/receiving a message, and handling errors.

The flow of this state machine is to connect, then repeatedly send and receive a message. If the connection is dropped, the state machine tries to reconnect.

Refer to the "Active partner symbol table" (Page 563) to see the symbol table for this program.

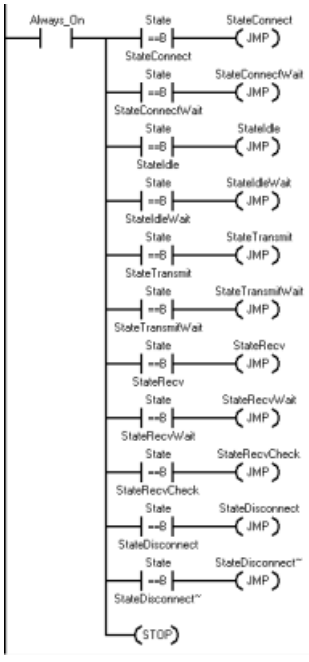
Network 1: On the first scan....

Initialize the State variable to initiate a connection. Clear the good and bad receive counts, and initialize some data to send.



Network 2: Process the state machine...

Determine the current state of the state machine and jump to the label for the state handler. If the state is ever illegal, the CPU goes to the STOP mode.

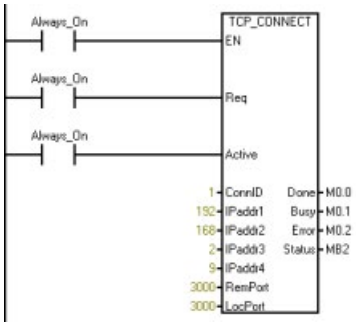


Network 3: State "Connect"...

Start the connection process.

Network 4: Create an active connection to the passive device.

The program only calls the TCP_CONNECT instruction one time in this state. Set the Req input to TRUE to start the connection process. Since this is the active side of the connection, the instruction sets the Active input to TRUE.

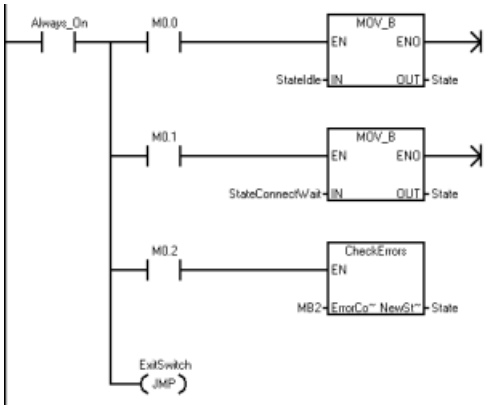


Network 5: If Done is TRUE, the CPU established the connection to go to the "Idle" state.

If Busy is TRUE, go to the "Connect Wait" state to wait for the connection to be established.

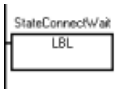
If Error is TRUE, there is probably an error with the input parameters, so check to see what state to go to next.

In all cases, exit the state machine for this scan. The program continues to the next state on the next scan.

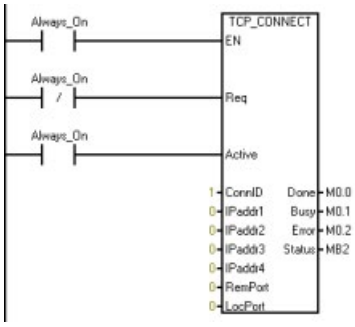


Network 6: State "Connect Wait"...

Wait in this state until there is a connection created to the passive device.



Network 7: Call the TCP_CONNECT instruction with Req = FALSE and the same Connection ID (ConnID) as above. Do this to check the connection status and ensure that the CPU established the connection.

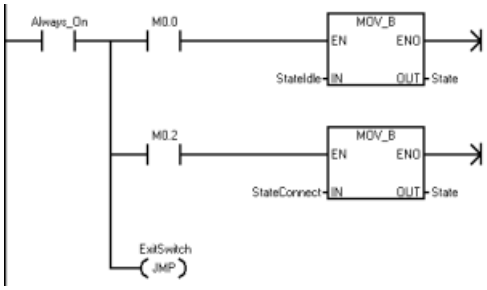


Network 8: If Done is TRUE, this means that the CPU established the connection, so continue in the "Idle" state.

If Busy is TRUE, stay in the "Connect Wait" state. The TCP_CONNECT instruction eventually times out and returns an error if the other device is not present, so you do not need to have a time out mechanism for the connect process.

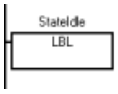
If Error is TRUE, there is a problem connecting to the passive device. In this case, just try again by going back to the "Connect" state. Note that if the passive device is present but it rejects the connection request, the connection errors very quickly and utilizes a great amount of bandwidth as the CPU continues to attempt to create a connection.

In all cases, exit the state machine for this scan. The program continues to the next state on the next scan.

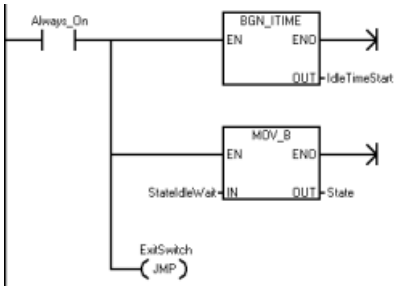


Network 9: State "Idle"...

This state puts a time delay between messages so that you do not flood the network. The symbol "IdleTimeDelay" specifies the delay time.



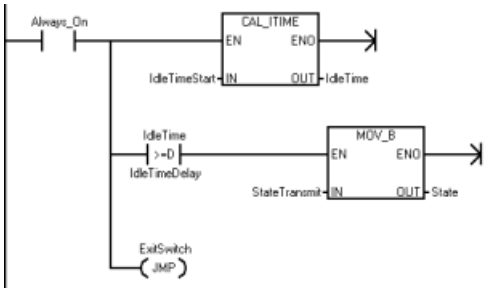
Network 10: Capture the interval timer and then change to the "Idle Wait" state to delay until it is time to transmit again.



Network 11: State "Idle Wait"...

Stay in this state for the number of milliseconds specified (IdleTimeDelay).

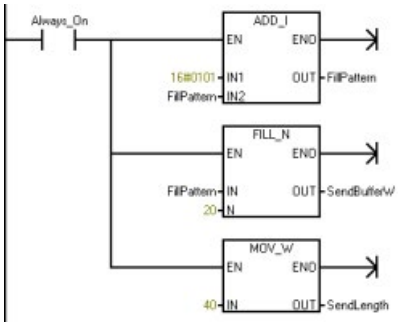
Network 12: Calculate the time since you entered the "Idle" state and, if this is greater than the "IdleTimeDelay" value, change the state to the "Transmit" state.



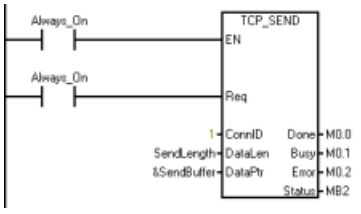
Network 13: State 'Transmit"...

Network 14: Create the message to send to the passive device.

For the test program, fill the send buffer with 40 bytes (20 words). Write the number of bytes to send into a variable so that you can use the same value in the "TransmitWait" state.



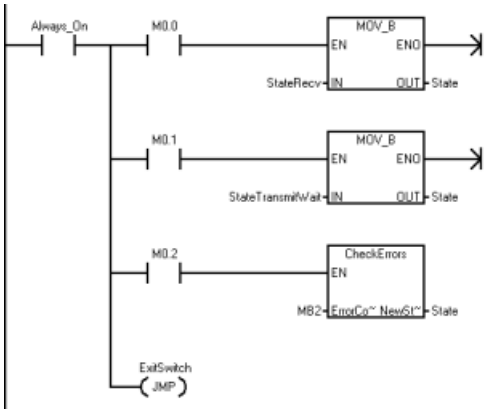
Network 15: Send the new message to the passive device. Set Req to TRUE to initiate the new send operation.



Network 16: If Done is TRUE, the send is complete (probably will never happen this quickly), so go to the "Receive" state on the next scan.

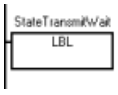
If Busy is TRUE (this is the normal situation), go to the "Transmit Wait" state to wait for the transmit to finish.

If Error is TRUE, check the reason, and you may need to change state if there is a connection issue.

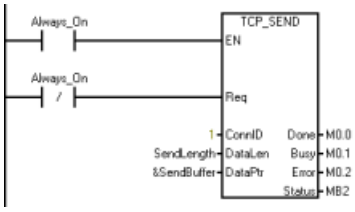


Network 17: State "Transmit Wait"...

Wait in this state until the transmit is complete.



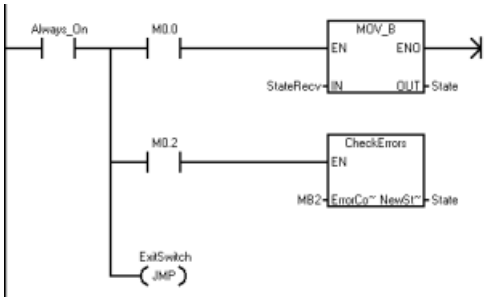
Network 18: Call the TCP_SEND instruction with Req = FALSE to determine if the send is complete. Be sure to use the same send length and buffer pointer that you used to initialize the send request.



Network 19: If Done is TRUE, the send is complete, so go to the "Receive" state.

If Busy is TRUE, stay in the "Transmit Wait" state.

If Error is TRUE, check the reason for the error and change the state if there is a connection issue.

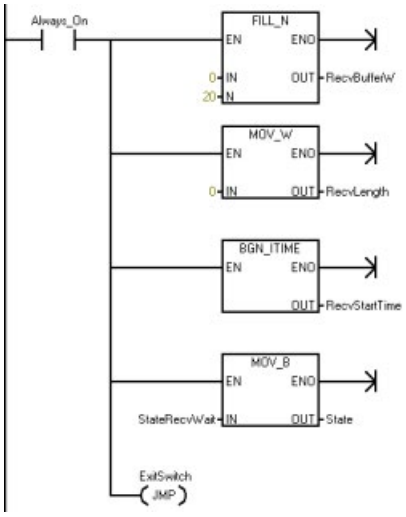


Network 20: State "Receive"...

Clean up the receive data area and prepare to receive the response.

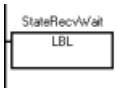
Network 21: Clear the receive buffer and the "RecvLength" so that there is no data left from the last received message.

Capture the current interval time value (in "RecvStartTime") to support a receive timeout and then go to the "Receive Wait" state.

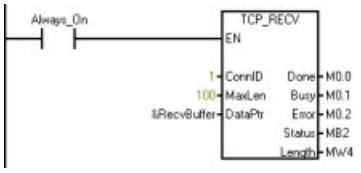


Network 22: State "Receive Wait"...

Stay in this state until you either receive some data or time out.



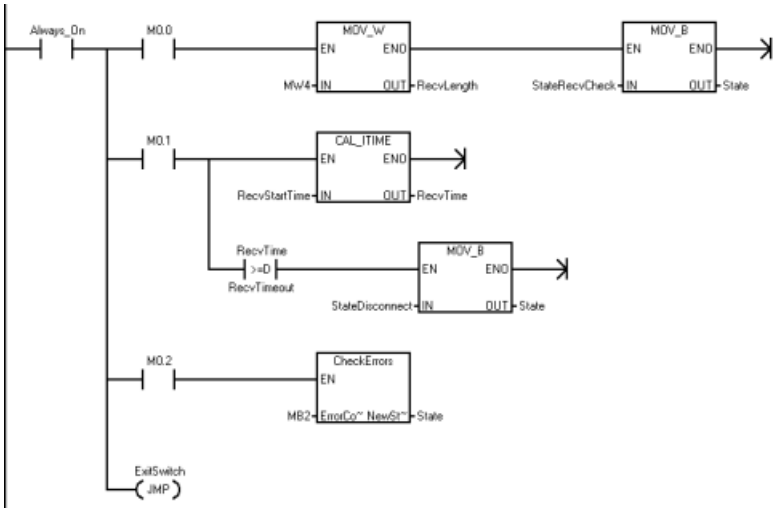
Network 23: Call the TCP_RECV instruction to get any messages that the CPU has received.



Network 24: If Done is TRUE, the instruction received new data, so go to the "Receive Check" state.

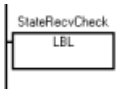
If Busy is TRUE, stay in the "Receive" state until you reach the receive timeout value. If you time out in this state, disconnect the device and then reconnect it.

If Error is TRUE, check the error code to determine what to do next.

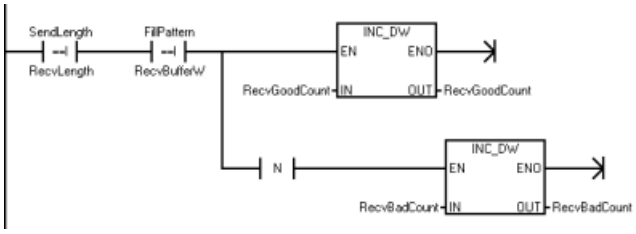


Network 25: State "RecvCheck"...

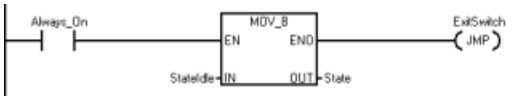
Process the data returned from the passive device.



Network 26: This program only checks the returned data to ensure that you received the same amount of data as was sent and that the first word matches the "Fill Pattern". Log whether the response was good or bad, and then go to the "Idle" state to wait to send the next message.



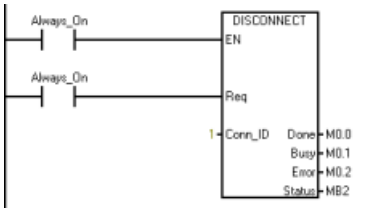
Network 27: Go to the "Idle" state in all situations.



Network 28: State "Disconnect"...

Initiate a disconnect.

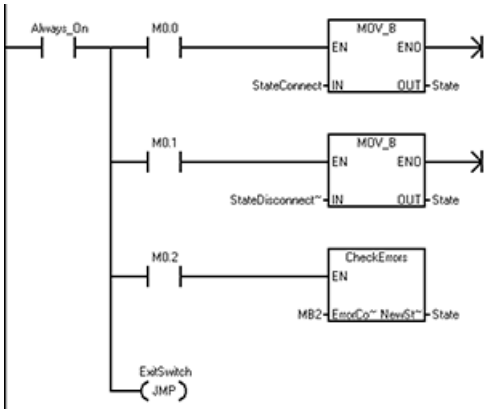
Network 29: Initiate the disconnect with the ConnID by calling the DISCONNECT instruction with Req = TRUE.



Network 30: If Done is TRUE, this means that the disconnect is complete (probably will never happen), so go try to reconnect.

If Busy is TRUE (this would be normal), go to the "Disconnect Wait" state.

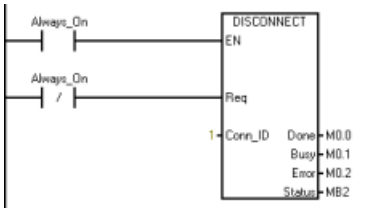
If Error is TRUE, check the reason, and you may need to change state if there is a connection issue.



Network 31: State "Disconnect Wait"...

Wait in this state for the disconnect operation to complete.

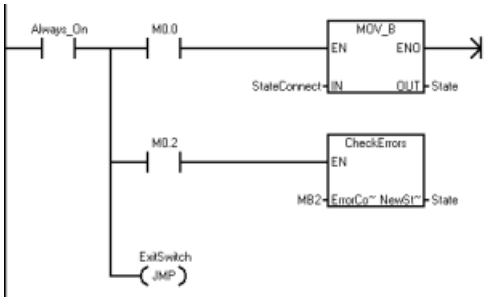
Network 32: Call the DISCONNECT instruction with Req = FALSE to check the status of the disconnect operation.



Network 33: If Done is TRUE, this means that the disconnect is complete, so try to reconnect on the next scan.

If Busy is TRUE (this would be normal), stay in the "Disconnect Wait" state.

If there is an error, check the reason, and you may need to change state based upon the error code.

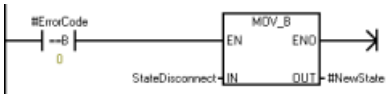


Network 34: Exit Switch.

9.5.4.2 CheckErrors subroutine

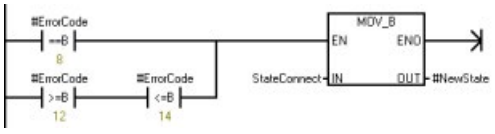
The CheckErrors subroutine checks the Open User Communication error codes and determines if the program requires a state change. You use the same CheckErrors subroutine for both the Active partner (client) and Passive partner (server).

Network 1: The program malfunctioned if there is no error code. Disconnect and then reconnect to correct this problem.



Network 2: If one of the partners drops the connection, the program displays error codes 8, 12, 13, and 14. The partners are currently disconnected.

In all these cases, reconnect to the partner. Set the state to "Connect".

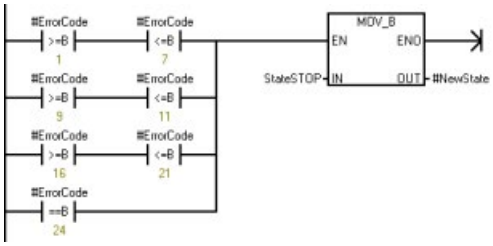


Network 3: If the error is a parameter error (error codes 1 - 7), stop the program because there is a configuration error with one of inputs to the function. Correct this in the program.

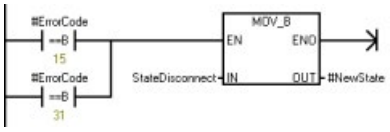
If the error code is 9 (connect in progress), 10 (disconnect in progress), or 11 (send in progress), stop because the state machine is broken. The state machine is programmed to stay in the associated waiting state until the operation is complete, and these errors should never occur.

If the error code is between 16 and 21, these constitute connect parameter errors and should not appear here. If these errors occur, there is a malfunction, so stop program execution.

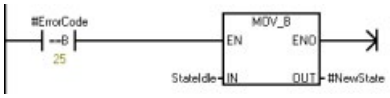
If the SEND and DISCONNECT instructions return error 24, there is no operation currently pending. This probably means that the operation is complete; however, error 24 should never appear based upon the state machine. Consider this an error, and go to stop.



Network 4: The program returns errors 15 and 31 if there are problems with the connection. Disconnect and then reconnect to correct this problem.



Network 5: The program returns error 25 when the RECV function receives more data than the buffer can support. In our case, continue as if nothing happened.



9.5.4.3 Active partner symbol table

The table below lists the Symbol names, addresses, and comments for the Active partner (client) program.

Symbol	Address	Comments
Always_On	SM0.0	Always ON
First_Scan_On	SM0.1	On for the first scan cycle only
State	VB0	
IdleTime	VD4	
IdleTimeStart	VD8	
RecvGoodCount	VD20	
RecvBadCount	VD24	
RecvStartTime	VD28	
RecvTime	VD32	
FillPattern	VW12	
RecvLength	VW16	
SendLength	VW18	
SendBufferW	VW1000	
RecvBufferW	VW2000	
StateConnect	1	
StateConnectWait	2	
StateIdle	3	
StateIdleWait	4	
StateTransmit	5	

Symbol	Address	Comments
StateTransmitWait	6	
StateRecv	7	
StateRecvWait	8	
StateRecvCheck	9	
StateDisconnect	10	
StateDisconnectWait	11	
ExitSwitch	19	
RecvTimeout	100	Milliseconds for receive timeout
IdleTimeDelay	250	Milliseconds between send commands

9.5.4.4 Passive partner (server)

This program implements a simple state machine to manage the opening of a connection, receiving a message, sending a response, and handling errors.

The flow of this state machine is to connect, then repeatedly receive a message and send a response. If the connection is dropped, the state machine reopens the passive connection.

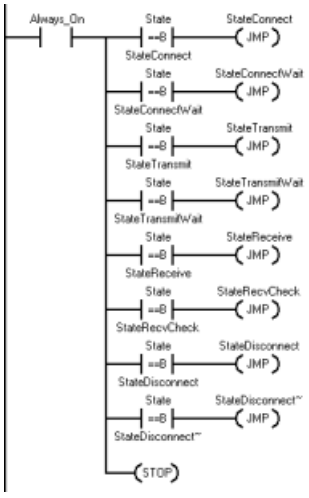
Refer to the "Passive partner symbol table" (Page 570) to see the symbol table for this program.

Network 1: On the first scan....

Initialize the State variable to initiate a connection.

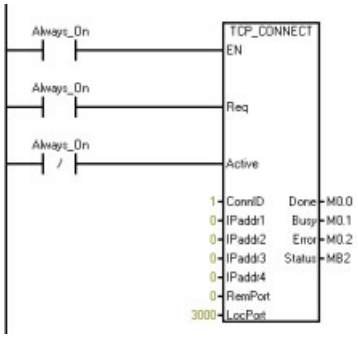
Network 2: Process the state machine...

Determine the current state of the state machine and jump to the label for the state handler. If the state is ever illegal, the CPU goes to the STOP mode.



Network 3: State Connect...

Network 4: Start the connection process. This is a server so set the Active input FALSE. Set the IPaddr inputs to zero so that the server accepts a connection from any address. Set the RemPort to zero because you do not need it for a server connection. Call the TCP_CONNECT instruction with the Req input TRUE to start the connection process.

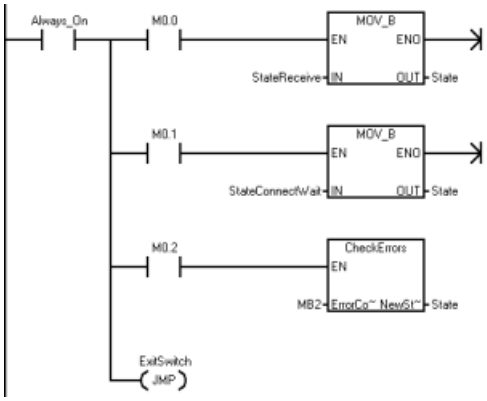


Network 5: If Done is TRUE, the CPU established the connection so go to the "Idle" state.

If Busy is TRUE, go to the "Connect Wait" state to wait for the connection to be established.

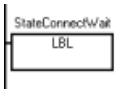
If Error is TRUE, there is probably an error with the input parameters, so check to see what state to go to next.

In all cases, exit the state machine for this scan. The program continues to the next state on the next scan.

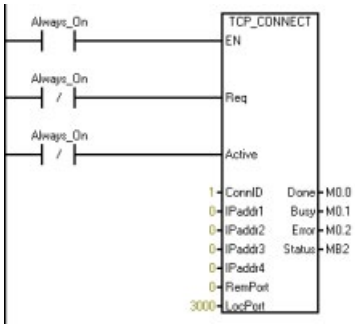


Network 6: State "Connect Wait"...

Wait in this state until the active partner creates a connection to this CPU.



Network 7: Call the TCP_CONNECT instruction with Req = FALSE and the same Connection ID (ConnID) as above. Do this to check the connection status.

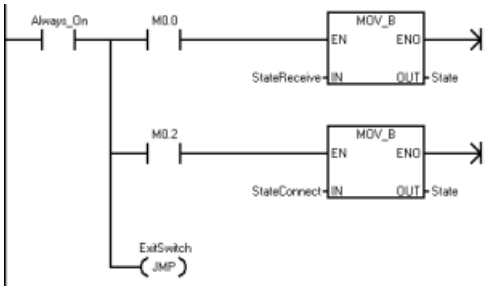


Network 8: If Done is TRUE, this means that the CPU established the connection so continue in the "Idle" state.

If Busy is TRUE, stay in the "Connect Wait" state. Since this is a passive connection, stay in the "Busy" state until the active partner connects to the CPU. There is no timeout for a passive connection.

If Error is TRUE, a problem occurred so go back and try again to connect on the next scan.

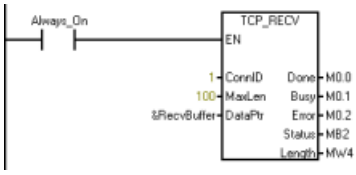
In all cases, exit the state machine for this scan. The program continues to the next state on the next scan.



Network 9: State Receive...

Stay in this state until the server receives some data.

Network 10: Call the TCP_RECV instruction to get any messages that the CPU has received.

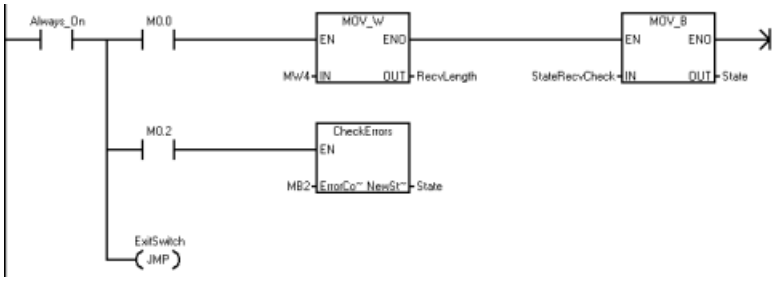


Network 11: If Done is TRUE, the CPU received new data so go to the "Receive Check" state.

If Busy is TRUE, stay in the "Receive" state until you receive some data.

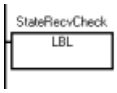
If Error is TRUE, check the error code to determine what to do next.

In all cases, exit the state machine for this scan. The program continues to the next state on the next scan.



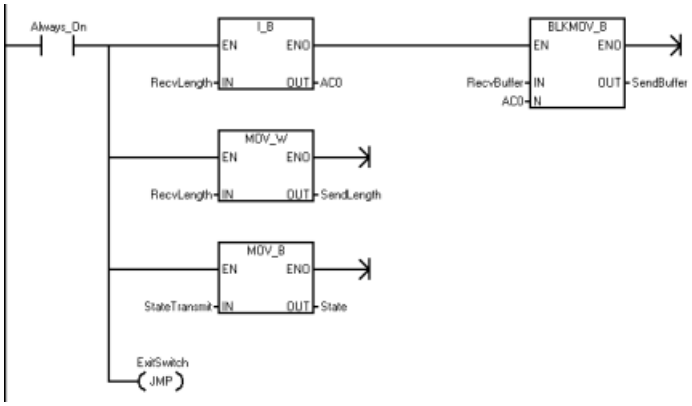
Network 12: State "Receive Check"...

Check and process the received data.



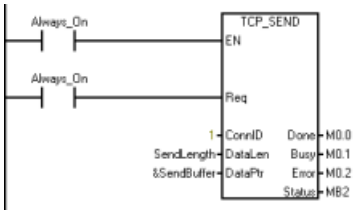
Network 13: In this program, you echo the data back to the partner. Copy all of the received bytes to the send buffer.

Change the state to "Transmit", and exit until the next scan.



Network 14: State "Transmit"...

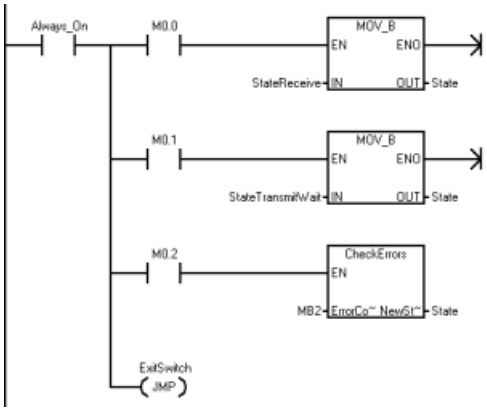
Network 15: Send the data back to the partner.



Network 16: If Done is TRUE, the send is complete (probably never happen this quickly), so go to the "Receive" state on the next scan.

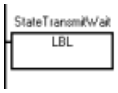
If Busy is TRUE (this is the normal situation), go to the "Transmit Wait" state to wait for the transmit to finish.

If Error is TRUE, check the reason, and you may need to change state if there is a connection issue.

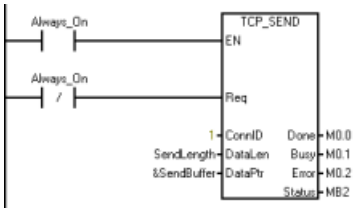


Network 17: State "Transmit Wait"...

Wait in this state until the transmit is complete.



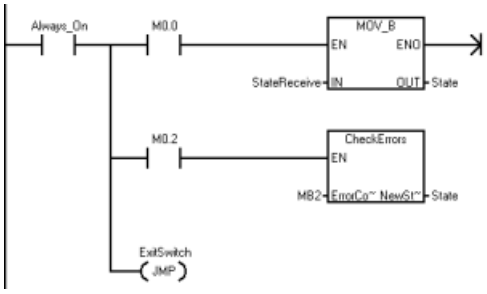
Network 18: Call the TCP_SEND instruction with Req = FALSE to determine when the send is complete. Be sure to use the same send length and buffer pointer that you used to initialize the send request.



Network 19: If Done is TRUE, the send is complete so go to the "Receive" state.

If Busy is TRUE, stay in the "Transmit Wait" state.

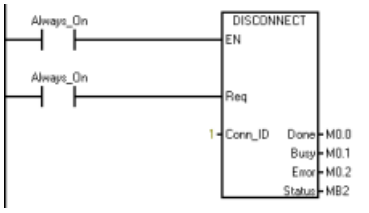
If Error is TRUE, check the reason for the error and change the state if there is a connection issue.



Network 20: State "Disconnect"...

Initiate a disconnect.

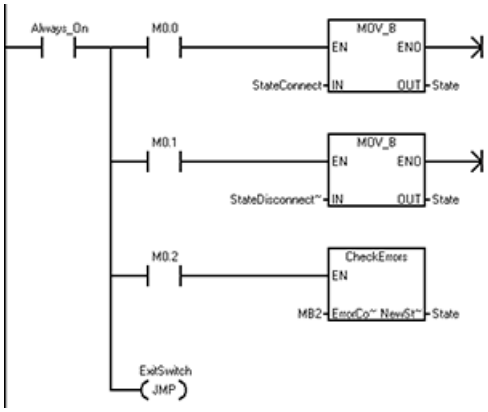
Network 21: Initiate the disconnect with the ConnID by calling the DISCONNECT instruction with Req = TRUE.



Network 22: If Done is TRUE, this means that the disconnect is complete (probably will never happen), so try to reconnect.

If Busy is TRUE (this would be normal), go to the "Disconnect Wait" state.

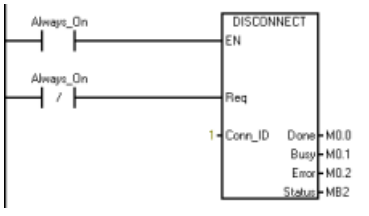
If Error is TRUE, check the reason, and you may need to change state if there is a connection issue.



Network 23: State "Disconnect Wait"...

Wait in this state for the disconnect operation to complete.

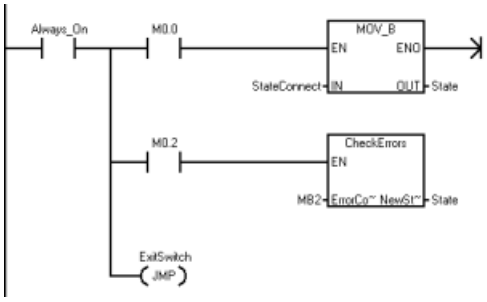
Network 24: Call the DISCONNECT instruction with Req = FALSE to check the status of the disconnect operation.



Network 25: If Done is TRUE, this means that the disconnect is complete, so try to reconnect on the next scan.

If Busy is TRUE (this would be normal), stay in the "Disconnect Wait" state.

If there is an error, check the reason, and you may need to change state based upon the error code.



Network 26: Exit Switch.

9.5.4.5 CheckErrors subroutine

The CheckErrors subroutine (Page 562) checks the Open User Communication error codes and determines if the program requires a state change. You use the same CheckErrors subroutine for both the Active partner (client) and Passive partner (server).

9.5.4.6 Passive partner symbol table

The table below lists the Symbol names, addresses, and comments for the Passive partner (server) program.

Symbol	Address	Comments
Always_On	SM0.0	Always ON
First_Scan_On	SM0.1	On for the first scan cycle only
State	VB0	
SendBuffer	VB1000	
RecvBuffer	VB2000	
RecvLength	VW16	
SendLength,	VW18	
SendBufferW	VW1000	
RecvBufferW	VW2000	
StateConnect	1	
StateConnectWait	2	
StateTransmit	3	

Symbol	Address	Comments
StateTransmitWait	4	
StateReceive	5	
StateRecvCheck	6	
StateDisconnect	7	
StateDisconnectWait	8	
ExitSwitch	10	

9.6 PN Read Write Record library

9.6.1 PN Read Write Record features

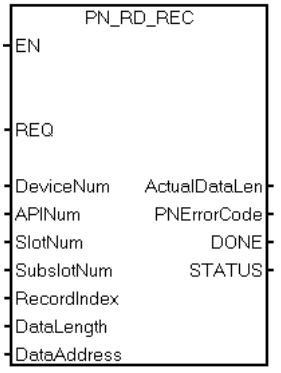
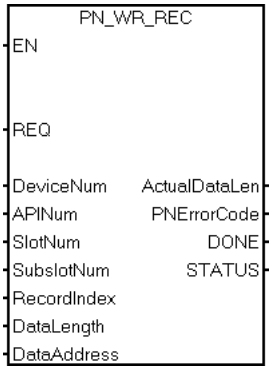
PN Read Write Record library includes the following two instructions:

- PN_RD_REC: Read a data record from any connected PROFINET device
- PN_WR_REC: Write a data record to any connected PROFINET device.

9.6.2 Input and output interface of PN Read Write Record library

The PN_RD_REC and PN_WR_REC instructions are as follows:

Table 9- 26 PN_RD_REC and PN_WR_REC

LAD/FBD	STL	Description
	CALL PN_RD_REC, REQ,DeviceNum, APINum, SlotNum, SubslotNum, RecordIndex, DataLength, DataAddress, ActualDataLen, PnErrorCode, Done, STATUS	Use the PN_RD_REC instruction to read a data record from PROFINET device.
	CALL PN_WR_REC, REQ,DeviceNum, APINum, SlotNum, SubslotNum, RecordIndex BufferLength, DataAddress, ActualDataLen, PnErrorCode, Done, STATUS	Use the PN_WR_REC instruction to write a data record to PROFINET device.

The parameters of the PN_RD_REC and PN_WR_REC instructions are as follows:

Table 9- 27 Parameters of PN_RD_REC and PN_WR_REC instructions

Parameter and type		Data type	Description
REQ	IN	BOOL	REQ=1: Transfer data record
Device Number	IN	WORD **Note:** The value range is from 1 to 8.	Device Number, API Number, Slot Number and SubSlot Number are used to address a submodule. You can find the Device Number, API Number, Slot Number and SubSlot Number in the PROFINET wizard.
API Number	IN	DWORD	
Slot Number	IN	WORD	
SubSlot Number	IN	WORD	

Parameter and type		Data type	Description
Record Index	Input	WORD	The Record Index includes the record index from protocol or the user-defined record index. For detailed information of the index from the protocol, refer to Technical Specification for PROFINET IO (Version 2.3).
Data Length	Input	WORD	This parameter refers to the number of bytes of the buffer. The buffer stores the data record read from or written to the device. The value range: from 1 to 1024.
Data Address	Input	DWORD	Address of the buffer read from or written to the device. **Note**: If the buffer length is greater than the actual record data length, the buffer contains all the record data.If the buffer length is smaller than actual record data length, the buffer contains partial record data and an error occurs.
Actual record data length	Output	WORD	This parameter is valid for the instruction RDREC and returns the actual data length specified by the device.
PROFINET Error Code	Output	DWORD	The error code defined by the PROFINET protocol. 0 = no error. If the value is not 0, check the specific error code in Technical Specification for PROFINET IO (Version 2.3).
DONE	OUT	BOOL	The instruction is completed.
STATUS	OUT	BYTE	The status of the current operation. For detailed information, refer to Definition of parameters for input signal "STATUS" (Page 573).

9.6.3 Definition of parameters for input signal "STATUS"

The following table lists the parameter information of "STATUS":

Table 9- 28 STATUS

Byte Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A [1]	E [2]	Error code [3]					

[1] A : 1 = a request is in process

[2] E : 1= an error occurs

[3] Error code: The system error code. For detailed information, refer to System_defined error code of the library PN Read Write Record (Page 574).

9.6.4 System_defined error code of the library PN Read Write Record

The error codes are as follows:

Error code	Description
0	No error.
1	The data length parameter is 0 or is greater than the supported maximum length (1024 bytes).
2	The data buffer is not in I, Q, M, or V memory areas.
3	The data buffer does not fit in the memory area.
4	The table doesn't match with the memory.
5	The device number is invalid and not within the range: from 1 to 8.
6	An instance mismatch: The connection is busy with another instance, whose device number, API number, slot number and subslot number are same as the requested instance, but with a different buffer size and data address.
7	The PROFINET device is not connected.
8	The size of the received buffer exceeds 1024 bytes.
9	Call sequence is invalid.
10	Parameters are invalid (for example, out-of-range).
11	The AR is created afresh in the meanwhile.
12	The RPC reports a timeout error.
13	The RPC reports a communication error.
14	The RPC Server of the IOD signaled “busy” (for example, the call can be repeated later).
15	CLRPC reports an error or the PDU cannot be parsed.
16	CM response is OK, but has a PROFINET protocol defined error.
17	The instruction parameter is invalid.
24	REQ is not enabled.
25	The buffer length is smaller than the actual data record length.
63	Unknown error.

9.7 USS library

9.7.1 USS communication overview

9.7.1.1 USS protocol overview

STEP 7-Micro/WIN SMART instruction libraries make controlling Siemens drives easier by including pre-configured subroutines and interrupt routines that are specifically designed for using the USS protocol to communicate with a motor drive. You can control the physical drive and the read/write drive parameters with the USS instructions.

Siemens designed the USS communications library for use with Siemens general purpose drives such as the Siemens Micromaster series. Siemens does not intend for the USS communications library to be used with special purpose drives such as the V90 servo drive. The control interface of the V90 servo drive is different from that of a general purpose drive. For this reason, do not use the USS communications library with the V90 servo drive.

You find these instructions in the "Libraries" folder of the STEP 7-Micro/WIN SMART instruction tree. When you select a USS instruction, one or more associated subroutines and interrupts are added automatically.

The USS protocol library overview discusses the following subjects:

- Requirements for using the USS protocol (Page 575)
- Calculating the time required for communicating with the drive (Page 576)

Refer to "Using the USS protocol instructions (Page 577)" for a listing of USS protocol instructions, error codes, and example programs.

Note

Only call the library functions from either the main program or from interrupt routines, but not both.

Note

For the compact CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s, do not connect pin 9 of the RS485 cable used for USS communication. The CRs CPU uses pin 9 to disable Freeport mode.

9.7.1.2 Requirements for using the USS protocol

The STEP 7-Micro/WIN SMART instruction libraries provide subroutines, interrupt routines, and instructions to support the USS protocol. The USS instructions use the following resources in the S7-200 SMART CPU:

- The USS protocol is an interrupt driven application. In the worst case, the receive message interrupt routine requires up to 2.5 ms to execute. During this time, all other interrupt events are queued for service after the receive message interrupt routine has been executed. If your application cannot tolerate this worst case delay, then you may want to consider other solutions for controlling drives.
- Initializing the USS protocol dedicates an S7-200 SMART CPU port for USS communications.

  You use the USS_INIT instruction to select either USS or PPI for port 0 or port 1. (USS refers to the USS protocol for Siemens drives.) When a port is set to use the USS protocol for communicating with drives, you cannot use the port for any other purpose, including communicating with an HMI. The second communications port allows STEP 7-Micro/WIN SMART to monitor the control program while USS protocol is running.
- The USS instructions affect all of the SM locations that are associated with Freeport communication on the assigned port.
- The USS subroutines and interrupt routines are stored in your program. The USS instructions increase the amount of memory required for your program by up to 3050

bytes. Depending on the specific USS instructions used, the support routines for these instructions can increase the overhead for the control program by at least 2150 bytes and up to 3050 bytes.

- The variables for the USS instructions require a 400-byte block of V memory. The starting address for this block is assigned by the user and is reserved for USS variables.
- Some of the USS instructions also require a 16-byte communications buffer. As a parameter for the instruction, you provide a starting address in V memory for this buffer. It is recommended that a unique buffer be assigned for each instance of USS instructions.
- When performing calculations, the USS instructions use accumulators AC0 to AC3. You can also use the accumulators in your program; however, the values in the accumulators will be changed by the USS instructions.
- The USS instructions cannot be used in an interrupt routine.

9.7.1.3 Calculating the time required for communicating with the drive

Communications with the drive are asynchronous to the S7-200 SMART CPU scan. The CPU typically completes several scans before one drive communications transaction is completed. The following factors help determine the amount of time required:

- Number of drives present
- Baud rate
- Scan time of the CPU

Some drives require longer delays when using the parameter access instructions. The amount of time required for a parameter access is dependent on the drive type and the parameter being accessed.

After a USS_INIT instruction assigns Port 0 to use the USS Protocol (or USS_INIT_P1 for port 1), the CPU regularly polls all active drives at the intervals shown in the following table. You must set the time-out parameter of each drive to allow for this task:

Table 9- 29 Communications times

Baud rate	Time between polls of active drives (with no parameter access instructions active)
1200	240 ms (maximum) times the number of drives
2400	130 ms (maximum) times the number of drives
4800	75 ms (maximum) times the number of drives
9600	50 ms (maximum) times the number of drives
19200	35 ms (maximum) times the number of drives
38400	30 ms (maximum) times the number of drives
57600	25 ms (maximum) times the number of drives
115200	25 ms (maximum) times the number of drives

9.7.2 USS program instructions

9.7.2.1 Using the USS protocol instructions

Procedure

To use the USS protocol instructions in your S7-200 SMART program, follow these steps:

1. Insert the USS_INIT instruction in your program and execute the USS_INIT instruction for one scan only. You can use the USS_INIT instruction either to initiate or to change the USS protocol communication parameters.

   When you insert the USS_INIT instruction, several hidden subroutines and interrupt routines are automatically added to your program.

2. Place only one USS_CTRL instruction in your program for each active drive.

   You can add as many USS_RPM_x and USS_WPM_x instructions as required, but only one of these can be active at a time.

3. Click the Memory button Memory from the Libraries area of the File menu ribbon strip to assign a starting address for the V Memory that the USS library requires. Alternatively, you can right-click the Program Block node in the project tree and select "Library Memory" from the context menu.

4. Configure the drive parameters to match the baud rate and address used in the program.

5. Connect the communications cable between the S7-200 SMART CPU and the drives.

   Ensure that all of the control equipment, such as the S7-200 SMART CPU, that is connected to the drive be connected by a short, thick cable to the same ground or star point as the drive.

   CAUTION

   Avoiding unwanted current flow

   Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable. These unwanted currents can cause communications errors or damage equipment.

   Ensure that all equipment that is connected with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows.

   The shield must be tied to chassis ground or pin 1 on the 9-pin connector. It is recommended that you tie terminal 2-0V on the drive to chassis ground.

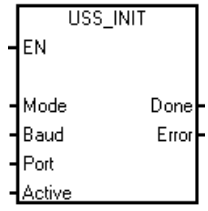
The USS protocol instructions consist of the following:

- USS_INIT (Page 578)
- USS_CTRL (Page 580)
- USS_RPM_X (Page 583)
- USS_WPM_x (Page 585)

USS protocol program examples (Page 588) and a listing of USS protocol error codes (Page 588) are also discussed in this section.

9.7.2.2 USS_INIT instruction

Table 9- 30 USS_INIT instruction

LAD / FBD	STL	Description
	`CALL USS_INIT, Mode, Baud, Port, Active, Done, Error`	The USS_INIT instruction is used to enable and initialize, or to disable Siemens drive communications. Before any other USS instruction can be used, the USS_INIT instruction must be executed without errors. The instruction completes and the "Done" bit is set immediately, before continuing to the next instruction.

The instruction is executed on each scan when the "EN" input is on.

Execute the USS_INIT instruction only once for each change in communications state. Use an edge detection instruction to pulse the "EN" input on. To change the initialization parameters, execute a new USS_INIT instruction.

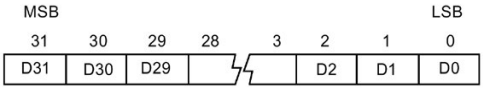
Table 9- 31 Parameters for the USS_INIT instruction

Inputs/outputs	Data type	Operands
Mode, Port	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Baud, Active	DWORD	VD, ID, QD, MD, SD, SMD, LD, Constant, AC *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

Table 9- 32 USS_INIT parameter descriptions

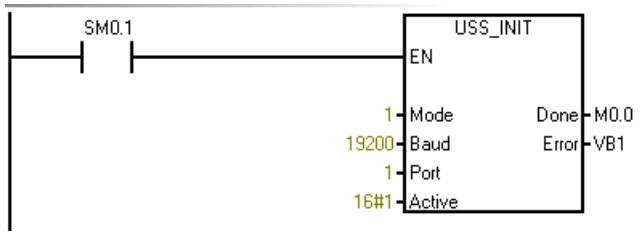
Parameter	Description
Mode	This value selects the communications protocol: • An input value of 1 assigns the port to USS protocol and enables the protocol. • An input value of 0 assigns the port to PPI and disables the USS protocol.
Baud	Sets the baud rate at 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200
Port	Sets the physical communication port (0 = RS485 integrated in CPU, 1 = RS485 or RS232 located on the optional CM01 signal board)
Active	Indicates which drives are active. Some drives only support addresses 0 through 30.
Done	Turned on when the USS_INIT instruction completes
Error	This output byte contains the result of executing the instruction. The USS protocol execution error codes (Page 588) define the error conditions that could result from executing the instruction.

Table 9- 33 Format for the Active drive parameter

	This figure shows the description and format of the active drive input. Any drive that is marked as "Active" is automatically polled in the background to control the drive, collect status, and prevent serial link time-outs in the drive.
• D0 (Drive 0 active bit): – 0 - drive not active – 1 - drive active • D1 (Drive 1 active bit): – 0 - drive not active – 1 - drive active • ...	

Refer to the USS protocol execution error codes (Page 588) to compute the time between status polls and the error conditions that could result from executing the instruction.

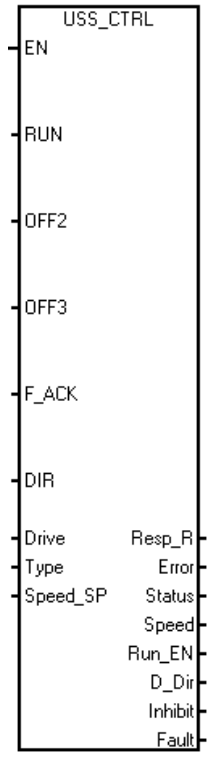
Table 9- 34 USS_INIT example program

Network 1	Network 1
	**LD SM0.1** **CALL USS_INIT, 1, 19200, 1, 16#1, M0.0, VB1**

Refer to "Using the USS protocol instructions" (Page 577) for and a listing of USS protocol instructions and error codes and example programs.

9.7.2.3 USS_CTRL instruction

Table 9- 35 USS_CTRL instruction

LAD / FBD	STL	Description
	**CALL USS_CTRL, RUN, OFF2, OFF3, F_ACK, DIR, Drive, Type, Speed_SP, Resp_R, Error, Status, Speed, Run_EN, D_Dir, Inhibit, Fault**	The USS_CTRL instruction is used to control an active Siemens drive. The USS_CTRL instruction places the selected commands in a communications buffer, which is then sent to the addressed drive ("Drive" parameter), if that drive has been selected in the "Active" parameter of the USS_INIT instruction.

Only one USS_CTRL instruction should be assigned to each drive.

Some drives report speed only as a positive value. If the speed is negative, the drive reports the speed as positive, but reverses the "D_Dir" (direction) bit.

The "EN" bit must be on to enable the USS_CTRL instruction. This instruction should always be enabled.

Table 9- 36 Parameters of the USS_CTRL instruction

Inputs/outputs	Data types	Operands
RUN, OFF 2, OFF 3, F_ACK, DIR	BOOL	I, Q, M, S, SM, T, C, V, L, Power Flow
Resp_R, Run_EN, D_Dir, Inhibit, Fault	BOOL	I, Q, M, S, SM, T, C, V, L
Drive, Type	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD, Constant
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD
Status	WORD	VW, T, C, IW, QW, SW, MW, SMW, LW, AC, AQW, *VD, *AC, *LD
Speed_SP	REAL	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD, Constant
Speed	REAL	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD

RUN parameter

RUN (RUN/STOP) indicates whether the drive is on (1) or off (0). When the "RUN" bit is on, the drive receives a command to start running at the specified speed and direction. In order for the drive to run, the following must be true:

- Drive must be selected as "Active" in USS_INIT.
- "OFF2" and "OFF3" must be set to 0.
- "Fault" and "Inhibit" must be 0.

When "RUN" is off, a command is sent to the drive to ramp the speed down until the motor comes to a stop:

- The "OFF2" bit is used to allow the drive to coast to a stop.
- The "OFF3" bit is used to command the drive to stop quickly.

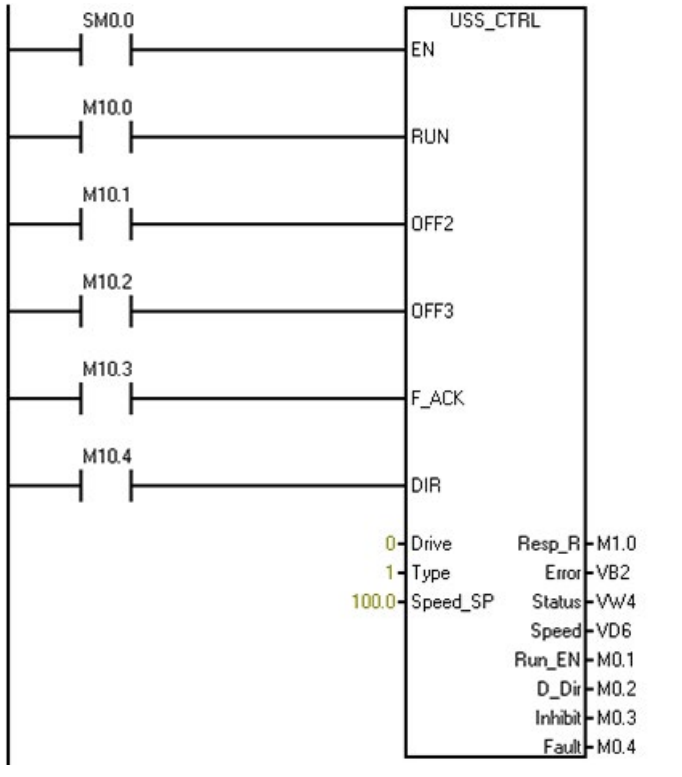
Resp_R parameter

The "Resp_R" (response received) bit acknowledges a response from the drive. All the Active drives are polled for the latest drive status information. Each time the CPU receives a response from the drive, the "Resp_R" bit is turned on for one scan and all the following values are updated:

Parameter	Description
F_ACK (fault acknowledge)	Bit that acknowledges a fault in the drive. The drive clears the fault ("Fault" bit) when "F_ACK" goes from 0 to 1.
DIR (direction)	Bit that indicates in which direction the drive should move.
Drive (drive address)	Input for the address of the drive to which the USS_CTRL command is to be sent. Valid addresses: 0 to 31
Type (drive type)	Input that selects the type of drive
Speed_SP (speed setpoint)	Drive speed as a percentage of full speed: • Negative values of "Speed_SP" cause the drive to reverse its direction of rotation. • Range: -200.0% to 200.0%
Error	Byte that contains the result of the latest communications request to the drive. The USS protocol execution error codes (Page 588) define the error conditions that could result from executing the instruction.
Status	Raw value of the status word returned by the drive. The figures below show the status bits for standard status word and main feedback.
Speed	Drive speed as a percentage of full speed. Range: -200.0% to 200.0%
Run_EN (RUN enable)	Indicates the drive condition: • Running (1) • Stopped (0)
D_Dir	Indicates the drive's direction of rotation

Parameter	Description
Inhibit	Indicates the state of the "Inhibit" bit on the drive: • 0: not inhibited • 1: inhibited To clear the "Inhibit" bit, the following bits must be OFF: • "Fault" • "RUN" • "OFF2" • "OFF3"
Fault	Indicates the state of the "Fault" bit: • 0: no fault • 1: fault The drive displays the fault code. (Refer to the manual for your drive). To clear the "Fault" bit, correct the cause of the fault and turn on the "F_ACK" bit.

Table 9- 37 USS_CTRL example program



To display in LAD or FBD:

```
Network 1 // Control box for drive 0
LD SM0.0
= L60.0
LD M10.0
= L63.7
LD M10.1
= L63.6
LD M10.2
= L63.5
LD M10.3
= L63.4
LD M10.4
= L63.3
LD L60.0
CALL USS_CTRL, L63.7, L63.6, L63.5, L63.4,
L63.3, 0, 1, 100.0, M1.0, VB2, VW4, VD6,
M0.1, M0.2, M0.3, M0.4
```

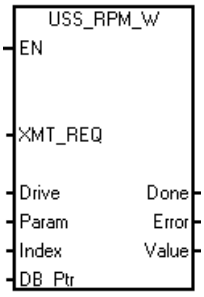
To display in STL only:

```
Network 1 // Control box for drive 0
LD SM0.0
CALL USS_CTRL, M10.0, M10.1, M10.2, M10.3,
M10.4, 0, 1, 100.0, M1.0, VB2, VW4, VD6,
M0.1, M0.2, M0.3, M0.4
```

Refer to "Using the USS protocol instructions" (Page 577) for a listing of USS protocol instructions and error codes and example programs.

9.7.2.4 USS_RPM_x instruction

Table 9- 38 USS_RPM_x instructions

LAD / FBD	STL	Description
	**CALL USS_RPM_W, XMIT_REQ, Drive, Param, Index, DB_Ptr, Done, Error, Value** **CALL USS_RPM_D, XMIT_REQ, Drive, Param, Index, DB_Ptr, Done, Error, Value** **CALL USS_RPM_R, XMIT_REQ, Drive, Param, Index, DB_Ptr, Done, Error, Value**	There are three read instructions for the USS protocol: • USS_RPM_W instruction reads an unsigned word parameter. • USS_RPM_D instruction reads an unsigned double word parameter. • USS_RPM_R instruction reads a floating-point parameter. • Only one read (USS_RPM_x) or write (USS_WPM_x) instruction can be active at a time.

The USS_RPM_x transactions complete when the drive acknowledges receipt of the command or when an error condition is posted. The logic scan continues to execute while this process awaits a response.

Table 9- 39 Valid operands for the USS_RPM_x instructions

Inputs/outputs	Data type	Operands
XMT_REQ	BOOL	I, Q, M, S, SM, T, C, V, L, Power Flow conditioned by a rising edge detection element
Drive	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD, Constant
Param, Index	WORD	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AIW, *VD, *AC, *LD, Constant
DB_Ptr	DWORD	&VB
Value	WORD DWORD, REAL	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AQW, *VD, *AC, *LD VD, ID, QD, MD, SD, SMD, LD, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC. *VD, *AC, *LD

The "EN" bit must be on to enable transmission of a request, and should remain on until the "Done" bit is set, signaling completion of the process. For example, a USS_RPM_x request is transmitted to the drive on each scan when the "XMT_REQ" input is on. Therefore, the "XMT_REQ" input should be pulsed on through an edge detection element which causes one request to be transmitted for each positive transition of the "EN" input.

Table 9- 40 USS_RPM_x parameter descriptions

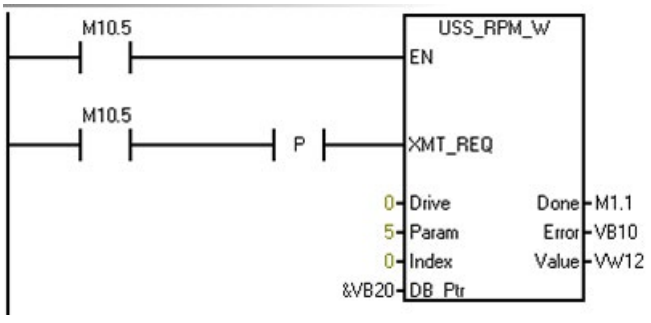
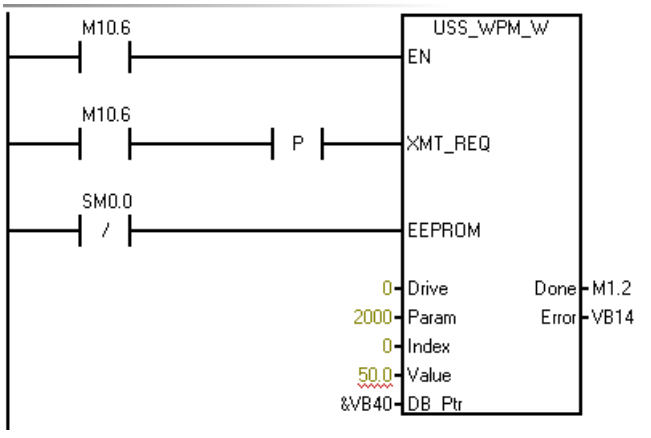
Parameter	Description
XMT_REQ (transmit request)	When ON, a USS_RPM_x request is transmitted to the drive on every scan.
Drive	Address of the drive to which the USS_RPM_x command is to be sent. Valid addresses of individual drives are 0 to 31.
Param	Parameter number
Index	Index value of the parameter that is to be read

Parameter	Description
DB_Ptr	The address of a 16-byte buffer must be supplied to the "DB_Ptr" input. This buffer is used by the USS_RPM_x instruction to store the results of the command issued to the drive.
Done	Turned on when the USS_RPM_x instruction completes
Error	This output byte contains the result of executing the instruction. The USS protocol execution error codes (Page 588) define the error conditions that could result from executing the instruction.
Value	Parameter value returned

When the USS_RPM_x instruction completes, the "Done" output is turned on and the "Error" output byte and the "Value" output contain the results of executing the instruction. The "Error" and "Value" outputs are not valid until the "Done" output turns on.

USS_RPM_x and USS_WPM_x example program

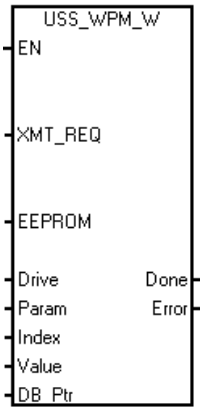
Table 9- 41 USS_RPM_x and USS_WPM_x example program

Network 1	Network 1
	**LD M10.5** **= L60.0** **LD M10.5** **EU** **= L63.7** **LD L60.0** **CALL USS_RPM_W, L63.7, 0, 5, 0, &VB20, M1.1, VB10, VW12**
Network 2	Network 2
	**LD M10.6** **= L60.0** **LD M10.6** **EU** **= L63.7** **LDN SM0.0** **= L63.6** **LD L60.0** **CALL USS_WPM_W, L63.7, L63.6, 0, 2000, 0, 50.0, &VB40, M1.2, VB14**

Refer to "Using the USS protocol instructions" (Page 577) for and a listing of USS protocol instructions and error codes and example programs.

9.7.2.5 USS_WPM_x instruction

Table 9- 42 USS_WPM_x instructions

LAD / FBD	STL	Description
	**CALL USS_WPM_W, XMT_REQ, EEPROM, Drive, Param, Index, Value, DB_Ptr, Done, Error** **CALL USS_WPM_D, XMT_REQ, EEPROM, Drive, Param, Index, Value, DB_Ptr, Done, Error** **CALL USS_WPM_R, XMT_REQ, EEPROM, Drive, Param, Index, Value, DB_Ptr, Done, Error**	There are three write instructions for the USS protocol: • USS_WPM_W instruction writes an unsigned word parameter. • USS_WPM_D instruction writes an unsigned double word parameter. • USS_WPM_R instruction writes a floating-point parameter. Only one read (USS_RPM_x) or write (USS_WPM_x) instruction can be active at a time.

The USS_WPM_x transactions complete when the drive acknowledges receipt of the command or when an error condition is posted. The logic scan continues to execute while this process awaits a response.

Table 9- 43 Valid operands for the USS_WPM_x instructions

Inputs/outputs	Data type	Operands
XMT_REQ	BOOL	I, Q, M, S,SM,T,C,V,L, Power Flow conditioned by a rising edge detection element
EEPROM	BOOL	I, Q, M, S, SM, T, C, V, L, Power Flow
Drive	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD, Constant
Param, Index	WORD	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AIW, *VD, *AC, *LD, Constant
DB_Ptr	DWORD	&VB
Value	WORD DWORD, REAL	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AQW, *VD, *AC, *LD VD, ID, QD, MD, SD, SMD, LD, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC. *VD, *AC, *LD

The "EN" bit must be on to enable transmission of a request, and should remain on until the "Done" bit is set, signaling completion of the process. For example, a USS_WPM_x request is transmitted to the drive on each scan when "XMT_REQ" input is on. Therefore, the "XMT_REQ" input should be pulsed on through an edge detection element which causes one request to be transmitted for each positive transition of the "EN" input.

Table 9- 44 USS_WPM_x parameter descriptions

Parameter	Description
XMT_REQ (transmit request)	When ON, a USS_WPM_x request is transmitted to the drive on every scan.
EEPROM	This input enables writing to both RAM and EEPROM of the drive when it is on and only to the RAM when it is off.
Drive	Address of the drive to which the USS_WPM_x command is to be sent. Valid addresses of individual drives are 0 to 31.
Param	Parameter number
Index	Index value of the parameter that is to be written
Value	Parameter value to be written to the RAM in the drive.
DB_Ptr	The address of a 16-byte buffer must be supplied to the "DB_Ptr" input. This buffer is used by the USS_RPM_x instruction to store the results of the command issued to the drive.
Done	Turned on when the USS_RPM_x instruction completes
Error	This output byte contains the result of executing the instruction. The USS protocol execution error codes (Page 588) define the error conditions that could result from executing the instruction.

When the USS_WPM_x instruction completes, the "Done" output is turned on and the "Error" output byte contains the result of executing the instruction. The "Error" output is not valid until the "Done" output turns on.

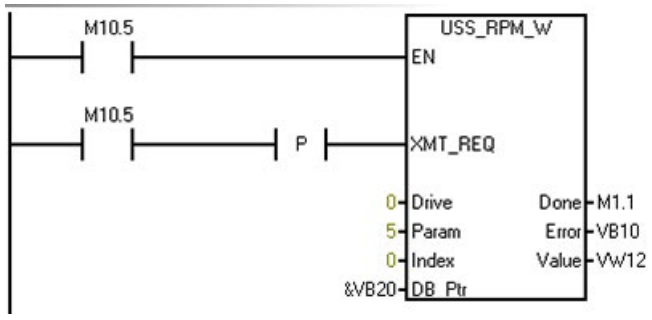
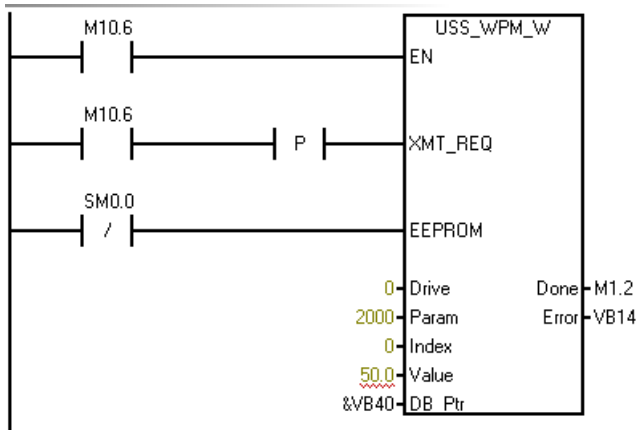
EEPROM

When the "EEPROM" input is turned on, the instruction writes to both the RAM and the EEPROM of the drive. When the input is turned off, the instruction writes only to the RAM of the drive.

NOTICE
Do not exceed the maximum number of write cycles to the EEPROM When you use an USS_WPM_x instruction to update the parameter set stored in drive EEPROM, you must ensure that the maximum number of write cycles (approximately 50,000) to the EEPROM is not exceeded. Exceeding the maximum number of write cycles will result in corruption of the stored data and subsequent data loss, and possible property damage. The number of read cycles is unlimited. Do not exceed the maximum number of write cycles to the EEPROM.

USS_RPM_x and USS_WPM_x example program

Table 9- 45 USS_RPM_x and USS_WPM_x example program

Network 1	Network 1
	LD M10.5 = L60.0 LD M10.5 EU = L63.7 LD L60.0 CALL USS_RPM_W, L63.7, 0, 5, 0, &VB20, M1.1, VB10, VW12
Network 2	**Network 2**
	LD M10.6 = L60.0 LD M10.6 EU = L63.7 LDN SM0.0 = L63.6 LD L60.0 CALL USS_WPM_W, L63.7, L63.6, 0, 2000, 0, 50.0, &VB40, M1.2, VB14

Refer to "Using the USS protocol instructions" (Page 577) for and a listing of USS protocol instructions and error codes and example programs.

9.7.2.6 USS protocol execution error codes

Table 9- 46 USS protocol execution error codes

Error code	Description
0	No error
1	Drive did not respond.
2	A checksum error in the response from the drive was detected.
3	A parity error in the response from the drive was detected.
4	An error was caused by interference from the user program.
5	An illegal command was attempted.
6	An illegal drive address was supplied.
7	The communications port was not set up for USS protocol.
8	The communications port is busy processing an instruction.
9	The drive speed input is out-of-range.
10	The length of the drive response is incorrect.
11	The first character of the drive response is incorrect.
12	The length character in the drive response is not supported by USS instructions.
13	The wrong drive responded.
14	The DB_Ptr address supplied is incorrect.
15	The parameter number supplied is incorrect.
16	An invalid protocol was selected.
17	USS is active; change is not allowed.
18	An illegal baud rate was specified.
19	No communications: the drive is not ACTIVE.
20	The parameter or value in the drive response is incorrect or contains an error code.
21	A double word value was returned instead of the word value requested.
22	A word value was returned instead of the double word value requested.
23	Invalid port number
24	Signal board (SB) port 1 is missing or not configured.

Refer to "Using the USS protocol instructions" (Page 577) for and a listing of USS protocol instructions and error codes and example programs.

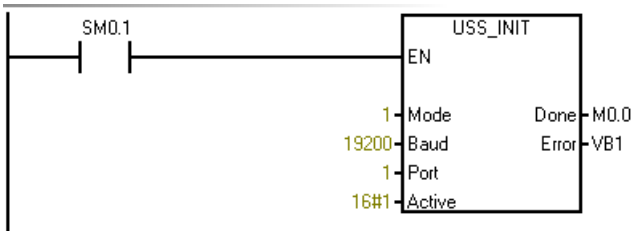
9.7.2.7 USS protocol example program

Table 9- 47 Sample USS program

Network 1	Network 1:

Initialize USS protocol: On the first scan, enable USS protocol for port 1 at 19200 with drive address "0" active.

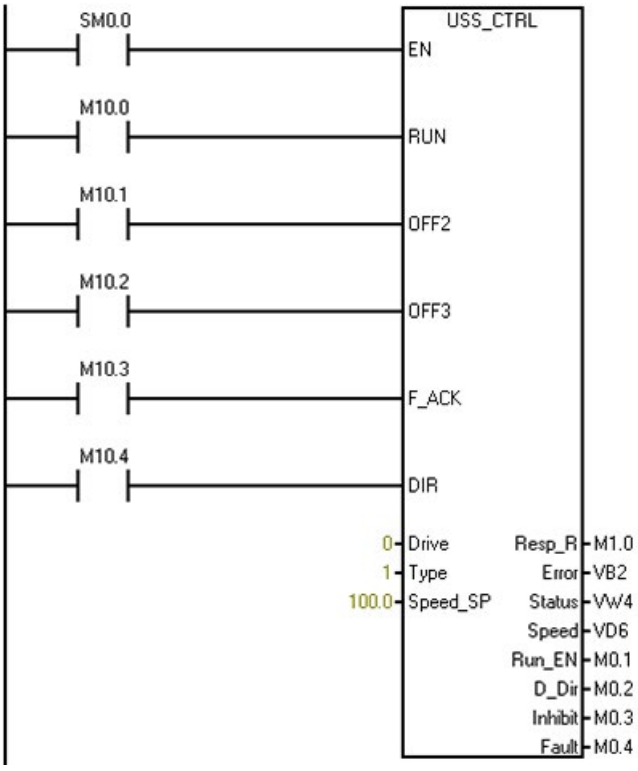
```
LD SM0.1
CALL USS_INIT, 1, 19200, 16#00000001,
Q0.0, VB1
```



Network 2

Network 2:
Control parameters for Drive 0

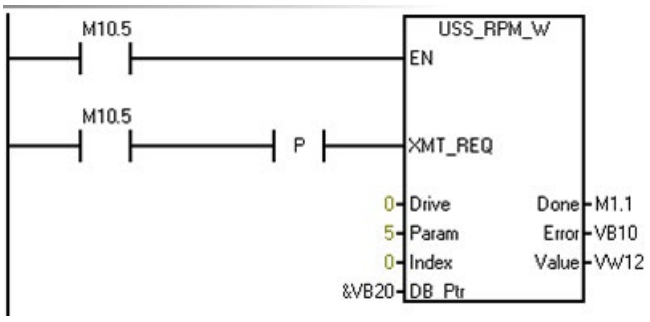
```
LD SM0.0
CALL USS_CTRL, M10.0, M10.1, M10.2,
M10.3, M10.4, 0, 1, 100.0, M1.0, VB2,
VW4, VD6, M0.1, M0.2, M0.3, M0.4
```



Network 3

Network 3:
Read a Word parameter from Drive 0.
Read parameter 5, index 0:

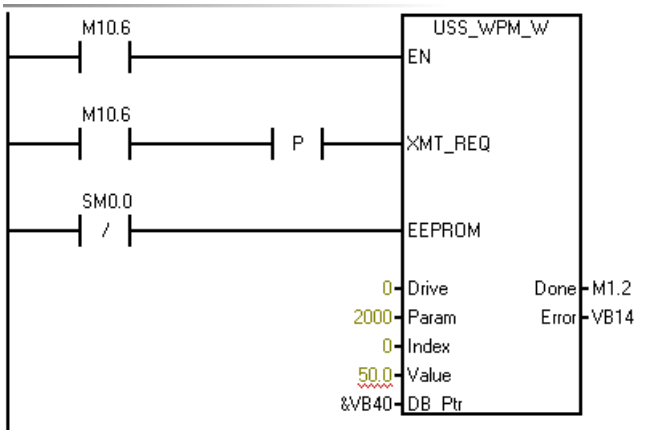
1. Save the state of M10.5 to a temporary location so that this network displays in LAD.
2. Save the rising edge pulse of I0.5 to a temporary L location so that it can be passed to the subroutine.




```
LD M10.5
= L60.0
LD M10.5
EU
= L63.7
LD L60.0
CALL USS_RPM_W, L63.7, 0, 5, 0, &VB20,
M1.1, VB10, VW12
```

Network 4

Network 4:
Write a Word parameter to Drive 0. Write parameter 2000, index 0.
Note: This STL code does not compile to LAD or FBD.




```
LD M10.6
= L60.0
LD M10.6
EU
= L63.7
LDN SM0.0
= L63.6
LD L60.0
CALL USS_WPM_R, L63.7, L63.6, 0, 2000,
0, 50.0, &VB40, M1.2, VB14
```

Refer to "Using the USS protocol instructions" (Page 577) for and a listing of USS protocol instructions and error codes and example programs.

9.8 SINAMICS Library

The SINAMICS library includes pre-configured subroutines that make controlling the drives easier. You can control the position and speed of physical drive, and read or modify the drive parameters with the SINAMICS library.

STEP 7-Micro/WIN SMART provides the following two groups of SINAMICS library instructions:

- SINAMICS_Control:
  - SINA_POS (Page 592): Control the drive position with 8 different operating modes
  - SINA_SPEED (Page 628): Control the drive speed
- SINAMICS_Parameter:

  SINA_PARA_S (Page 635): Read the parameters from the drive or modify the parameters to the drive

Open the Libraries folder in the Instruction folder of the project tree for access to the SINAMICS library instructions. When you place a SINAMICS library instruction in your program, STEP 7-Micro/WIN SMART places one or more associated subroutines in your project.

SINAMICS_Control

SINAMICS_Control uses the following program entities:

- 2 subroutines:
  - SINA_POS
  - SINA_SPEED
- 3867 bytes of program space
- A 188-byte block of V memory for the instruction symbols

SINAMICS_Parameter

SINAMICS_Parameter uses the following program entities:

- 4 subroutines:
  - PN_RD_REC_PARA_S
  - PN_WR_REC_PARA_S
  - SINA_PARA_S
  - ERROR_HANDLER
- 5050 bytes of program space
- A 1314-byte block of V memory for the instruction symbols

Note

For the SINAMICS_Parameter instruction, if the program is large, you need to use the CPU ST30, ST40 and ST60.

9.8.1 SINA_POS instruction

9.8.1.1 Prerequisite of using the SINA_POS instruction

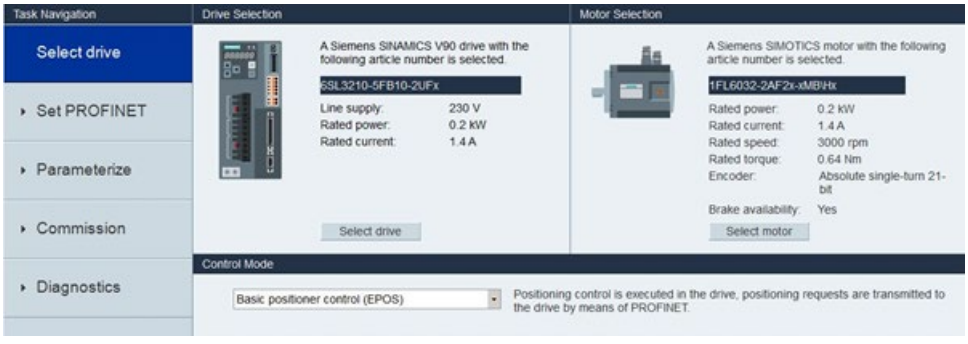
The prerequisite for using the SINA_POS instruction is as follows:

- The SINAMICS V90 PN drive and the servo motor are ready.
- The PROFINET network is connected between the drive and the S7-200 SMART CPU.
- The software V-assistant is connected with SINAMICS V90 PN. For the detailed information, refer to SINAMICS V-ASSISTANT Online Help (https://support.industry.siemens.com/cs/ww/en/view/109738316). You can download the SINAMICS V-ASSISTANT software (https://support.industry.siemens.com/cs/ww/en/view/109738387) and the SINAMICS V90: PROFINET GSD file (https://support.industry.siemens.com/cs/ww/en/view/109737269) in the SIEMENS Industry Online Support Website.

Procedure for configuring SINAMICS V90 PN parameters with the V-assistant

Configure the SINAMICS V90 PN parameters with the V-assistant as follows:

1. Start the V-assistant software.
2. Click "Online" to select the working mode.
3. Click the connected drive and click the "OK" button.
4. Click "Select drive" from the navigation tree, and select "Basic positioner control (EPOS)" in the "Control Mode" field.

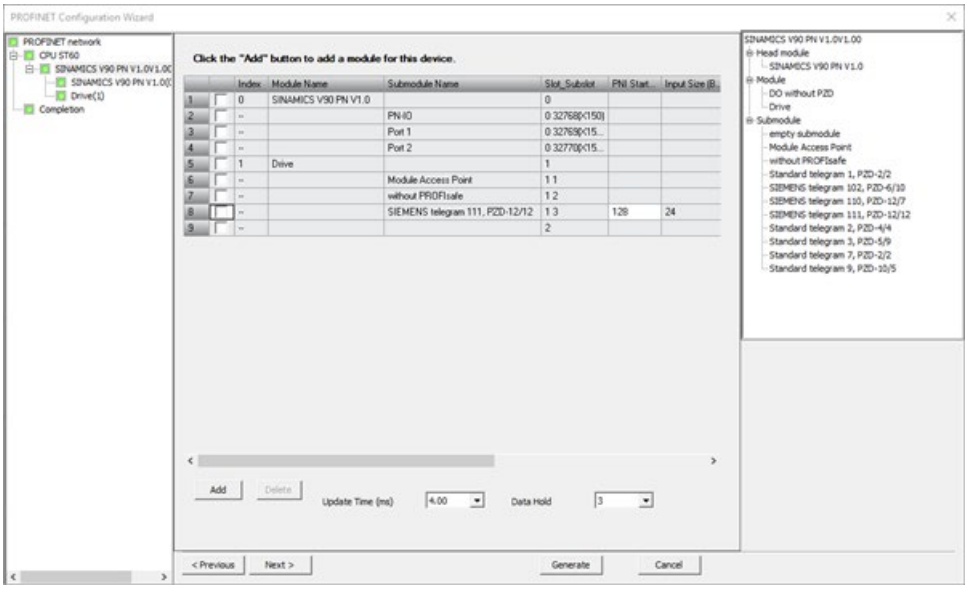


5. Click "Set PROFINET" from the navigation tree and click "Select telegram".

6. In the "Selection of telegram" field, select "SIEMENS telegram 111" as the current telegram. The SINA_POS instruction only supports SIEMENS telegram 111.

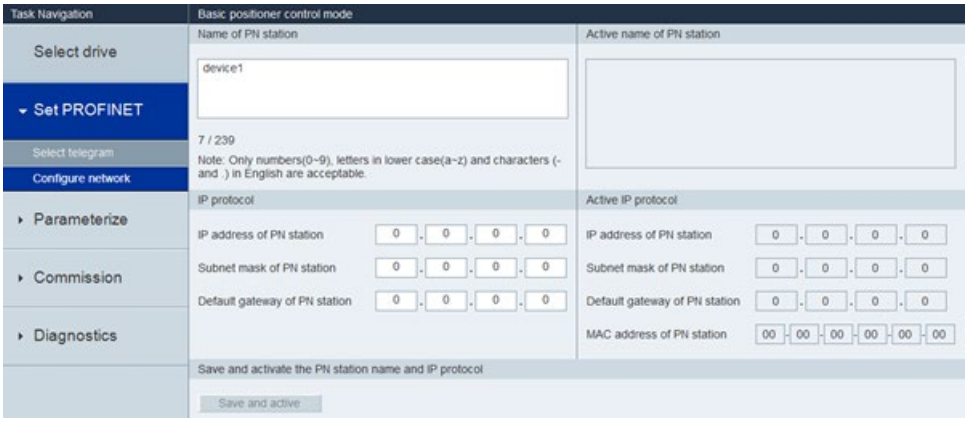
Note

When you configure the PROFINET network in PROFINET wizard, keep the telegram consistent with that in this step.



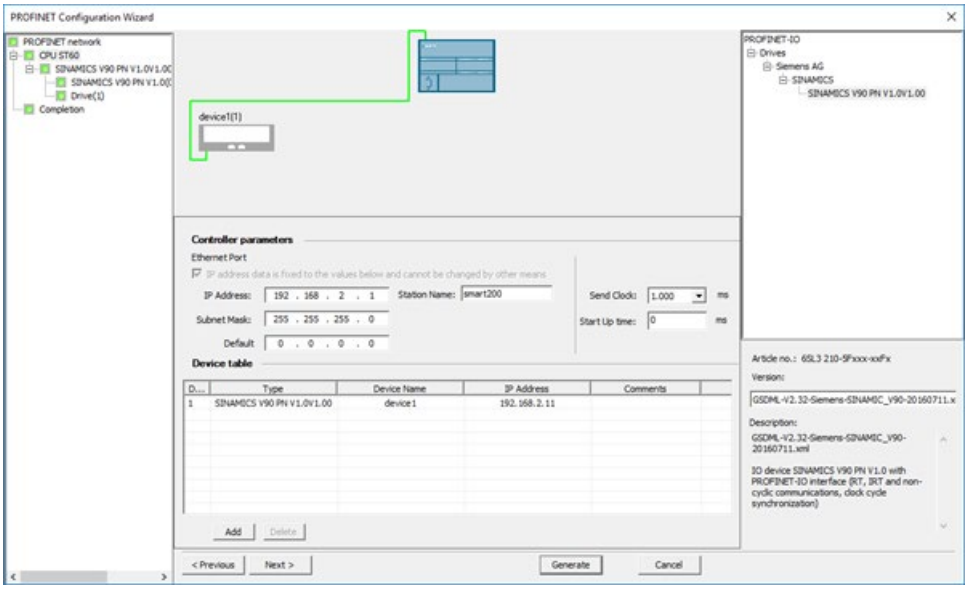
7. Click "Configure network" from the navigation tree.

8. Define the PN station name in the "Name of PN station" field.



Note

When you configure the PROFINET network in PROFINET wizard, keep the device name consistent with the PN station name in this step.



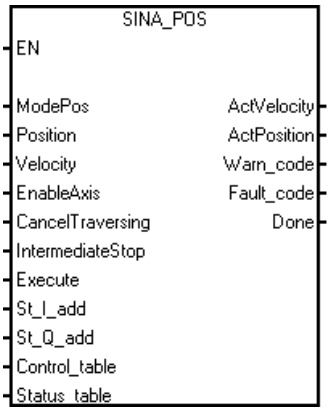
9. Click the "Save and active" button.

Result: The drive automatically restarts and the configured parameters take effect.

9.8.1.2 Input and output interface of SINA_POS instruction

The SINA_POS instruction can control and set the position for drives.

Table 9- 48 SINA_POS instruction

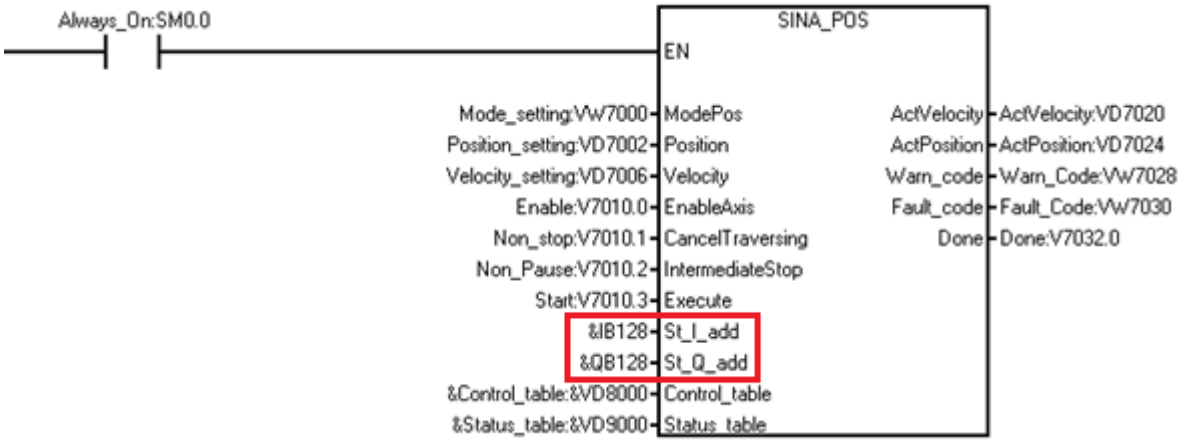
LAD/ FBD	STL	Description
	CALL SINA_POS, ModePos, Position, Velocity, EnableAxis, CancelTraversing, IntermediateStop, Execute, St_I_add, St_Q_add, Control_table, Status_table, ActVelocity, ActPosition, Warn_code, Fault_code, Done	The SINA_POS instruction enables the position control of the drive movement.

Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing.

You must enter an ampersand (&) at the beginning of the input operand and keep the offset consistent with that in the PROFINET wizard.

You can take the Relative positioning for example:



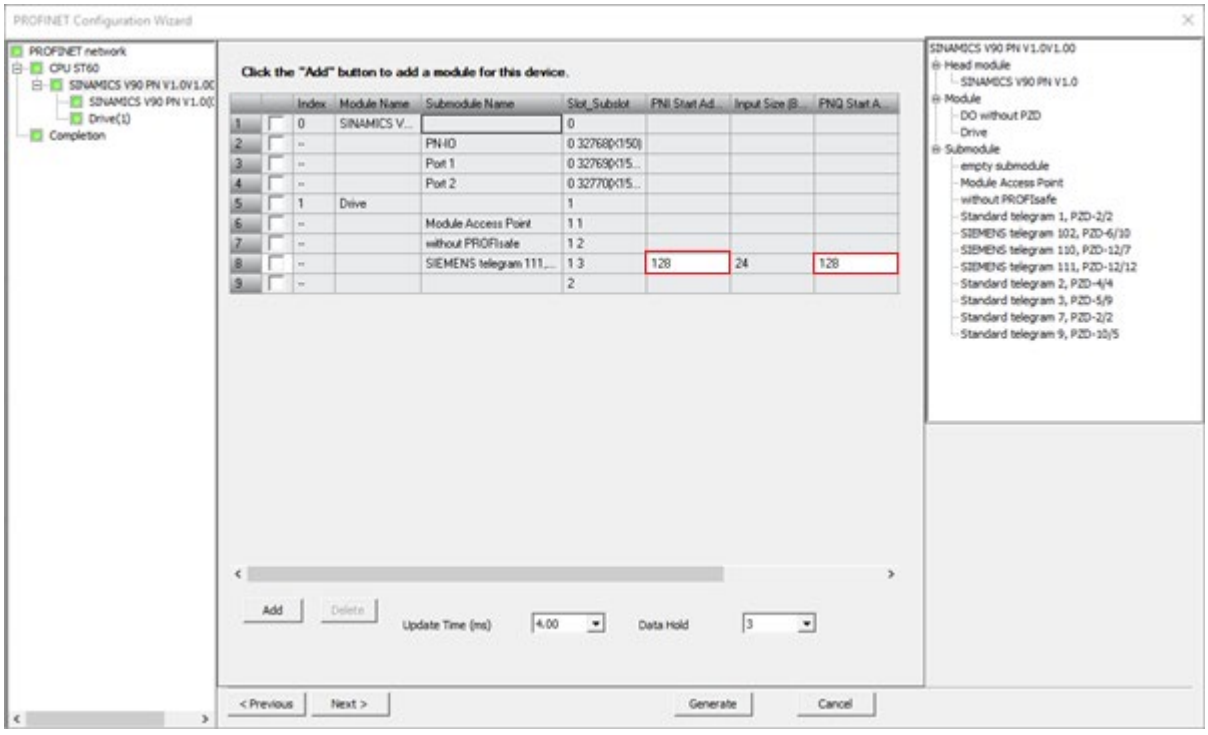


Table 9- 49 Parameters of SINA_POS instruction

Parameter and type		Data type	Description
ModePos	IN	INT	Operating mode: 1 = relative positioning 2 = absolute positioning 3 = positioning as setup 4 = referencing (active homing) 5 = referencing (set reference point) 6 = traversing block 0 – 15 7 = jog mode 8 = incremental jog
Position	IN	DINT	Position setpoint in [LU] for direct setpoint input / MDI mode or traversing block number for traversing block mode. (Default = 0)
Velocity	IN	DINT	Velocity in [LU/min] for MDI mode. (Default value = 0 [1000LU/min])
EnableAxis	IN	BOOL	Switching command: 0 = OFF, 1 = ON
CancelTraversing	IN	BOOL	0 = reject active traversing task 1 = do not reject (Default)
IntermediateStop	IN	BOOL	0 = active traversing command is interrupted 1 = no intermediate stop (Default)
Execute	IN	BOOL	Activate traversing task/setpoint acceptance/ activate reference function.
St_I_add	IN	DWORD	Pointer of I memory area start address for PROFINET IO. For example, &IB128.

Parameter and type		Data type	Description
St_Q_add	IN	DWORD	Pointer of Q memory area start address for PROFINET IO. For example, &QB128.
Control_table	IN	DWORD	Pointer of the start address of control_table (Page 597). For example, &VD8000.
Status_table	IN	DWORD	Pointer of the start address of Status_table (Page 599). For example, &VD9000.
ActVelocity	OUT	DWORD	Actual velocity
ActPosition	OUT	DWORD	Actual position in LU
Warn_code	OUT	WORD	The warning code information from V90. For detailed information, refer to *SINAMICS V90, SIMOTICS S-1FL6 Operating Instruction*.
Fault_code	OUT	WORD	The fault code information from V90. For detailed information, refer to *SINAMICS V90, SIMOTICS S-1FL6 Operating Instruction*.
Done	OUT	BOOL	Target position is reached when the operating mode is relative positioning or absolute positioning.

Definition of "Control_table" parameters

The parameters of "Control_table" are as follows:

Table 9- 50 Control_table parameter

Byte Offset	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	Reserved	Reserved	AckError [1]	FlyRef [2]	Jog2 [3]	Jog1 [4]	Negative [5]	Positive [6]
1	Reserved							
2	OverV [7]							
3								
4	OverAcc [8]							
5								
6	OverDec [9]							
7								
8	ConfigEpos [10]							
9								
10								
11								

[1] AckError: Acknowledging errors. (1= Acknowledging errors is valid, 0 =Acknowledging errors is invalid)

[2] FlyRef: flying reference selection (1 = Reference point setting is activated, 0 = Reference point setting is not activated)

[3] Jog2: Jog signal source 2 (1= positive jog is activated, 0 = positive jog is not activated)

[4] Jog1: Jog signal source 1 (1= negative jog is activated, 0 = negative jog is not activated)

[5] Negative: Negative direction (1 = negative rotation is activated, 0 = negative rotation is not activated)

[6] Positive: Positive direction (1 = positive rotation is activated, 0 = positive rotation is not activated)

[7] OverV: Velocity override is active for all modes. The value range is 0%- 199% and the default value is 100%. For example, you can set OverV as 60%.

[8] OverAcc: Acceleration override is active. The value range is 0%-100% and the default value is 100%. For example, you can set OverAcc as 70%.

[9] OverDec: Deceleration override is active. The value range is 0%-100% and the default value is 100%. For example, you can set OverAcc as 50%.

[10] ConfigEpos: One input to control the EPos functions that are not directly specified at the block. For the detailed information, refer to Description of the configuration input "ConfigEPos" (Page 598).

Description of the configuration input "ConfigEPos"

The following table lists the bit mapping between "ConfigEpos" and "Telegram 111":

ConfigEpos	Telegram 111
ConfigEPos.%X0	STW1.%X1
ConfigEPos.%X1	STW1.%X2
ConfigEPos.%X2	EPosSTW2.%X14
ConfigEPos.%X3	EPosSTW2.%X15
ConfigEPos.%X4	EPosSTW2.%X11
ConfigEPos.%X5	EPosSTW2.%X10
ConfigEPos.%X6	EPosSTW2.%X2
ConfigEPos.%X7	STW1.%X13
ConfigEPos.%X8	EPosSTW1.%X12
ConfigEPos.%X9	STW2.%X0
ConfigEPos.%X10	STW2.%X1
ConfigEPos.%X11	STW2.%X2
ConfigEPos.%X12	STW2.%X3
ConfigEPos.%X13	STW2.%X4
ConfigEPos.%X14	STW2.%X7
ConfigEPos.%X15	STW1.%X14
ConfigEPos.%X16	STW1.%X15
ConfigEPos.%X17	EPosSTW1.%X6
ConfigEPos.%X18	EPosSTW1.%X7
ConfigEPos.%X19	EPosSTW1.%X11
ConfigEPos.%X20	EPosSTW1.%X13
ConfigEPos.%X21	EPosSTW2.%X3
ConfigEPos.%X22	EPosSTW2.%X4
ConfigEPos.%X23	EPosSTW2.%X6
ConfigEPos.%X24	EPosSTW2.%X7
ConfigEPos.%X25	EPosSTW2.%X12
ConfigEPos.%X26	EPosSTW2.%X13

ConfigEpos	Telegram 111
ConfigEPos.%X27	STW2.%X5
ConfigEPos.%X28	STW2.%X6
ConfigEPos.%X29	STW2.%X8
ConfigEPos.%X30	STW2.%X9

Definition of "Status_table" parameters

The definition of the "Status_table" bits is as follows:

Table 9- 51 Status_table

Byte offset	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	Reserved	Over-range_Error [1]	AxisError [2]	AxisWarn [3]	Lockout [4]	AxisRef [5]	AxisPosOk [6]	Axisena-bled [7]
1	Error ID [8]							
2	Actmode [9]							
3								
4	Epos_zsw1 [10]							
5								
6	Epos_zsw2 [11]							
7								

[1] Overrange_Error: The data you enter is out of the range. For detailed information, refer to Error code 3, 4, 5 (Page 600).

[2] AxisError: The drive has an error. (Default = 0)

[3] AxisWarn: Drive alarm is active. (Default = 0)

[4] Lockout: Switching-on inhibit. (Default = 0)

[5] AxisRef: Reference point set. (Default = 0)

[6] AxisPosOk: Target position of the axis is reached. (Default = 0)

[7] Axisenabled: Drive is ready and switched on. (Default = 0)

[8] Error ID: Identify the error type. For the detailed information, refer to Error codes for the "Status_table" parameter (Page 600).

[9] Actmode: Currently active mode. (Default = 0)

[10] Epos_zsw1: Status of EPos_zsw1 (bit-granular). For the detailed information, refer to Assignment of "Epos_zsw1" (Page 600).(Default = 0)

[11] Epos_zsw2: Status of EPos_zsw2 (bit-granular). For the detailed information, refer to Assignment of "Epos_zsw2" (Page 601).(Default = 0)

Error codes for the "Status_table" parameter

The following table lists the error code of the "Status_table" parameter:

Table 9- 52 Error codes for the "Status_table" parameter

Error Code	Description
0	No error.
1	An error from the drive is detected.
2	The drive is disabled.
3	The selected mode is not supported.
4	The rate of parameters OverV, OverAcc and OverDec exceeds the supported value range.
5	The selected block is out of range under the motion mode "traversing block".

Assignment of "Epos_zsw1"

The following table lists the assignment information of "Epos_zsw1":

Table 9- 53 Epos_zsw1

Bit	Abbr.	Designation	Drive parameter	Function chart
0	ActTrvBit0	Active traversing block, bit 0	r2670.0	3650
1	ActTrvBit1	Active traversing block, bit 1	r2670.1	3650
2	ActTrvBit2	Active traversing block, bit 2	r2670.2	3650
3	ActTrvBit3	Active traversing block, bit 3	r2670.3	3650
4	ActTrvBit4	Active traversing block, bit 4	r2670.4	3650
5	ActTrvBit5	Active traversing block, bit 5	r2670.5	3650
6	Bit6	Reserved		
7	Bit7	Reserved		
8	StpCamMinAct	STOP cam minus active	r2684.13	3630
9	StpCamPlsAct	STOP cam plus active	r2684.14	3630
10	JogAct	Jog mode is active	r2094.0 [1]	2460
11	RefAct	Reference point approach mode active	r2094.1 [1]	2460
12	FlyRefAct	Flying referencing active	r2684.1 [1]	3630
13	TrvBlAct	Traversing blocks mode active	r2094.2 [1]	2460
14	MdiStupAct	In the direct setpoint input / MDI mode, setup is active	r2094.4 [1]	2460
15	MdiPosAct	In the direct setpoint input / MDI mode, positioning is active	r2094.3 [1]	2460

[1] r2669 (function diagram 3630) displays bit-granular. P2099[0] = r2699 is interconnected at the input of the connector-bivector converter for this purpose.

Assignment of "Epos_zsw2"

The following table lists the assignment information of "Epos_zsw2":

Table 9- 54 Epos_zsw2

Bit	Abbr.	Designation	Drive parameter	Function chart
0	TrkModeAct	Follow-up/tracking mode active	r2683.0	3645
1	VeloLimAct	Velocity limitation active	r2683.1	3645
2	SetPStat	Setpoint static	r2683.2	3645
3	PrntMrkOut	Print mark outside outer window	r2683.3	3614
4	FWD	Axis moves forward	r2683.4	3635
5	BWD	Axis moves backward	r2683.5	3635
6	SftSwMinAct	Minus software limit switch actuated	r2683.6	3635
7	SftSwPlsAct	Plus software limit switch actuated	r2683.7	3635
8	PosSmCam1	Position actual value <= cam switching position 1	r2683.8	4025
9	PosSmCam2	Position actual value <= cam switching position 2	r2683.9	4025
10	TrvOut1	Direct output 1 with the traversing block	r2683.10	3616
11	TrvOut2	Direct output 2 with the traversing block	r2683.11	3616
12	FxStpRd	Fixed stop reached	<not used> (r2683.12)	3645
13	FxStpTrRd	Fixed stop clamping torque reached	<not used> (r2683.13)	3645
14	TrvFxStpAct	Travel to fixed stop active	<not used> (r2683.14)	3645
15	CmdAct	Traversing active	r2683.15	3645

9.8.1.3 Mode selection of SINAMICS with the SINA_POS instruction

The "ModePos" input is used for the operating mode selection. There are eight operating modes:

- Relative positioning
- Absolute positioning
- Setup mode
- Referencing (active referencing)
- Referencing (set reference point)
- Traversing blocks
- Jog
- Incremental jog

Basic requirements

The input operands "CancelTraversing" and "IntermediateStop" are relevant for all modes except for jog and must be set to "1" when using SINAMICS.

In each operating mode, follow these steps to enable the drive:

- Set the input operand "CancelTraversing" to 1.
- Set the input operand "Intermediatestop" to 1.
- In the Control_table, set "ConfigEpos" to 3 according to Decimal numeral system.

To enable the axis, set the input operand "EnableAxis" to 1.

You can use the input operand "ModePos" to set or change the operating mode.

9.8.1.4 Relative positioning

Relative positioning mode enables the motor axis start the positioning motion relative to the start position. The distance is incremental in each movement.

The Relative positioning mode is implemented with the "MDI relative positioning" function of SINAMICS V90 PN. It enables the position-controlled traversing of traversing paths using the integrated position controller of the SINAMICS V90 PN.

Requirements

- The mode is selected with ModePos=1.
- The device is switched on with "EnableAxis".
- The axis does not have to be referenced or the encoder adjusted.
- The axis is at standstill if selected by an operating mode greater than 3. A change within the MDI operating modes (1,2,3) is possible at any time.

Sequence

You configure the traversing path and dynamic responses with the following inputs:

- "Position"
- "Velocity"
- "OverV" (velocity override)
- "OverAcc" (acceleration override)
- "OverDec" (deceleration override)

The velocity override refers to the "Velocity". For example, the velocity that takes effect is 500LU/min and the OverV is 120%, the velocity in SINAMICS V90 is 600LU/min.

You must set the signal inputs "CancelTraversing" and "IntermediateStop" to "1". "Jog1" and "Jog2" have no effect and you must set them to "0" (false).

The direction of travel in relative positioning always results from the sign of the traversing path. For example, if the position is -1000, the direction is negative. If the position is 1000, the direction is positive.

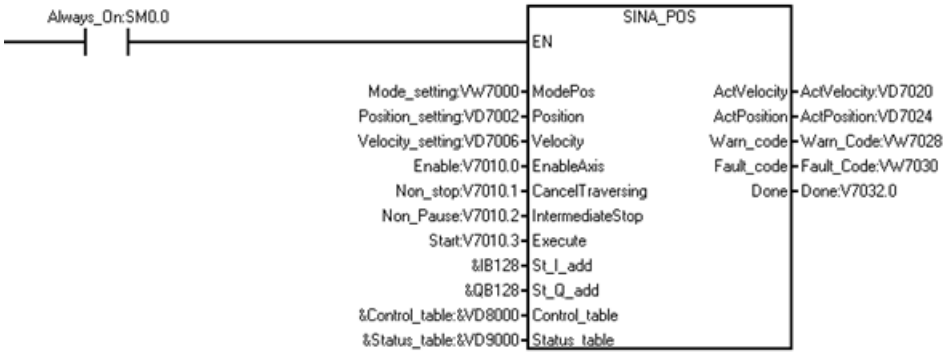
Traversing motion is started with a positive edge at "Execute". You can track the current state of the active command with "EPos_zsw1 / EPos_zsw2".

If the target position is reached, the value of bit "AxisPosOK" in the output signal "Status_table" is 1. If an error occurs during the traversing motion, the bit "AxisError" in the output signal "Status_table" is issued.

Note

The current command can be replaced on the fly by a new command with "ExecuteMode". This is only possible for the "ModePos" 1, 2 or 3 modes.

Example of the Relative positioning mode



Set the Relative positioning mode as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	1		
Position_setting	VD7002	Position length	Position	DWORD	2500		
Velocity_setting	VD7006	Velocity	Velocity	DWORD	500		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermediateStop	BOOL	1		
Start	V7010.3	Start the drive.	Execute	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	VW8002	OverV	100
					VW8004	OverAcc	100
					VW8006	OverDec	100

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
					VD8008	ConfigEpos	3

Note

The variable value in this example is for your reference, and you need to create the variable according to your actual situation.

2. Enter the data in the inputs "St_I_add" and "St_Q_add".

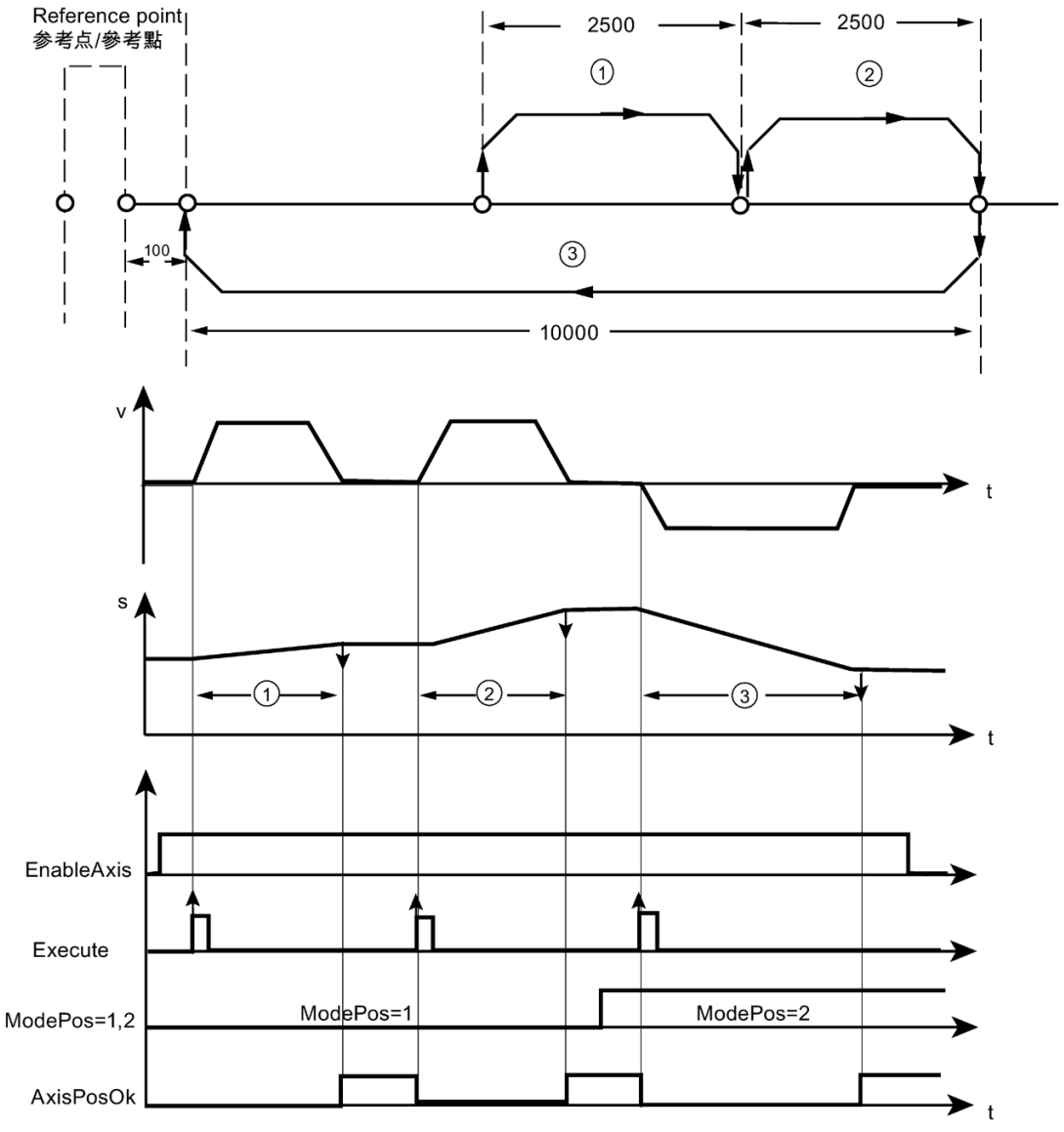
Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

For the input operand "St_I_add" and "St_Q_add", keep the offset consistent with that in the PROFINET wizard.

Result: The drive moves 2500 LU on the basis of the previous position distance. For example, if the previous distance is 5000 LU, the drive stays at the distance of 7500 LU.

The following illustration displays the movement trajectory and dynamic parameters:



① Relative positioning

② Relative positioning

③ Absolute positioning

In this illustration, "v" refers to velocity, "s" refers to position, and "t" refers to time.

9.8.1.5 Absolute positioning

The Absolute positioning mode enables the motor axis start the positioning motion to an absolute position.

The Absolute positioning mode is implemented with the "MDI absolute positioning" function of SINAMICS V90 PN. It enables the absolute movement using the integrated position controller of the SINAMICS V90 PN.

Requirements

- The mode is selected with "ModePos"=2.
- The device is switched on with "EnableAxis".
- The axis must be referenced or the encoder adjusted.
- The axis is at a standstill if selected by an operating mode greater than 3. A change within the MDI operating modes (1,2 or 3) is possible at any time.

Sequence

The traversing path and dynamic responses are specified with the following inputs:

- "Position"
- "Velocity"
- "OverV" (velocity override)
- "OverAcc" (acceleration override)
- "OverDec" (deceleration override)

The velocity override refers to the "Velocity". For example, the velocity that takes effect is 500LU/min and the OverV is 120%, the velocity in SINAMICS V90 is 600LU/min.

You must set the input signal "CancelTraversing" and "IntermediateStop" to "1". "Jog1" and "Jog2" have no effect and you must set them to "0".

The direction of travel in Absolute positioning always results from the shortest distance to the target position. The inputs "Positive " and "Negative" are "0".

Note

You can use the parameters "Positive" or "Negative" to specify a preferred direction to approach the target position for an axis.

Note

You can select the absolute positioning direction with the following positioning control words:

- POS_STW1.9:
- POS_STW1.10:
  - 1 = Absolute positioning/MDI direction selectionpositive
  - 2 = Absolute positioning/MDI direction selectionnegative
  - 3 = Absolute positioning through the shortest distance

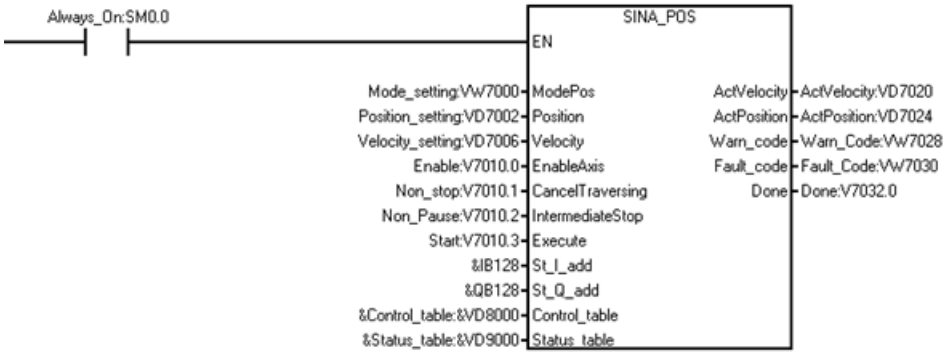
Traversing motion is started with a positive edge at "Execute". You can track the current state of the active command with "EPoszsw1 / EPoszsw2".

If the target position is reached, the value of bit "AxisPosOK" in the output signal "Status_table" is 1. If an error occurs during the traversing motion, the bit "AxisError" in the output signal "Status_table" is issued.

Note

The current command can be replaced on-the-fly by a new command via "Execute". This is only possible for the "ModePos" 1, 2, 3 modes.

Example of the Absolute positioning mode



Set the Absolute positioning mode as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	2		
Position_setting	VD7002	Position length	Position	DWORD	100		
Velocity_setting	VD7006	Velocity	Velocity	DWORD	500		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermidateStop	BOOL	1		
Start	V7010.3	Start the drive.	Execute	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	VW8002	Over V	100
					VW8004	Over Acc	100

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
					VW8006	Over Dec	100
					VD8008	ConfigEpos	3

Note

The variable value in this example is for your reference, and you need to create the variable according to your actual situation.

2. Enter the data in the input "St_I_add" and "St_Q_add".

Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

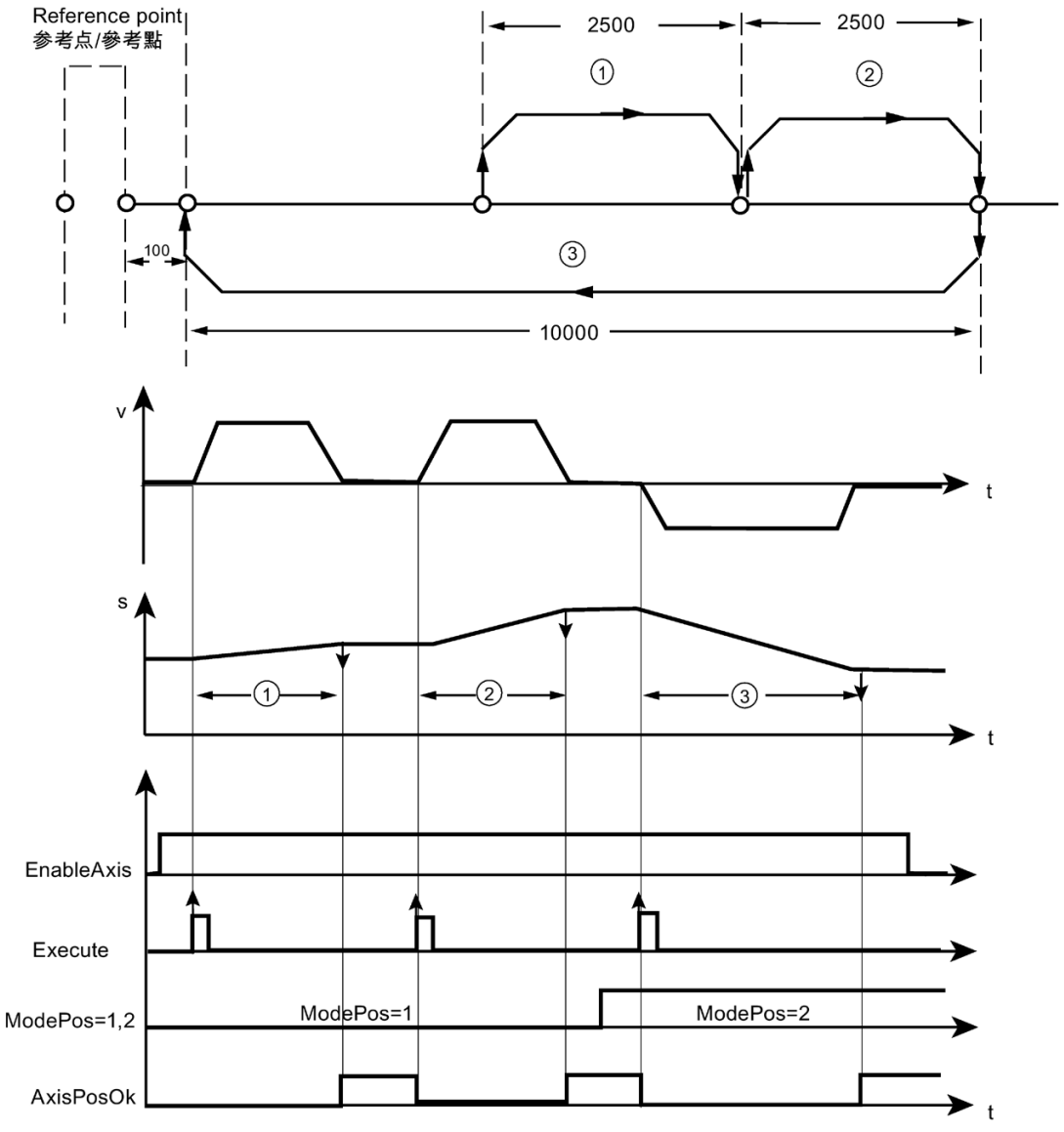
For the input operand "St_I_add" and "St_Q_add", keep the offset consistent with that in PROFINET wizard.

Result: Then the drive stays at the position of 100.

Note

For the operating mode "absolute positioning", you must set a valid reference position by configuring it in the mode "Referencing (active referencing)" or the "Referencing (set reference point)".

The following illustration displays the movement trajectory and dynamic parameters:



① Relative positioning

② Relative positioning

③ Absolute positioning

In this illustration, "v" refers to velocity, "s" refers to position, and "t" refers to time.

9.8.1.6 Setup mode

The Setup mode enables the position-controlled traversing of the axis in the positive or negative direction with constant velocity without specification of a target position with the "MDI set up" function of SINAMICS V90 PN.

Requirements

- The mode is selected with "ModePos" = 3.
- The device is switched on using "EnableAxis".
- The axis does not have to be referenced or the encoder adjusted.
- The axis is at a standstill if the operating mode is greater than 3. A change within the MDI operating modes (1, 2 or 3) is possible at any time.

Sequence

The traversing path and dynamic responses are specified with the following inputs:

- "Positive" or "Negative"
- "Velocity"
- "OverV" (velocity override)
- "OverAcc" (acceleration override)
- "OverDec" (deceleration override)

You must set the input signal "CancelTraversing" and "IntermediateStop" to "1". "Jog1" and "Jog2" have no effect and you must set them to "0".

The travel direction is determined via "Positive" and "Negative". Simultaneous selection stops the axis without further alarms or faults.

Traversing motion is started with a positive edge at "Execute". You can track the current state of the active command with "EPos_zsw1 / EPos_zsw2".

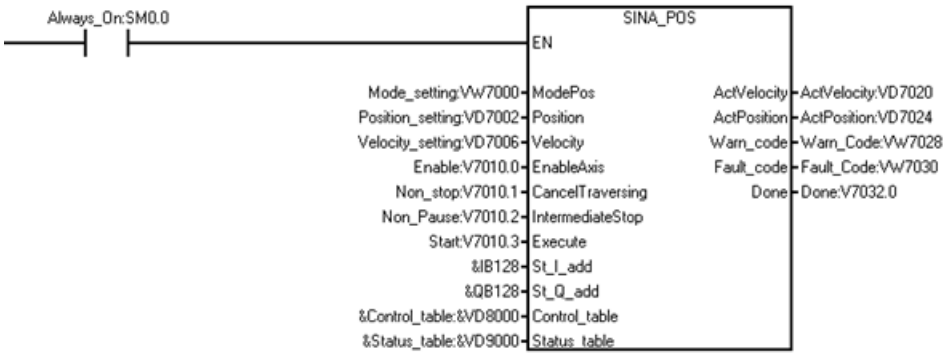
The output signal "AxisPosOk" is set when the setup mode is terminated with reject traversing task and the axis has stopped.

If an error occurs during the traversing motion, the bit "AxisError" in the output signal "Status_table" is issued.

Note

The current command can be replaced on-the-fly by a new command via "Execute". This is only possible for the "ModePos" 1, 2, 3 modes.

Example of the Setup mode



Set the Setup mode as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	3		
Velocity_setting	VD7006	Velocity	Velocity	DWORD	500		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermediateStop	BOOL	1		
Start	V7010.3	Start the drive.	Execute	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	VW8002	OverV	100
					VW8004	OverAcc	100
					VW8006	OverDec	100
					VD8008	ConfigEpos	3
					V8000.0 [1]	Positive	1
					V8000.1 [1]	Negative	0

[1] The value of V8000.0 and V8000.1 cannot be 1 or 0 at the same time.

Note

The variable value in this example is for your reference, and you need to create the variable according to your actual situation.

2. Enter the data in the input "St_I_add" and "St_Q_add".

Note

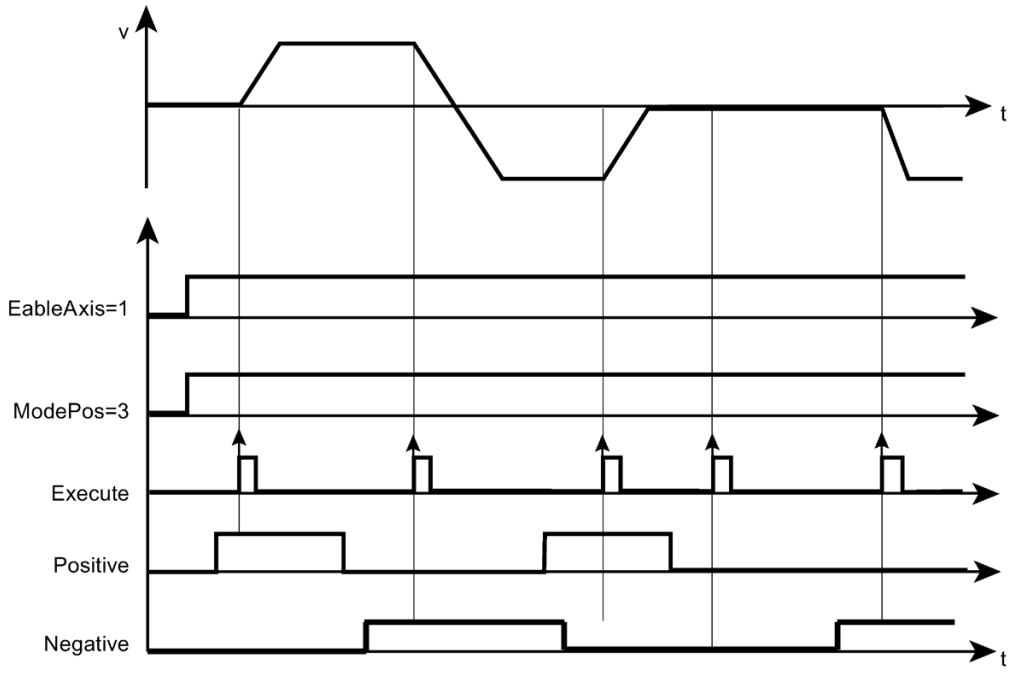
For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

For the input operands "St_I_add" and "St_Q_add", keep the offset consistent with that in the PROFINET wizard.

You have the following options for the variables:

- In the input signal "Control_table" :
  - To make the drive move toward the positive direction, set V8000.0 as 1 and V8000.1 as 0.
  - To make the drive moves toward the negative direction, set V8000.1 as 1 and V8000.0 as 0.
- To stop the drive, set the variable "Non_stop" to 0.
- To pause the drive, set the variable "Non_Pause" to 0.

The following picture displays the movement trajectory and dynamic parameters:



In this illustration, "v" refers to velocity and "t" refers to time.

9.8.1.7 Referencing (active referencing)

The Referencing (active referencing) mode enables the reference point approach of the axis in the positive or negative direction with predefined velocity and reference mode with the "Active referencing" function of SINAMICS V90 PN.

Requirements

- The mode is selected with "ModePos"=4.
- The device is switched on using "EnableAxis".
- The axis is at a standstill.

Sequence

The required velocity is saved as a velocity profile in the SINAMICS V90. Further, the preset acceleration and deceleration values are active in the traversing profile of the axis. The "OverV" velocity override affects the preconfigured traversing velocity. For example, the velocity that takes effect is 500LU/min and the OverV is 120%, the velocity in SINAMICS V90 is 600LU/min.

You must set the input signal "CancelTraversing" and "IntermediateStop" to "1". "Jog1" and "Jog2" have no effect and you must set them to "0".

The travel direction is determined via "Positive" and "Negative". Simultaneous selection is not permitted and results in a fault.

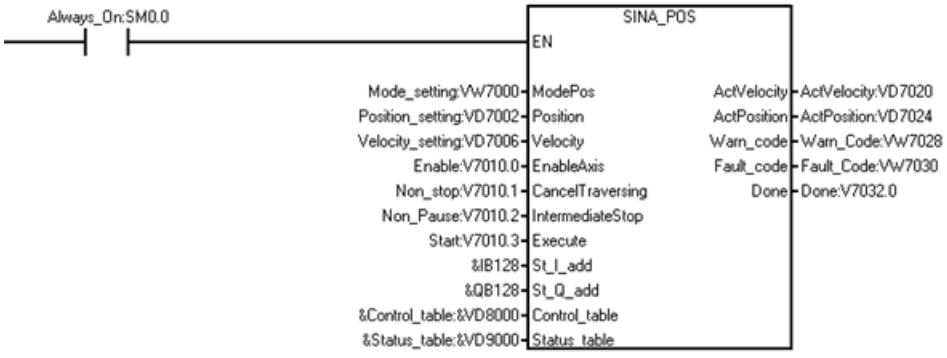
The reference point approach is started with a positive edge at "Execute".

Traversing motion is started with a positive edge at "Execute". You can track the current state of the active command with "EPos_zsw1 / EPos_zsw2".

The bit "AxisRef" in the output signal "Status_table" is set if the reference cam is appropriately found and evaluated.

If an error occurs during traversing motion, the bit "AxisError" in the output signal "Status_table" is issued.

Example of the Referencing (active referencing) mode



Set the operating mode "Referencing (active referencing)" as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	4		
Position_setting	VD7002	Position length	Position	DWORD	2500		
Velocity_setting	VD7006	Velocity	Velocity	DWORD	500		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermediateStop	BOOL	1		
Start	V7010.3	Start the drive.	Execute	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	V8000.0 [1]	Positive	1
					V8000.1 [1]	Negative	0

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
					VD8008	ConfigEpos	3
						V8011.6 (Bit 7)	1

[1] The value of V8000.0 and V8000.1 cannot be 1 or 0 at the same time.

Note

The variable value in this example is for your reference, and you need to create the variable according to your actual situation.

2. Enter the data in the input "St_I_add" and "St_Q_add".

Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

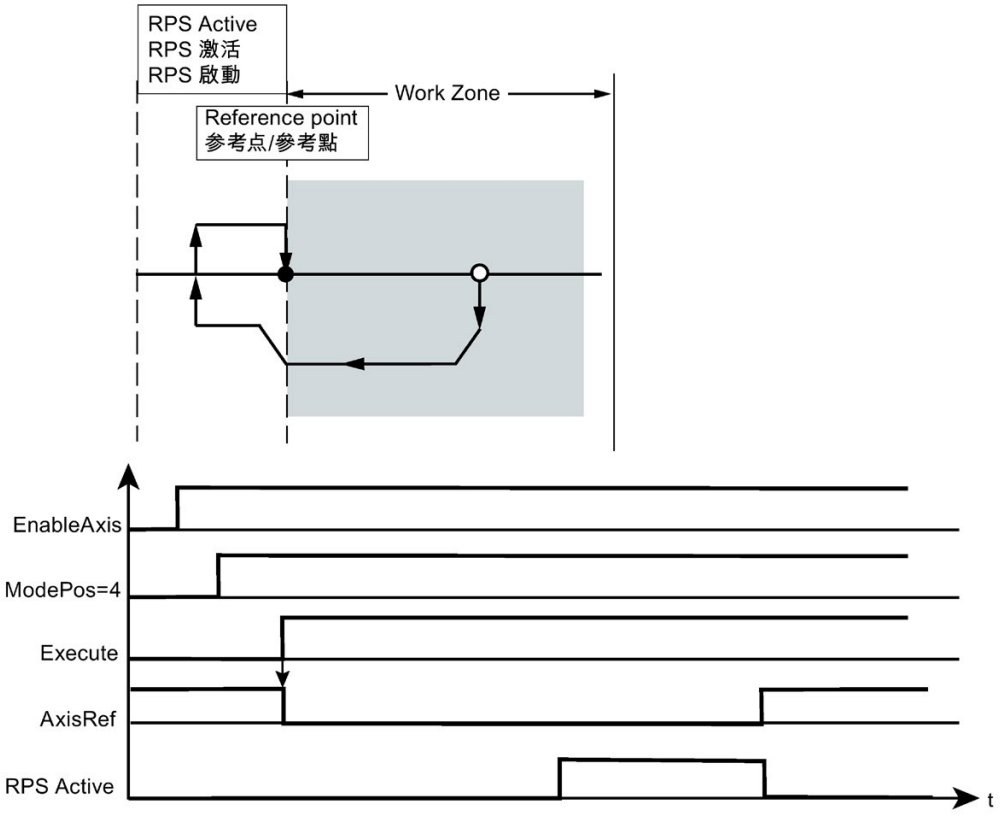
For the input operands "St_I_add" and "St_Q_add", keep the offset consistent with that in the PROFINET wizard.

3. In the variable "Control_table", if you set V8000.0 as 1 and V8000.1 as 0, the drive moves toward the positive direction to find the reference point. If you set V8000.1 as 1 and V8000.0 as 0, the drive moves toward the negative direction to find the reference point.

Note

When the external reference signal connects to PLC directly, you can set V8011.6 as 1 through digital inputs. The drive stops and you set the reference point successfully.

The following picture displays the movement trajectory and dynamic parameters:



In this illustration, "t" refers to time.

9.8.1.8 Referencing (set reference point)

The Referencing (set reference point) mode enables the referencing of the axis at an arbitrary position and is performed through the "Set reference point" function of SINAMICS V90 PN.

Requirements

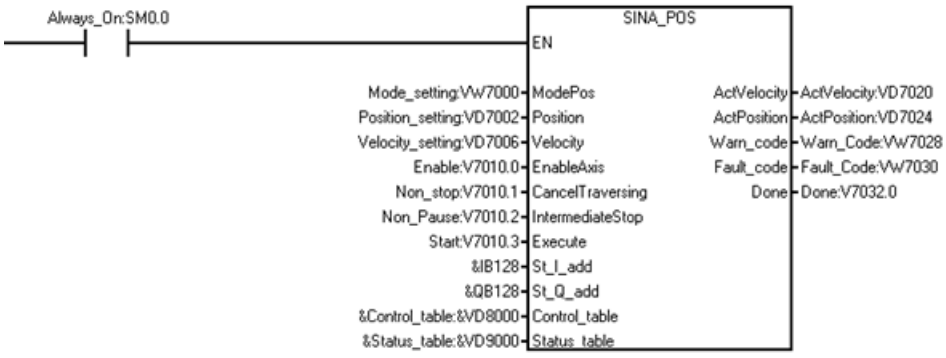
- The mode is selected with "ModePos"=5.
- The axis can be in closed-loop control, but must be at a standstill.

Sequence

The axis is at a standstill and the reference point is set with a positive edge at "Execute".

If an error occurs while setting the reference point, the bit "AxisError" in the output signal "Status_table" is issued.

Example of the Referencing (set reference point) mode



Set the operating mode as "Referencing (set reference point)" as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	5		
Position_setting	VD7002	Position length	Position	DWORD	2500		
Velocity_setting	VD7006	Velocity	Velocity	DWORD	500		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermediateStop	BOOL	1		
Start	V7010.3	Start the drive.	Execute	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	VD8008	ConfigEpos	3
Status_table	VD9000	The status parameters.	Status_table	DWORD	V9000.2	AxisRef	1

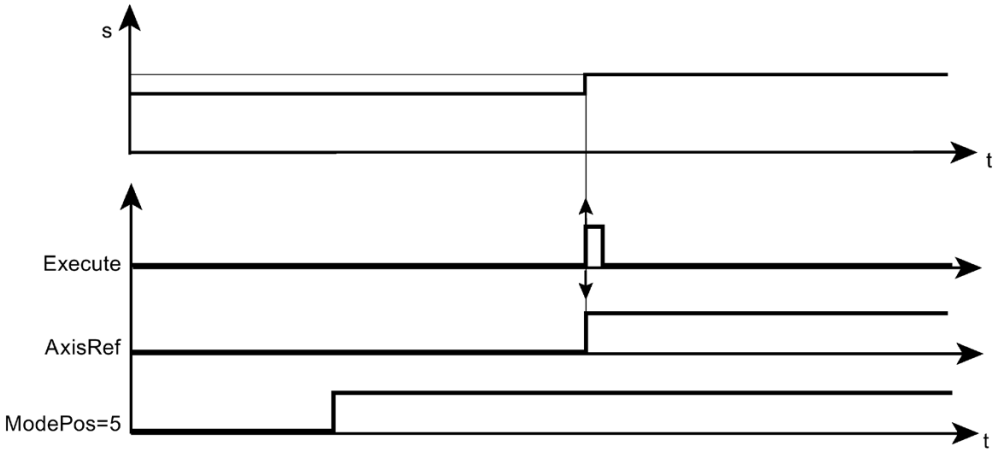
2. Enter the data in the input "St_I_add" and "St_Q_add".

Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

For the input operands "St_I_add" and "St_Q_add", keep the offset consistent with that in PROFINET wizard.

The following picture displays the movement trajectory and dynamic parameters:



In this illustration, "s" refers to position, and "t" refers to time.

9.8.1.9 Traversing blocks

The Traversing blocks mode is implemented through the "Traversing blocks" function of SINAMICS V90 PN.

Requirements

- The mode is selected with "ModePos"=6.
- The device is switched on using "EnableAxis".
- The axis is at a standstill.
- The axis must be referenced or the encoder adjusted.

Sequence

Note

You can use the "Position" input to select the traversing task to start. The value is from 0 to 15.

If the value is out of the range, the error code 5 (Page 600)displays in the "Status_table" bit "Overrange_Error".

The traversing block parameters in the SINAMICS V90 specify the task modes, the target positions and dynamic responses. The velocity override "OverV" refers to the velocity setpoint stored in the traversing block. You must set the operating conditions "CancelTraversing" and "IntermediateStop" to "1". "Jog1" and "Jog2" have no effect and you must set them to "0".

The travel direction results from the task mode and the set position setpoint. The "Positive" and "Negative" are not relevant in this case and you must set them to "0".

Note

You can use the parameters "Positive" or "Negative" to specify a preferred direction to approach the target position for an axis.

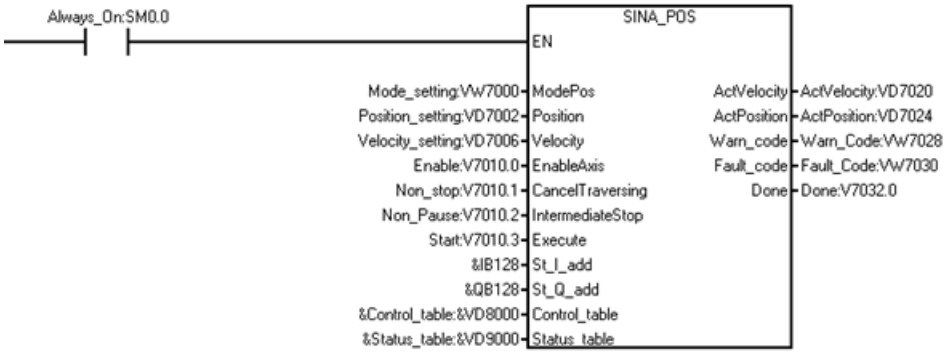
Traversing motion is started with a positive edge at "ExecuteMode". You can track the current state of the active command with "EPos_zsw1 / EPos_zsw2".

The SINA_POS instruction acknowledges when the end of the traversing path is reached successfully with "AxisPosOk". If an error occurs during the traversing motion, the bit "AxisError" in the output signal "Status_table" is issued.

Note

The current command can be replaced on-the-fly by a new command via "Execute". This is only possible for the same operating mode.

Example of the Traversing blocks mode



Set the operating mode as "Traversing blocks" as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	6		
Position_setting	VD7002	Position length	Position	DWORD	Enter the index of the desired blocks. The maximum supported blocks are 16, so the value in this variable is from 0 to 15.		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermediateStop	BOOL	1		
Start	V7010.3	Start the drive.	Execute	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	VD8008	ConfigEpos	3

Note

The variable value in this example is for your reference, and you need to create the variable according to your actual situation.

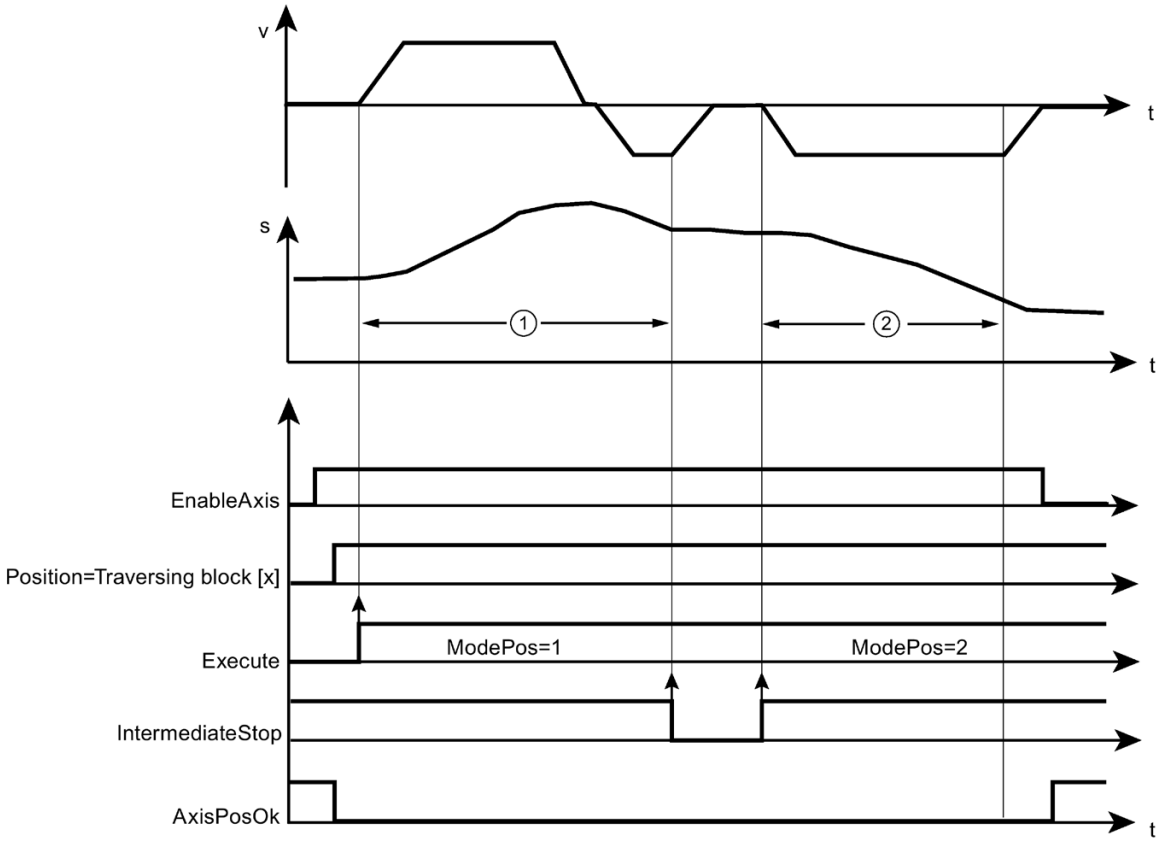
2. Enter the data in the input "St_I_add" and "St_Q_add".

Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

For the input operands "St_I_add" and "St_Q_add", keep the offset consistent with that in the PROFINET wizard.

The following picture displays the movement trajectory and dynamic parameters:



In this illustration, "v" refers to velocity, "s" refers to position, and "t" refers to time.

9.8.1.10 Jog

The Jog mode is implemented using the "Jog" function of SINAMICS V90 PN. It enables the position-controlled, velocity-dependent traversing of axes using the integrated position controller of the SINAMICS V90 PN.

Requirements

- The mode is selected with "ModePos" = 7.
- The device is switched on using "EnableAxis".
- The axis is at standstill.
- The axis does not have to be referenced or adjusted.

Sequence

The specification of the jog velocity is performed via the V-assistant form or the SINA_PARA_S instruction for the configuration of the operating mode in the SINAMICS V90. The SINAMICS V90 uses the acceleration and deceleration for the dynamic responses of the axis.

The velocity override also applies in this operating mode and is set through "OverV". For example, the velocity that takes effect is 500LU/min and the OverV is 120%, the velocity in SINAMICS V90 is 600LU/min.

The input signal "CancelTraversing" and "IntermediateStop" are not relevant for the operating mode and you must set them to "1".

Note

"Jog1" and "Jog2" are the signal sources for the jog mode in SINA_POS. "Jog1" is the signal source for negative, and "Jog2" is the signal source for positive.

You can configure the distance of traversing motion in the SINAMICS V90 PN.

The velocity setpoint sets the travel direction for jogging.

The inputs "Positive" and "Negative" are not relevant for the operating mode and you can set them to "0".

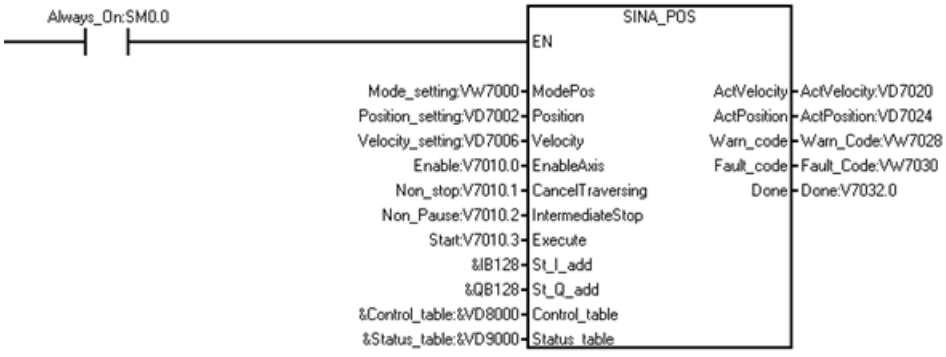
You can track the current state of the active command with "EPos_zsw1 / EPos_zsw2".

The SINA_POS instruction displays the current command processing with "AxisEnabled" and acknowledges the termination of the jog function ("Jog1" or "Jog2" = 0) when the axis is at standstill with "AxisPosOk". If an error occurs during the traversing motion, the bit "AxisError" in the output signal "Status_table" is issued.

Note

The current command can be replaced on-the-fly by a new command via "Jog1" or "Jog2". This is only possible for the "ModePos" 1, 2 or 3 mode.

Example of the Jog mode



Set the Jog mode as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	7		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermediateStop	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	V8000.2 [1]	Jog1	0
					V8000.3 [1]	Jog2	1
					VD8008	ConfigEpos	3

[1] The value of V8000.2 and V8000.3 cannot be 1 or 0 at the same time.

Note

The variable value in this example is for your reference, and you need to create the variable according to your actual situation.

2. Enter the data in the input "St_I_add" and "St_Q_add".

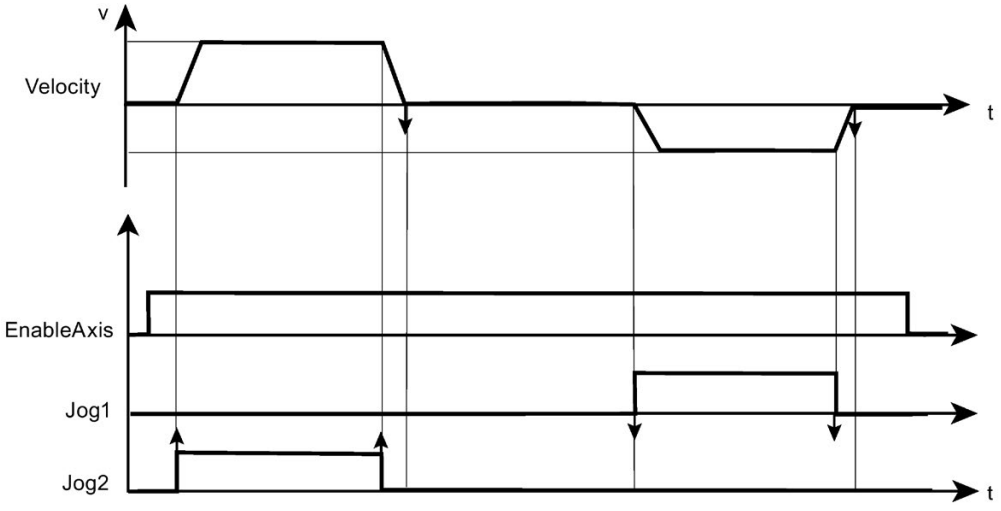
Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

For the input operands "St_I_add" and "St_Q_add", keep the offset consistent with that in the PROFINET wizard.

3. In the variable "Control_table", if you set V8000.2 as 1 and V8000.3 as 0, the drive moves toward the negative direction. If you set V8000.3 as 1 and V8000.2 as 0 , the drive moves toward the positive direction.
4. To pause the drive, set the variable V8000.2 or V8000.3 as 0.

The following illustration displays the movement trajectory and dynamic parameters:



In this illustration, "v" refers to velocity and "t" refers to time.

9.8.1.11 Incremental jog

The Incremental jog mode is implemented through the "Jog" function of SINAMICS V90 PN. It enables the position-controlled, distance-dependent traversing of axes using the integrated position controller of the SINAMICS V90 PN.

Requirements

- The mode is selected with "ModePos" = 8.
- The device is switched on via "EnableAxis".
- The axis is at standstill.
- The axis does **not** have to be referenced or adjusted.

Sequence

The specification of the jog velocity is performed via the V-assistant or the SINA_PARA_S instruction for the configuration of the operating mode in the SINAMICS V90. The SINAMICS V90 uses the acceleration and deceleration for the dynamic responses of the axis.

The velocity override also applies in this operating mode and is set through "OverV". For example, the velocity that takes effect is 500LU/min and the OverV is 120%, the velocity in SINAMICS V90 is 600LU/min.

The operating conditions "CancelTraversing" and "IntermediateStop" are not relevant for the operating mode and can be set to "1".

Note

"Jog1" and "Jog2" are the signal sources for the jog mode in SINA_POS. Jog 1 is the signal source for negative, and Jog 2 is the signal source for positive.

You can configure the distance of incremental traversing motion in the SINAMICS V90 PN.

The travel direction when jogging results from the set velocity setpoint. The inputs "Positive" and "Negative" are not relevant for the operating mode and can be set to "0" as standard.

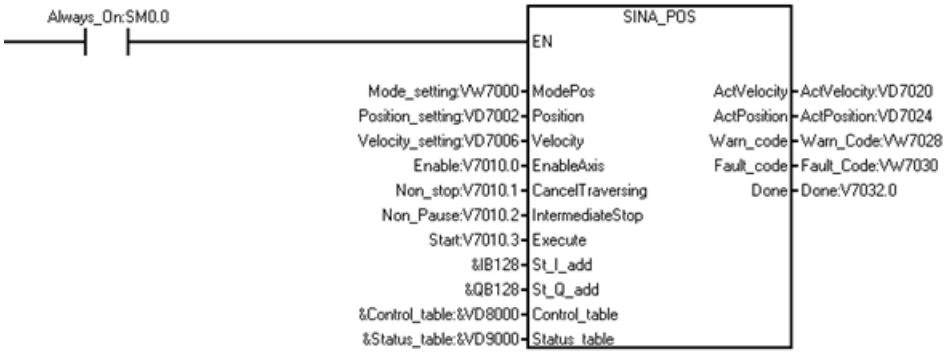
The current state of the active command can be tracked via "EPosZSW1 / EPosZSW2".

The block displays the current command processing with "AxisEnabled" and acknowledges the termination of the jog function ("Jog1" or "Jog2" = 0) when the axis is at standstill with bit "AxisPosOk". If an error occurs during the traversing motion, the bit "AxisError" in the output signal "Status_table" is issued.

Note

The current command can be replaced on-the-fly by a new command via "Jog1" or "Jog2". This is only possible when you remain in one of the jog modes.

Example of the Incremental jog mode



Set the operating mode as "Incremental Jog" as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value		
Mode_setting	VW7000	Mode selection	ModePos	WORD	8		
Enable	V7010.0	Enable the drive.	EnableAxis	BOOL	1		
Non_stop	V7010.1	The status is non-stop.	CancelTraversing	BOOL	1		
Non_Pause	V7010.2	No pausing.	IntermediateStop	BOOL	1		
Control_table	VD8000	The control parameters.	Control_table	DWORD	V8000.2 [1]	Jog1	0
					V8000.3 [1]	Jog2	1
					VD8008	ConfigEpos	3

[1] The value of V8000.2 and V8000.3 cannot be 1 or 0 at the same time.

Note

The variable value in this example is for your reference, and you need to create the variable according to your actual situation.

2. Enter the data in the input "St_I_add" and "St_Q_add".

Note

For the four inputs "St_I_add", "St_Q_add", "Control_table", and "Status_table", the mode of addressing instruction operands is the indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

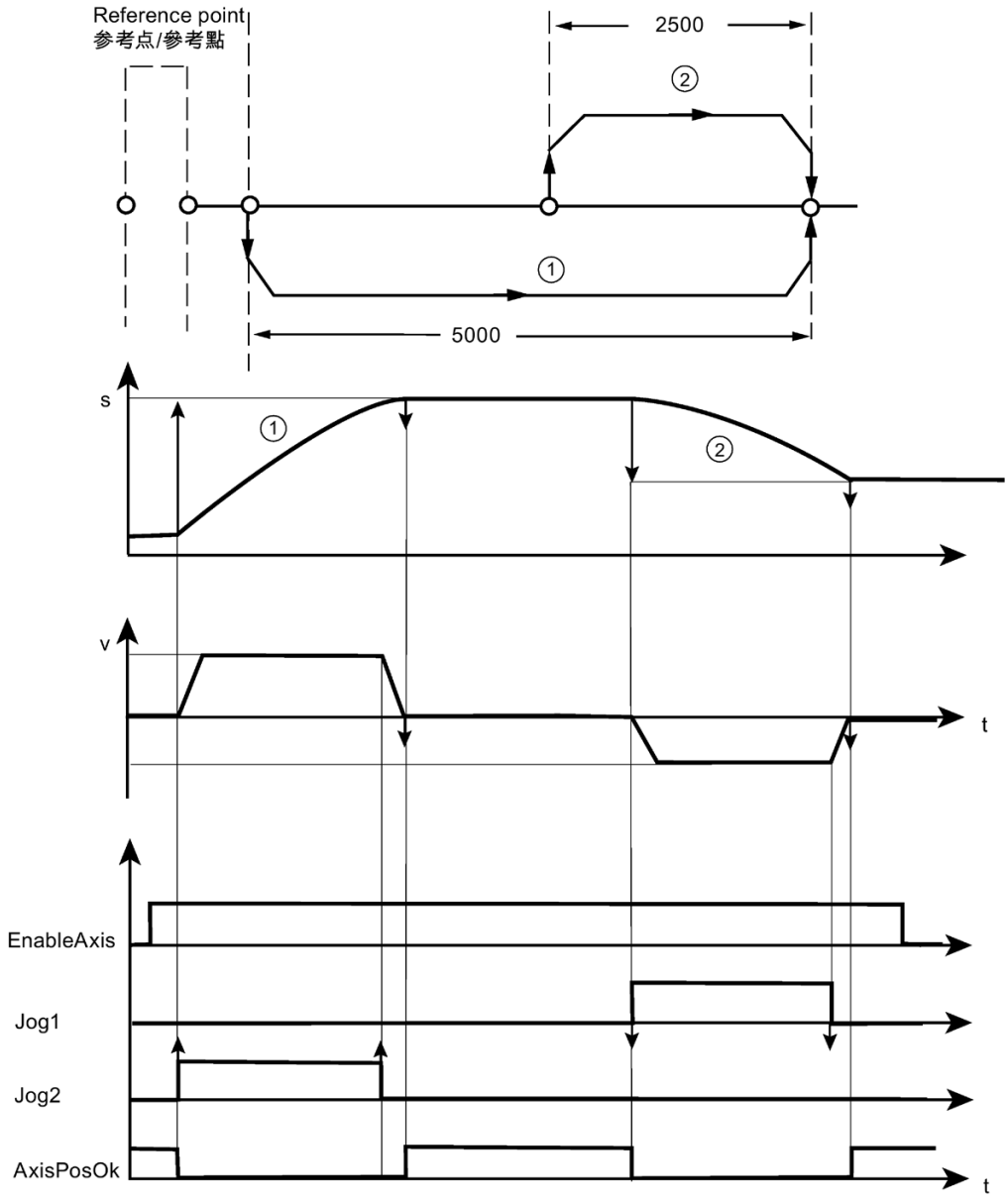
For the input operand "St_I_add" and "St_Q_add", keep the offset consistent with that in PROFINET wizard.

You have the following options for the variable:

- To make the drive rotate in the negative position, set V8000.2 as 1, V8000.3 as 0.
- To make the drive rotate in the positive position, set V8000.3 as 1, V8000.2 as 0.

Result: The drive runs at the incremental jog speed set in V-assistant. When the incremental distance is reached, the motor stops, no matter whether "Jog1" or "Jog2" is set as 1.

The following picture displays the movement trajectory and dynamic parameters:



In this illustration, "v" refers to velocity, "s" refers to position, and "t" refers to time.

9.8.2 SINA_SPEED instruction

9.8.2.1 Prerequisite of using the SINA_SPEED instruction

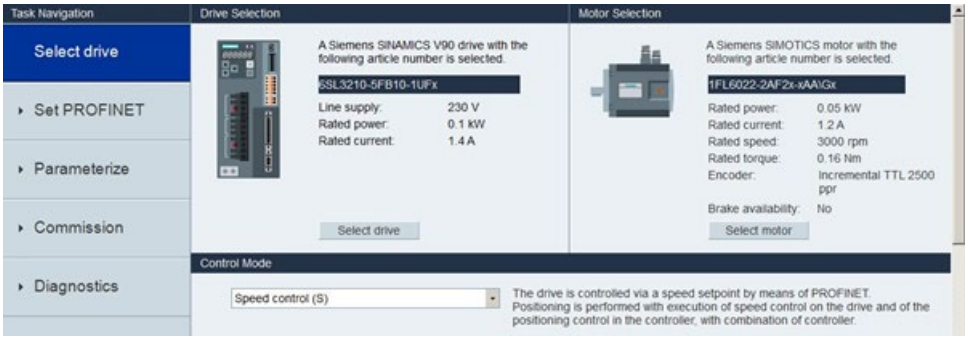
The prerequisite for using the Speed_control instruction is as follows:

- The SINAMICS V90 PN drive and the servo motor are ready.
- The PROFINET network is connected between the drive and the S7-200 Smart CPU.
- The software V-assistant is connected with V90 PN. For the detailed information, refer to SINAMICS V-ASSISTANT Online Help (https://support.industry.siemens.com/cs/ww/en/view/109738316). You can download the SINAMICS V-ASSISTANT software (https://support.industry.siemens.com/cs/ww/en/view/109738387) and the SINAMICS V90: PROFINET GSD file (https://support.industry.siemens.com/cs/ww/en/view/109737269) in the SIEMENS Industry Online Support Website.

Procedure of configuring SINAMICS V90 PN parameters with V-assistant

Configure the SINAMICS V90 PN parameters as follows:

1. Start the software V-assistant.
2. Click "Online" to select the working mode.
3. Click the connected drive and click the "OK" button.
4. Click "Select drive" from the navigation tree and select "Speed control (s)" in the "Control Mode" field.

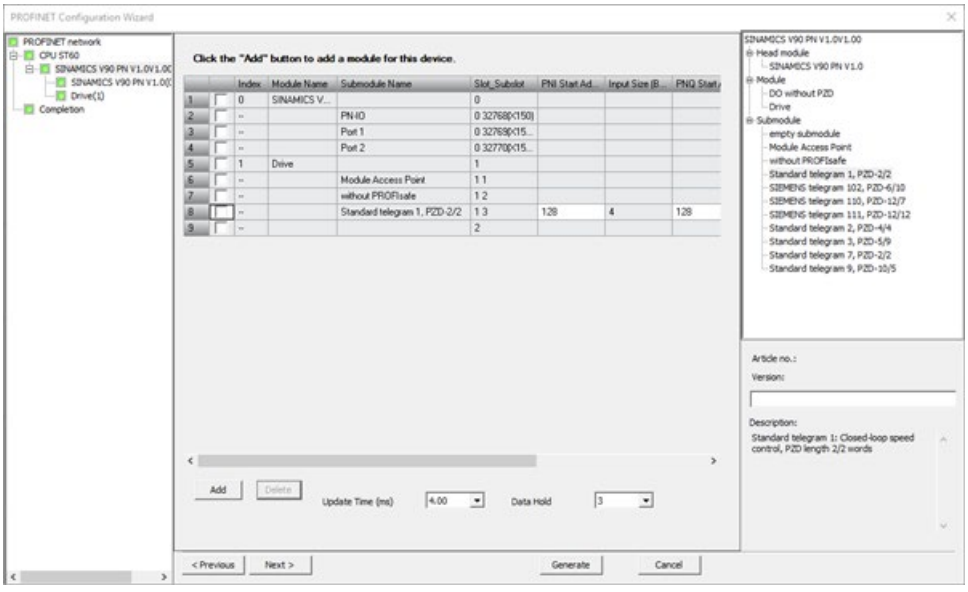


5. Click "Set PROFINET" from the navigation tree and click "Select telegram".

6. In the "Selection of telegram" field, select "Standard telegram 1" as the current telegram. The SINA_SPEED instruction only supports Standard telegram 1.

Note

When you configure the PROFINET network in PROFINET wizard, keep the telegram consistent with that in this step.

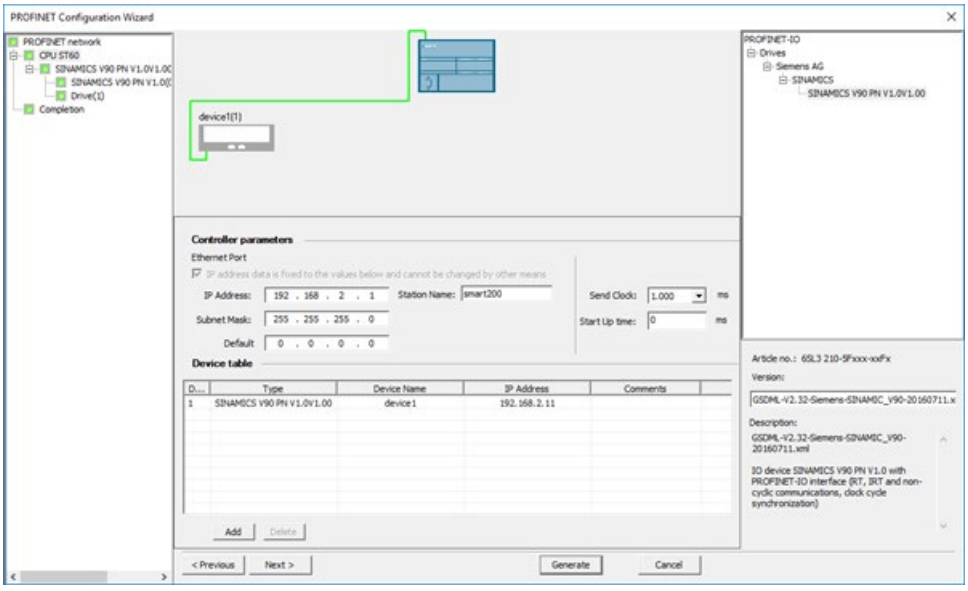


7. Click "Configure network" from the navigation tree.

8. Define the PN station name in the "Name of PN station".

Note

When you configure the PROFINET network in PROFINET wizard, keep the device name consistent with the PN station name in this step.



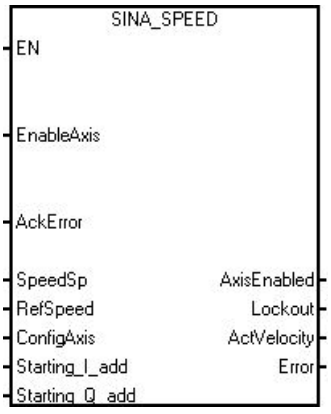
9. Click the "Save and active" button.

Result: The drive restarts and the configured parameters take effect.

9.8.2.2 Input and output interface of SINA_SPEED instruction

The SINA_SPEED instruction is used for setting the drive speed.

Table 9- 55 SINA_SPEED instruction

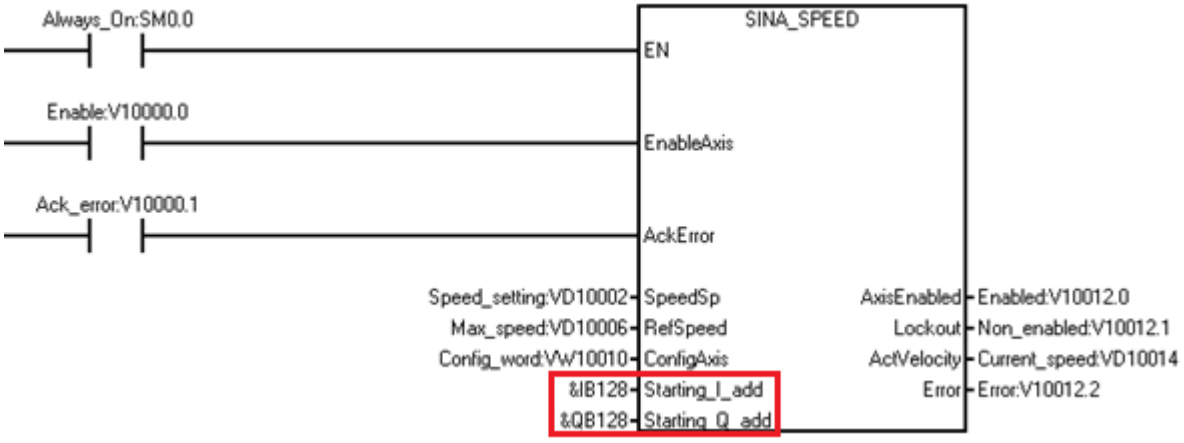
LAD/ FBD	STL	Description
	`CALL SINA_SPEED,EnableAxis, AckError, SpeedSp, Ref-Speed, ConfigAxis Start-ing_I_add,Starting_Q_add, AxisEnabled, Lockout, ActVelocity, Error`	The SINA_SPEED instruction enables the speed control of drive movement.

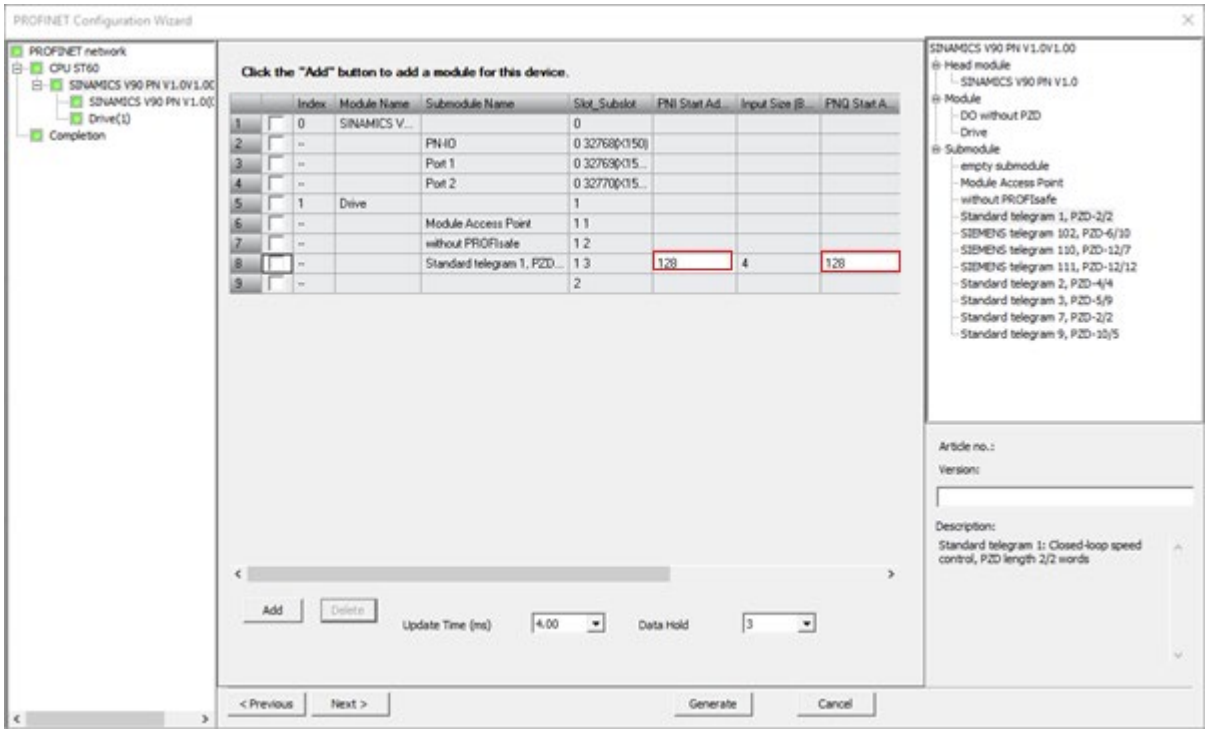
Note

For the two inputs "St_I_add" and "St_Q_add", the mode of addressing instruction operands is indirect addressing.

You must enter an ampersand (&) at the beginning of the input operand and keep the offset consistent with that in the PROFINET wizard.

You can take the following pictures for reference:





The parameters of SINA_SPEED instruction are as follows:

Table 9- 56 Parameters of SINA_SPEED instruction

Parameter and type		Data type	Description
EnableAxis	IN	BOOL	"EnableAxis" = 1 -> switches on the drive
AckError	IN	BOOL	Acknowledges axis faults -> "AckFlt"=1
SpeedSp	IN	REAL	Speed setpoint. SpeedSp value changes with Refspeed. For example, if you set Refspeed as 1000 rpm, the SpeedSp value range is (0, 1000 rpm).
Refspeed	IN	REAL	Rated speed of the drive. The value range is (6, 210000 rpm).
ConfigAxis	IN	WORD	One input to control the SINA_SPEED functions that are not directly specified at the block. For the detailed information, refer to Definition of "ConfigAxis" parameters (Page 633). (Default value = 0)
Starting_I_add	IN	DWORD	Pointer of I memory area start address for PROFINET IO.
Starting_Q_add	IN	DWORD	Pointer of Q memory area start address for PROFINET IO.
AxisEnabled	OUT	BOOL	The drive is being executed or enabled.
Lockout	OUT	BOOL	1 = switching-on inhibited active.
ActVelocity	OUT	REAL	Actual velocity ->dependent on scaling factor RefSpeed.
Error	OUT	BOOL	1 = group fault active.

9.8.2.3 Definition of "ConfigAxis" parameters

The following table lists the definition of "ConfigAxis" parameters:

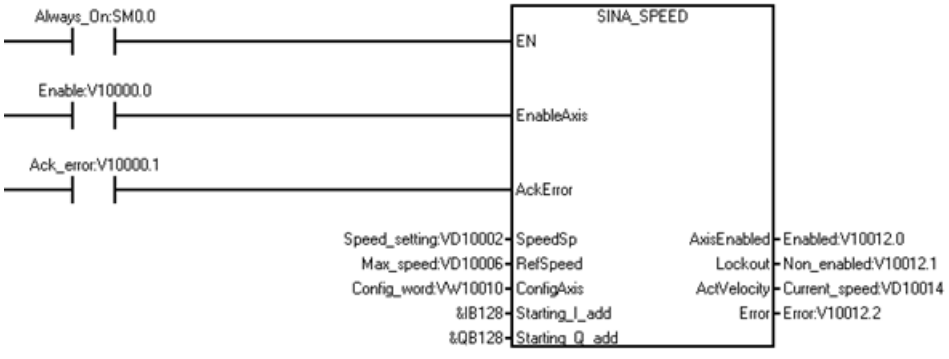
Table 9- 57 ConfigAxis

ConfigAxis	Meaning
Bit0	OFF2
Bit1	OFF3
Bit2	Inverter enable
Bit3	Enable ramp-function generator
Bit4	Continue ramp-function generator
Bit5	Enable speed setpoint
Bit6	Direction of rotation
Bit7	Unconditionally open holding brake
Bit8	Motorized potentiometer increase setpoint
Bit9	Motorized potentiometer, decrease setpoint
Bit10	Reserved
Bit11	Reserved
Bit12	Reserved
Bit13	Reserved
Bit14	Reserved
Bit15	Reserved

9.8.2.4 Example of SINA_SPEED instruction

The SINA_SPEED instruction is used for controlling the drive cyclically with standard telegram 1.

Example



Insert and configure the SINA_SPEED instruction as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Comment	Corresponding input operand	Data type	Value
Speed_setting	VD10002	Speed	SpeedSp	REAL	100
Max_speed	VD10006	Maximum speed	RefSpeed	REAL	3000
Config_word	VW10010	Parameter table of ConfigAxis	ConfigAxis	WORD	63 **Note:** To enable the drive, you must set this variable as 63 (Decimal).
Enable	V10000.0	Enable the drive.	EnableAxis	BOOL	1
Ack_error	V10000.1	Acknowledge the fault.	AckError	BOOL	
Enabled	V10012.0	The axis is enabled.	AxisEnabled	BOOL	
Non_enabled	V10012.1	The drive is not enabled.	Lockout	BOOL	
Current_speed	VD10014	Actual velocity	ActVelocity	REAL	
Error	V10012.2	Error from the drive	Error	REAL	

2. Enter the data in the input "St_I_add" and "St_Q_add".

Note

For the inputs "St_I_add" and "St_Q_add", the mode of addressing instruction operands is indirect addressing. You must enter an ampersand (&) at the beginning of the input operand.

For the input operands "St_I_add" and "St_Q_add", keep the offset consistent with that in the PROFINET wizard.

9.8.3 SINA_PARA_S instruction

9.8.3.1 Prerequisite of using the SINA_PARA_S instruction

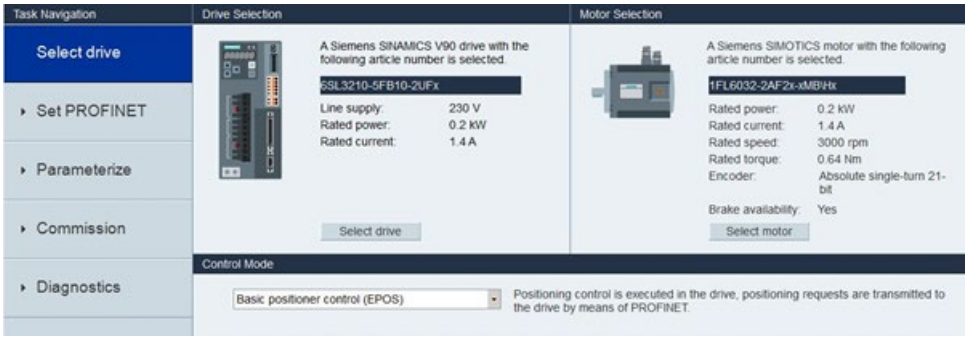
The prerequisite for using the SINA_PARA_S instruction is as follows:

- The SINAMICS V90 PN drive and the servo motor are ready.
- The PROFINET network is connected between the drive and the S7-200 Smart CPU.
- The software V-assistant is connected with SINAMICS V90 PN. For the detailed information, refer to SINAMICS V-ASSISTANT Online Help (https://support.industry.siemens.com/cs/ww/en/view/109738316). You can download the SINAMICS V-ASSISTANT software (https://support.industry.siemens.com/cs/ww/en/view/109738387) and the SINAMICS V90: PROFINET GSD file (https://support.industry.siemens.com/cs/ww/en/view/109737269) in the SIEMENS Industry Online Support Website.

Procedure of configuring SINAMICS V90 PN parameters with V-assistant

Configure the V90 PN parameters with V-assistant as follows:

1. Start the software V-assistant.
2. Click "Online" to select the working mode.
3. Click the connected drive and click the "OK" button.
4. Click "Select drive" from the navigation tree, and select "Speed control (s)" or "Basic positioner control (EPOS)" in the "Control Mode" field.

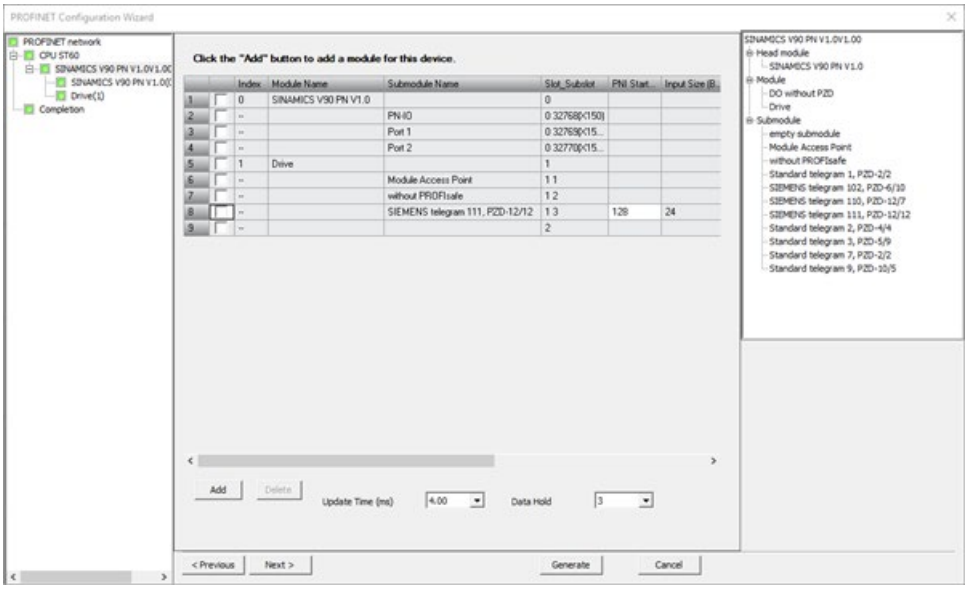


5. Click "Set PROFINET" from the navigation tree and click "Select telegram".

6. In the "Selection of telegram" field, select your desired telegram as the current telegram. The SINA_PARA_S instruction only supports SIEMENS telegram 111 or Standard telegram 1.

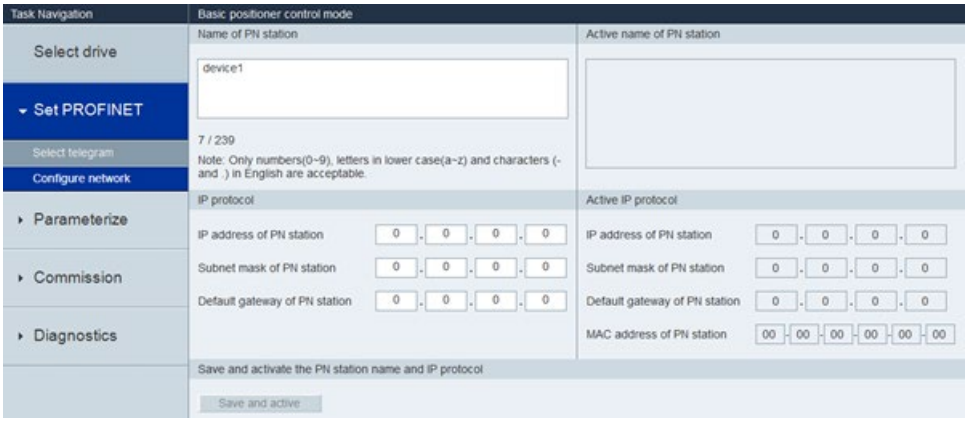
Note

When you configure the PROFINET network in PROFINET wizard, keep the telegram consistent with that in this step.



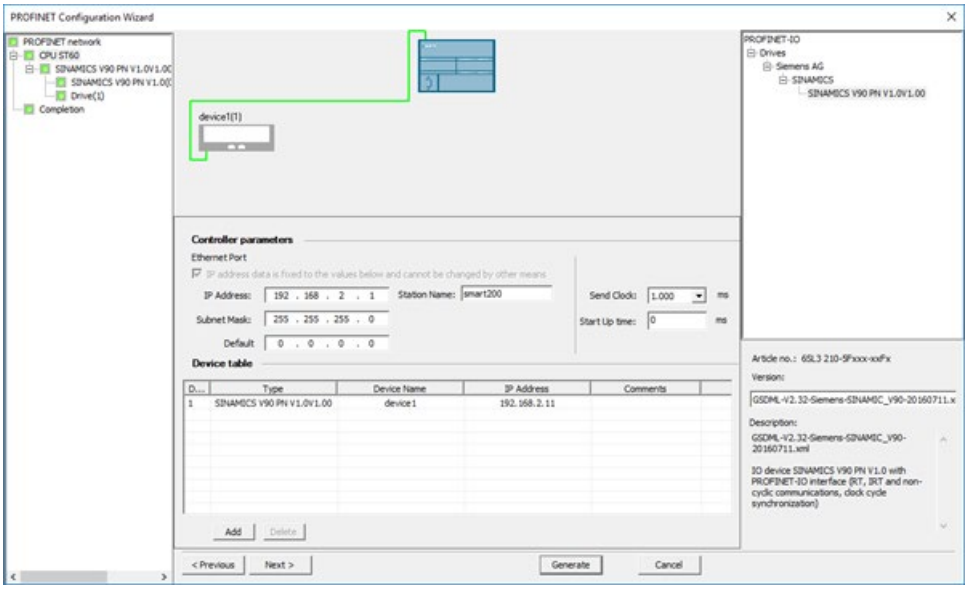
7. Click "Configure network" from the navigation tree.

8. Define the PN station name in the "Name of PN station".



Note

When you configure the PROFINET network in PROFINET wizard, keep the device name consistent with the PN station name that in this step.



9. Click the "Save and active" button.

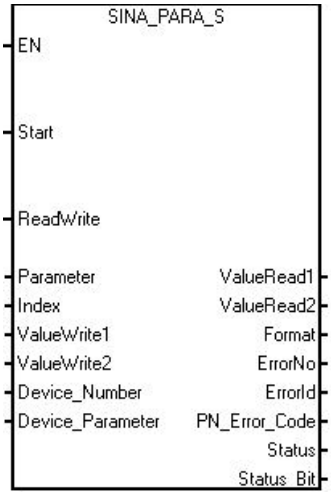
Result: The drive restarts and the configured parameters take effect.

9.8.3.2 Input and output interface of SINA_PARA_S instruction

The SINA_PARA_S instruction is used for managing the acyclic parameters.

The SINA_PARA_S instruction is as follows:

Table 9- 58 SINA_PARA_S instruction

LAD/FBD	STL	Description
	CALL SINA_PARA_S,Start, ReadWrite, Parameter, Index, ValueWrite1, ValueWrite2, Device_Number, Device_Parameter, ValueRead1, ValueRead2, Format, ErrorNo, ErrorId, PN_Error_Code, Status, Status_Bit	Use the SINA_PARA_S instruction to set the drive parameters or to read the parameter from drive.

The following table lists the input and output signals of SINA_PARA_S insrtruction:

Table 9- 59 Parameters of SINA_PARA_S instruction

Parameter and type		Data type	Description
Start	IN	BOOL	Start of the task (0= no task or cancel task; 1= start and execute task)
ReadWrite	IN	BOOL	Task type selection 0 = read, 1= write
Parameter	IN	INT	Parameter number
Index	IN	INT	Index of the parameter
ValueWrite1	IN	REAL	Value of the parameter in REAL format
ValueWrite2	IN	DINT	Value of the parameter in DINT format
DeviceNo	IN	WORD	Device number
De-vice_Parameter	IN	DWORD	Pointer of the start address of "Device_Parameter" (Page 639). "De-vice_Parameter" refers to the parameter of the PROFINET slave.
ValueRead1	OUT	REAL	Value of the parameter read from the drive (REAL format)
ValueRead2	OUT	DINT	Value of the parameter read from the drive (DINT format)
Format	OUT	BYTE	Format of the read parameter (Page 639)
ErrorNo	OUT	WORD	Error number according to PROFIdrive profile (Page 640)
ErrorID	OUT	DWORD	Error ID (Page 641). First word: binary-coded indicating which parameter access is faulted Second word: type of fault
PN_Error_Code	OUT	DINT	Error code according to the PROFINET protocol. For detailed information, refer to *Technical Specification for PROFINET IO (Version 2.3).*

Parameter and type		Data type	Description
Status	OUT	BYTE	The status of the current operation (Page 641).
Status_bit	OUT	BYTE	Status table (Page 642)

Definition of "Device_parameter"

The following table list the bit definitions of Device_parameters:

Table 9- 60 Device_parameters

Byte Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Axis number: Select 2 for the V90 drive. For other drives, refer to the related manual.							
1	Reserved							
2	API number[1]							
3								
4								
5								
6	Slot number[1]							
7								
8	Subslot number[1]							
9								

Note

[1] For the API number, Slot number and Subslot number, move to the PROFINET wizard (Page 414)to check the information.

Definition of "Format" parameters

The format of parameter read from the drive is as follows:

Table 9- 61 Format

ID	Description
02	Integer 8
03	Integer 16
04	Integer 32
05	Unsigned 8
06	Unsigned 16
07	Unsigned 32
08	Floating Point
10	Octet String 8 (16 bit)

ID	Description
13	Time Difference (32 bit)
41	Byte
42	Word
43	Double word
44	Error

Definition of "ErrorNo" parameters

ErrorNo refers to the error code defined by the PROFIdrive protocol.

Table 9- 62 ErrorNo

Error ID	Description
00 hex	Illegal parameter number (access to a parameter that does not exist)
01 hex	Parameter value cannot be changed (change request for a parameter value that cannot be changed)
02 hex	Lower or upper value limit exceeded (change request with a value outside the value limits)
03 hex	Incorrect subindex (access to a parameter index that does not exist)
04 hex	No array (access with a subindex to non-indexed parameters)
05 hex	Incorrect data type (change request with a value that does not match the data type of the parameter)
06 hex	Setting not permitted, only resetting (change request with a value not equal to 0 without permission)
07 hex	Descriptive element cannot be changed (change request to a descriptive element that cannot be changed)
09 hex	Description data not available (access to a description that does not exist, parameter value is available)
0B hex	No master control (change request but with no master control)
0F hex	Text array does not exist (although the parameter value is available, the request is made to a text array that does not exist)
11 hex	Request cannot be executed due to the operating state (access is not possible for temporary reasons that are not specified)
14 hex	Inadmissible value (change request with a value that is within the limits but which is illegal for other permanent reasons, for example, a parameter with defined individual values)
15 hex	Response too long (the length of the actual response exceeds the maximum transfer length)
16 hex	Illegal parameter address (illegal or unsupported value for attribute, number of elements, parameter number, subindex or a combination of these)
17 hex	Illegal format (change request for an illegal or unsupported format)
18 hex	Number of values not consistent (number of values of the parameter data to not match the number of elements in the parameter address)
19 hex	Drive object does not exist (access to a drive object that does not exist)
20 hex	Parameter text cannot be changed
21 hex	Service is not supported (illegal or not support request ID)
6B hex	A change request for a controller that has been enabled is not possible. (The drive rejects the change request because the motor is switched on. Please observe the "Can be changed" parameter attribute (U, T) as given in the parameter list relevant section in the SINAMICS V90, SIMOTICS S-1FL6 Operating Instructions
6C hex	Unknown unit.
77 hex	Change request is not possible during download

Error ID	Description
81 hex	Change request is not possible during download
82 hex	Accepting the master control is inhibited
83 hex	Desired interconnection is not possible (the connector output does not supply a float value although the connector input requires a float value)
84 hex	Inverter does not accept a change request (inverter is busy with internal calculations)
85 hex	No access methods defined
87 hex	Know-how protection active, access locked
C8 hex	Change request below the currently valid limit (change request to a value that lies within the "absolute" limits, but is however below the currently valid lower limit)
C9 hex	Change request above the currently valid limit (example: a parameter value is too large for the drive power)
CC hex	Change request not permitted (change is not permitted as the access code is not available)

Definition of "ErrorID" parameters

There are two types of error:

- ErrorID[1]
- ErrorID[2]

Table 9- 63 ErrorID[1]

ID	Description
0×000	No fault active
0×001	Internal telegram error active
0×002	Parameterization error active
0×003	RDREC and WRREC call error active
0×004	Cancelation of the job during the active data transfer by resetting the Start input to "0"
0×005	Unknown data type detected; evaluation of the ErrorID[2] shows the parameter with unknown data type in the highest value bit

Table 9- 64 ErrorID[2]

ID	Description
0×01	Unknown error

Definition of "STATUS" parameter

The parameters of STATUS are as follows:

Table 9- 65 STATUS

Byte Offset	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A [1]	E [2]	Error code [3]				

[1] A: 1 = a request is in process

[2] E: 1= an error occurs

[3] Error code: The system error code. For detailed information, refer to System-defined error code of the instructions RDREC and WRREC (Page 377).

Definition of "Status_bit" parameters

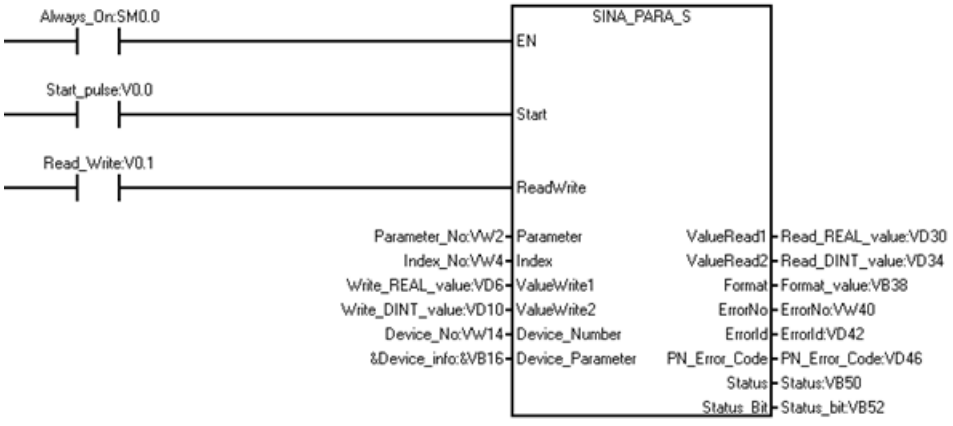
Table 9- 66 Status_bit

Byte Offset	Bit 3	Bit 2	Bit 1	Bit 0
0	Error	Done	Busy	Ready

9.8.3.3 Example of the SINA_PARA_S instruction

The SINA_PARA_S instruction can read and write the drive parameters in acyclic communication.

Example



Modify the drive parameters with SINA_PARA_S instruction as follows:

1. Create the following variables and write the variables to the corresponding input operands.

Symbol	Address	Corresponding input operand	Data type	Value		
Start_pulse	V0.0	Start	BOOL	1		
Read_Write	V0.1	ReadWrite	BOOL	0 or 1		
Parameter_No	VW2	Parameter	INT	29070		
Index_No	VW4	Index	INT	1		
Write_REAL_value	VD6	ValueWrite1	REAL			
Write_DINT_value	VD10	ValueWrite2	DINT			
Device_No	VW14	Device_number	WORD	1		
Device_info	VB16	Device_Parameter	DWORD	VB16	Axis number	2
				VD18	API number	14848
				VW22	Slot number	1
				VW24	Subslot number	3
Read_REAL_value	VD30	ValueRead1	REAL			
Read_DINT_value	VD34	ValueRead2	DINT			
Format_value	VB38	Format	BYTE			
ErrorNo	VW40	ErrorNo	WORD			
ErrorId	VD42	ErrorID	DWORD			
PN_Error_Code	VD46	PN_Error_Code	DINT			
Status	VB50	Status	BYTE			
Status_bit	VB52	Status bit	BYTE	V52.0	Ready	
				V52.1	Busy	
				V52.2	Done	
				V52.3	Error	

2. Read the parameters from the drive to get the parameter data type information.
   - Set the variable "Read_Write" to 0 to initiate a task of reading drive parameters.
   - Enter the device parameter information in the variable "Device_info".
   - Enter the axis number in VB16 "Axis number".
   - Enter the parameter number in the variable "Parameter_No". Enter the index in the variable "Index_No".
   - Set the variable "Start_pulse" to 1 to start the task.

   Result:

   If the parameter data type is REAL, the variable "Read_REAL_value" displays the value.

   If the parameter data type is DINT, the variable "Read_DINT_value" displays the value.
3. Set the variable "Read_Write" to 1 to initiate a task of writing the parameters to the drive.
4. Modify the parameter in the variable "Write_REAL_value" or "Write_DINT_value":

   If in step 2, the variable "Format_value" shows the following data: 16#02, 16#05, 16#41, 16#42, 16#03, 16#06, 16#0A or 16#08, you modify the parameter in the varaible "Write_REAL_value" .

   If in step 2, the variable "Format_value" shows the following data: 16#43, 16#04, 16#07 or 16#0D, you modify the parameter in the varaible "Write_DINT_value" .
5. Set the variable "Start_pulse" to 1.

Result:

In the software V-assistant, check whether you modify the parameter successfully.

9.9 Creating a user-defined library of instructions

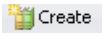
STEP 7-Micro/WIN SMART allows you either to create a custom library of instructions, or to use a library created by someone else.

Creating a library

To create a user-defined library of instructions, you create standard STEP 7-Micro/WIN SMART subroutines and group them together. By grouping these subroutines into a library, you can hide the code. Putting code into a library helps prevent unwanted changes and provides a higher degree of protection for the technology (know-how) of the author.

To create a user-defined library, perform the following tasks:

1. Write the program as a standard project and put the function to be included in the library into subroutines.
2. Ensure that you have assigned a symbolic name to all V memory addresses in the subroutines or interrupt routines. To minimize the amount of V memory that the library requires, use sequential V memory addresses.
3. Rename the subroutines to the names that you want to appear in the instruction library.

4. Select the subroutines that you want to include in the library.
5. Click the Create button  from the Libraries area of the File menu ribbon strip to compile and create the new library. If the subroutine references interrupts, STEP 7-Micro/WIN SMART also includes the interrupt routines in the library.

The libraries that you create are available for new or existing projects. They are not available for the project from which you created them.

Further information

See the STEP 7-Micro/WIN SMART online help library topics for library programming tips and a user-defined library example.

Note

Using custom libraries

You must be responsible for testing any libraries that you add to your project. Siemens bears no responsibility for custom libraries.

If the CPU in your project does not support the instructions in your library, you will get errors when you compile.

Debugging and troubleshooting 10

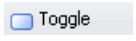
STEP 7-Micro/WIN SMART provides software tools to help you debug and test your program. These features include viewing the status of the program as it is executed by the CPU, selecting to run the CPU for a specified number of scans, and forcing values.

Use the hardware troubleshooting guide (Page 658) as a guide for determining the cause and possible solution when troubleshooting problems with the hardware.

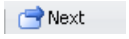
10.1 Debugging your program

10.1.1 Bookmark functions

You can set bookmarks in your program to make it easy to move back and forth between designated networks in a long program:

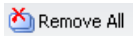


Toggle Bookmark: Click this button to set or remove a bookmark at the program network designated by the current cursor location.

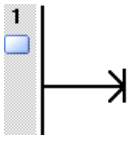


Next Bookmark: Click this button to move to the next bookmarked network of your program.

Previous Bookmark: Click this button to move to the previous bookmarked network of your program.



Remove All Bookmarks: Click this button to remove all bookmarks in your program.



10.1.2 Cross reference table

Note

You must compile your program in order to view the cross reference table.

Use the cross reference table when you want to know whether a symbolic name or memory assignment is already in use in your program, and where it is used. The cross reference table identifies all operands used in the program, and identifies the POU, network or line

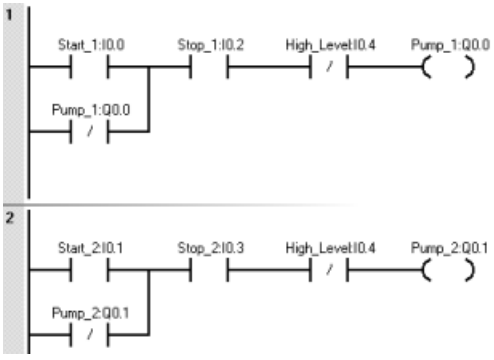
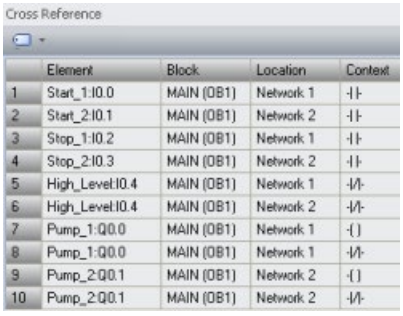
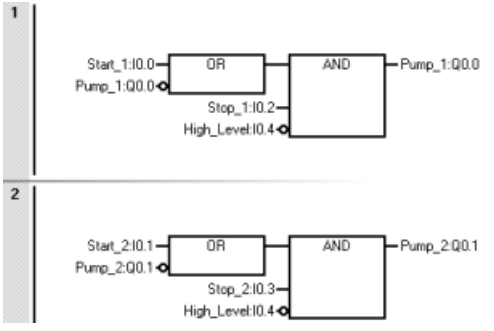
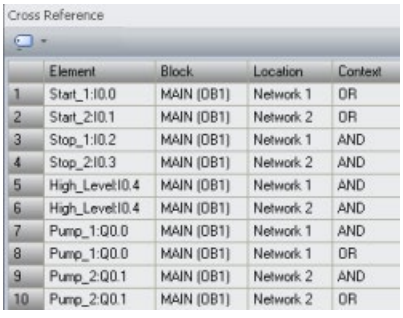
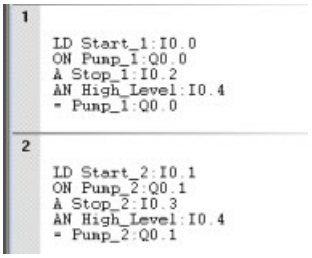
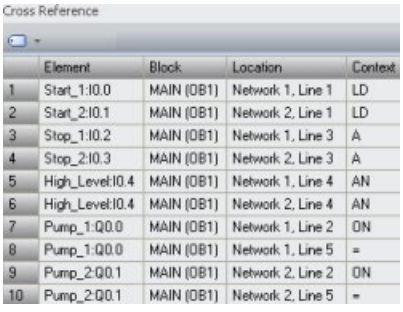
location, and instruction context of the operand each time it is used. Double-clicking an element in the cross-reference table displays that part of your POU.

Element refers to the operands used in your program. You can use the toggle button to toggle between symbolic and absolute addressing to change the representation of all operands.

- **Block** refers to the POU where the operand is used.
- **Location** refers to the line or network where the operand is used.
- **Context** refers to the program instruction where the operand is used.

Examples

The following examples show the cross-reference table for a simple program in all three languages: LAD, FBD, and STL.

Language	Program	Cross reference
LAD		 Element / Block / Location / Context 1 Start_1:I0.0 / MAIN (OB1) / Network 1 / -| |- 2 Start_2:I0.1 / MAIN (OB1) / Network 2 / -| |- 3 Stop_1:I0.2 / MAIN (OB1) / Network 1 / -| |- 4 Stop_2:I0.3 / MAIN (OB1) / Network 2 / -| |- 5 High_Level:I0.4 / MAIN (OB1) / Network 1 / -|/|- 6 High_Level:I0.4 / MAIN (OB1) / Network 2 / -|/|- 7 Pump_1:Q0.0 / MAIN (OB1) / Network 1 / -( ) 8 Pump_1:Q0.0 / MAIN (OB1) / Network 1 / -|/|- 9 Pump_2:Q0.1 / MAIN (OB1) / Network 2 / -( ) 10 Pump_2:Q0.1 / MAIN (OB1) / Network 2 / -|/|-
FBD		 Element / Block / Location / Context 1 Start_1:I0.0 / MAIN (OB1) / Network 1 / OR 2 Start_2:I0.1 / MAIN (OB1) / Network 2 / OR 3 Stop_1:I0.2 / MAIN (OB1) / Network 1 / AND 4 Stop_2:I0.3 / MAIN (OB1) / Network 2 / AND 5 High_Level:I0.4 / MAIN (OB1) / Network 1 / AND 6 High_Level:I0.4 / MAIN (OB1) / Network 2 / AND 7 Pump_1:Q0.0 / MAIN (OB1) / Network 1 / AND 8 Pump_1:Q0.0 / MAIN (OB1) / Network 1 / OR 9 Pump_2:Q0.1 / MAIN (OB1) / Network 2 / AND 10 Pump_2:Q0.1 / MAIN (OB1) / Network 2 / OR
STL		 Element / Block / Location / Context 1 Start_1:I0.0 / MAIN (OB1) / Network 1, Line 1 / LD 2 Start_2:I0.1 / MAIN (OB1) / Network 2, Line 1 / LD 3 Stop_1:I0.2 / MAIN (OB1) / Network 1, Line 3 / A 4 Stop_2:I0.3 / MAIN (OB1) / Network 2, Line 3 / A 5 High_Level:I0.4 / MAIN (OB1) / Network 1, Line 4 / AN 6 High_Level:I0.4 / MAIN (OB1) / Network 2, Line 4 / AN 7 Pump_1:Q0.0 / MAIN (OB1) / Network 1, Line 2 / ON 8 Pump_1:Q0.0 / MAIN (OB1) / Network 1, Line 5 / = 9 Pump_2:Q0.1 / MAIN (OB1) / Network 2, Line 2 / ON 10 Pump_2:Q0.1 / MAIN (OB1) / Network 2, Line 5 / =

10.2 Displaying program status

10.2.1 Displaying status in the program editor

To display current data values and I/O status in the program editor, click the Program Status ON/OFF button from either the program editor toolbar or from the Debug menu ribbon strip Program Status.

Status data collection begins and shows the results of all logic operations during the execution of the program. You can also pause and resume program status collection with the Pause Status ON/OFF button from either the program editor toolbar or from the Debug menu ribbon strip Pause Status.

A status chart (Page 652) displays values at the end of scan.

Execution status coloring

- The power rail (LAD) is colored when the program is being scanned.
- Power flow or logic flow in the diagrams is indicated by coloring.
- Contacts and coils (LAD) that have power or are logically true are colored blue.

  You can assign your own color choice from the Options settings of the Tool menu ribbon strip and selecting the Colors tab.
- Box instructions – The box instructions are colored when the instruction has power and the instruction successfully executes without an error.
- Green timers and counters indicate that the Timer or Counter has valid data.
- Red indicates an instruction executed with an error.
- Jump and Label instructions display in the powerflow color when active. If not active, they display in gray.
- Gray color (default assignment) indicates no power flow - the instruction not scanned (jumped over or not called), or the PLC is in STOP mode.
- Blue color for Boolean Powerflow bits (FBD only).
- LAD, FBD, and STL program editors display the value of operands and indicate powerflow as each instruction executes during the execute program phase of a scan cycle. Execution status can display intermediate data values that may be overwritten by executing subsequent program instructions. All displayed PLC data values are collected from a single program scan cycle.

Example program status in an STL program

When you start program status in STL, the program editor window is divided into a code region and a status region. You can customize the status region according to the types of values that you want to monitor.

There are three categories of values that you can monitor in STL Status:

Operands	You can monitor up to 17 operands per instruction.
Logic stack	You can monitor up to the four most recent values from the logic stack.
Instruction status bits	You can monitor up to eleven status bits.

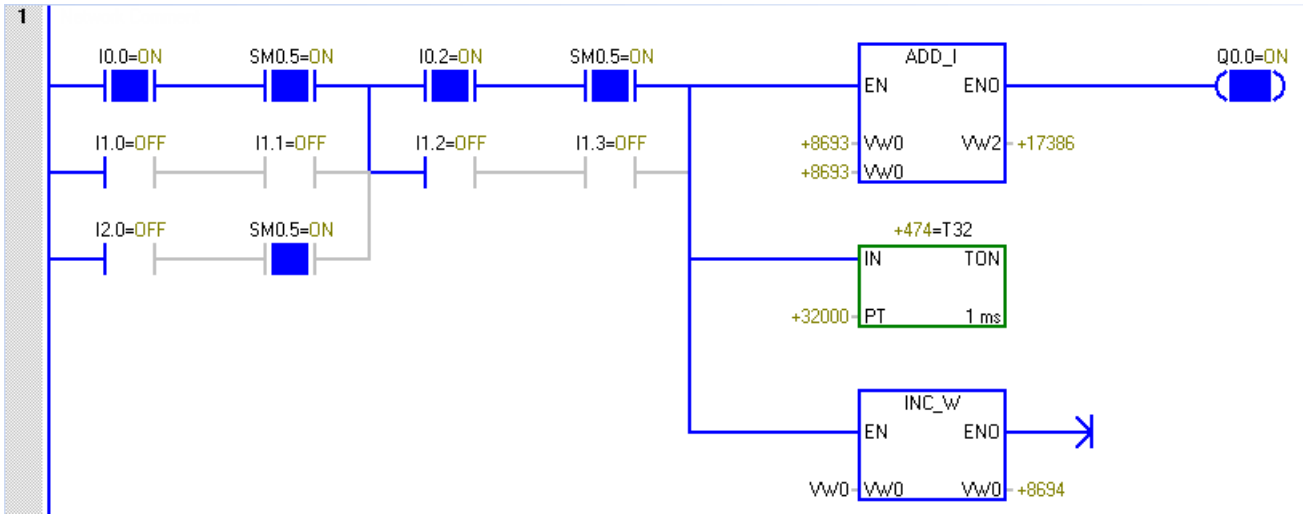
The STL Status tab of the Program Editor options (Page 652) in the Options settings of the Tools menu ribbon strip allows you to select or deselect any of these categories of values. If you deselect an item, it does not appear in the status display.

1

		Op 1	Op 2	Op 3	0123	
LD	I0.0	ON			1000	1
A	SM0.5	ON			1000	1
LD	I1.0	OFF			0100	0
A	I1.1	OFF			0100	0
OLD					1000	1
LD	I2.0	OFF			0100	0
A	SM0.5	ON			0100	1
OLD					1000	1
LD	I0.2	ON			1100	1
A	SM0.5	ON			1100	1
LD	I1.2	OFF			0110	0
A	I1.3	OFF			0110	0
OLD					1100	1
ALD					1000	1
LPS					1100	1
MOVW	VW0, VW2	+8525	+8525		1100	1
AENO					1100	1
+I	VW0, VW2	+8525	+17050		1100	1
AENO					1100	1
=	Q0.0	ON			1100	1
LRD					1100	1
TON	T32, +32000	+294	+32000		1100	1
LRD					1100	1
INCW	VW0	+8526			1100	1
LRD					1100	1
INCW	VW1000	+8526			1100	1

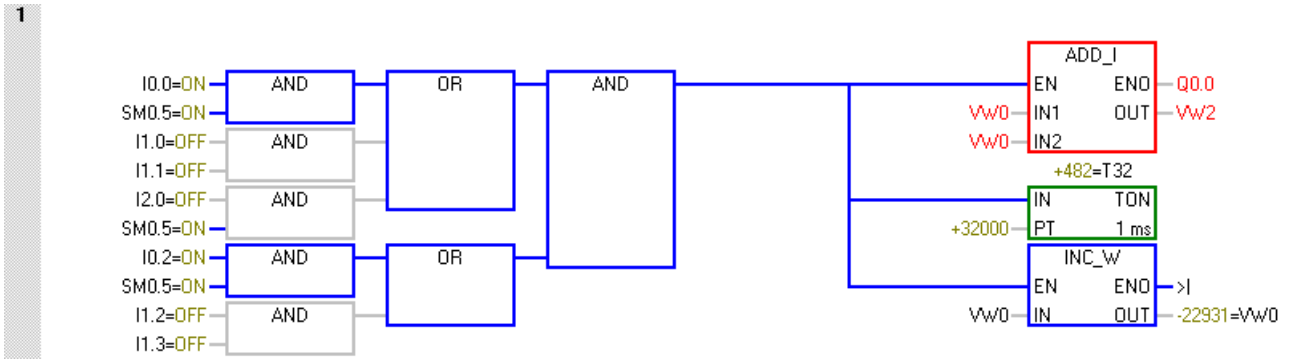
Example program status in the LAD program editor

The example below shows status in the LAD program editor. The program editor displays the values of operands and indicates powerflow as each instruction executes during the execute program phase of a scan cycle.



Example program status in the FBD program editor

The example below shows status in the FBD program editor. The red color of the ADD_I instruction box indicates errors in the operands.



10.2.2 Configuring the STL status options

To configure the STL program status display options, follow these steps:

1. Click the Options button from the Settings area of the Tools menu ribbon strip.

2. Under Options, click Program Editor > STL > Status.
3. Configure the following STL program status options:

Type	Select the font type for STL program status text.
Style	Select Regular, Italic, Bold, or Bold Italic for the text style.
Size	Select the point size for the font.
Watch Values	The check boxes and selection boxes allow you to include or remove operands, stack values, and instruction status bits (that is, flags) from the Program Status display.
Number of operands	If you chose to include operands in the Program Status display, you can edit the Operands list box to display more or fewer operands. The maximum number possible is 17.
Number of Stack Bits	If you chose to include logic stack values in the Program Status display, you can edit the Logic Stack list box to display more or fewer stack values. The maximum number possible is four.
Instruction Status Bits	If you chose to include instruction status bits in the program status display, select the Instruction Status Bits that you want to be shown and which (if any) should be omitted. A check mark indicates that you are choosing to watch a particular status bit in the program status display; if you deselect the checkbox, STEP 7-Micro/WIN SMART does not display that status bit in the Program Status.

See also

How to display status in the program editor (Page 649)

10.3 Using a status chart to monitor your program

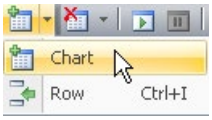
In a status chart, you can enter addresses or defined symbol names to monitor or modify the status of program inputs, outputs, or variables by displaying the current values. The status chart also allows you to force or change the values of the process variables. You can create multiple status charts in order to view elements from different portions of your program.

You can display timer and counter values as either bits or words. If you display a timer or counter value as a bit, you see the output status of the instruction (0 or 1). If you display a timer or counter value as a word, you see the current value of the timer or counter.

Creating a new chart

To create a new status chart, make sure that Chart Status and Program Status is off and use one of these methods to create a new chart:

- From the project tree, right-click the Status Chart folder and select the context menu command **Insert > Chart**.
- From the Insert area of the Edit menu ribbon strip, click the down arrow beneath "Object" and select "Chart" from the drop-down menu.
- From a status chart tab in the status chart editor, or from any cell in an existing status chart, right-click and select the context menu command **Insert > Chart**.
- From the status chart toolbar, click the insert button and select "Chart".



After you successfully insert a new status chart, the new chart appears under "Status Chart" in the project tree and a new tab appears at the bottom of the Status Chart window.

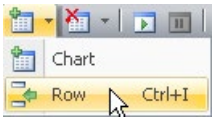
Opening an existing chart

If the status chart editor is not open, you can open an existing status chart from the project tree, navigation bar, or from the Component drop-down list in the Windows area of the View menu. If the status chart editor is open, you can click a status chart tab in the editor to switch to that status chart.

Building a status chart

To build a status chart, follow these steps:

1. Enter the address (or symbol name) for each desired value in the Address field. Symbol names must be names that you have already defined in the symbol table.
2. If the element is a bit (I, Q, or M, for example), the format is set as bit in the Format column. If the element is a byte, word, or double word, select the drop-down list in the Format column and select the valid format from the available options.
3. To insert an additional row, use one of the following methods:
   - Click the insert button on the status chart toolbar and select "Row".

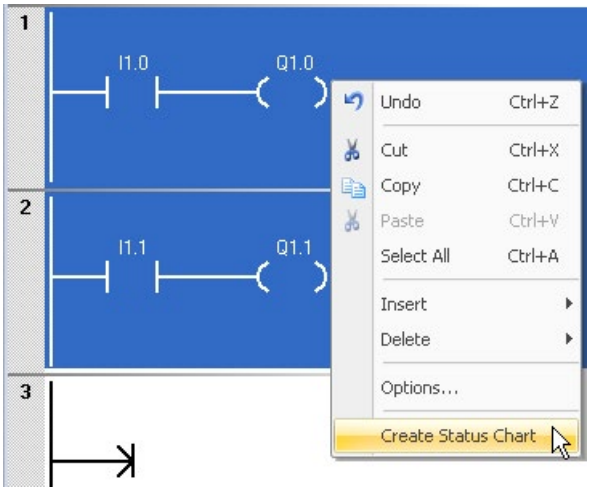
     

   - From the Insert area of the Edit menu ribbon strip, click the "Row" button.
   - Right-click a cell in the status chart to bring up a context menu, and select the menu command **Insert > Row**.

     The new row is inserted above the current location of the cursor in the status chart. You can also place the cursor in the last cell of the last row and press the DOWN ARROW key to insert a row at the bottom of the status chart.

Building a status chart from a section of program code

Highlight a selection of networks in the program editor, right-click, and select "**Create Status Chart**" from the context menu. The new chart contains an entry for each unique operand in the selected region for which status can be gathered. STEP 7-Micro/WIN SMART places the entries in the order of their occurrence in the program, gives the chart a default name, and adds this chart after the last tab in the status chart editor.



When creating a chart from the program editor, note that only the first 150 addresses can be added each time you select **"Create Status Chart"**. After STEP 7-Micro/WIN SMART creates the status chart, you can edit the chart entries.

You can also add an entry to a status chart by pressing the Ctrl key and dragging an operand from the LAD or FBD program editor to the status chart. From STL, you can select an address and drag it to a status chart.

Additionally, you can also copy and paste data from a Microsoft Excel spreadsheet.

Note

A project can store a maximum of 32 status charts.

Copying symbols from the symbol table to a status chart

You can copy addresses or symbol names from the symbol table and paste them into the status chart to build your chart more quickly.

10.4 Forcing specific values

The rule of forcing specific values is as follows:

- The CPU allows you to force any or all of the I/O points (I and Q bits).
- You can force up to 16 memory values (V or M) or analog I/O values (AI or AQ).
- You can force up to 100 bytes for PROFINET I/O values.

- V memory, M memory or PROFINET I/O values can be forced in bytes, words, or double words.
- Analog values are forced as words only, on even-numbered byte boundaries, such as AIW6 or AQW14. All forced values are stored in the non-volatile memory of the CPU.

Note

You cannot transfer force information for PROFINET ports with a SD card.

Because the forced data might be changed during the scan cycle (either by the program, by the I/O update cycle, or by the communication-processing cycle), the CPU reapplies the forced values at various times in the scan cycle.

- *Reading the inputs:* The CPU applies the forced values to the inputs as they are read.
- *Executing the control logic in the program:* The CPU applies the forced values to all immediate I/O accesses. Forced values are applied for up to 16 memory values after the program has been executed.
- *Processing any communications requests:* The CPU applies the forced values to all read/write communications accesses.
- *Writing to the outputs:* The CPU applies the forced values to the outputs as they are written.

Note

The Force function overrides a Read Immediate or Write Immediate instruction. The Force function also overrides the STOP mode values that you configured in the system block (Page 129). If the CPU goes to STOP mode, the output reflects the forced value and not the STOP mode value that you configured for the output in the system block.

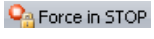
You can use the status chart to force values.

1. To force a new value, enter the value in the New Value column of the Status Chart, then click the Force button on the status chart toolbar, or right-click in the New Value column and select "Force" from the context menu.
2. To force an existing value, select the value in the Current Value column and click the Force button on the status chart toolbar, or right-click the value in the Current Value column and select "Force" from the context menu.

The status LEDs (Page 92) indicate whether the CPU has any forced data.

10.5 Writing and forcing outputs in STOP mode

To enable Write and Force functions while in STOP mode, click the "Force in Stop" button from the Settings area of the Debug menu ribbon strip.



The S7-200 SMART PLCs support writing and forcing outputs (both analog and digital) while the PLC is in STOP mode. However, as a safety precaution, you must specifically enable this functionality in STEP 7-Micro/WIN SMART with the "Force in Stop" setting.

WARNING

Effect on process equipment of writing or forcing outputs

If you have connected the S7-200 SMART PLC to process equipment when you write or force an output, the PLC can transmit these changes to the equipment. This could result in unanticipated activity in the equipment, which could cause death or serious injury to personnel, and/ or property damage.

Only write or force outputs when your process equipment can safely accept those changes.

By default, STEP 7-Micro/WIN SMART does not enable "Force in STOP". STEP 7-Micro/WIN SMART prevents you from writing or forcing outputs while the PLC is in STOP mode. Clicking the "Force in STOP" button from the Debug menu enables writing and forcing for the current editing session with the current project. When you open a different project, "Force in STOP" returns to its default state and STEP 7-Micro/WIN SMART prevents you from either writing or forcing output addresses while the PLC is in STOP mode.

The status LEDs (Page 92) indicate whether the CPU has any forced data in STOP mode.

10.6 How to execute a limited number of scans

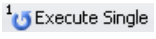
You can specify that the PLC execute your program for a limited number of scans (from 1 scan to 65,535 scans). By selecting the number of scans for the PLC to run, you can monitor the program as it changes the process variables.

On the first scan, the value of SM0.1 is one (ON).

Before executing a single scan or multiple scans, change the PLC to STOP mode (Page 41) if the PLC is not already in STOP mode.

Executing a single scan

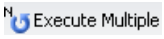
To execute a single scan, click the "Execute Single" button from the Scan area of the Debug menu ribbon strip.



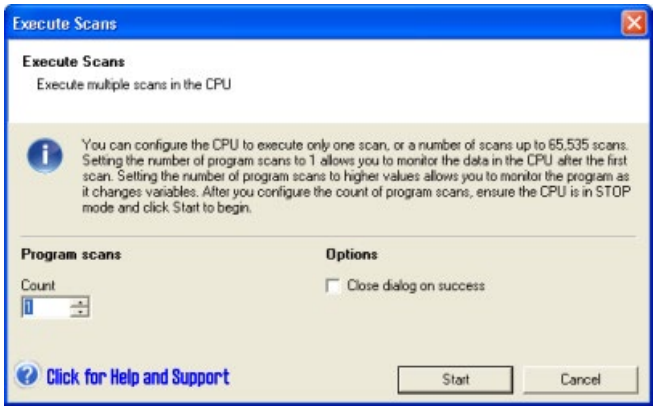
Executing multiple scans

To execute multiple scans, follow these steps:

1. Click the "Execute Multiple" button from the Scan area of the Debug menu ribbon strip.

   The Execute Scans dialog box appears.

2. Enter a value for the desired number of scans, and click "Start" to execute the number of scans you entered.

Note

When you are ready to resume normal program operation, change the PLC back to RUN mode (Page 41).

See also

Overview of debugging and monitoring features (Page 647)

How to display status in the editor windows (Page 649)

How to display status in a status chart (Page 652)

How to download a program (Page 80)

Timestamp mismatch error (Page 859) (ensuring that project in programming device matches project in PLC)

Cross reference and element usage (Page 647) (ensuring that program edits do not cause duplicate assignments)

Forcing values (Page 654)

Forcing outputs in STOP mode (Page 655)

10.7 Hardware troubleshooting guide

Table 10- 1 Troubleshooting guide for the S7-200 SMART hardware

Symptom	Possible cause	Possible solution
Outputs stop working	The device being controlled has caused an electrical surge that damaged the output	When connecting to an inductive load (such as a motor or relay), a proper suppression circuit should be used. Refer to the wiring guidelines in Chapter 3.
	Wiring loose or incorrect	Check wiring and correct
	Excessive load	Check load against contact ratings
	Output point is forced	Check the CPU for forced I/O
ERROR light on the CPU turns on (Red)	Electrical noise	Refer to the wiring guidelines in Chapter 3. It is very important that the control panel is connected to a good ground and that high voltage wiring is not run in parallel with low voltage wiring. Connect the M terminal on the 24 V DC Sensor Power Supply to ground
	Component damage	Send in hardware for repair or replacement
None of the CPU LEDs turn on	Blown fuse	Use a line analyzer and monitor the input power to check the magnitude and duration of the over-voltage spikes. Based on this information, add the proper type surge arrestor device to your power wiring.
	Reversed 24 V power wires	Refer to the wiring guidelines in Chapter 3 for information about installing the field wiring.
	Incorrect voltage	
Intermittent operation associated with high energy devices	Improper grounding	Refer to the wiring guidelines in Chapter 3.
	Routing of wiring within the control cabinet	It is very important that the control panel is connected to a good ground and that high voltage wiring is not run in parallel with low voltage wiring. Connect the M terminal on the 24 V DC Sensor Power Supply to ground.
	Time delay on input filters too short	Increase the input filter delay in the system data block.

Symptom	Possible cause	Possible solution
Serial communications (RS-485 or RS-232) are damaged when connecting to an external device. Either the port on the external device or the port on the CPU is damaged.	The communications cable can provide a path for unwanted currents if all non-isolated devices, such as PLCs, computers, or other devices do not share the same network circuit common reference. The unwanted currents can cause communications errors or damage to electric circuits.	• Refer to the wiring guidelines in Chapter 3 and to the network guidelines in Chapter 8. • Purchase network isolators or isolated RS485-to-RS485 repeaters when you connect devices that do not have a common electrical reference. Refer to Appendix F for information about article numbers for S7-200 SMART equipment.
Other communications problems (STEP 7-Micro/WIN SMART)		Refer to Chapter 8 for information about network communications.
Error handling		Refer to Appendix C for information about error codes.

PID loops and tuning

11

PID auto-tune capability is incorporated into the CPU, and STEP 7-Micro/WIN SMART adds a PID Tune control panel. Together, these two features greatly enhance the utility and ease-of-use of the PID function provided.

You can initiate auto-tune in the user program, from an operator panel, or by the PID Tune control panel. The PID Auto-Tuner computes suggested (near optimum) values for the gain, integral time (reset), and derivative time (rate) tuning values. You can select tuning for fast, medium, slow, or very slow response of your loop.

With the PID Tune control panel, you can initiate the auto-tuning process, abort the auto-tuning process, and monitor the results in a graphical form. The control panel displays any error conditions or warnings that might be generated. You can apply the gain, reset, and rate values computed by auto-tune.

The purpose of the PID Auto-Tuner is to determine a set of tuning parameters that provide a reasonable approximation to the optimum values for your loop. Starting with the suggested tuning values will allow you to make fine tuning adjustments and truly optimize your process. The auto-tuning algorithm used in the CPU is based upon a technique called relay feedback suggested by K. J. Åström and T. Hägglund in 1984. Over the past twenty years, relay feedback has been used across a wide variety of industries.

The relay feedback concept produces a small, but sustained oscillation in an otherwise stable process. Based upon the period of the oscillations and the amplitude changes observed in the process variable, the ultimate frequency and the ultimate gain of the process are determined. Then, using the ultimate gain and ultimate frequency values, the PID Auto-tuner suggests a value for the gain, reset, and rate tuning values.

The values suggested depend upon your selection for speed of response of the loop for your process. You can select fast, medium, slow, or very slow response. Depending upon your process, a fast response can overshoot and corresponds to an underdamped tuning condition. A medium speed response can be on the verge of overshoot and corresponds to a critically-damped tuning condition. A slow response cannot have any overshoot and corresponds to an overdamped tuning condition. A very slow response cannot have overshoot and corresponds to a heavily overdamped tuning condition.

In addition to suggesting tuning values, the PID Auto-tuner can automatically determine the values for hysteresis and peak PV deviation. You use these parameters to reduce the effect of the process noise, while limiting the amplitude of the sustained oscillations set up by the PID Auto-Tuner.

The PID Auto-Tuner can determine suggested tuning values for both direct-acting and reverse-acting P, PI, PD, and PID loops.

11.1 PID loop definition table

Eighty (80) bytes are allocated for the loop table from the starting address you enter for Table (TBL) in the PID instruction box. The PID instruction for the S7-200 SMART CPU references this loop table that contains the loop parameters.

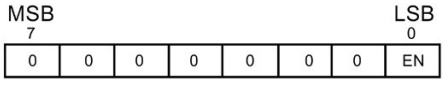
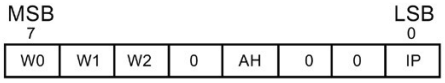
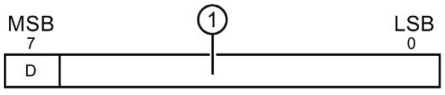
If you use the PID Tune control panel, all interaction with the PID loop table is handled for you by the control panel. If you need to provide auto-tuning capability from an operator panel, your program must provide the interaction between the operator and the PID loop table to initiate and monitor the auto-tuning process, and then apply the suggested tuning values.

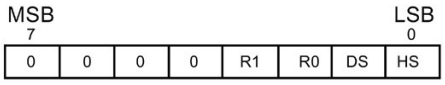
Table 11- 1 Loop table

Offset	Field	Format	Type	Description
0	Process variable (PV_n)	REAL	In	Contains the process variable, which must be scaled between 0.0 and 1.0.
4	Setpoint (SP_n)	REAL	In	Contains the setpoint, which must be scaled between 0.0 and 1.0.
8	Output (M_n)	REAL	In/Out	Contains the calculated output, scaled between 0.0 and 1.0.
12	Gain (K_C)	REAL	In	Contains the gain, which is a proportional constant. Can be a positive or negative number.
16	Sample time (T_S)	REAL	In	Contains the sample time, in seconds. Must be a positive number.
20	Integral time or reset (T_I)	REAL	In	Contains the integral time or reset, in minutes.
24	Derivative time or rate (T_D)	REAL	In	Contains the derivative time or rate, in minutes.
28	Bias (MX)	REAL	In/Out	Contains the bias or integral sum value between 0.0 and 1.0.
32	Previous process variable (PV_{n-1})	REAL	In/Out	Contains the value of the process variable stored from the last execution of the PID instruction.
36	PID Extended Table ID	ASCII	Constant	'PIDA' (PID Extended Table, Version A): ASCII constant
40	AT Control (ACNTL)	BYTE	In	See the following table
41	AT Status (ASTAT)	BYTE	Out	See the following table
42	AT Result (ARES)	BYTE	In/Out	See the following table
43	AT Config (ACNFG)	BYTE	In	See the following table
44	Deviation (DEV)	REAL	In	Normalized value of the maximum PV oscillation amplitude (range: 0.025 to 0.25).
48	Hysteresis (HYS)	REAL	In	Normalized value of the PV hysteresis used to determine zero crossings (range: 0.005 to 0.1). If the ratio of DEV to HYS is less than 4, a warning will be indicated during auto-tune.
52	Initial Output Step (STEP)	REAL	In	Normalized size of the step change in the output value used to induce oscillations in the PV (range: 0.05 to 0.4).
56	Watchdog Time (WDOG)	REAL	In	Maximum time allowed between zero crossings in seconds (range: 60 to 7200).
60	Suggested Gain (AT_K_C)	REAL	Out	Suggested loop gain as determined by the auto-tune process.

Offset	Field	Format	Type	Description
64	Suggested Integral Time (AT_T_I)	REAL	Out	Suggested integral time as determined by the auto-tune process.
68	Suggested Derivative Time (AT_T_D)	REAL	Out	Suggested derivative time as determined by the auto-tune process.
72	Actual Step size (ASTEP)	REAL	Out	Normalized output step size value as determined by the auto-tune process.
76	Actual Hysteresis (AHYS)	REAL	Out	Normalized PV hysteresis value as determined by the auto-tune process.

Table 11- 2 Expanded description of control and status fields

Field	Description	
AT Control (ACNTL) Input - Byte		
	EN	Set to 1 to start auto-tune; set to 0 to abort auto-tune
AT Status (ASTAT) Output - Byte		
	W0	Warning: The deviation setting is not four times greater than the hysteresis setting.
	W1	Warning: Inconsistent process deviations may result in incorrect adjustment of the output step value.
	W2	Warning: Actual average deviation is not four times greater than the hysteresis setting.
	AH	Auto-hysteresis calculation in progress: 0 - not in progress 1 - in progress
	IP	Auto-tune in progress: 0 - not in progress 1 - in progress
	Each time an auto-tune sequence is started the CPU clears the warning bits and sets the in progress bit. Upon completion of auto-tune, the CPU clears the in progress bit.	
AT Result (ARES) Input/Output - Byte	 ① Result code	
	D	Done bit: 0 - auto-tune not complete 1 - auto-tune complete Must be set to 0 before auto-tune can start
	Result Code	00 - completed normally (suggested tuning values available) 01 - aborted by the user 02 - aborted, watchdog timed out waiting for a zero crossing 03 - aborted, process (PV) out-of-range 04 - aborted, maximum hysteresis value exceeded 05 - aborted, illegal configuration value detected 06 - aborted, numeric error detected 07 - aborted, PID instruction executed without having power flow (loop in manual mode) 08 - aborted, auto-tuning allowed only for P, PI, PD, or PID loops 09 to 7F - reserved

Field	Description		
AT Config (ACNFG) Input - Byte			
	R1	R0	Dynamic response
	0	0	Fast response
	0	1	Medium response
	1	0	Slow response
	1	1	Very slow response
	DS	Deviation setting: 0 - use deviation value from loop table 1 - determine deviation value automatically	
	HS	Hysteresis setting: 0 - use hysteresis value from loop table 1 - determine hysteresis value automatically	

Note

The standard PID instruction (Page 293) is not used directly by projects with PID wizard configurations. If you use a PID wizard configuration, then your program must use "PIDx_CTRL", to activate the PID wizard subroutine.

To simplify the use of PID loop control in your application, STEP 7-Micro/WIN SMART provides a PID wizard (Page 295) to configure your PID loops.

11.2 Prerequisites

The loop that you want to auto-tune must be in automatic mode. The loop output must be controlled by the execution of the PID instruction. Auto-tune will fail if the loop is in manual mode.

Before initiating an auto-tune operation, your process must be brought to a stable state which means that the PV has reached setpoint (or for a P-type loop, a constant difference between PV and setpoint) and the output is not changing erratically.

Ideally, the loop output value needs to be near the center of the control range when auto-tuning is started. The auto-tune procedure sets up an oscillation in the process by making small step changes in the loop output. If the loop output is close to either extreme of its control range, the step changes introduced in the auto-tune procedure can cause the output value to attempt to exceed the minimum or the maximum range limit.

If this happens, the generation of an auto-tune error condition results, and the determination of less than near optimal suggested values certainly results.

11.3 Auto-hysteresis and auto-deviation

Hysteresis parameter

The hysteresis parameter specifies the excursion (plus or minus) from setpoint that the PV (process variable) is allowed to make without causing the relay controller to change the output. This value is used to minimize the effect of noise in the PV signal to more accurately determine the natural oscillation frequency of the process.

If you select to automatically determine the hysteresis value, the PID Auto-Tuner will enter a hysteresis determination sequence. This sequence involves sampling the process variable for a period of time and then performing a standard deviation calculation on the sample results.

In order to have a statistically meaningful sample, a set of at least 100 samples must be acquired. For a loop with a sample time of 200 msec, acquiring 100 samples takes 20 seconds. For loops with a longer sample time it will take longer. Even though 100 samples can be acquired in less than 20 seconds for loops with sample times less than 200 msec, the hysteresis determination sequence always acquires samples for at least 20 seconds.

Once all the samples have been acquired, the standard deviation for the sample set is calculated. The hysteresis value is defined to be two times the standard deviation. The calculated hysteresis value is written into the actual hysteresis field (AHYS) of the loop table.

Note

While the auto-hysteresis sequence is in progress, the normal PID calculation is not performed. Therefore, it is imperative that the process be in a stable state prior to initiating an auto-tune sequence. This will yield a better result for the hysteresis value and it will ensure that the process does not go out of control during the auto-hysteresis determination sequence.

Deviation parameter

The deviation parameter specifies the desired peak-to-peak swing of the PV around the setpoint. If you select to automatically determine this value, the desired deviation of the PV is computed by multiplying the hysteresis value by 4.5. The output will be driven proportionally to induce this magnitude of oscillation in the process during auto-tuning.

11.4 Auto-tune sequence

The auto-tuning sequence begins after the hysteresis and deviation values have been determined. The tuning process begins when the initial output step is applied to the loop output.

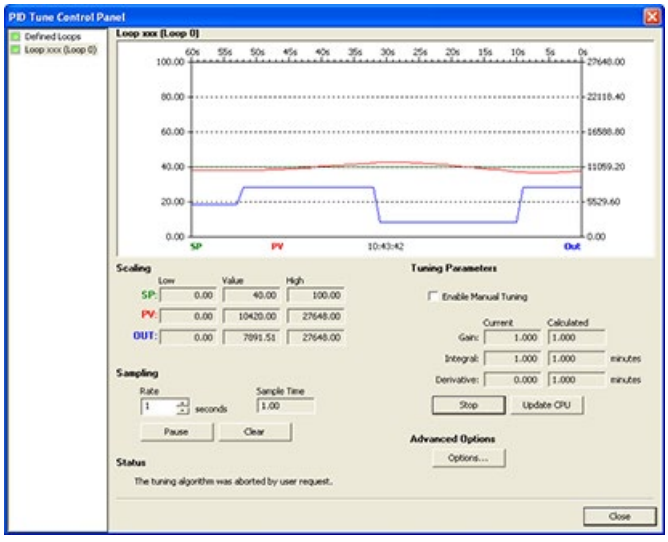
This change in output value causes a corresponding change in the value of the process variable. When the output change drives the PV away from setpoint far enough to exceed the hysteresis boundary, a zero-crossing event is detected by the auto-tuner. Upon each zero-crossing event, the auto-tuner drives the output in the opposite direction.

The tuner continues to sample the PV and waits for the next zero-crossing event. The tuner requires a total of twelve zero-crossings to complete the sequence. The magnitude of the observed peak-to-peak PV values (peak error) and the rate at which zero-crossings occur are directly related to the dynamics of the process.

Early in the auto-tuning process, the output step value is proportionally adjusted once to induce subsequent peak-to-peak swings of the PV to more closely match the desired deviation amount. Once the adjustment is made, the new output step amount is written into the Actual Step Size field (ASTEP) of the loop table.

If the time between zero-crossings exceeds the zero-crossing watchdog interval time, the auto-tuning sequence is terminated with an error. The default value for the zero-crossing watchdog interval time is two hours.

The following figure shows the output and process variable behaviors during an auto-tuning sequence on a direct acting loop. You use the PID Tune control panel to initiate and monitor the tuning sequence.



Notice how the auto-tuner switches the output to cause the process (as evidenced by the PV value) to undergo small oscillations. The frequency and the amplitude of the PV oscillations are indicative of the process gain and natural frequency.

Based upon the information collected about the frequency and gain of the process during the auto-tune process, the ultimate gain and ultimate frequency values are calculated. From these values, the suggested values for gain (loop gain), reset (integral time), and rate (derivative time) are calculated.

Note

Your loop type determines which tuning values are calculated by the auto-tuner. For example, for a PI loop, the auto-tuner will calculate gain and integral time values, but the suggested derivative time will be 0.0 (no derivative action).

Once the auto-tune sequence has completed, the output of the loop is returned to its initial value. The next time the loop is executed, the normal PID calculation will be performed.

11.5 Exception conditions

Warning conditions

Tuning execution can generate three warning conditions. Tuning execution reports these warnings in three bits of the ASTAT field of the loop table, and, once set, these bits remain set until the next auto-tune sequence is initiated:

- Warning 0: Generated if the deviation value is not at least 4X greater than the hysteresis value. This check is performed when the hysteresis value is actually known, which depends upon the auto-hysteresis setting.
- Warning 1: Generated if there is more than an 8X difference between the two peak error values gathered during the first 2.5 cycles of the auto-tune procedure
- Warning 2: Generated if the measured average peak error is not at least 4X greater than the hysteresis value

Error conditions

In addition to the warning conditions, several error conditions are possible. The following table lists the error conditions, along with a description of the cause of each error.

Table 11- 3 Error conditions during tuning execution

Result code (in ARES)		Condition
01	aborted by user	EN bit cleared while tuning is in progress
02	aborted due to a zero-crossing watchdog timeout	Half-cycle elapsed time exceeds zero-crossing watchdog interval
03	aborted due to the process out-of-range	PV goes out-of-range: • during the auto-hysteresis sequence, or • twice before the fourth zero-crossing, or • after the fourth zero crossing
04	aborted due to hysteresis value exceeding maximum	User-specified hysteresis value, or automatically determined hysteresis value > maximum
05	aborted due to illegal configuration value	The following range checking errors: • Initial loop output value is < 0.0 or > 1.0 • User-specified deviation value is <= hysteresis value, or is > maximum • Initial output step is <= 0.0 or is > maximum • Zero-crossing watchdog interval time is < minimum • Sample time value in loop table is negative
06	aborted due to a numeric error	Illegal floating point number or divide by zero encountered
07	PID instruction was executed with no power flow (manual mode)	PID instruction executed with no power flow while auto-tuning is in progress or is requested
08	auto-tuning allowed only for P, PI, PD, or PID loops	Loop type is not P, PI, PD, or PID

11.6 Notes concerning PV out-of-range (result code 3)

The process variable is considered to be in-range by the auto-tuner if its value is greater than 0.0 and less than 1.0.

If the PV is detected to be out-of-range during the auto-hysteresis sequence, then the tuning is immediately aborted with a process out-of-range error result.

If the PV is detected to be out-of-range between the starting point of the tuning sequence and the fourth zero-crossing, then the output step value is cut in half and the tuning sequence is restarted from the beginning. If a second PV out-of-range event is detected after the first zero-crossing following the restart, then the tuning is aborted with a process out-of-range error result.

Any PV out-of-range event occurring after the fourth zero-crossing results in an immediate abort of the tuning and a generation of a process out-of-range error result.

11.7 PID Tune control panel

STEP 7-Micro/WIN SMART includes a PID Tune control panel that allows you to graphically monitor the behavior of your PID loops. In addition, the control panel allows you to initiate the auto-tune sequence, abort the sequence, and apply the suggested tuning values or your own tuning values.

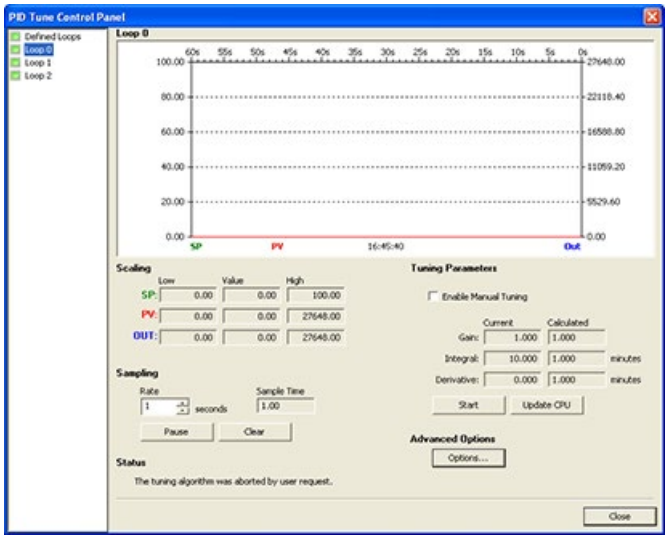
To use the control panel, you must be communicating with the CPU and a wizard-generated configuration for a PID loop must exist in the CPU. The CPU must be in RUN mode for the control panel to display the operation of a PID loop.

To open the PID control panel, use one of the following methods:

- Click the "PID Control Panel" button from the Tools area of the Tools menu ribbon strip.

- Open the Tools folder in the project tree, select the "PID Tune Control Panel" node and press Enter; or double-click the "PID Tune Control Panel" node.

STEP 7-Micro/WIN SMART opens the PID control panel if the connected CPU is in RUN mode:



The PID control panel includes the following fields:

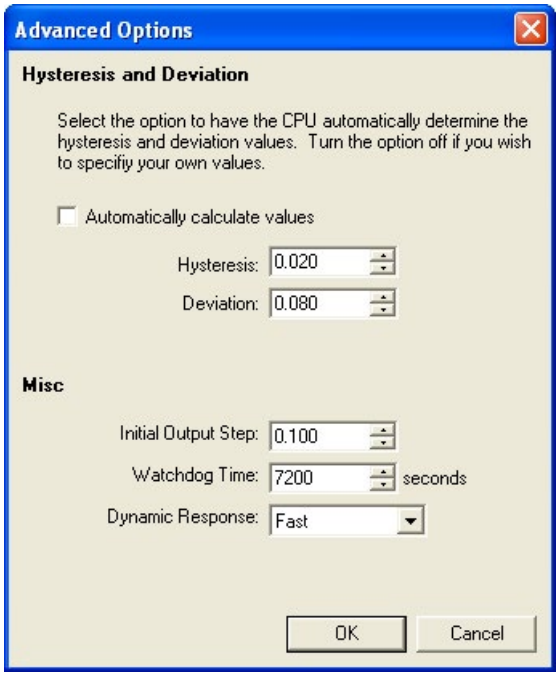
- Current values: The values of the SP (Setpoint), PV (Process Variable), OUT (Output), Sample Time, Gain, Integral time, and Derivative time are displayed. The SP, PV, OUT are shown in green, red, and blue, respectively; the same color legend is used to plot the PV, SP, and OUT values.
- Graphical display: The graphical display shows color-coded plots of the PV, SP, and Output as a function of time. The PV and SP share the same vertical scale which is located at the left hand side of the graph while the vertical scale for the output is located on the right hand side of the graph.
- Tuning Parameters: At the bottom left-hand side of the screen are the Tuning Parameters (Minutes). Here, the Gain, Integral Time, and Derivative Time values are displayed. You click in the "Calculated" column to modify any one of the three sources for these values.
- "Update CPU" button: You can use the" Update CPU" button to transfer the displayed Gain, Integral Time, and Derivative Time values to the CPU for the PID loop that is being monitored. You can use the "Start" button to initiate an auto-tuning sequence. Once an auto-tuning sequence has started, the "Start" button becomes a "Stop" button.

- Sampling: In the "Sampling" area, you can select the graphical display sampling rate from 1 to 480 seconds per sample.

  You can freeze the graph by clicking the "Pause" button. Click the "Resume" button to resume sampling data at the selected rate. To clear the graph, select "Clear" from the right-mouse button within the graph.

- Advanced Options: You can use the "Options" button to further configure parameters for the auto-tuning process. (See the figure below.)

  From the advanced screen, you can check the box that causes the auto-tuner to automatically determine the values for the Hysteresis and Deviation (default setting), or you can enter the values for these fields that minimize the disturbance to your process during the auto-tune procedure.



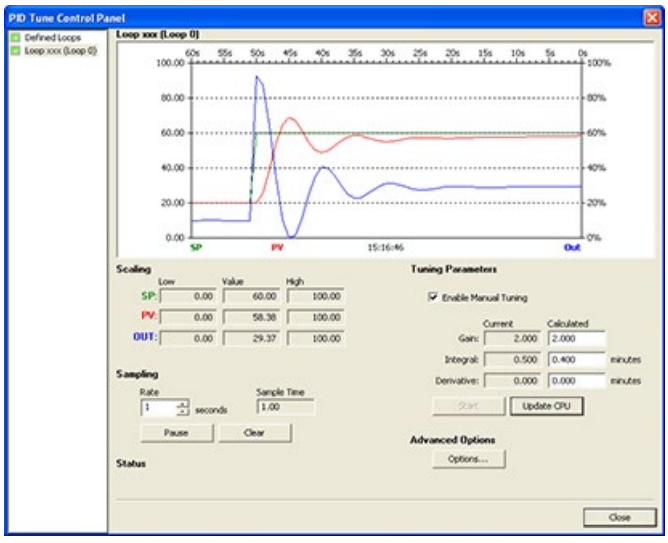
  In the "Dynamic Response" field, use the dropdown button to select the type of loop response (Fast, Medium, Slow, or Very Slow) that you wish to have for your process. Depending upon your process, a fast response may have overshoot and would correspond to an underdamped tuning condition. A medium speed response may be on the verge of having overshoot and would correspond to a critically damped tuning condition. A slow response may not have any overshoot and would correspond to an overdamped tuning condition. A very slow response may not have overshoot and would correspond to a heavily overdamped tuning condition.

  Once you have made the desired selections, click OK to return to the main screen of the PID Tune Control Panel.

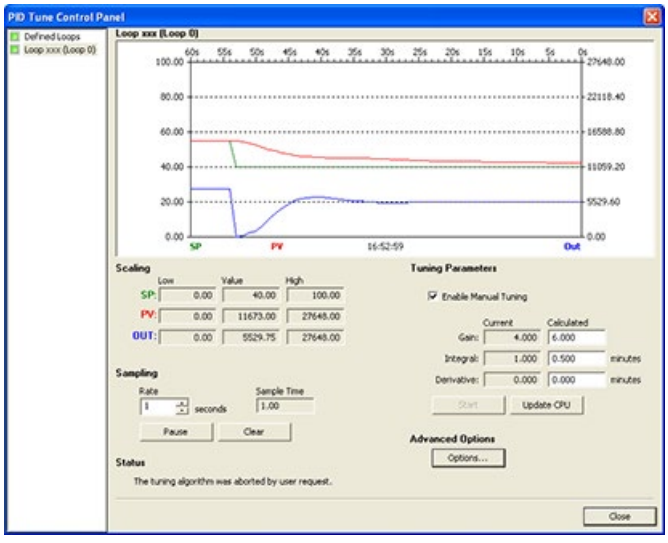
Loop monitoring

After you have completed the auto-tune sequence and have transferred the suggested tuning parameters to the CPU, you can use the control panel to monitor your loop's response to a step change in the setpoint.

In the figure below, the loop responds to a setpoint change with the original tuning parameters (before running auto-tune). Notice the overshoot and the long, damped ringing behavior of the process using the original tuning parameters.



The loop responds to the same setpoint change after applying the values determined by the auto-tune process using the selection for a fast response. Notice that for this process there is no overshoot, but there is just a little bit of ringing.



To eliminate the ringing at the expense of the speed of response, select a medium or a slow response and re-run the auto-tuning process.

After you have a good starting point for the tuning parameters for your loop, you can use the control panel to tweak the parameters. Then you can monitor the loop's response to a setpoint change. In this way, you can fine tune your process for an optimum response in your application.

To simplify the use of PID loop control in your application, STEP 7-Micro/WIN SMART provides a PID wizard (Page 295) to configure your PID loops.

Open loop motion control

12

The S7-200 SMART CPU provides three methods of open loop motion control:

- Pulse Train Output (PTO): Built into the CPU for speed and position control. See Pulse Output instruction. (Note: There is no PTO wizard available. Use the Motion wizard instead.)
- Pulse Width Modulation (PWM): Built into the CPU for speed, position, or duty cycle control. See Pulse Output instruction.
- Axis of Motion: Built into the CPU for speed and position control

The CPU provides three digital outputs (Q0.0, Q0.1, and Q0.3) that you can configure as PTO or PWM outputs by the PLS Instruction, PWM outputs by the PWM wizard, or as motion control outputs by the Motion wizard.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s do not support motion control.

When you configure an output for PTO operation, the CPU generates a 50% duty cycle pulse train for open loop control of the speed and position for either stepper motors or servo motors. The built-in PTO function only provides the pulse train output. Your application program must supply direction and limit controls using I/O built into the PLC or provided by expansion modules.

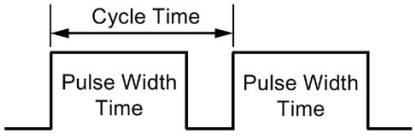
When you configure an output for PWM operation, the CPU fixes the cycle time of the output, and your program controls the pulse width or duty cycle of the pulse. You can use the variations in pulse width to control the speed or position in your application.

The Axis of Motion provides a single pulse train output with integrated direction control and disable outputs. It also includes programmable inputs which allow the CPU to be configured for several modes of operation, including automatic reference point seek. The Axis of Motion provides a unified solution for open loop control of the speed and position for either stepper motors or servo motors.

To simplify the use of motion control in your application, STEP 7-Micro/WIN SMART provides a Motion wizard to configure Axis of Motion and a PWM wizard to configure PWM. The wizards generate motion instructions that you can use to provide dynamic control of speed and motion in your application. For the Axis of Motion, STEP 7-Micro/WIN SMART also provides a control panel that allows you to control, monitor, and test your motion operations.

12.1 Using the PWM output

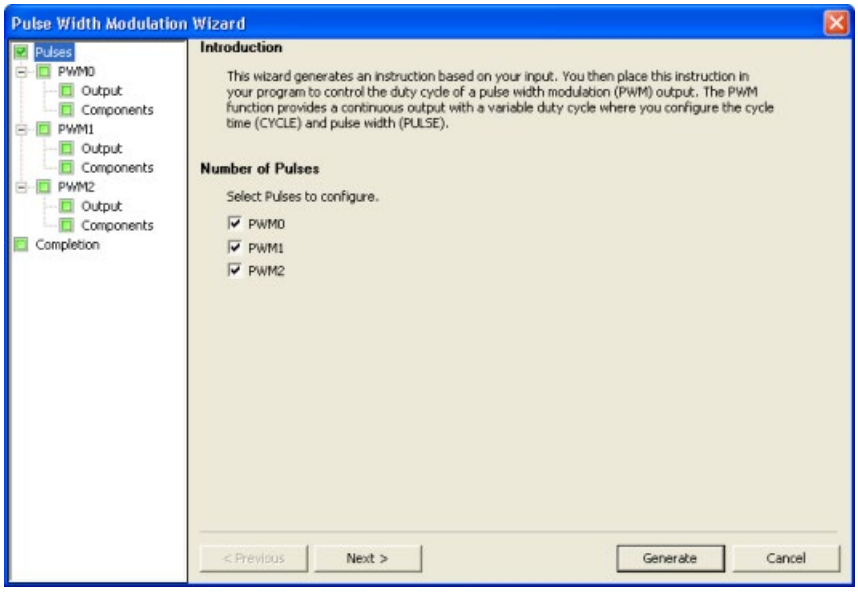
PWM provides a fixed cycle time output with a variable duty cycle. The PWM output runs continuously after being started at the specified frequency (cycle time). The pulse width is varied as required to affect the desired control. Duty cycle can be expressed as a percentage of the cycle time or as a time value corresponding to pulse width. The pulse width can vary from 0% (no pulse, always off) to 100% (no pulse, always on). See the following figure.



Since the PWM output can be varied from 0% to 100%, it provides a digital output that in many ways is analogous to an analog output. For example the PWM output can be used to control the speed of a motor from stop to full speed or it can be used to control the position of a valve from closed to full open.

12.1.1 Configuring the PWM output

To configure one of the built-in outputs for PWM control, use the PWM wizard.



Use one of the following methods to open the PWM wizard:

- Click the "PWM" button from the Wizards area of the Tools menu.

- Open the Wizards folder in the project tree and double-click "PWM", or select "PWM" and press the Enter key.

1. Select a pulse generator.
2. Change the name of a PWM channel, if required.
3. Configure the PWM channel output time base.
4. Generate project components.
5. Use the PWMx_RUN subroutine to control the duty cycle of your PWM output.

Note

PWM channels are hard-coded to specific outputs:

- PWM0 is assigned to Q0.0.
- PWM1 is assigned to Q0.1.
- PWM2 is assigned to Q0.3.

12.1.2 PWMx_RUN subroutine

To simplify the use of Pulse Width Modulation (PWM) control in your application, STEP 7-Micro/WIN SMART provides a PWM wizard (Page 674) to configure your on-board PWM generators and control the duty cycle of a PWM output.

The PWMx_RUN subroutine is used to execute PWM under program control.

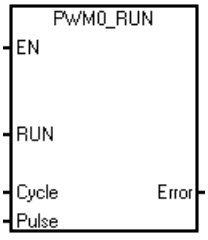
LAD / FBD	STL	Description
	`CALL PWMx_RUN, Cycle, Pulse, Error`	The PWMx_RUN subroutine allows you to control the duty cycle of the output by varying the pulse width from 0 to the pulse width of the cycle time.

Table 12- 1 Parameters for the PWMx_RUN subroutine

Inputs/Outputs	Data types	Operands
Cycle, Pulse	Word	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *AC, *LD, Constant
Error	Byte	IB, QB, VB, MBV, SMB, LB, AC, *VD, *AC, *LD, Constant

The Cycle input is a word value that defines the cycle time for the pulse width modulation (PWM) output. The allowed range is from 2 to 65535 when milliseconds is the time base and 10 to 65535 when microseconds is the time base.

The Pulse input is a word value that defines the pulse width (Duty Cycle) for the PWM output. The allowed range of values is from 0 to 65535 units of the time base (microseconds or milliseconds) that was specified within the PWM wizard.

The Error is a byte value returned by the PWMx_RUN subroutine that indicates the result of execution. See the following table for a description of the possible error codes.

Table 12- 2 PWMx_RUN instruction error codes

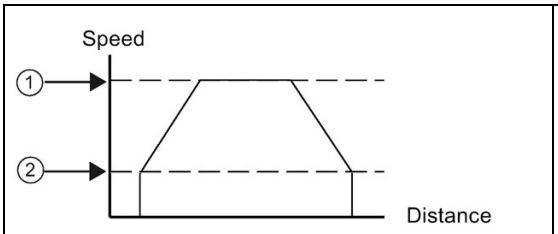
Error code	Description
0	No error, normal completion
131	Pulse generator already in use by another PWM or by a motion axis, or illegal time base change

12.2 Using motion control

The motion control built into the CPU uses an Axis of Motion to control both the speed and motion of a stepper motor or a servo motor.

Using the Axis of Motion requires expertise in the field of motion control. This chapter is not meant to educate the novice in this subject. However, it provides fundamental information that will help as you use the Motion wizard to configure the Axis of Motion for your application.

12.2.1 Maximum and start/stop speeds



The Motion wizard prompts you for the maximum speed (MAX_SPEED) and Start/Stop Speed (SS_SPEED) for your application.

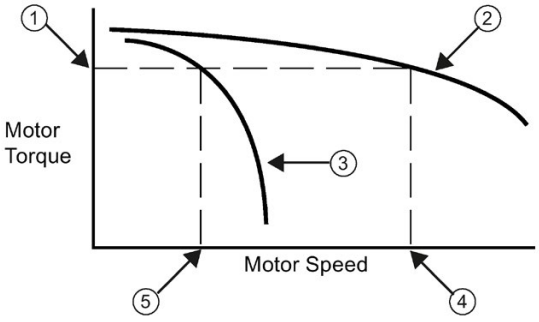
① MAX_SPEED

② SS_SPEED

- MAX_SPEED: Enter the value for the optimum operating speed of your application within the torque capability of your motor. The torque required to drive the load is determined by friction, inertia, and the acceleration/deceleration times.
- The Motion wizard calculates and displays the minimum speed that can be controlled by the Axis of Motion based upon the MAX_SPEED that you specify.
- SS_SPEED: Enter a value within the capability of your motor to drive your load at low speeds. If the SS_SPEED value is too low, the motor and load could vibrate or move in short jumps at the beginning and end of travel. If the SS_SPEED value is too high, the motor could lose pulses on start up, and the load could overdrive the motor when attempting to stop.

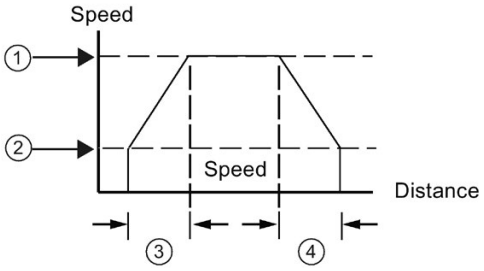
Motor data sheets have different ways of specifying the start/stop (or pull-in/pull-out) speed for a motor and given load. Typically, a useful SS_SPEED value is 5% to 15% of the

MAX_SPEED value. To help you select the correct speeds for your application, refer to the data sheet for your motor. The following figure shows a typical motor torque/speed curve.



① Torque required to drive the load
② Motor torque versus speed characteristic
③ Start/Stop speed versus torque: This curve moves towards lower speed as the load inertia increases.
④ Maximum speed that the motor can drive the load: MAX_SPEED should not exceed this value.
⑤ Start/Stop speed (SS_SPEED) for this load

12.2.2 Entering the acceleration and deceleration times



As part of the configuration, you set the acceleration and deceleration times. The default setting for both the acceleration time and the deceleration time is 1 second. Typically, motors can work with less than 1 second. See the following figure.

① MAX_SPEED
② SS_SPEED
③ ACCEL_TIME
④ DECEL_TIME

Note

Motor acceleration and deceleration times are determined by trial and error. You should start by entering a large value. Optimize these settings for the application by gradually reducing the times until the motor starts to stall.

Specify the following times in milliseconds:

- ACCEL_TIME: Time required for the motor to accelerate from SS_SPEED to MAX_SPEED. Default = 1000 ms
- DECEL_TIME: Time required for the motor to decelerate from MAX_SPEED to SS_SPEED. Default = 1000 ms

12.2.3 Configuring the motion profiles

A profile is a pre-defined motion description consisting of one or more speeds of movement that effect a change in position from a starting point to an ending point. You do not have to define a profile in order to use the Axis of Motion. The Motion wizard provides instructions for you to use to control moves without running a profile.

A profile is programmed in steps consisting of an acceleration/deceleration to a target speed followed by a fixed number of pulses at the target speed. In the case of single step moves or the last step in a move, there is also a deceleration from the target speed (last target speed) to stop.

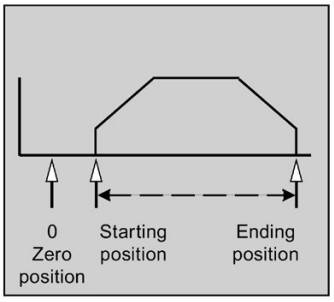
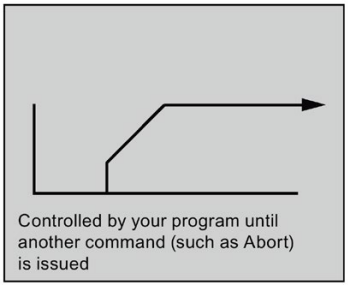
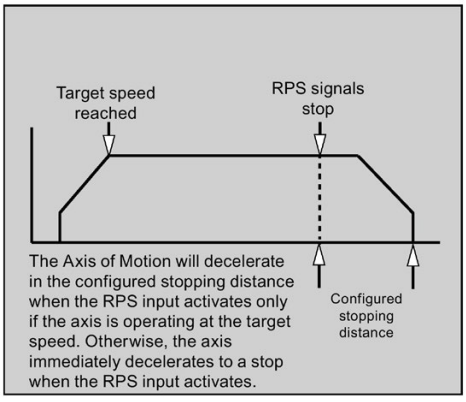
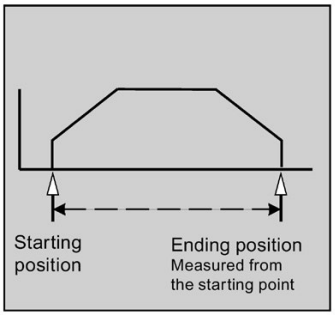
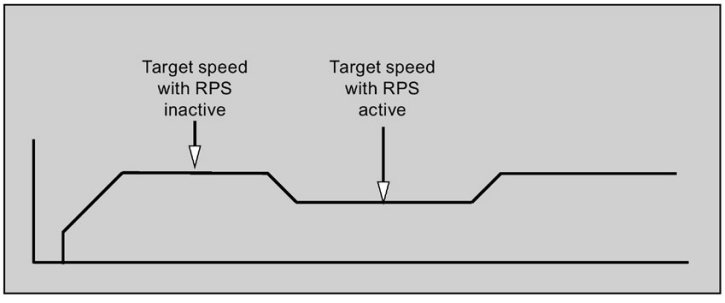
The Axis of Motion supports a maximum of 32 profiles.

Defining the motion profile

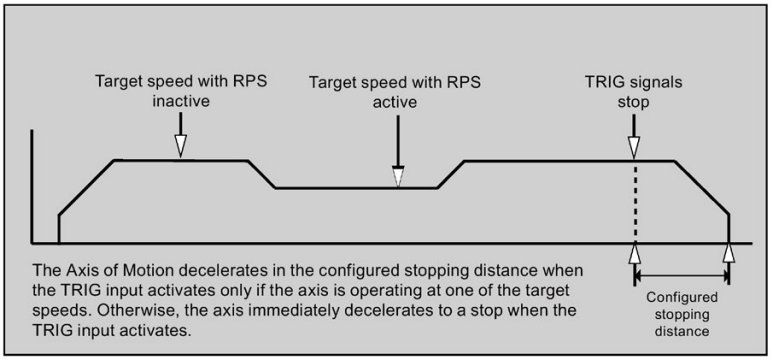
The Motion wizard guides you through a motion profile definition, where you define each motion profile for your application. For each profile, you select the operating mode and define the specifics of each individual step for the profile. The Motion wizard also allows you to define a symbolic name for each profile by simply entering the symbol name as you define the profile.

Selecting the mode of operation for the profile

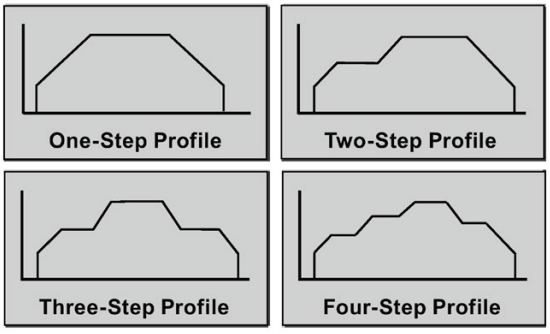
You configure the profile according to the mode of operation desired. The Axis of Motion supports absolute position, relative position, single-speed continuous rotation, and two-speed continuous rotation. The following figure shows the different modes of operation.

<table>
<tr><th colspan="2">Mode selections for the Axis of Motion</th></tr>
<tr><td>Absolute Position</td><td>Single-Speed, Continuous Rotation</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Single-speed, Continuous Rotation, with triggered stop</td><td>Relative Position</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Two-Speed, Continuous Rotation</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td colspan="2">Two-Speed, Continuous Rotation, with triggered stop</td></tr>
</table>

Mode selections for the Axis of Motion



Creating the steps of the profile



A step is a fixed distance that a tool moves, including the distance covered during acceleration and deceleration times. The Axis of Motion supports a maximum of 16 steps in each profile.

You specify the target speed and ending position or number of pulses for each step. Additional steps are entered one at a time. The figure illustrates a one-step, two-step, three-step, and a four-step profile.

Notice that a one-step profile has one constant speed segment, a two-step profile has two constant speed segments, and so on. The number of steps in the profile matches the number of constant speed segments of the profile.

12.3 Features of motion control

Motion control provides the functionality and performance that you need for open-loop position control in up to three Axes of Motion:

- Provides high-speed control, with a range from 20 pulses per second up to 100,000 pulses per second
- Supports both jerk (S curve) or linear acceleration and deceleration
- Provides a configurable measuring system that allows you to enter data either as engineering units (such as inches or centimeters) or as a number of pulses
- Provides configurable backlash compensation
- Supports absolute, relative, and manual methods of position control

- Provides continuous operation
- Provides up to 32 motion profiles, with up to 16 speed changes per profile
- Provides four different reference-point seek modes, with a choice of the starting seek direction and the final approach direction for each sequence
- Provides support for SINAMICS V90 drives

You use STEP 7-Micro/WIN SMART to create all of the configuration and profile information used by the Axis of Motion. This information is downloaded to the CPU with your program blocks.

Motion control provides six digital inputs and four digital outputs that provide the interface to your motion application. See the following table. These inputs and outputs are local to the CPU. The CPU technical specifications (Page 756) provide detailed information for the CPUs and include wiring diagrams for connecting each CPU to some of the more common motor driver/amplifier units.

Table 12- 3 Motion control CPU inputs to configure

Signal	Description
STP	The STP input causes the CPU to stop the motion in progress. You can select the desired operation of STP within the Motion wizard.
RPS	The RPS (Reference Point Switch) input establishes the reference point or home position for absolute move operations. In some modes, the RPS input is also used to stop the motion in progress after travelling a specified distance.
ZP	The ZP (Zero Pulse) input helps establish the reference point or home position. Typically, the motor driver/amplifier pulses ZP once per motor revolution. Note: Only used in RP Seek Modes of 3 and 4.
LMT+ LMT-	LMT+ and LMT- inputs establish the maximum limits for motion travel. The Motion wizard allows you to configure the operation of LMT+ and LMT- inputs.
TRIG	The TRIG (Trigger) input causes the CPU, in some modes, to stop the motion in progress after travelling a specified distance.

⚠ WARNING

Risks with changes to filter time for digital input channel

If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 12.8 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 12.8 ms may not be detected or counted.

This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

To ensure that a new filter time goes immediately into effect, cycle the power of the CPU.

Table 12- 4 Motion control CPU hard-coded outputs

Signal	Description
P0 P1	P0 and P1 are pulse outputs that control the movement and direction of movement of the motor.
DIS	DIS is an output used to disable or enable the motor driver/amplifier.

12.4 Programming an Axis of Motion

STEP 7-Micro/WIN SMART provides easy-to-use tools for configuring and programming the Axis of Motion. Simply follow these steps:

1. Configure the Axis of Motion: STEP 7-Micro/WIN SMART provides a Motion wizard for creating the configuration/profile table and the position instructions. See "Configuring the Axis of Motion" for information about configuring the Axis of Motion.
2. Test the operation of the Axis of Motion: STEP 7-Micro/WIN SMART provides a Motion control panel for testing the wiring of the inputs and outputs, the configuration of the Axis of Motion, and the operation of the motion profiles. See "Monitoring the Axis of Motion with the Motion control panel" for information about the Motion control panel.
3. Create the program to be executed by the CPU: The Motion wizard automatically creates the motion instructions that you insert into your program. See "Instructions created by the Motion wizard for the Axis of Motion" for information about the motion instructions. Insert the following instructions into your program:
   - To enable the Axis of Motion, insert an AXISx_CTRL instruction. Use SM0.0 (Always On) to ensure that this instruction is executed every scan.
   - To move the motor to a specific location, use an AXISx_GOTO or an AXISx_RUN instruction. The AXISx_GOTO instruction moves to a location specified by the inputs from your program. The AXISx_RUN instruction executes the motion profiles you configured with the Motion wizard.
   - To use absolute coordinates for your motion, you must establish the zero position for your application. Use an AXISx_RSEEK or an AXISx_LDPOS instruction to establish the zero position.
   - The other instructions that are created by the Motion wizard provide functionality for typical applications and are optional for your specific application.
4. Compile your program and download the system block, data block, and program block to the CPU.

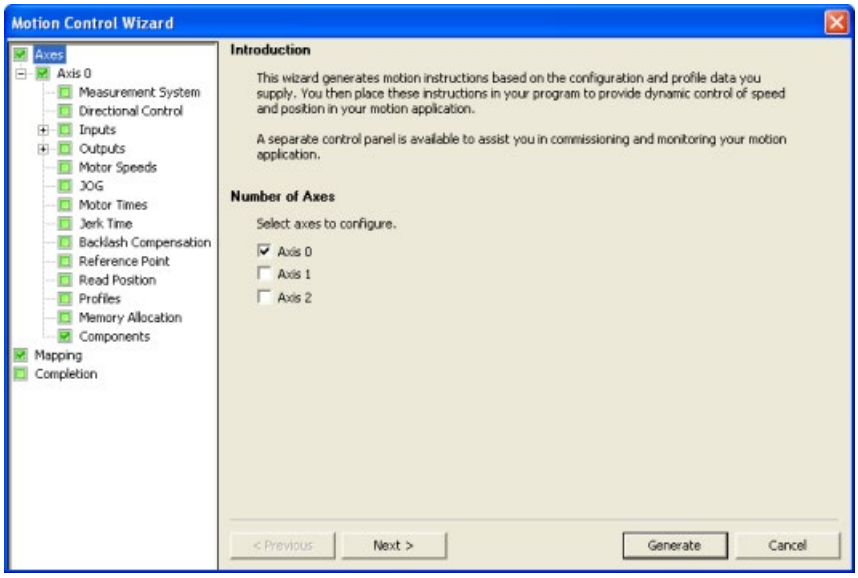
Note

Make sure to match the measurement system configuration to the pulses/revolution and distance/revolution specifications of your stepper/servo motor controller system.

12.5 Configuring an Axis of Motion

Configuration/profile table

You must create a configuration/profile table for the Axis of Motion in order for the CPU to control your motion application. The Motion wizard makes the configuration process quick and easy by leading you step-by-step through the configuration process. Refer to the "Advanced Topics" section of this chapter for detailed information about the configuration/profile table.



The Motion wizard also allows you to create the configuration/profile table offline. You can create the configuration without being connected to a CPU.

Starting the Motion wizard

To start the Motion wizard, either click the Tools icon in the navigation bar and then double-click the Motion wizard icon, or select the Tools > Motion wizard menu command.

Selecting type of measurement

Select the measurement system: You can select either engineering units or pulses:

- If you select pulses, no other information is required.
- If you select engineering units, enter the number of pulses required to produce one revolution of the motor (refer to the data sheet for your motor or drive), the base unit of measurement (such as inch, foot, millimeter, or centimeter), and the distance traveled in one revolution of the motor.

If you change the measurement system later, you must delete the entire configuration including any instructions generated by the Motion wizard. You must then enter your selections consistent with the new measurement system.

Configuring the input pin locations

You can program inputs related to motion control, to include STP, LMT-, LMT+, RPS, TRIG, and ZP, with a configuration in SDB0.

Table 12- 5 STP, RPS, LMT+, LMT-, TRIG, and ZP pin locations

Pin definition for inputs	Description
LMT+, LMT-, STP, RPS, TRIG	Input pin 0 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.0).
	Input pin 1 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.1).
	Input pin 2 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.2).
	Input pin 3 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.3).
	Input pin 4 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.4).
	Input pin 5 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.5).
	Input pin 6 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.6).
	Input pin 7 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.7).
	Input pin 8 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.0).
	Input pin 9 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.1).
	Input pin 10 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.2).
	Input pin 11 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.3).
ZP HSC	HSC0 of the CPU acts as the ZP input (I0.0).
	HSC1 of the CPU acts as the ZP input (I0.1).
	HSC2 of the CPU acts as the ZP input (I0.2).
	HSC3 of the CPU acts as the ZP input (I0.3).
	HSC4 of the CPU acts as the ZP input (I0.6).
	HSC5 of the CPU acts as the ZP input (I1.0).

Note

After you configure an input to a specific function (for example, RPS) for a particular Axis of Motion, you cannot use that input for any other Axis of Motion or for any other input, counter, or interrupt function.

Note

High-speed input wiring must use shielded cables

Use shielded cable with a maximum length of 50 m, when connecting HSC input channels I0.0, I0.1, I0.2, and I0.3.

Mapping the I/O

STEP 7-Micro/WIN SMART enforces a fixed output assignment for PWM and Axis of Motion.

P0 and P1 outputs

At a minimum, any axis that is enabled has a P0 output pin configured for it. It may also have P1 output if its "Phase" configuration is anything other than "Single-phase (1 output)". Refer to the "Editing default input and output configuration" section for more information. These output pins are hard-coded to a specific output, depending on the following criteria:

Axis 0	• P0 for Axis 0 is always configured for Q0.0. • P1 for Axis 0 is configured for Q0.2 if the "Phase" for the axis is not configured to "Single phase (1 output)".
Axis 1	• P0 for Axis 1 is always configured for Q0.1. • P1 for Axis 1 is mapped in two possible locations based upon axis configuration as follows: – If the "Phase" for Axis 1 is configured for "Single-phase (1 output)", then no P1 output is assigned. – If the "Phase" for Axis 1 is configured for "Two-phase (2 output)" or "AB quadrature phase (2 output)", then P1 is configured for Q0.3. – Else, P1 for Axis 1 is configured for Q0.7.
Axis 2	• P0 for Axis 2 is always configured for Q0.3. • P1 for Axis 2 is configured for Q1.0 if the "Phase" for the axis is not configured to "Single phase (1 output)". • Axis 2 is not available for use if the configured "Phase" for Axis 1 is configured to "Two-phase (2 output)" or "AB quadrature phase (2 output)".

DIS outputs

If an axis has a DIS output configured, then an entry is present in the mapping table for that output. The DIS output is also hard-coded to specific outputs based upon the following rules:

- DIS for Axis 0 is always configured for Q0.4.
- DIS for Axis 1 is always configured for Q0.5.
- DIS for Axis 2 is always configured for Q0.6.

Pulse output units are connected to standard 24V outputs.

Editing default input and output configuration

To change or view the default configuration of the integrated inputs/outputs select the required input/output node:

- In the "Active level" field, use the dropdown list to select the active level (High or Low). When the level is set to High, a logic 1 is read when current is flowing in the input. When the level is set to Low, a logic 1 is read when there is no current flow in the input. A logic 1 level is always interpreted as meaning the condition is active. The LEDs are illuminated when current flows in the input, regardless of activation level. (Default = active high)
- In the "System Block", "Digital Inputs" node, you can select the filter time constant (0.20 ms to 12.80 ms) for the STP, RPS, LMT+, LMT-, and TRIG inputs. Increasing the filter time constant eliminates more noise, but it also slows down the response time to a signal state change. (Default = 6.4 ms)

> **⚠ WARNING**
>
> **Risks with changes to filter time for digital input channel**
>
> If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 12.8 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 12.8 ms may not be detected or counted.
>
> This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.
>
> To ensure that a new filter time goes immediately into effect, a power cycle of the CPU must be applied.

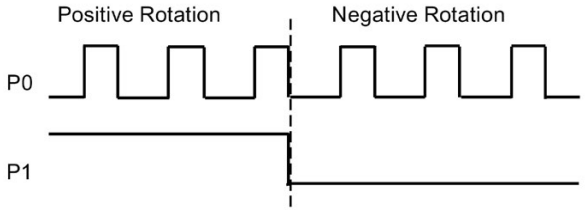
- In the "Directional Control" node, you can select the following "Phasing" modes:
  - Single phase (2 output)
  - Two-phase (2 output)
  - AB quadrature phase (2 output)
  - Single phase (1 output)

  You can also select the "Polarity" (positive or negative) of the outputs.

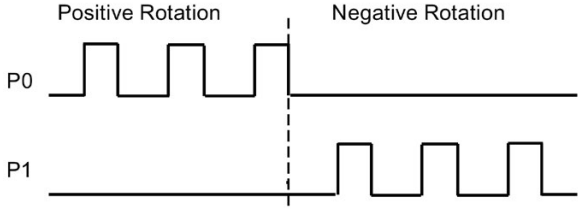
Phasing

You have four options for the "Phasing" interface to the stepper/servo drive. These options are as follows:

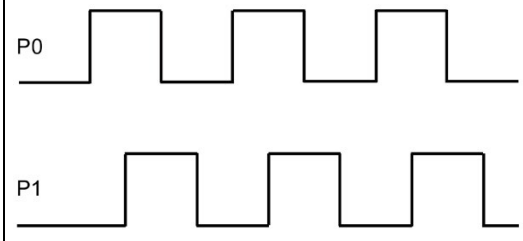
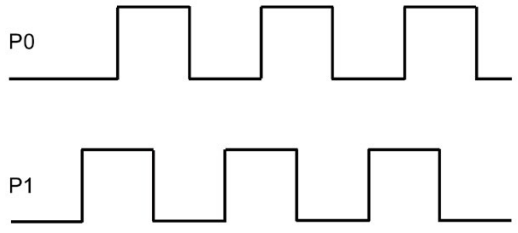
- Single phase (2 output): If you select the single phase (2 output) option, then one output (P0) controls the pulsing, and one output (P1) controls the direction. P1 is high (active) if pulsing is in the positive direction. P1 is low (inactive) if pulsing is in the negative direction. Single phase (2 output) is shown in the figure below (assuming positive polarity):



- Two-phase (2 output): If you select the Two-phase (2 output) option, then one output (P0) pulses for positive directions, and a different output (P1) pulses for negative directions. Two-phase (2 output) is shown in the figure below (assuming positive polarity):



- AB quadrature phase (2 output): If you select the AB quadrature phase (2 output) option, then both outputs pulse at the speed specified, but 90 degrees out-of-phase. The AB quadrature phase (2 output) is a 1X configuration, meaning a generated pulse is measured from one positive transition of the output to the next positive transition of the same output. In this case, the direction is determined by which output transitions high first. P0 leads P1 for the positive direction. P1 leads P0 for the negative direction. AB quadrature phase (2 output) is shown in the figures below (assuming positive polarity):

AB quadrature phase (2 output)	
(Positive polarity): positive rotation	(Positive polarity): negative rotation
	
P0 leads P1	P1 leads P0

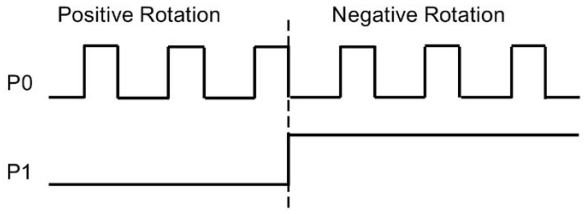
- Single phase (1 output): If you select the single phase (1 output) option, then the output (P0) controls the pulsing. Only positive motion commands are accepted by the CPU in this mode. The Motion wizard restricts you from making illegal negative configurations when you select this mode. You can save an output if your motion application is in one

direction only. Single phase (1 output) is shown in the figure below (assuming positive polarity):

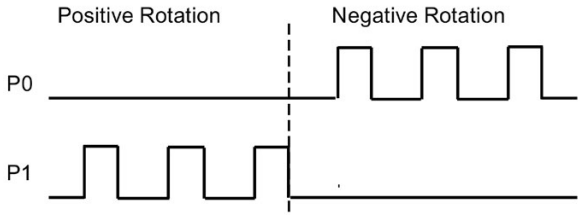
Polarity

You can switch positive and negative directions with the "Polarity" parameter. If the motor is wired in the wrong direction, this is typically done. You can avoid re-wiring the hardware by setting this parameter to negative. The negative setting changes the output operation as follows:

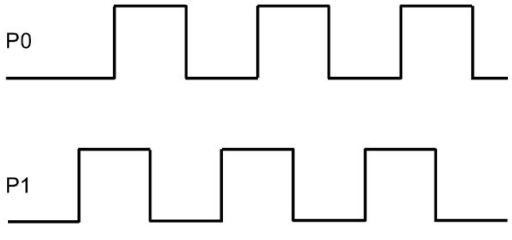
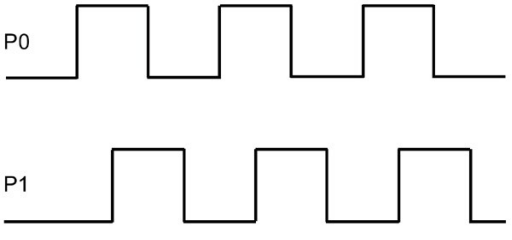
- Single phase (2 output): P1 is low (inactive) if pulsing is in the positive direction. P1 is high (active) if pulsing is in the negative direction. This is shown in the figure below:



- Two-phase (2 output): P0 pulses for negative directions. P1 pulses for positive directions. This is shown in the figure below:



- AB quadrature phase (2 output): P0 leads P1 for a negative direction. P1 leads P0 for a positive direction. This is shown in the figure below:

AB quadrature phase (2 output)	
(Negative polarity): positive rotation	(Negative polarity): negative rotation
	
P1 leads P0	P0 leads P1

- Single phase (1 output): Negative polarity is not allowed in this phasing mode.

The default setting for the "Directional Control" dialog is "Single phase (2 output)" and "Positive polarity".

Note

You cannot choose to which pins P0 and P1 are configured; this is hardcoded to a specific pin. Refer to the Mapping I/O section (preceding this section) for the pin mapping list.

WARNING

Safety precautions when using an Axis of Motion

The limit and stop functions in the Axis of Motion are electronic logic implementations that do not provide the level of protection provided by electromechanical controls.

Control devices and Axis of Motion functions can fail in unsafe conditions, which can result in unpredictable operation of controlled equipment. Such unpredictable operations could result in death or serious personal injury, and/or property damage.

Consider using an emergency stop function, electromechanical overrides, or redundant safeguards that are independent of the Axis of Motion and the CPU.

Configure response to physical inputs

1. Select the response to the LMT+, the LMT-, and the STP inputs.
2. Use the dropdown list to select: decelerate to a stop (default) or immediate stop.

Entering maximum start and stop speed

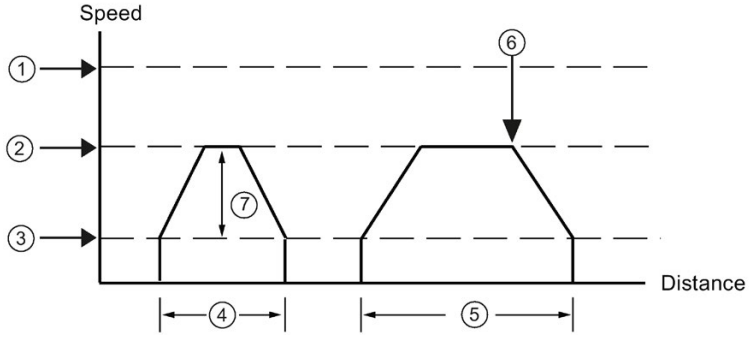
Enter the maximum speed (MAX_SPEED) and Start/Stop Speed (SS_SPEED) for your application.

Entering jog parameters

Enter the JOG_SPEED and the JOG_INCREMENT values:

- JOG_SPEED: The JOG_SPEED (Jog speed for the motor) is the maximum speed that can be obtained while the JOG command remains active.
- JOG_INCREMENT: Distance that the tool is moved by a momentary JOG command.

  The following figure shows the operation of the Jog command. When the Axis of Motion receives a Jog command, it starts a timer. If the Jog command is terminated before 0.5 seconds has elapsed, the Axis of Motion moves the tool the amount specified in the JOG_INCREMENT at the speed defined by JOG_SPEED. If the Jog command is still active when the 0.5 seconds have elapsed, the Axis of Motion accelerates to the JOG_SPEED. Motion continues until the Jog command is terminated. The Axis of Motion then performs a decelerated stop. You can enable the Jog command either from the Motion Control Panel or with a motion instruction. A representation of a JOG operation is shown in the figure below.



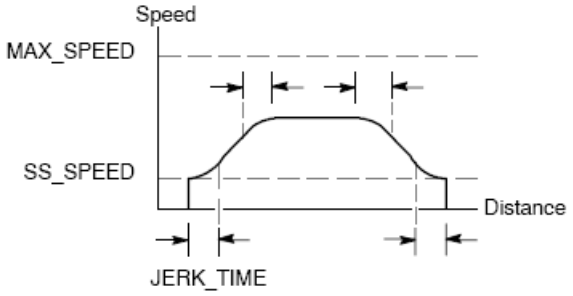
① MAX_SPEED
② JOG_SPEED
③ SS_SPEED
④ JOG_INCREMENT: JOG command is active for less than 0.5 seconds.
⑤ JOG command is active for more than 0.5 seconds.
⑥ JOG command is terminated (Starts ramp from JOG_SPEED to SS_SPEED).
⑦ The speed reached can be anywhere from the SS_SPEED to the JOG_SPEED, depending on the length of the JOG_INCREMENT.

Entering acceleration time

Enter the acceleration and deceleration times in the edit boxes.

Entering jerk time

Available on certain types of moves, jerk compensation provides smoother position control by reducing the jerk (rate of change) in the acceleration and deceleration parts of the motion profile. See the following figure:



Jerk compensation is also known as "S curve profiling". This compensation is applied equally to the beginning and ending portions of both the acceleration and deceleration curve. Jerk compensation is not applied to the initial and final step between zero speed and SS_SPEED.

Specify the jerk compensation by entering a time value (JERK_TIME). This is the time required for acceleration to change from zero to the maximum acceleration rate. A longer jerk time yields smoother operation with a smaller increase in total cycle time than would be obtained by decreasing the ACCEL_TIME and DECEL_TIME. A value of 0 ms (the default value) indicates that no compensation is to be applied.

Note

A good first value for JERK_TIME is 40% of ACCEL_TIME.

Note

Jerk compensation is not available for two speed moves, manual speed change moves, aborted moves, and automatic deceleration reactions upon reaching a limit or STP input.

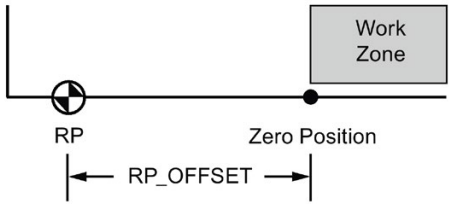
Configuring the Backlash compensation

Backlash compensation: Distance that the motor must move to eliminate the backlash (slack) in the system on a direction change. Backlash compensation is always a positive value:

- Default = 0
- Choose a Reference Point search sequence to use backlash.

Configuring reference point and seek parameters

1. Select using a reference point or not using a reference point for your application:
   - If your application requires that movements start from or be referenced to an absolute position, you must establish a reference point (RP) or zero position that fixes the position measurements to a known point on the physical system.
   - If a reference point is used, you will want to define a way to automatically relocate the reference point. The process of automatically locating the reference point is called Reference Point Seek. Defining the Reference Point Seek process requires two steps in the wizard.
2. Enter the Reference Point seek speeds (a fast seek speed and a slow seek speed):
   - **RP_FAST** is the initial speed the module uses when performing an RP seek command. Typically, the RP_FAST value is approximately 2/3 of the MAX_SPEED value.
   - **RP_SLOW** is the speed of the final approach to the RP. A slower speed is used on approach to the RP, so as not to miss it. Typically, the RP_SLOW value is the SS_SPEED value.
3. Define the initial seek direction and the final reference point approach direction:
   - **RP_SEEK_DIR** is the initial direction for the RP seek operation. Typically, this is the direction from the work zone to the vicinity of the RP. Limit switches play an important role in defining the region that is searched for the RP. When performing a RP seek operation, encountering a limit switch can result in a reversal of the direction, which allows the search to continue. (Default = Negative)
   - **RP_APPR_DIR** is the direction of the final approach to the RP. To reduce backlash and provide more accuracy, the reference point should be approached in the same direction used to move from the RP to the work zone. (Default = Positive)
4. The Motion wizard provides advanced reference point options that allow you to specify an RP offset (RP_OFFSET), which is the distance from the RP to the zero position. See the following figure:
   - **RP_OFFSET**: Distance from the RP to the zero position of the physical measuring system.
   - Default = 0



5. The Axis of Motion provides a reference point switch (RPS) input that is used when seeking the RP. The RP is identified by a method of locating an exact position with respect to the RPS. The RP can be centered in the RPS Active zone, the RP can be located on the edge of the RPS Active zone, or the RP can be located a specified number of zero pulse (ZP) input transitions from the edge of the RPS Active zone.
6. You can configure the sequence that the Axis of Motion uses to search for the reference point. The following figure shows a simplified diagram of the default RP search sequence. Select one of the following options for the RP search sequence:
   - **RP Seek mode 0**: Does not perform a RP seek sequence
   - **RP Seek mode 1**: The RP is where the RPS input goes active on the approach from the work zone side. (Default)
   - **RP Seek mode 2**: The RP is centered within the active region of the RPS input.
   - **RP Seek mode 3**: The RP is located outside the active region of the RPS input. RP_Z_CNT specifies how many ZP (Zero Pulse) input counts should be received after the RPS becomes inactive.
   - **RP Seek mode 4**: The RP is generally within the active region of the RPS input. RP_Z_CNT specifies how many ZP (Zero Pulse) input counts should be received after the RPS becomes active.

RP seek mode 1

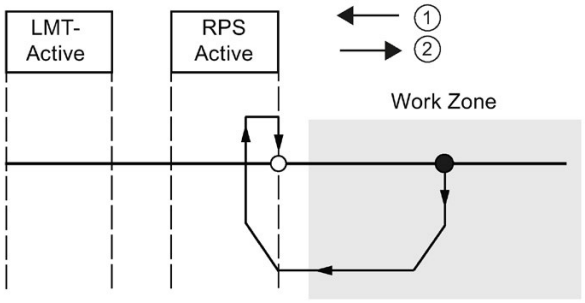


Figure 12-1 ①: RP seek direction
②: RP approach direction

Note

The RPS Active region (which is the distance that the RPS input remains active) must be greater than the distance required to decelerate from the RP_FAST speed to the RP_SLOW speed. If the distance is too short, the Axis of Motion generates an error.

Defining the motion profile

1. In the motion profile definition screen, click the new profile button to enable defining the profile.
2. Choose the desired mode of operation:
   - For an absolute position profile:

     Fill in the target speed and the ending position.

     If more than one step is needed, click the new step button and fill in the step information as required.
   - For a relative position profile:

     Fill in the target speed and the ending position.

     If more than one step is needed, click the new step button and fill in the step information as required.
   - For a single-speed, continuous rotation:

     Enter the target speed value in the edit box.

     Select the direction of rotation.

     If you wish to terminate the single speed, continuous rotation move using the RPS input, click the checkbox.

     Fill in the distance to move after the RPS input goes active (RPS input must be enabled).
   - For a two-speed, continuous rotation (RPS input must be enabled):

     Enter the target speed value when RPS is inactive in the edit box.

     Enter the target speed value when RPS is active in the edit box.

     Select the direction of rotation.

     If you wish to terminate the two speed, continuous rotation move using the TRIG input, click the checkbox. (TRIG input must be enabled.)

     Fill in the distance to move after the TRIG input goes active.
3. Define as many profiles and steps as you need to perform the desired movement.

Finishing the configuration

1. After you have configured the operation of the Axis of Motion, you simply click "Generate".

   The Motion wizard performs the following tasks:

   – Inserts the axis configuration and profile table into the system block and data block for your CPU program

   – Creates a global symbol table for the motion parameters

   – Adds the motion instruction subroutines into the project program block for you to use in your application

2. You can run the Motion wizard again in order to modify any configuration or profile information.

Note

Because the Motion wizard makes changes to the program block, the data block, and the system block, you must download all three blocks to the CPU. Otherwise, the Axis of Motion will not have all the program components that it needs for proper operation.

12.6 Subroutines created by the Motion wizard for the Axis of Motion

You must ensure for each motion action that only one motion subroutine is active at a time in addition to the AXISx_CTRL, which must be active every scan. Each motion subroutine is prefixed with an "AXISx_" where "x" is the axis number channel. There are 13 motion subroutines.

Motion subroutine	Description
AXISx_CTRL (Page 697)	Provides initialization and overall control of the axis
AXISx_MAN (Page 698)	Used for manual mode operation of the axis
AXISx_GOTO (Page 700)	Commands the axis to go to a specified location
AXISx_RUN (Page 701)	Commands the axis to execute a configured motion profile
AXISx_RSEEK (Page 702)	Initiates a reference point seek operation
AXISx_LDOFF (Page 703)	Establishes a new zero position that is offset from the reference point position
AXISx_LDPOS (Page 704)	Changes the axis position to a new value
AXISx_SRATE (Page 705)	Modifies the configured acceleration, deceleration, and jerk compensation times
AXISx_DIS (Page 706)	Controls the DIS output

Motion subroutine	Description
AXISx_CFG (Page 706)	Reads the configuration block and updates the axis setup as required
AXISx_CACHE (Page 707)	Pre-caches a configured motion profile
AXISx_RDPOS (Page 708)	Returns the current axis position
AXISx_ABSPOS (Page 709)	Reads the absolute position value from a SINAMICS V90 servo drive

Note

The motion subroutines increase the amount of memory required for your program by up to 1700 bytes. You can delete unused motion subroutines to reduce the amount of memory required. To prevent the generation of unneeded motion subroutines, uncheck the "Generate" box for each unneeded subroutine in the "Components" node of the Motion wizard. To restore generation of a particular motion subroutine, start the Motion wizard again, navigate to the "Components" node, and check the "Generate" box for the subroutine. Click the "Generate" button to rebuild the wizard-generated subroutines.

See also

Using the Motion wizard (Page 673)

12.6.1 Guidelines for using the Motion subroutines

You must ensure that only one motion subroutine is active at a time.

You can execute the AXISx_RUN and AXISx_GOTO from an interrupt routine as long as the interrupt is called cyclically. However, it is very important that you do not attempt to start a motion subroutine in an interrupt routine if the Axis of Motion is busy processing another command. If you start a subroutine in an interrupt routine, then you can use the outputs of the AXISx_CTRL subroutine to monitor when the Axis of Motion has completed the movement.

The Motion wizard automatically configures the values for the speed parameters (Speed and C_Speed) and the position parameters (Pos or C_Pos) according to the measurement system that you selected. For pulses, these parameters are DINT values. For engineering units, the parameters are REAL values for the type of unit that you selected. For example: selecting centimeters (cm) stores the position parameters as REAL values in centimeters and stores the speed parameters as REAL values in centimeters per second (cm/sec).

Some "generate" guidelines when using motion subroutines are as follows:

- Insert the AXISx_CTRL subroutine in your program, and use the SM0.0 contact to execute it every scan.
- To specify motion to an absolute position, you must first use either an AXISx_RSEEK or an AXISx_LDPOS subroutine to establish the zero position.

- To move to a specific location, based upon inputs from your program, use the AXISx_GOTO subroutine.
- To run the motion profiles you configured with the Motion wizard, use the AXISx_RUN subroutine.

12.6.2 AXISx_CTRL subroutine

Table 12- 6 AXISx_CTRL

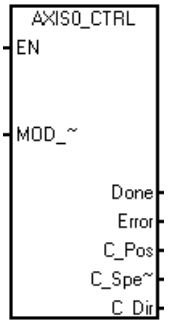
LAD / FBD	STL	Description
	**CALL AXISx_CTRL, MOD_EN, Done, Error, C_Pos, C_Speed, C_Dir**	The AXISx_CTRL subroutine (Control) enables and initializes the Axis of Motion by automatically commanding the Axis of Motion to load the configuration/profile table each time the CPU changes to RUN mode. Use this subroutine only once in your project per motion axis, and ensure that your program calls this subroutine every scan. Use SM0.0 (Always On) as the input for the EN parameter.

Table 12- 7 Parameters for the AXISx_CTRL subroutine

Inputs/Outputs	Data type	Operands
MOD_EN	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done, C_Dir	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

The MOD_EN parameter must be on to enable the other motion subroutines to send commands to the Axis of Motion. If the MOD_EN parameter turns off, then the Axis of Motion aborts any command that is in progress and performs a decelerated stop.

The output parameters of the AXISx_CTRL subroutine provide the current status of the Axis of Motion.

The Done parameter turns on when the Axis of Motion completes any subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

The C_Pos parameter is the current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter provides the current speed of the Axis of Motion. If you configured the measurement system for the Axis of Motion for pulses, C_Speed is a DINT value containing the number of pulses/second. If you configured the measurement system for

engineering units, C_Speed is a REAL value containing the selected engineering units/second (REAL).

The C_Dir parameter indicates the current direction of the motor:

- Signal state of 0 = positive
- Signal state of 1 = negative

Note

The Axis of Motion reads the configuration/profile table only at power-up or when commanded to load the configuration.

- If you use the Motion wizard to modify the configuration, then the AXISx_CTRL subroutine automatically commands the Axis of Motion to load the configuration/profile table every time the CPU changes to RUN mode.
- If you use the Motion control panel to modify the configuration, clicking the Update Configuration button commands the Axis of Motion to load the new configuration/profile table.
- If you use another method to modify the configuration, then you must also issue an AXISx_CFG command to the Axis of Motion to load the configuration/profile table. Otherwise, the Axis of Motion continues to use the old configuration/profile table.

12.6.3 AXISx_MAN subroutine

Table 12- 8 AXISx_MAN

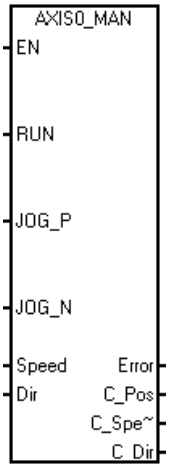
LAD / FBD	STL	Description
	`CALL AXISx_MAN, RUN, JOG_P, JOG_N, Speed, Dir, Error, C_Pos, C_Speed, C_Dir`	The AXISx_MAN subroutine (Manual Mode) puts the Axis of Motion into manual mode. This allows the motor to be run at different speeds or to be jogged in a positive or negative direction. You can enable only one of the RUN, JOG_P, or JOG_N inputs at a time.

Table 12- 9 Parameters for the AXISx_MAN subroutine

Inputs/Outputs	Data type	Operands
RUN, JOG_P, JOG_N	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant
Dir, C_Dir	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Enable the RUN (Run/Stop) parameter to command the Axis of Motion to accelerate to the specified speed (Speed parameter) and direction (Dir parameter). You can change the value of the Speed parameter while the motor is running, but the Dir parameter must remain constant. Disabling the RUN parameter commands the Axis of Motion to decelerate until the motor comes to a stop.

Enable the JOG_P (Jog Positive Rotation) or the JOG_N (Jog Negative Rotation) parameter to command the Axis of Motion to jog in either a positive or negative direction. If the JOG_P or JOG_N parameter remains enabled for less than 0.5 seconds, the Axis of Motion issues pulses to travel the distance specified in JOG_INCREMENT. If the JOG_P or JOG_N parameter remains enabled for 0.5 seconds or longer, the Axis of Motion begins to accelerate to the specified JOG_SPEED.

The Speed parameter determines the speed when RUN is enabled. If you configured the measuring system of the Axis of Motion for pulses, the speed is a DINT value for pulses/second. If you configured the measuring system of the Axis of Motion for engineering units, the speed is a REAL value for units/second. You can change this parameter while the motor is running.

Note

The Axis of Motion may not react to small changes in the Speed parameter, especially if the configured acceleration or deceleration time is short and the difference between the configured maximum speed and start/stop speed is large.

The Dir parameter determines the direction to move when RUN is enabled. You cannot change this value when the RUN parameter is enabled.

The Error parameter (Page 732) contains the result of this subroutine.

The C_Pos parameter contains the current position of the Axis of Motion. Based upon the units of measurement selected, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter contains the current speed of the Axis of Motion. Based upon the units of measurement selected, the value is either the number of pulses/second (DINT) or the engineering units/second (REAL).

The C_Dir parameter indicates the current direction of the motor:

- Signal state of 0 = positive
- Signal state of 1 = negative

12.6.4 AXISx_GOTO subroutine

Table 12- 10 AXISx_GOTO

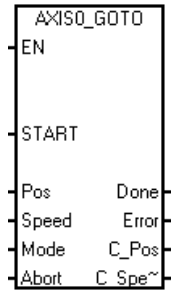
LAD / FBD	STL	Description
	**CALL AXISx_GOTO, START, Pos, Speed, Mode, Abort, Done, Error, C_Pos, C_Speed**	The AXISx_GOTO subroutine commands the Axis of Motion to go to a desired location.

Table 12- 11 Parameters for the AXISx_GOTO subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Pos, Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant
Mode	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Abort, Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the DONE bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a GOTO command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a GOTO command to the Axis of Motion. To ensure that only one GOTO command is sent, use an edge detection element to pulse the START parameter on.

The Pos parameter contains a value that signifies either the location to move (for an absolute move) or the distance to move (for a relative move). Based upon the units of measurement selected, the value is either a number of pulses (DINT) or the engineering units (REAL).

The Speed parameter determines the maximum speed for this movement. Based upon the units of measurement, the value is either a number of pulses/second (DINT) or the engineering units/second (REAL).

The Mode parameter selects the type of move:

- 0: Absolute position
- 1: Relative position
- 2: Single-speed, continuous positive rotation
- 3: Single-speed, continuous negative rotation

The Done parameter turns on when the Axis of Motion completes this subroutine.

Turn on the Abort parameter to command the Axis of Motion to stop execution of this command and decelerate until the motor comes to a stop.

The Error parameter (Page 732) contains the result of this subroutine.

The C_Pos parameter contains current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter contains the current speed of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses/second (DINT) or the engineering units/second (REAL).

12.6.5 AXISx_RUN subroutine

Table 12- 12 AXISx_RUN

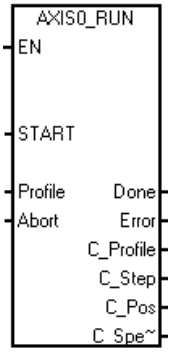
LAD / FBD	STL	Description
	`CALL AXISx_RUN, START, Profile, Abort, Done, Error, C_Profile, C_Step, C_Pos, C_Speed`	The AXISx_RUN subroutine (Run Profile) commands the Axis of Motion to execute the motion operation in a specific profile stored in the configuration/profile table.

Table 12- 13 Parameters for the AXISx_RUN subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Profile	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Abort, Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error, C_Profile, C_Step	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a RUN command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a RUN command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Profile parameter contains the number or the symbolic name for the motion profile. The "Profile" input must be between 0 - 31. If not, the subroutine will return an error.

Turn on the Abort parameter to command the Axis of Motion to stop the current profile and decelerate until the motor comes to a stop.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

The C_Profile parameter contains the profile currently being executed by the Axis of Motion.

The C_Step parameter contains the step of the profile currently being executed.

The C_Pos parameter contains the current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter contains the current speed of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses/second (DINT) or the engineering units/second (REAL).

12.6.6 AXISx_RSEEK subroutine

Table 12- 14 AXISx_RSEEK

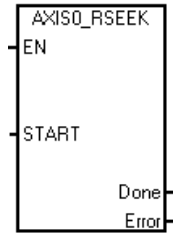
LAD / FBD	STL	Description
	`CALL AXISx_RSEEK, START, Done, Error`	The AXISx_RSEEK subroutine (Seek Reference Point Position) initiates a reference point seek operation, using the search method in the configuration/profile table. When the Axis of Motion locates the reference point and motion has stopped, the Axis of Motion loads the RP_OFFSET parameter value into the current position.

Table 12- 15 Parameters for the AXISx_RSEEK subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

The default value for RP_OFFSET is 0. You can use the Motion wizard, the Motion Control Panel, or the AXISx_LDOFF (Load Offset) subroutine to change the RP_OFFSET value.

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a RSEEK command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the

subroutine sends a RSEEK command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

12.6.7 AXISx_LDOFF subroutine

Table 12- 16 AXISx_LDOFF

LAD / FBD	STL	Description
	CALL AXISx_LDOFF, START, Done, Error	The AXISx_LDOFF subroutine (Load Reference Point Offset) establishes a new zero position that is at a different location from the reference point position. Before executing this subroutine, you must first determine the position of the reference point. You must also move the machine to the starting position. When the subroutine sends the LDOFF command, the Axis of Motion computes the offset between the starting position (the current position) and the reference point position. The Axis of Motion then stores the computed offset to the RP_OFFSET parameter and sets the current position to 0. This establishes the starting position as the zero position. In the event that the motor loses track of its position (such as on loss of power or if the motor is repositioned manually), you can use the AXISx_RSEEK subroutine to re-establish the zero position automatically.

Table 12- 17 Parameters for the AXISx_LDOFF subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a LDOFF command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a LDOFF command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

12.6.8 AXISx_LDPOS subroutine

Table 12- 18 AXISx_LDPOS

LAD / FBD	STL	Description
	CALL AXISx_LDPOS, START, New_Pos, Done, Error, C_Pos	The AXISx_LDPOS subroutine (Load Position) changes the current position value in the Axis of Motion to a new value. You can also use this subroutine to establish a new zero position for any absolute move command.

Table 12- 19 Parameters for the AXISx_LDPOS subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
New_Pos, C_Pos	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a LDPOS command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a LDPOS command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The New_Pos parameter provides the new value to replace the current position value that the Axis of Motion reports and uses for absolute moves. Based upon the units of measurement, the value is either a number of pulses (DINT) or the engineering units (REAL).

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

The C_Pos parameter contains the current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

12.6.9 AXISx_SRATE subroutine

Table 12- 20 AXISx_SRATE

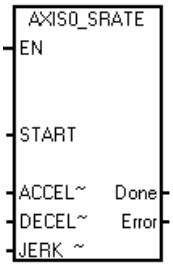
LAD / FBD	STL	Description
	**CALL AXISx_SRATE, START, ACCEL_Time, DECEL_Time, JERK_Time, Done, Error**	The AXISx_SRATE subroutine (Set Rate) commands the Axis of Motion to change the acceleration, deceleration, and jerk times.

Table 12- 21 Parameters for the AXISx_SRATE subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L
ACCEL_Time, DECEL_Time, JERK_Time	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to copy the new time values to the configuration/profile table and sends a SRATE command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends an SRATE command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The ACCEL_Time, DECEL_Time, and JERK_Time parameters determine the new acceleration time, deceleration time, and jerk time in milliseconds (ms).

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

12.6.10 AXISx_DIS subroutine

Table 12- 22 AXISx_DIS

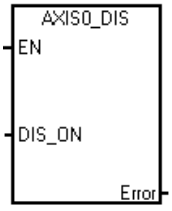
LAD / FBD	STL	Description
	**CALL AXISx_DIS, DIS_ON, Error**	The AXISx_DIS subroutine turns the DIS output of the Axis of Motion on or off. This allows you to use the DIS output for disabling or enabling a motor controller. If you use the DIS output on the Axis of Motion, then this subroutine can be called every scan or only when you need to change the value of the DIS output.

Table 12- 23 Parameters for the AXISx_DIS subroutine

Inputs/Outputs	Data type	Operands
DIS_ON	BOOL	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

When the EN bit turns on to enable the subroutine, the DIS_ON parameter controls the DIS output of the Axis of Motion.

Note

If you have not defined a "DIS" output in the Motion wizard, the AXISx_DIS subroutine will return an error.

The Error parameter (Page 732) contains the result of this subroutine.

12.6.11 AXISx_CFG subroutine

Table 12- 24 AXISx_CFG

LAD / FBD	STL	Description
	CALL AXISx_CFG, START, Done, Error	The AXISx_CFG subroutine (Reload Configuration) commands the Axis of Motion to read the configuration block from the location specified by the configuration/profile table pointer. The Axis of Motion then compares the new configuration with the existing configuration and performs any required setup changes or recalculations.

Table 12- 25 Parameters for the AXISx_CFG subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a CFG command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a CFG command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

12.6.12 AXISx_CACHE subroutine

Table 12- 26 AXISx_CACHE

LAD / FBD	STL	Description
	`CALL AXISx_CACHE, START, Profile, Done, Error`	The AXISx_CACHE subroutine (Cache Profile) commands a caching of a motion profile before the profile is executed. This allows you to pre-cache needed commands before execution. Pre-caching reduces the amount of time and provides consistency between executing a motion instruction and starting the move.

Table 12- 27 Parameters for the AXISx_CACHE subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Profile	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Abort, Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error, C_Profile, C_Step	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a CACHE command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the

subroutine sends a CACHE command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Profile parameter contains the number or the symbolic name for the motion profile. The "Profile" input must be between 0 - 31. If not, the subroutine will return an error.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

12.6.13 AXISx_RDPOS subroutine

Table 12- 28 AXISx_RDPOS

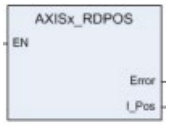
LAD / FBD	STL	Description
	`CALL AXISx_RDPOS, Error, I_Pos`	The AXISx_RDPOS subroutine returns the current motion axis position.

Table 12- 29 Parameters for the AXISx_RDPOS subroutine

Inputs/Outputs	Data type	Operands
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
I_Pos	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine.

The Error parameter (Page 732) contains the result of this subroutine.

The I_Pos parameter contains the current motion axis position.

Note

Execution of this command returns the actual current position of the axis. Position status values provided in other motion subroutines, such as AXISx_CTRL, AXISx_GOTO, and so forth, are updated periodically. Therefore, the position values reported by those commands and the position value reported by this command may differ somewhat as a result and is normal.

12.6.14 AXISx_ABSPOS subroutine

Table 12- 30 AXISx_ABSPOS

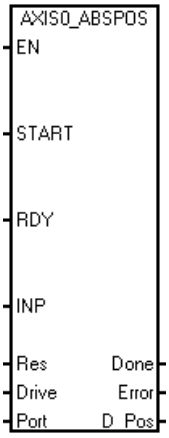
LAD / FBD	STL	Description
	**CALL AXISx_ABSPOS, START, RDY, INP, Res, Drive, Port, Done, Error, D_Pos**	The AXISx_ABSPOS subroutine reads the absolute position from certain Siemens servo drives, such as the V90. You read the absolute position value in order to update the current position value in the Axis of Motion. This capability is supported when using a SINAMICS V90 servo drive combined with a SIMOTICS-1FL6 servo motor that has an absolute encoder installed.

Table 12- 31 Parameters for the AXISx_ABSPOS subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
RDY, INP	BOOL	I, Q, V, M, SM, S, T, C, L
Res	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant
Drive	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Port	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
D_Pos	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the DONE bit signals that the execution of the subroutine has completed.

Turn on the START parameter to obtain the current absolute position from the specified drive. To ensure that only one operation to read the current position is performed, use an edge detection element to pulse the START parameter on.

The RDY parameter indicates the readiness state of the servo drive, which is typically provided by a digital output signal from the drive. This subroutine will read the absolute position from the drive only if this parameter is on.

The INP parameter indicates the standstill state of the motor, which is typically provided by a digital output signal from the drive. This subroutine will read the absolute position from the drive only if this parameter is on.

The Res parameter must be set to the resolution of the absolute encoder connected to your servo motor. For example, the single turn resolution of a SIMOTICS S-1FL6 servo motor with absolute encoder is 20 bits or 1048576.

Set the Drive parameter to match the RS485 address of the servo drive to be accessed by this subroutine. Valid addresses of individual drives are 0 to 31.

Set the Port parameter to designate the CPU port to be used to communicate with the servo drive:

- 0: Onboard RS485 port (Port 0)
- 1: RS485/RS232 signal board (if present, Port 1)

The Done parameter turns on when the subroutine's work is complete.

The Error parameter (Page 732) contains the result of this subroutine.

The D_Pos parameter contains the current absolute position returned by the servo drive.

Note

To use this subroutine, you must configure the measurement system setting for this Axis of Motion to Engineering Units.

Note

Additional subroutines

The Motion wizard also creates the subroutines ABSPOS_SBR and ABSPOS_INT when you enable read position in the wizard configuration to read an absolute position from a drive.

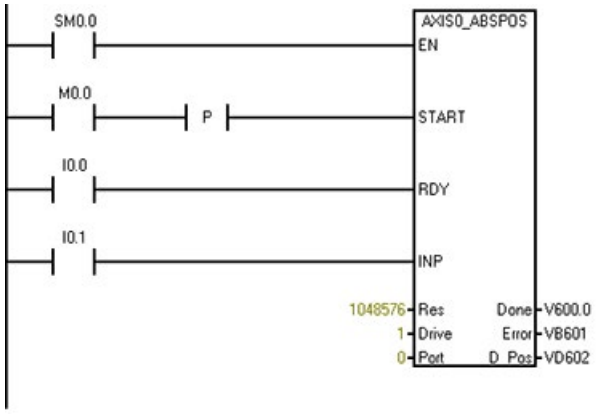
12.7 Using the AXISx_ABSPOS subroutine to read the absolute position from a SINAMICS servo drive

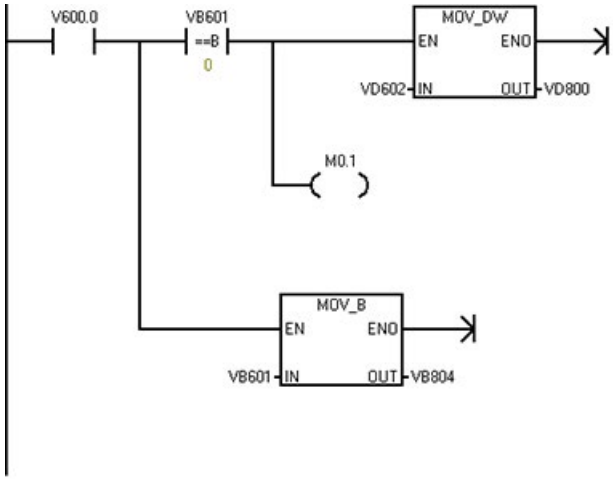
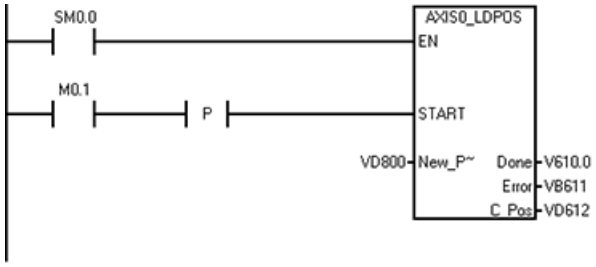
The following sections provide information about how to use the AXISx_ABSPOS subroutine in your project to read the absolute position from a SINAMICS V90 servo drive.

12.7.1 AXISx_ABSPOS and AXISx_LDPOS subroutines usage examples

The absolute position is valid only after successful completion of the AXISx_ABSPOS subroutine (Done parameter = ON and Error parameter = "no error") when executed with the START parameter on. Since the Error and D_Pos parameters revert to default values when the subroutine is executed with the START input off, you must include instructions in your program to capture the valid absolute position value after completion of the subroutine.

Table 12- 32 Example: Using the AXISx_ABSPOS subroutine to read the absolute position from a SINAMICS V90 servo drive

LAD/FBD	Description	STL
Network 1: 	Read the servo position from the drive.	LD SM0.0 = L60.0 LD M0.0 EU = L63.7 LD I0.0 = L63.6 LD I0.1 = L63.5 LD L60.0 CALL AXIS0_ABSPOS, L63.7, L63.6, L63.5, 1048576,1, 0, V600.0, VB601, VD602

Network 2: 	When the operation is done, capture the error code and also capture the servo position, if no error.	**LD V600.0** **LPS** **AB= VB601, 0** **MOVD VD602, VD800** **= M0.1** **LPP** **MOVB VB601, VB804**
Network 3: 	Update the current position in this Axis of Motion with the captured servo position value.	**LD SM0.0** **= L60.0** **LD M0.1** **EU** **= L63.7** **LD L60.0** **CALL AXIS0_LDPOS, L63.7, VD800, V610.0, VB611, VD612**

12.7.2 Interconnections

Digital I/O

Refer to the section "Connection examples with PLCs" in the *SINAMICS V90 / SIMOTICS S-1FL6 Operating Instructions* document to find wiring diagrams for connection of the suggested digital control signals between an S7-200 SMART CPU and a V90 servo drive.

Communications

The AXISx_ABSPOS subroutine obtains the position data from the drive using serial communications on the RS485 link between the two devices. Therefore, connect a cable between the RS485 port on the S7-200 SMART CPU (or optionally the S7-200 SMART CM01 signal board, if your CPU model supports it) and the RS485 port on the V90 servo drive.

Refer to the appropriate sections of the *S7-200 SMART System Manual* and the *SINAMICS V90 / SIMOTICS S-1FL6 Operating Instructions* documents for descriptions of the RS485 ports on the S7-200 SMART CPU and V90 servo drive.

12.7.3 Commissioning

12.7.3.1 Control mode

"PTI" mode is the drive control mode setting that allows movement speed and distance to be controlled from an external pulse train. The default control mode in the V90 servo drive is basic "PTI" mode, but you can check the mode setting by reading the value of parameter "p29003" and verifying that the value = "0". It is possible to use compound control modes (PTI/S and PTI/T) with the pulse train output from the S7-200 SMART CPU. These are advanced features and are not within the scope of this document. For assistance with these features, refer to the *SINAMICS V90 / SIMOTICS S-1FL6 Operating Instructions* document.

12.7.3.2 Setpoint pulse input channel

For correct operation with the digital outputs of the S7-200 SMART CPU, you must select the "24 V DC single end pulse train input" setting for the setpoint pulse input channel parameter (parameter "p29014" = 1) in the V90 servo drive.

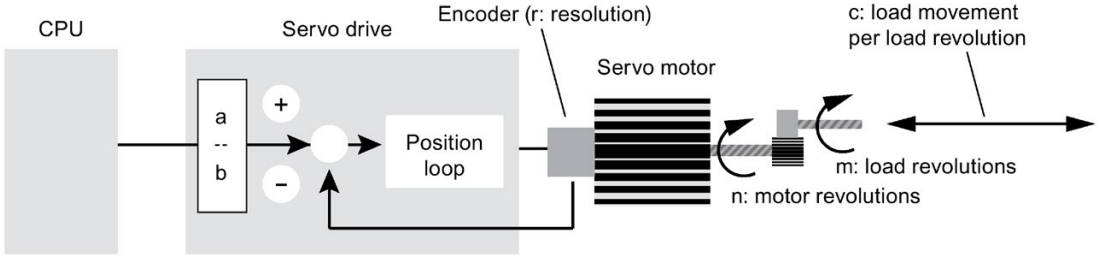
12.7.3.3 Setpoint pulse train input format

Ensure that the CPU's Axis of Motion output phasing and polarity settings (established in the "Directional Control" dialog of the STEP 7-Micro/WIN SMART Motion wizard) are consistent with the V90 servo drive's setpoint pulse train input format setting (parameter "p29010").

12.7.3.4 Common engineering units basis

When using a motion axis on the S7-200 SMART CPU to control the movement speed and distance of a servo motor, you must establish a common definition of the engineering units between the Axis of Motion (CPU) and the drive.

The following diagram shows the elements of a motion system:



To establish a common engineering unit definition between the CPU and the servo drive, you must consider the following motion system variables when commissioning your system:

Electronic gearing

In the V90 servo drive, the "a" and "b" values determine the drive's electronic gear ratio, a feature that allows a frequency conversion on the pulse train issued from the CPU. Since the maximum pulse frequency issued from an Axis of Motion in the S7-200 SMART CPU is 100 kHz, while the encoder resolution of SIMOTICS S-1FL6 servo motors installed with absolute encoders is 2^20 pulses per revolution, use of the drive's electronic gear feature is likely, in many applications, to achieve higher motor speeds. For example, to achieve a 10x increase in the setpoint pulse frequency within the servo drive compared to the frequency of the CPU pulse train supplied to the drive, then you must set the electronic gear ratio to "10:1".

In the V90 servo drive, setting parameter "p29012[0]" establishes the numerator of the electronic gearing ratio ("a"), while setting parameter "p29013" establishes the denominator of the ratio ("b"). Also, when using electronic gearing, set the parameter "p29011" value to "0". The valid range for the electronic gear ratio (a / b) in the V90 servo drive is between "0.02" and "200".

Refer to the "Electronic Gear Ratio" section of the *SINAMICS V90 / SIMOTICS S-1FL6 Operating Instructions* document for more information.

Mechanical factors

The "m" and "n" values establish the mechanical relationship between a load revolution and a motor revolution, applicable when a gearing mechanism is used. When the V90 servo drive is in "PTI" control mode, its internal mechanical gearing ratio parameters are fixed at "1:1", but the physical "m" and "n" values are important in establishing the correct engineering unit conversion factors for the Axis of Motion, as shown below.

The "c' value establishes the relationship between load movement in the specified engineering unit, and load revolutions. "20 cm of load movement per load revolution" and "360 degrees of load movement per load revolution" are examples of this conversion factor.

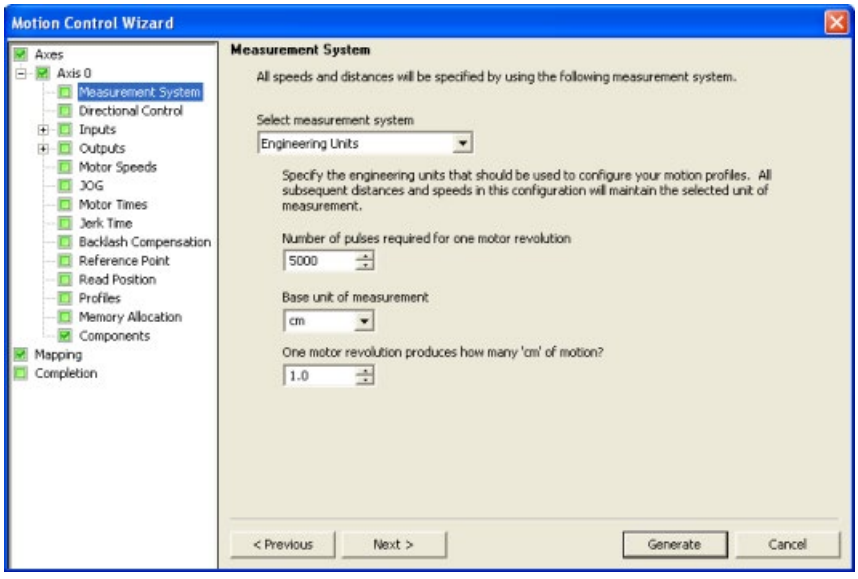
Encoder resolution

The "r" value is the resolution of the absolute encoder in your servo motor. As stated above, the encoder resolution of SIMOTICS S-1FL6 servo motors installed with absolute encoders is 2^20 pulses per revolution or "1048576". When the V90 servo drive is paired with a motor containing an absolute encoder, the drive automatically detects the encoder type and obtains its resolution. However, in your program, you must specify this resolution value in the AXISx_ABSPOS subroutine's "Res" input parameter and also in one of the engineering unit conversion factor calculations shown below.

Measurement system settings in the Motion wizard

When using the STEP 7-Micro/WIN SMART Motion wizard to configure the measurement system for a CPU Axis of Motion, you must assign the three conversion settings:

- First setting: Relates CPU pulses to motor revolutions
- Second setting: Establishes the base engineering unit name
- Third setting: Relates motor revolutions to load movement



Setting 1: "Number of pulses required for one motor revolution"

This setting defines the relationship between CPU pulses and motor revolutions. The relevant equation that yields the correct value for this setting follows:

(1) Number of pulses required for one motor revolution = r * (b / a)

where, "r" = encoder resolution, expressed as encoder pulses per motor revolution, "a" and "b" = electronic gearing (E-gear) ratio parameters ("a" = value of V90 parameter "p29012[0]" and "b" = value of V90 parameter "p29013")

For example, if the desired E-gear ratio is "128:1" and the motor's absolute encoder resolution is 2^20 or "1048576", then:

"Number of pulses required for one motor revolution" = **1048576 * (1 / 128) = 8192**

Setting 2: "Base unit of measurement"

This setting establishes the base engineering unit name for speed and distance settings throughout the Motion wizard. To avoid confusion, the selection should match the engineering unit relevant at the load. For example, if load movement and speed is to be expressed in "cm" and "cm / second", then the "cm" selection should be chosen for this setting.

Setting 3: "One motor revolution produces how many "xxx" of motion?"

This setting defines the relationship between motor revolutions and load movement in the defined engineering unit (for example, cm and degrees). The relevant equation that yields the correct value for this setting is as follows:

(2) One motor revolution produces how many "xxx" of motion = c * (m / n)

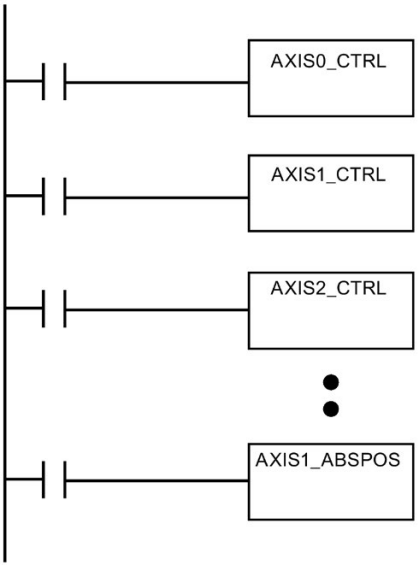
where, "c" = load movement (in the defined engineering unit) per load revolution, "m/n" = external gearing ratio expressed as load revolutions per motor revolution

For example, if the mechanical gear ratio is "1:2" and the load movement per load revolution is 10 cm, then:

"One motor revolution produces how many cm of motion" = **10 * (1 / 2) = 5**

12.7.4 Important facts to know

- Do not call the AXISx_ABSPOS subroutine from within an interrupt routine or from a subroutine called within an interrupt routine.
- If you have configured multiple Axes of Motion in your CPU project, ensure that the AXISx_CTRL subroutines for all axes are executed prior to executing the first AXISx_ABSPOS subroutine for any axis. The AXISx_CTRL subroutine contains code to initialize the V memory area used commonly by all instances of the AXISx_ABSPOS subroutine in your program to manage the communications with the servo drive.



- If you configure your motion axis measurement system to the "relative pulses" setting instead of the "engineering units" setting, you can still use the AXISx_ABSPOS subroutine to return position information from the V90 servo drive. Note, however, that the position value returned in the "D_pos" parameter of the subroutine will then be of type DINT and is the actual position value reported by the servo drive (there are no engineering unit conversions performed on the value).

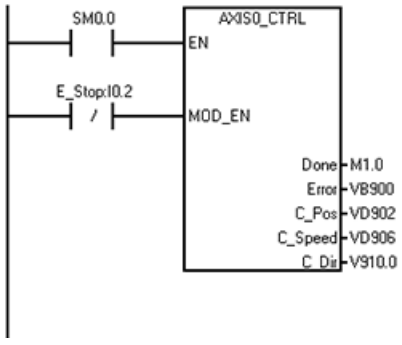
12.8 Axis of Motion example programs

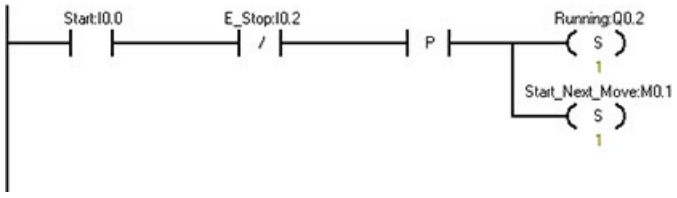
12.8.1 Axis of Motion simple relative move (cut-to-length application) example

The example program shows a simple relative move that uses the AXISx_CTRL and AXISx_GOTO subroutines to perform a cut-to-length operation. This program does not require an RP seek mode or a motion profile, and the length can be measured in either pulses or engineering units. Enter the length (VD500) and target speed (VD504). When I0.0 (Start) turns on, the machine starts. When I0.1 (Stop) turns on, the machine finishes the current operation and stops. When I0.2 (E_Stop) turns on, the machine aborts any motion and immediately stops.

Table 12- 33 Example: Axis of Motion simple relative move (cut-to-length application)

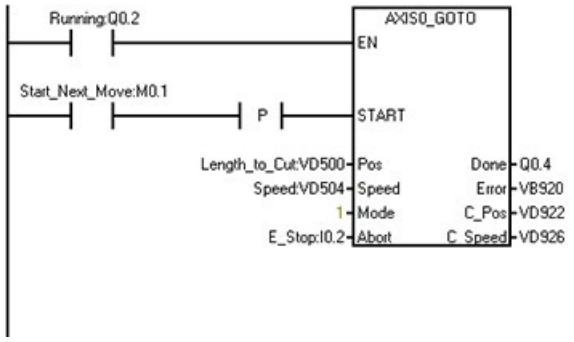
LAD/FBD	Description	STL

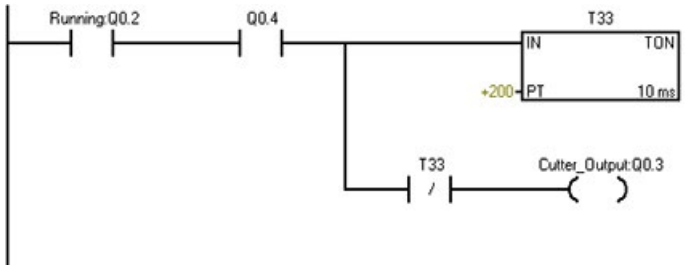
Network 1:
 
	Control instruction	`LD SM0.0`
		`= L60.0`
		`LDN I0.2`
		`= L63.7`
		`LD L60.0`
		`CALL AXIS0_CTRL,`
		`L63.7, M1.0,`
		`VB900, VD902,`
		`VD906, V910.0`

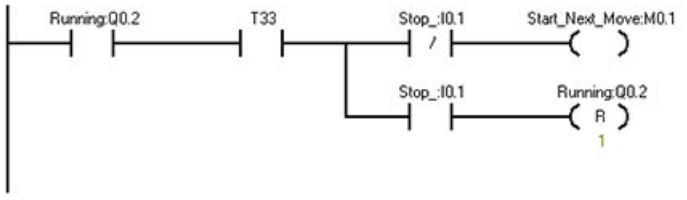
Network 2:
 
	Start puts machine into automatic mode.	`LD I0.0`
		`AN I0.2`
		`EU`
		`S Q0.2, 1`
		`S M0.1, 1`

Network 3: 	E_Stop: Stops immediately and turns off automatic mode.	`LD I0.2` `R Q0.2, 1`

LAD/FBD	Description	STL

Network 4:
 
	1. Move to a certain point:
	2. Enter the length to cut.
	3. Enter the target speed into Speed.
	4. Set the mode to 1 (Relative mode).
		`LD Q0.2`
		`= L60.0`
		`LD M0.1`
		`EU`
		`= L63.7`
		`LD L60.0`
		`CALL AXIS0_GOTO, L63.7, VD500, VD504, 1, I0.2, Q0.4, VB920, VD922, VD926`

Network 5:
 
	When in position, turn on the cutter for 2 seconds to finish the cut.	`LD Q0.2`
		`A Q0.4`
		`TON T33, +200`
		`AN T33`
		`= Q0.3`

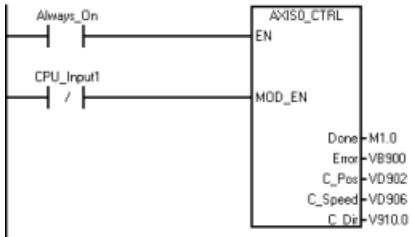
Network 6:
 
	When the cut is finished, then restart unless the Stop is active.	`LD Q0.2`
		`A T33`
		`LPS`
		`AN I0.1`
		`= M0.1`
		`LPP`
		`A I0.1`
		`R Q0.2, 1`

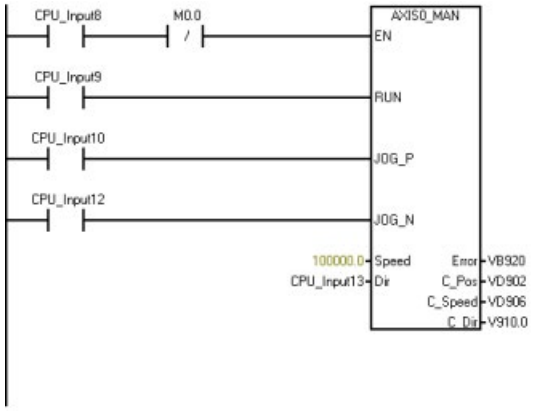
12.8.2 Axis of Motion AXISx_CTRL, AXISx_RUN, AXISx_SEEK, and AXISx_MAN example

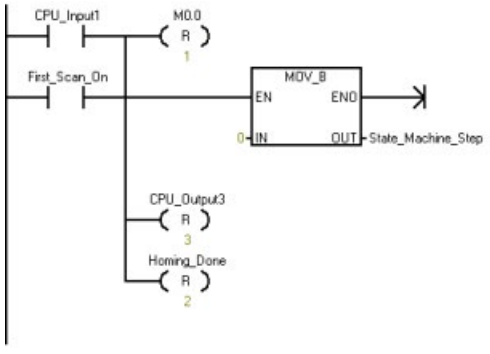
This program provides an example of the AXISx_CTRL, AXISx_RUN, AXISx_RSEEK, and AXISx_MAN subroutines. You must configure the RP seek mode and a motion profile.

Table 12- 34 Example: Axis of Motion AXISx_CTRL, AXISx_RUN, AXISx_SEEK, and AXISx_MAN subroutines application

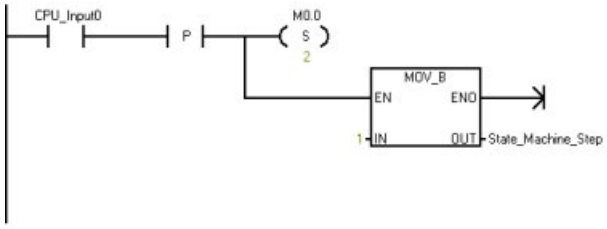
LAD/FBD	Description	STL

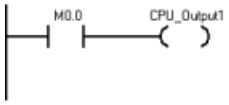
Network 1
 
	Enable the axis by turning CPU_Input1 off.
	
	Symbol and Address: [1]
	• Always_On = SM0.0
	• AXIS0_CTRL = SBR1
	• CPU_Input1 = I0.1
		`LD Always_On`
		`= L60.0`
		`LDN CPU_Input1`
		`= L63.7`
		`LD L60.0`
		`CALL AXIS0_CTRL,`
		`L63.7, M1.0,`
		`VB900, VD902,`
		`VD906, V910.0`

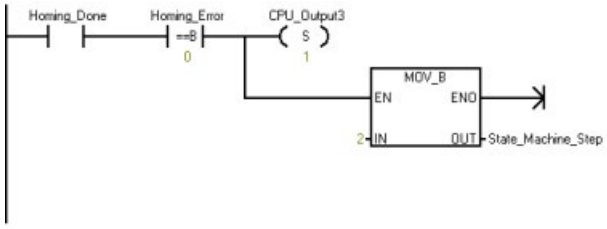
Network 2
 
	Move the axis to a known position using the Jog command. You can now move the axis manually.
	
	Symbol and Address: [1]
	• AXIS0_MAN = SBR2
	• CPU_Input10 = I1.2
	• CPU_Input12 = I1.4
	• CPU_Input13 = I1.5
	• CPU_Input8 = I1.0
	• CPU_Input9 = I1.1
		`LD CPU_Input8`
		`AN M0.0`
		`= L60.0`
		`LD CPU_Input9`
		`= L63.7`
		`LD CPU_Input10`
		`= L63.6`
		`LD CPU_Input12`
		`= L63.5`
		`LD L60.0`
		`CALL AXIS0_MAN,`
		`L63.7, L63.6,`
		`L63.5, 100000.0,`
		`CPU_Input13,`
		`VB920, VD902,`
		`VD906, V910.0`

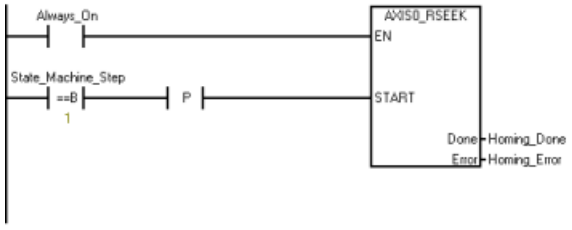
Network 3
 
	Reset the process. Set the initial step to "0".
	
	Symbol and Address: [1]
	• CPU_Input1 = I0.1
	• CPU_Output3 = Q0.3
	• First_Scan_On = SM0.1
	• Homing_Done = M1.1
	• State_Machine_Step = VB1500
		`LD CPU_Input1`
		`O First_Scan_On`
		`R M0.0, 1`
		`MOVB 0,`
		`State_Machine_`
		`Step`
		`R CPU_Output3, 3`
		`R Homing_Done, 2`

LAD/FBD	Description	STL

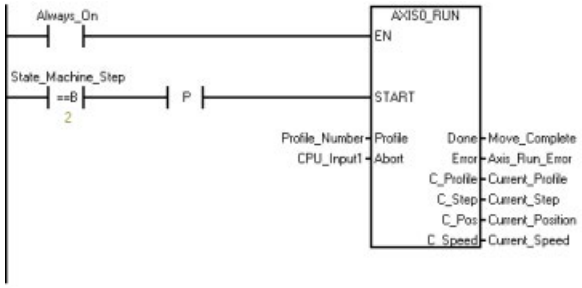
Network 4
 
	Start the process. When CPU_Input0 is toggled from off to on, the State_Machine_Step is set to "1".
	
	Symbol and Address: [1]
	• CPU_Input0 = I0.0
	• State_Machine_Step = VB1500
		`LD CPU_Input0`
		`EU`
		`S M0.0, 2`
		`MOVB 1, State_Machine_Step`

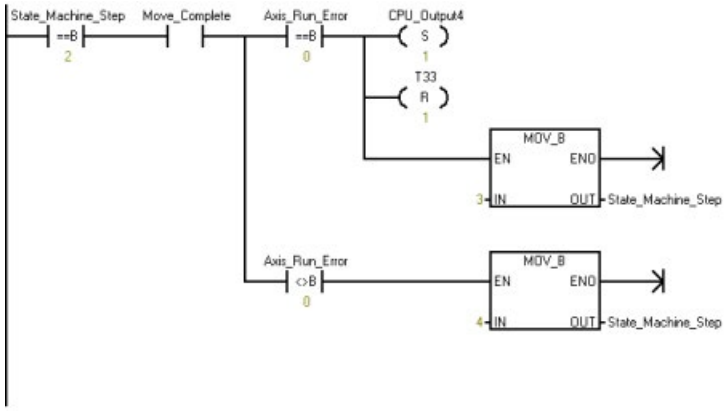
Network 5 	This network turns CPU_Input1 on. Symbol and Address: [1] • CPU_Input1 = I0.1	`LD M0.0` `= CPU_Input1`

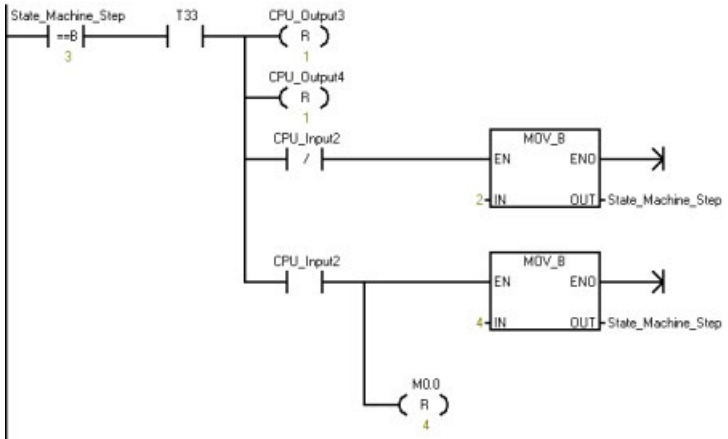
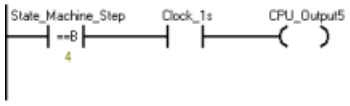
Network 6
 
	When the homing is complete, move to the next step.
	
	Symbol and Address: [1]
	• CPU_Output3 = Q0.3
	• Homing_Done = M1.1
	• Homing_Error = VB930
	• State_Machine_Step = VB1500
		`LD Homing_Done`
		`AB= Homing_Error`
		`S CPU_Output3`
		`MOVB 2, State_Machine_Step`

Network 7
 
	When the state machine is in Step 1, the system automatically homes the axis. If there is a Homing error, the Homing_Error output displays the error code.
	
	Symbol and Address: [1]
	• Always_On = SM0.0
	• AXIS0_RSEEK = SBR5
	• Homing_Done = M1.1
	• Homing_Error = VB930
	• State_Machine_Step = VB1500
		`LD Always_On`
		`= L60.0`
		`LDB State_Machine_Step, 1`
		`EU`
		`= L63.7`
		`LD L60.0`
		`CALL AXIS0_RSEEK, L63.7, Homing_Done, Homing_Error`

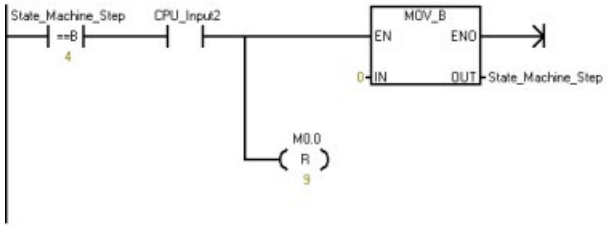
LAD/FBD	Description	STL

Network 8
 
	When the State Machine enters Step 2, it executes a move of the selected profile.
	
	Symbol and Address: [1]
	• Always_On = SM0.0
	• AXIS0_RUN = SBR4
	• Axis_Run_Error = VB940
	• CPU_Input1 = I0.1
	• Current_Position = VD914
	• Current_Profile = VB941
	• Current_Speed = VD948
	• Current_Step = VB942
	• Move_Complete = M1.2
	• Profile_Number = VB228
	• State_Machine_Step = VB1500
		`LD Always_On`
		`= L60.0`
		`LDB= State_Machine_ Step, 2`
		`EU`
		`= L63.7`
		`LD L60.0`
		`CALL AXIS0_RUN, L63.7, Profile_Number, CPU_Input1, Move_Complete, Axis_Run_Error, Current_Profile, Current_Step, Cur- rent_Position, Current_Speed`

Network 9
 
	When the State Machine is in Step 2 and completes the move, you evaluate the error status. If there is no error, the State Machine transitions to Step 3. If there is an error, the State Machine transitions to Step 4 for error handling.
	
	Symbol and Address: [1]
	• Axis_Run_Error = VB940
	• CPU_Output4 = Q0.4
	• Move_Complete = M1.2
	• State_Machine_Step = VB1500
		`LDB= State_Machine_ Step, 2`
		`A Move_Complete`
		`LPS`
		`AB= Axis_Run_Error, 0`
		`S CPU_Output4, 1`
		`R T33, 1`
		`MOVB 3, State_Machine_ Step`
		`LPP`
		`AB<> Axis_Run_Error, 0`
		`MOVB 4, State_Machine_ Step`

LAD/FBD	Description	STL
Network 10 	Wait for Step 3. Symbol and Address: [1] • State_Machine_Step = VB1500	**LDB= State_Machine_ Step, 3 TON T33, 200**
Network 11 	If the State Machine moves to Step 3, wait for 2 s. Then, evaluate the status of the switches, and restart the move or stop. Symbol and Address: [1] • CPU_Input2 = I0.2 • CPU_Output3 = Q0.3 • CPU_Output4 = Q0.4 • State_Machine_Step = VB1500	**LDB= State_Machine_ Step, 3 A T33 LPS R CPU_Output3, 1 R CPU_Output4, 1 AN CPU_Input2 MOVB 2, State_Machine_ Step LPP A CPU_Input2 MOVB 4, State_Machine_ Step R M0.0, 4**
Network 12 	If the State Machine moves to Step 4, clear the outputs. Symbol and Address: [1] • CPU_Output3 = Q0.3 • State_Machine_Step = VB1500	**LDB= State_Machine_ Step, 4 R CPU_Output3, 2**
Network 13 	If the State Machine is in Step 4, flash output 5 to indicate an error. Symbol and Address: [1] • Clock_1s = SM0.5 (Clock pulse that is ON for 0.5 s and OFF for 0.5 s for a duty cycle time of 1 s.) • CPU_Output5 = Q0.5 • State_Machine_Step = VB1500	**LDB= State_Machine_ Step, 4 A Clock_1s = CPU_Output5**

LAD/FBD	Description	STL

Network 14
 
	If the State Machine is in Step 4, you must acknowledge the error by toggling Input I0.2. This action resets the state to Step 0.
	
	Symbol and Address: [1]
	• CPU_Input2 = I0.2
	• State_Machine_Step = VB1500
		`LDB= State_Machine_ Step, 4`
		`A CPU_Input2`
		`MOVB 0, State_Machine_ Step`
		`R M0.0, 9`

[1] The program addresses shown are example addresses. Your program addresses could vary.

12.9 Monitoring the Axis of Motion

To aid you in the development of your motion control solution, STEP 7-Micro/WIN SMART provides the Motion control panel.

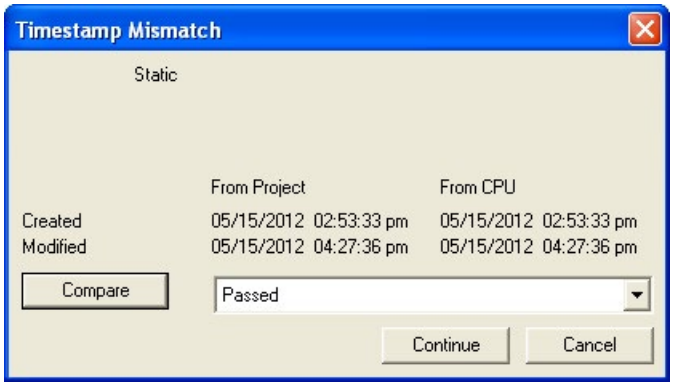
Opening the Motion control panel

To open the Motion control panel, use one of the following methods:

- Click the "Motion Control Panel" button from the Tools area of the Tools menu ribbon strip.

- Open the Tools folder in the project tree, select the "Motion Control Panel" node and press Enter; or double-click the "Motion Control Panel" node.

At this point, a comparison between the CPU and STEP 7-Micro/WIN SMART is executed to ensure that the configurations are the same. (See the figure below.)



The Axis of Motion Operation (Page 724), Configuration (Page 730), and Profile Configuration (Page 730) settings make it easy for you to monitor and control the operation of the Axis of Motion during the startup and test phases of your development process.

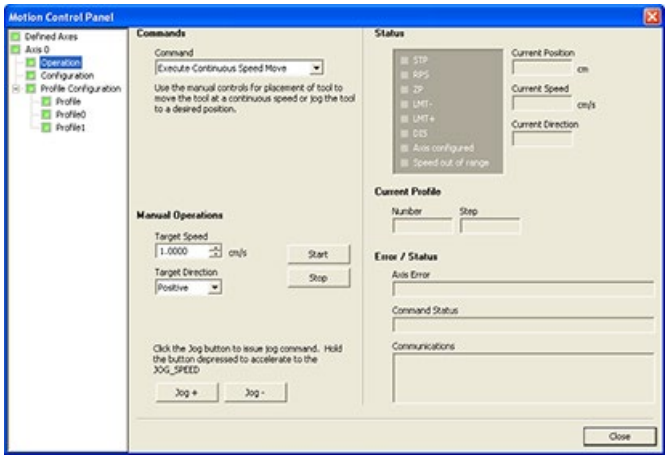
Use the Motion control panel to verify that the Axis of Motion is wired correctly, to adjust the configuration data, and to test each movement profile.

If additional changes need to be made in the Axis of Motion, refer to the Motion wizard (Page 683).

For error code listings, refer to the Axis of Motion error codes (Page 731) and the Motion instruction error codes (Page 732).

12.9.1 Displaying and controlling the operation of the Axis of Motion

In the Operation node, you can interact with the operations of the Axis of Motion. The control panel displays the current speed, the current position, and the current direction of the Axis of Motion. You can also see the status of the input and output LEDs (excluding the Pulse LEDs).



The control panel allows you to interact with the Axis of Motion by changing the speed and direction, by stopping and starting the motion, and by jogging the tool (if the CPU is stopped).

Note

You cannot execute a motion command while the CPU is running. The CPU must be in STOP mode in order to change the speed and direction, stop and start the motion, and use the jog tool.

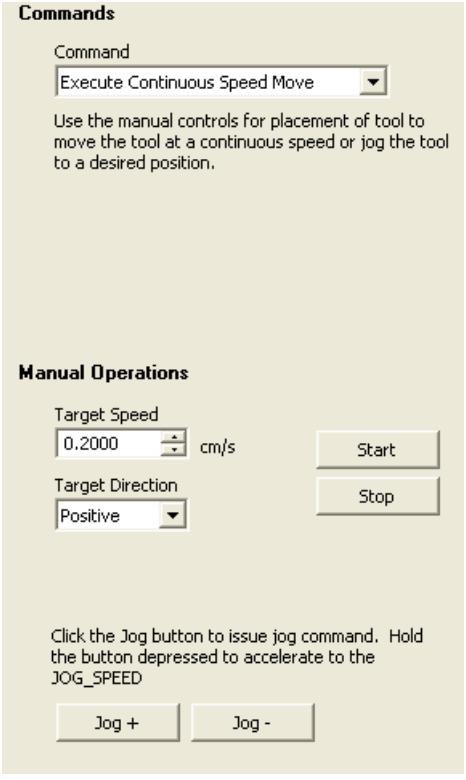
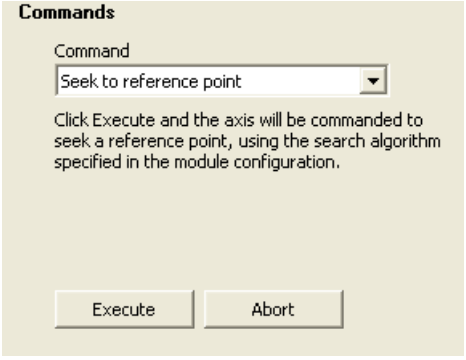
Note

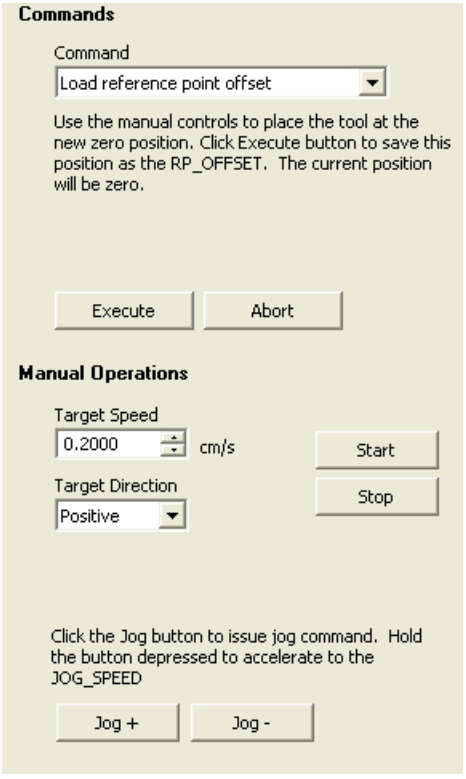
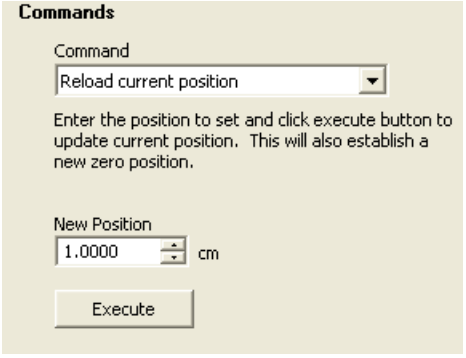
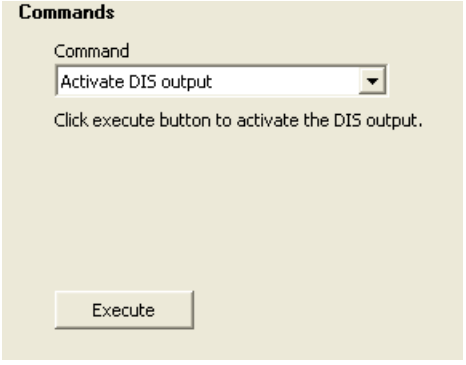
Exiting the Motion control panel or a loss of communications while a motion command is active causes the axis to stop its motion after a 5 second timeout.

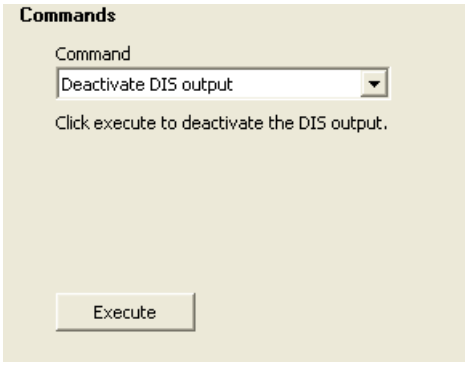
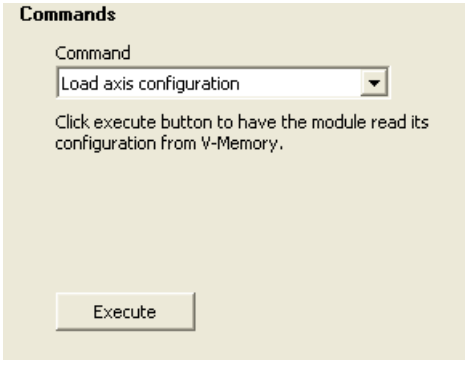
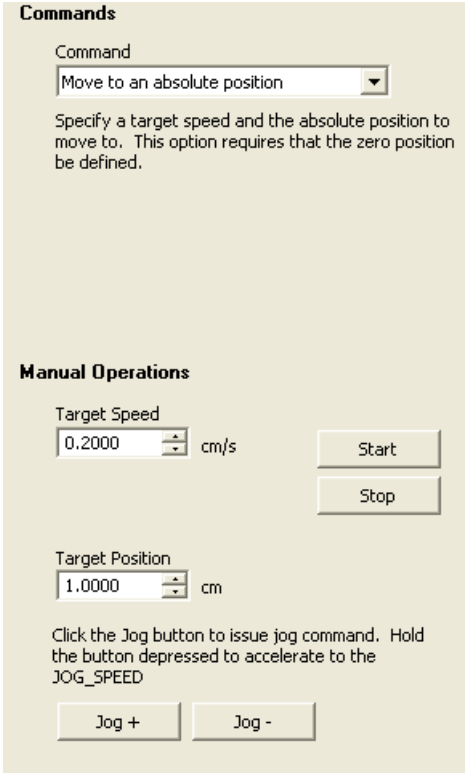
Motion commands

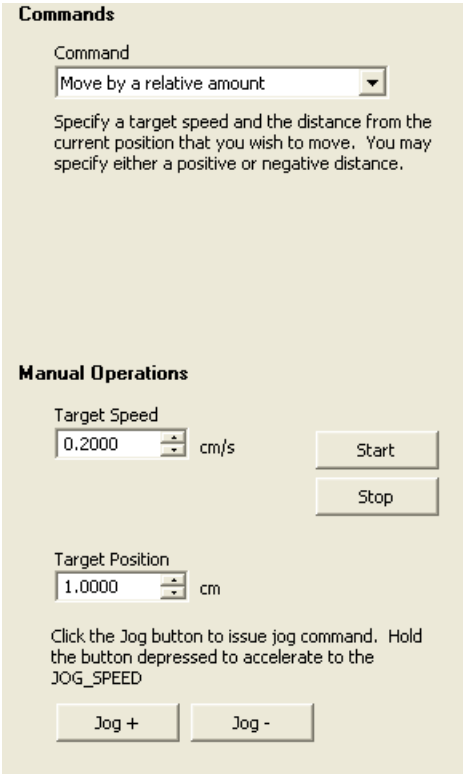
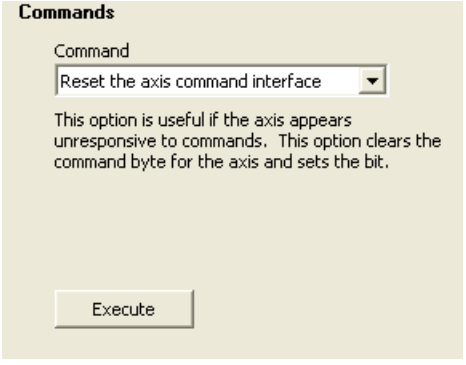
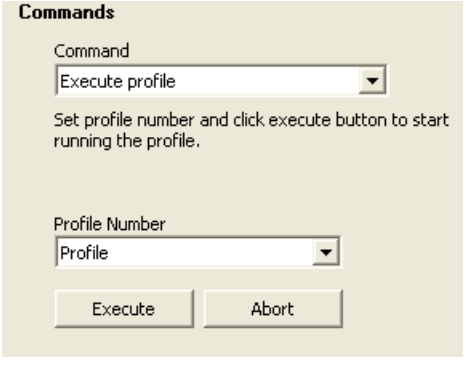
You can also generate the following motion commands:

Table 12- 35 Motion control panel commands

Command	Description
	Execute continuous speed move: This command allows you to use the manual controls for positioning the tool. Enter "Target Speed" and "Direction", and click "Start" to execute continuous move. Motion will continue until "Stop" is clicked (or error condition).
	Seek to a reference point: This command finds the reference point by using the configured search mode. Click "Execute", and the axis will command a "Seek to Reference Point", using the search algorithm specified in the axis configuration. Click "Abort" to stop a seek process before the reference point has been found.

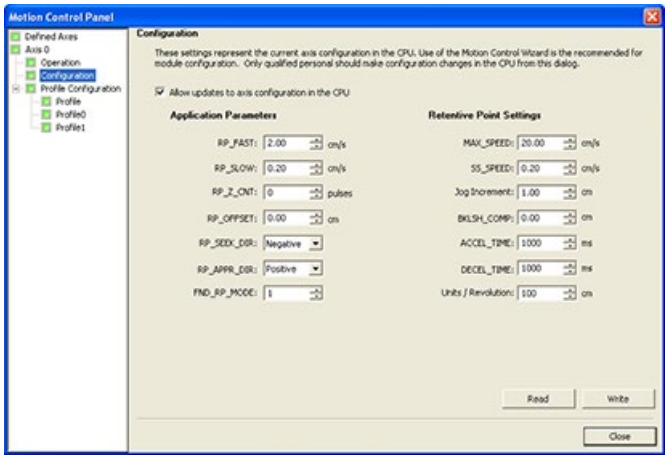
Command	Description
	Load reference point offset: After you use the manual controls to jog the tool to the new position, you then load the "Reference Point Offset". Use the manual controls to place the tool at the new position. Click "Execute" to save this position as the "RP_OFFSET". The current position will be set to zero.
	Reload current position: This command updates the current position value and establishes a new zero position. Enter the position to set and click "Execute" to update the current position. This will also establish a new zero position.
	Activate the DIS output: This command turns the DIS output of the Axis of Motion on. Click "Execute" to activate the DIS output.

Command	Description
	De-activate the DIS output: This command turns the DIS output of the Axis of Motion off. Click "Execute" to de-activate the DIS output.
	Load axis configuration: This command loads a new configuration by commanding the Axis of Motion to read the configuration block from the V memory of the CPU. Click "Execute" to have the axis read its configuration from V memory.
	Move to an absolute position: This command allows you to move to a specified position at a target speed. Before using this command, you must have already established the zero position. Assign a "Target Speed" and the "Absolute Position" to move to. This option requires that the zero position be defined.

Command	Description
	Move by a relative amount: This command allows you to move a specified distance from the current position at a target speed. You can enter a positive or negative distance. Assign a "Target Speed" and a "Target Position" to move to.
	Reset the axis command interface: This command clears the axis command interface for the Axis of Motion and sets the "Done" bit. Use this command if the Axis of Motion appears to not be responding to commands.
	Execute profile: This command allows you to select a profile to be executed. The control panel displays the status of the profile which is being executed by the Axis of Motion. Select the profile that is to be executed, then click "Execute", and the axis will command the profile. Note: This command is only available if a profile has been defined in the Motion wizard.

12.9.2 Displaying and modifying the configuration of the Axis of Motion

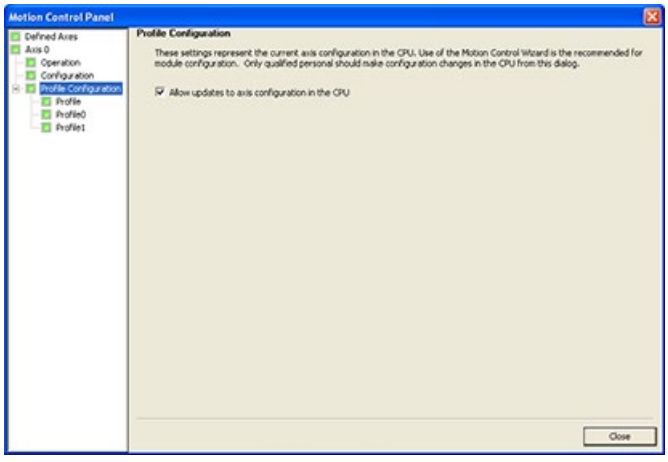
In the Configuration node, you can view and modify the configuration settings for the Axis of Motion that is stored in the data block of the CPU.



After you modify the configuration settings, you simply click the write button to send the data values to the CPU. These data values are not saved in your STEP 7-Micro/WIN SMART project. You must manually make changes to your project that reflects the final values of these fields.

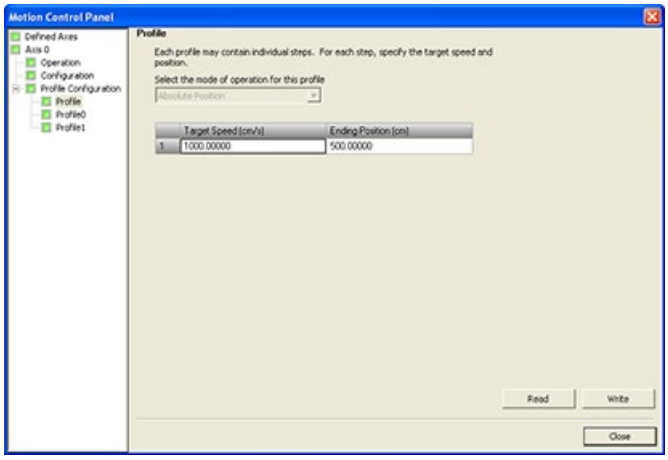
12.9.3 Displaying the profile configuration for the Axis of Motion

In the Profile Configuration node, you can view the configuration of each profile for the Axis of Motion.



Click each profile to view its mode of operation and data values.

Some data values of the profile can be modified in this dialog. After you modify the configuration settings, you simply click the write button to send the data values to the CPU. These data values are not saved in your STEP 7-Micro/WIN SMART project. You must manually make changes to your project that reflects the final values of these fields.



12.9.4 Error codes for the Axis of Motion (WORD at SMW620, SMW670, or SMW720)

Table 12- 36 Axis of Motion error codes

Error code	Description
0	No error
1	Reserved
2	Configuration block not present
3	Configuration block pointer error
4	Size of configuration block exceeds available V memory
5	Illegal configuration block format
6	Too many profiles specified
7	Illegal STP_RSP specification
8	Illegal LIM- specification
9	Illegal LIM+ specification
10	Illegal FILTER_TIME specification
11	Illegal MEAS_SYS specification
12	Illegal RP_CFG specification
13	Illegal PLS/REV value
14	Illegal UNITS/REV value
15	Illegal RP_ZP_CNT value
16	Illegal JOG_INCREMENT value
17	Illegal MAX_SPEED value
18	Illegal SS_SPD value
19	Illegal RP_FAST value
20	Illegal RP_SLOW value
21	Illegal JOG_SPEED value

Error code	Description
22	Illegal ACCEL_TIME value
23	Illegal DECEL_TIME value
24	Illegal JERK_TIME value
25	Illegal BKLSH_COMP value
26	AXIS not available
27	Invalid LMT+ location
28	Invalid LMT- location
29	Invalid STP location
30	Invalid RPS location
31	Invalid ZP location
32	Illegal output phase
33	No RPS input defined (If Homing is defined, then RPS must be defined also.)
34	Invalid TRIG location
35	Reserved
36	No ZP input defined (If Homing mode 3 or 4 is defined, then ZP must be defined also.)
37	Phase A (P0) output conflict
38	Phase B (P1) output conflict
39	DIS output conflict
40	Reserved
41	Invalid SDB0 record size
42	Illegal SDB0 format
43 to 127	Reserved

To verify that the Axis of Motion is wired correctly, to adjust the configuration data, and to test each movement profile, use the Motion control panel.

If additional changes need to be made in the Axis of Motion, go to the Motion wizard.

12.9.5 Error codes for the Motion instruction (seven LS bits of SMB634, SMB684, or SMB734)

In the SM table for each axis there is a byte reserved to display the result of the motion instruction (Offset 34). This byte indicates when an instruction is complete and if there was an error in the instruction.

Table 12- 37 Motion instruction error codes

Error code	Description
0	No error
1	Aborted by user
2	Configuration error (This error occurs if there is an error in the SDB0 configuration.)

Error code	Description
3	Illegal command
4	Aborted due to no valid configuration (This error occurs if there is an error in the configuration table.)
5	Reserved
6	Aborted due to no defined reference point
7	Aborted due to STP input active
8	Aborted due to LMT- input active
9	Aborted due to LMT+ input active
10	Aborted due to problem executing motion
11	No profile block configured for specified profile
12	Illegal operation mode
13	Operation mode not supported for this command
14	Illegal number of steps in profile block
15	Illegal direction change
16	Illegal distance
17	RPS/TRIG trigger occurred before target speed reached
18	Insufficient RPS active region width
19	Speed out-of-range
20	Insufficient distance to perform desired speed change
21	Illegal position
22	Zero position unknown
23	No DIS output is defined
24	Reserved
25	Aborted due to CPU going to stop
26	Aborted due to expiration of Motion control panel heartbeat
27 to 127	Reserved
128	Axis of Motion cannot process this instruction: either the Axis of Motion is busy with another instruction, or there was no Start pulse on this instruction.
129	Reserved
130	Axis of Motion is not enabled
131	Reserved
132	Reserved
133	Illegal profile specified. The AXISx_RUN and AXISx_CACHE instructions profile number range must be between 0 - 31.
134	Illegal mod specified in AXISx_GOTO instruction

To verify that the Axis of Motion is wired correctly, to adjust the configuration data, and to test each movement profile, use the Motion control panel.

If additional changes need to be made in the Axis of Motion, go to the Motion wizard.

12.10 Advanced topics

12.10.1 Understanding the configuration/profile table for the Axis of Motion

Overview

The Motion wizard has been developed to make motion applications easy by automatically generating the configuration and profile information based upon the answers you give about your motion control system. Configuration/profile table information is provided for advanced users who want to create their own motion control routines.

The configuration/profile table is located in the V memory area of the S7-200 SMART CPU. As shown in the table below, the configuration settings are stored in the following types of information:

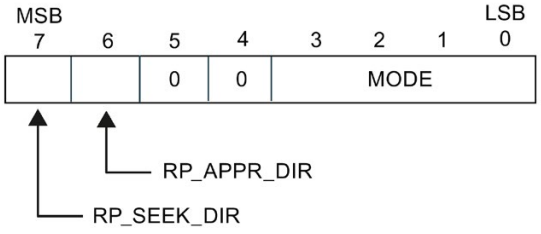
- **Configuration block:** Contains information used to setup the Axis of Motion in preparation for executing position commands
- **Interactive block:** Supports direct setup of position parameters by the user program
- **Profile block:** Describes a pre-defined move operation to be performed by the Axis of Motion. You can configure up to 32 profile blocks.

Note

The profile block of the configuration/profile table can contain up to 32 motion profiles. To create more than 32 move profiles, you can exchange configuration/profile tables by changing the value stored in the configuration/profile table pointer.

Table 12- 38 Configuration/Profile table: Configuration block

Configuration/Profile table			
Byte offset	**Name**	**Function description**	**Type**
Configuration block			
0	MOD_ID	Axis of Motion identification field	--
5	CB_LEN	Length of the configuration block in bytes (1 byte)	--
6	IB_LEN	Length of the interactive block in bytes (1 byte)	--
7	PF_LEN	Length of a single profile in bytes (1 byte)	--
8	STP_LEN	Length of a single step in bytes (1 byte)	--
9	STEPS	Number of steps allowed per profile (1 byte)	--
10	PROFILES	Number of profiles from 0 to 32 (1 byte)	--
11	--	Reserved: Set to 0	--
13	--	Reserved: Set to 0	--
14	STOP_RSP	Specifies the drive's response to the STP input (1 byte): • 0: No action. Ignore input condition. • 1: Decelerate to a stop and indicate STP input active. • 2: Terminate pulses and indicate STP input. • 3 to 255: Reserved (error if specified)	--
15	LMT-_RSP	Specifies the drive's response to the negative limit input (1 byte): • 0: No action. Ignore input condition. • 1: Decelerate to a stop and indicate limit reached. • 2: Terminate pulses and indicate limit reached. • 3 to 255: Reserved (error if specified)	--
16	LMT+_RSP	Specifies the drive's response to the positive limit input (1 byte): • 0: No action. Ignore input condition. • 1: Decelerate to a stop and indicate limit reached. • 2: Terminate pulses and indicate limit reached. • 3 to 255: Reserved (error if specified)	--
17	--	Reserved: Set to 0	--
18	MEAS_SYS	Specifies the measurement system used to describe moves (1 byte): • 0: Pulses (speed measured in pulses/sec and position values measured in pulses; values are double integer) • 1: Engineering units (speed measured in units/sec and position values measured in units; values are single precision real) • 2 to 255: Reserved (error if specified)	--

<table>
<tr><td colspan="4">Configuration/Profile table</td></tr>
<tr><td>Byte off-set</td><td>Name</td><td>Function description</td><td>Type</td></tr>
<tr><td colspan="4">Configuration block</td></tr>
<tr><td>19</td><td>--</td><td>Reserved: Set to 0</td><td>--</td></tr>
<tr><td>20</td><td>PLS/REV</td><td>Specifies the number of pulses per revolution of the motor, (only applicable when MEAS_SYS is set to 1) - (4 bytes)</td><td>DInt</td></tr>
<tr><td>24</td><td>UNITS/REV</td><td>Specifies the engineering units per revolution of the motor, (only applicable when MEAS_SYS is set to 1) - (4 bytes)</td><td>Real</td></tr>
<tr><td>28</td><td>UNITS</td><td>Reserved for STEP 7-Micro/WIN SMART to store a custom units string (4 bytes)</td><td>--</td></tr>
<tr><td>32</td><td>RP_CFG</td><td>Specifies the reference point search configuration (1 byte):
 

RP_SEEK_DIR: This bit specifies the starting direction for a reference point search (0 - positive direction, 1 - negative direction).
RP_APPR_DIR: This bit specifies the approach direction for terminating the reference point search (0 - positive direction, 1 - negative direction).
<table>
<tr><td>MODE</td><td>Specifies the reference point search method</td></tr>
<tr><td>'0000'</td><td>Reference point search disabled.</td></tr>
<tr><td>'0001'</td><td>The reference point is where the RPS input goes active.</td></tr>
<tr><td>'0010'</td><td>The reference point is centered within the active region of the RPS input.</td></tr>
<tr><td>'0011'</td><td>The reference point is outside the active region of the RPS input.</td></tr>
<tr><td>'0100'</td><td>The reference point is within the active region of the RPS input.</td></tr>
<tr><td>'0101' to '1111'</td><td>Reserved (error if selected)</td></tr>
</table></td><td>--</td></tr>
<tr><td>33</td><td>--</td><td>Reserved: Set to 0</td><td>--</td></tr>
<tr><td>34</td><td>RP_Z_CNT</td><td>Number of pulses of the ZP input used to define the reference point (4 bytes)</td><td>DInt</td></tr>
<tr><td>38</td><td>RP_FAST</td><td>Fast speed for the RP seek operation: MAX_SPD or less (4 bytes)</td><td>DInt/Real</td></tr>
</table>

Configuration/Profile table			
Byte offset	**Name**	**Function description**	**Type**
Configuration block			
42	RP_SLOW	Slow speed for the RP seek operation: Maximum speed from which the motor can instantly go to a stop or less (4 bytes)	DInt/Real
46	SS_SPEED	Start/Stop Speed. (4 bytes): The starting speed is the maximum speed to which the motor can instantly go from a stop and the maximum speed from which the motor can instantly go to a stop. Operation below this speed is allowed, but the acceleration and deceleration times do not apply.	DInt/Real
50	MAX_SPEED	Maximum operating speed of the motor (4 bytes)	DInt/Real
54	JOG_SPEED	Jog speed (4 bytes): MAX_SPEED or less (4 bytes)	DInt/Real
58	JOG_INCREMENT	Jog increment value: The distance (or number of pulses) to move in response to a single jog pulse (4 bytes).	DInt/Real
62	ACCEL_TIME	Time required to accelerate from minimum to maximum speed in msec (4 bytes)	DInt
66	DECEL_TIME	Time required to decelerate from maximum to minimum speed in msec (4 bytes)	DInt
70	BKLSH_COMP	Backlash compensation: The distance used to compensate for the system backlash on a direction change (4 bytes).	DInt/Real
74	JERK_TIME	Time during which jerk compensation is applied to the beginning and ending portions of an acceleration/deceleration curve (S-curve). Specifying a value of 0 disables jerk compensation. The jerk time is given in milliseconds (Range: 0 ms to 32000 ms. (4 bytes)	DInt

Table 12- 39 Configuration/Profile table: Interactive block

Configuration/Profile table			
Byte offset	**Name**	**Function description**	**Type**
Interactive block			
78	MOVE_CMD	Selects the mode of operation (1 byte): • 0: Absolute position • 1: Relative position • 2: Single-speed, continuous positive rotation • 3: Single-speed, continuous negative rotation • 4: Manual speed control, positive rotation • 5: Manual speed control, negative rotation • 6: Single-speed, continuous positive rotation with triggered stop (Activation of RPS triggers the stop; TARGET_POS contains distance to travel after signal) • 7: Single-speed, continuous negative rotation with triggered stop (Activation of RPS triggers the stop; TARGET_POS contains distance to travel after signal) • 8 to 255: Reserved (error if specified)	--
79	--	Reserved: Set to 0	--
80	TGT_POS	Target position to go to in this move (4 bytes)	DInt/Real
84	TGT_SPEED	Target speed for this move (4 bytes)	DInt/Real
88	RP_OFFSET	Absolute position of the reference point (4 bytes)	DInt/Real

Table 12- 40 Configuration/Profile table: Profile block 0

Configuration/Profile table			
Byte offset	**Name**	**Function description**	**Type**
		Profile block 0	
92(+0)	STEPS	Number of steps in this move sequence (1 byte)	--
93(+1)	MODE	Selects the mode of operation for this profile block (1 byte): • 0: Absolute position • 1: Relative position • 2: Single-speed, continuous positive rotation • 3: Single-speed, continuous negative rotation • 4: Reserved (error if specified) • 5: Reserved (error if specified) • 6: Single-speed, continuous positive rotation with triggered stop (RPS input signals stop) • 7: Single-speed, continuous negative rotation with triggered stop (RPS input signals stop) • 8: Two-speed, continuous positive rotation RPS selects speed) • 9: Two-speed, continuous negative rotation (RPS selects speed) • 10: Two-speed, continuous positive rotation with triggered stop (RPS selects speed, TRIG input signals stop) • 11: Two-speed, continuous negative rotation with triggered stop (RPS selects speed, TRIG input signals stop) • 12 to 255: Reserved (error if specified)	--
94(+2)	Step 0: POS	Position to go to in move step 0 (4 bytes)	DInt/Real
98(+6)	Step 0: SPEED	Target speed for move step 0 (4 bytes)	DInt/Real
102(+10)	Step 1: POS	Position to go to in move step 1 (4 bytes)	DInt/Real
106(+14)	Step 1: SPEED	Target speed for move step 1 (4 bytes)	DInt/Real
110(+18)	Step 2: POS	Position to go to in move step 2 (4 bytes)	DInt/Real
114(+22)	Step 2: SPEED	Target speed for move step 2 (4 bytes)	DInt/Real
118(+26)	Step 3: POS	Position to go to in move step 3 (4 bytes)	DInt/Real
122(+30)	Step 3: SPEED	Target speed for move step 3 (4 bytes)	DInt/Real
...	Step ...: POS	Position to go to in move step ... (4 bytes)	DInt/Real
...	Step ...: SPEED	Target speed for move step ... (4 bytes)	DInt/Real
214(+122)	Step 15: POS	Position to go to in move step 3 (4 bytes)	DInt/Real
218(+126)	Step 15: SPEED	Target speed for move step 3 (4 bytes)	DInt/Real

Note

You can have between 1 and 16 steps in the Configuration/Profile table, Profile block 0.

Table 12- 41 Configuration/Profile table: Profile block 1

Configuration/Profile table			
Byte offset	**Name**	**Function description**	**Type**
Profile block 1			
X[1]	STEPS	Number of steps in this move sequence (1 byte) **Note**: There can be up to 16 steps.	--
(X + 1)	MODE	Selects the mode of operation for this profile block (1 byte)	--
(X + 2)	Step 0: POS	Position to go to in move step 0 (4 bytes)	DInt/Real
(X + 4)	Step 0: SPEED	The target speed for move step 0 (4 bytes)	DInt/Real
...	...	...	...

[1] The offset of Profile block 1 and subsequent blocks is variable and dependent upon the number of steps configured in the largest profile. The offset is determined by the following formula:
Offset of Profile block x = CB_LEN + IB_LEN + (x * PF_LEN)

Table 12- 42 Profile detail for Mode 0 (Absolute position)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	n = Number of steps configured in this profile
+1		MODE	byte	0 = Absolute position
+2	0	POS	dint/fp	Destination position in step 0
+6		SPEED	dint/fp	Target speed for step 0
• • •				
(4 * n) + 2	n	POS	dint/fp	Destination position in step n
($ * N) + 6		SPEED	dint/fp	Target speed for step n

Table 12- 43 Profile detail for Mode 1 (Relative position)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	n = Number of steps configured in this profile
+1		MODE	byte	0 = Relative position
+2	0	POS	dint/fp	Distance to travel in step 0
+6		SPEED	dint/fp	Target speed for step 0
• • •				
(4 * n) + 2	n	POS	dint/fp	Distance to travel in step n
($ * N) + 6		SPEED	dint/fp	Target speed for step n

Table 12- 44 Profile detail for Mode 2 (Single-speed, continuous positive rotation) and Mode 3 (Single-speed, continuous negative rotation)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	1
+1		MODE	byte	2= Single-speed, continuous positive rotation or 3 = Single-speed, continuous negative rotation
+2	0	POS	dint/fp	n.a. (must be set to 0)
+6		SPEED	dint/fp	Target speed

Table 12- 45 Profile detail for Mode 6 (Single-speed, continuous positive rotation with triggered stop) and Mode 7 (Single-speed, continuous negative rotation with triggered stop)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	1
+1		MODE	byte	6 = Single-speed, continuous positive rotation with triggered stop or 7 = Single-speed, continuous negative rotation with triggered stop
+2	0	POS	dint/fp	Distance to travel after activation of RPS signal (value must be positive)
+6		SPEED	dint/fp	Target speed

Table 12- 46 Profile detail for Mode 8 (Two-speed, continuous positive rotation) and Mode 9 (Two-speed, continuous negative rotation)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	2
+1		MODE	byte	8 = Two-speed, continuous positive rotation or 9 = Two-speed, continuous negative rotation
+2	0	POS	dint/fp	n.a. (must be set to 0)
+6		SPEED	dint/fp	Target speed if RPS signal is inactive
+10	1	POS	dint/fp	n.a. (must be set to 0)
+14		SPEED	dint/fp	Target speed if RPS signal is active

Table 12- 47 Profile detail for Mode 10 (Two-speed, continuous positive rotation with triggered stop) and Mode 11 (Two-speed, continuous negative rotation with triggered stop)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	2
+1		MODE	byte	10 = Two-speed, continuous positive rotation with triggered stop or 11 = Two-speed, continuous negative rotation with triggered stop
+2	0	POS	dint/fp	Distance to travel after activation of TRIG signal (value must be positive)
+6		SPEED	dint/fp	Target speed if RPS signal is inactive
+10	1	POS	dint/fp	n.a. (must be set to 0)
+14		SPEED	dint/fp	Target speed if RPS signal is active

12.10.2 Special memory (SM) locations for the Axis of Motion

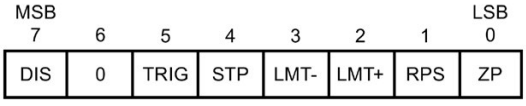
The CPU allocates 50 bytes of special memory (SM) to each Axis of Motion. (See the following table.) When the Axis of Motion detects an error condition or a change in status of the data, the Axis of Motion updates these SM locations. The first Axis of Motion updates SMB600 through SMB649 as required to report the error condition, the second Axis of Motion updates SMB650 through SMB699, and so on.

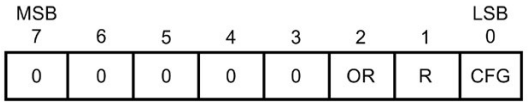
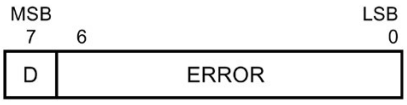
Table 12- 48 Special memory bytes SMB600 to SMB749

SM bytes for Axes of Motion:		
Axis of Motion 0	Axis of Motion 1	Axis of Motion 2
SMB600 to SMB649	SMB650 to SMB699	SMB700 to SMB749

The following table shows the structure of the SM data area allocated for an Axis of Motion. The definition uses Axis 0 as an example.

Table 12- 49 Special memory area definition for the Axis of Motion 0

SM address	Description
SMB600 to SMB615	Axis name (16 ASCII characters). SMB600 is the first character: "Axis 0"
SMB616 to SMB619	Reserved
SMW620	Axis 0: Error code (See "Axis of Motion error codes" (Page 731) list.)
SMB622	Axis 0: Input/output status: Reflects the status of the inputs and outputs  • DIS (Disable outputs): – 0 = No current flow – 1 = Current flow • TRIG (Stop input): – 0 = No current flow – 1 = Current flow • STP (Stop input): – 0 = No current flow – 1 = Current flow • LMT- (Negative travel limit input): – 0 = No current flow – 1 = Current flow • LMT+ (Positive travel limit input): – 0 = No current flow – 1 = Current flow • RPS (Reference point switch input): – 0 = No current flow – 1 = Current flow • ZP (Zero pulse input): – 0 = No current flow – 1 = Current flow

SM address	Description
SMB623	Axis 0 Instantaneous status: Reflects the status of the configuration and rotation direction status  • OR (Target speed out of range): – 0 = In range – 1 = Out of range • R (Direction of rotation): – 0 = Positive rotation – 1 = Negative rotation • CFG (Module configured): – 0 = Not configured – 1 = Configured
SMB624	Axis 0: CUR_PF is a byte that indicates the profile currently being executed.
SMB625	Axis 0: CUR_STP is a byte that indicates the step currently being executed in the profile.
SMD626	Axis 0: CUR_POS is a double-word value that indicates the current position of the Axis of Motion.
SMD630	Axis 0: CUR_SPD is a double-word value that indicates the current speed of the Axis of Motion.
SMB634	Axis 0: Result of the instruction. Error conditions above 127 are generated by the instruction subroutines created by the Motion wizard.  • D (Done bit): – 0= Operation in progress – 1= Operation complete (set by the Axis of Motion during initialization) • ERROR: (See "Motion instruction error codes" (Page 732) list.)
SMB635 to SMB645	Reserved
SMD646	Axis 0: Pointer to the V memory location of the configuration/profile table. A pointer value to an area other than V memory is not valid. The Axis of Motion monitors this location until it receives a non-zero pointer value.

12.11 Understanding the RP Seek modes of the Axis of Motion

The following figures provide diagrams of the different options for each RP Seek mode:

- RP Seek: Mode 1 shows two of the options for RP Seek mode 1. This mode locates the RP where the RPS input goes active on the approach from the work zone side.
- RP Seek: Mode 2 shows two of the options for RP Seek mode 2. This mode locates the RP in the center within the active region of the RPS input.
- RP Seek: Mode 3 shows two of the options for RP Seek mode 3. This mode locates the RP a specified number of zero pulses (ZP) outside the active region of the RPS input.
- RP Seek: Mode 4 shows two of the options for RP Seek mode 4. This mode locates the RP a specified number of zero pulses (ZP) within the active region of the RPS input.

For each mode, there are four combinations of RP Seek direction and RP Approach direction. (Only two of the combinations are shown.) These combinations determine the pattern for the RP Seek operation. For each of the combinations, there are also four different starting points:

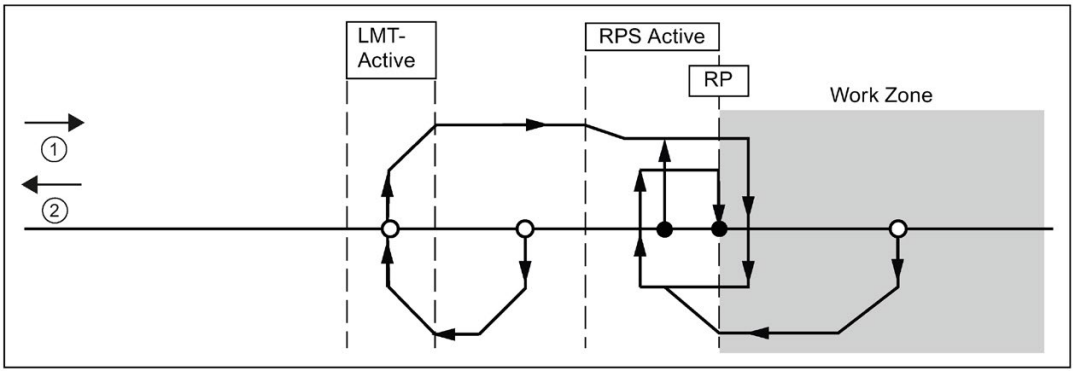
The work zones for each diagram have been located so that moving from the reference point to the work zone requires movement in the same direction as the RP Approach Direction. By selecting the location of the work zone in this way, all the backlash of the mechanical gearing system is removed for the first move to the work zone after a reference point seek.

Note

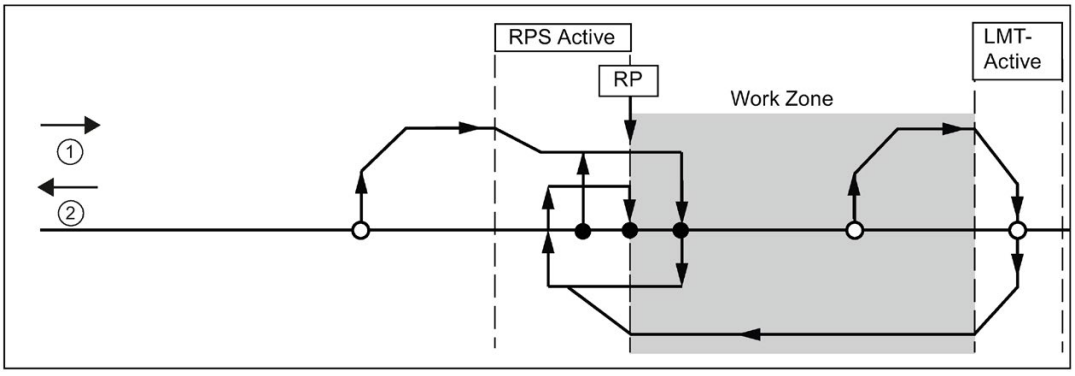
The RPS input must be enabled to use the RP Seek functionality. The ZP input must also be enabled if RP Seek modes 3 or 4 are to be used, unless you configure the number of ZP pulses to be received after entering the RPS active region to a value of "0".

RP seek mode 1

Default configuration: RP seek direction: negative and RP approach direction: positive



Default configuration: RP seek direction: positive and RP approach direction: positive

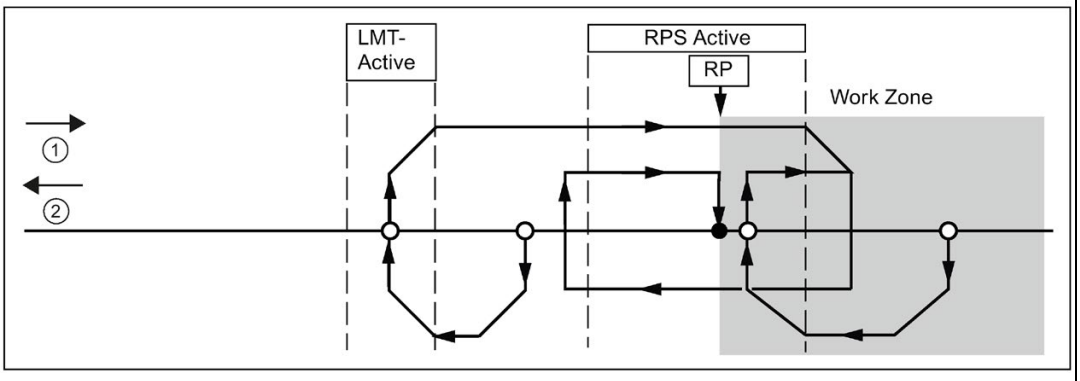


①: Positive motion

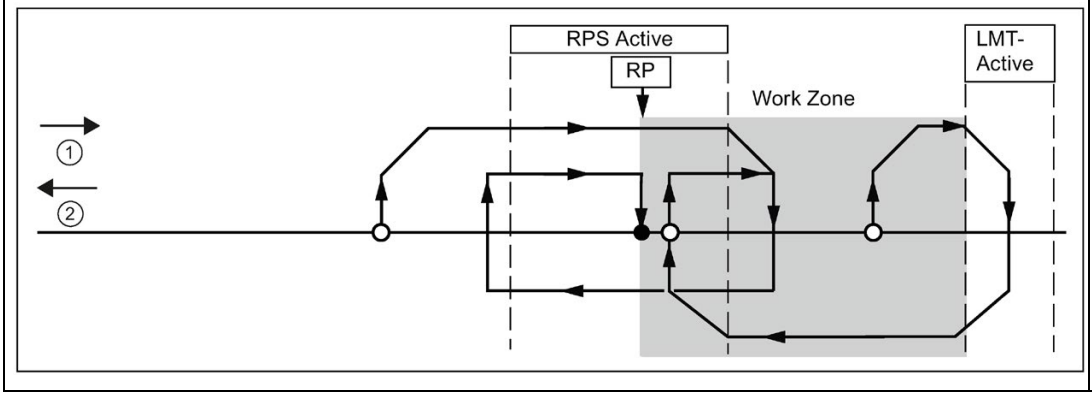
②: Negative motion

RP seek mode 2

Default configuration: RP seek direction: negative and RP approach direction: positive

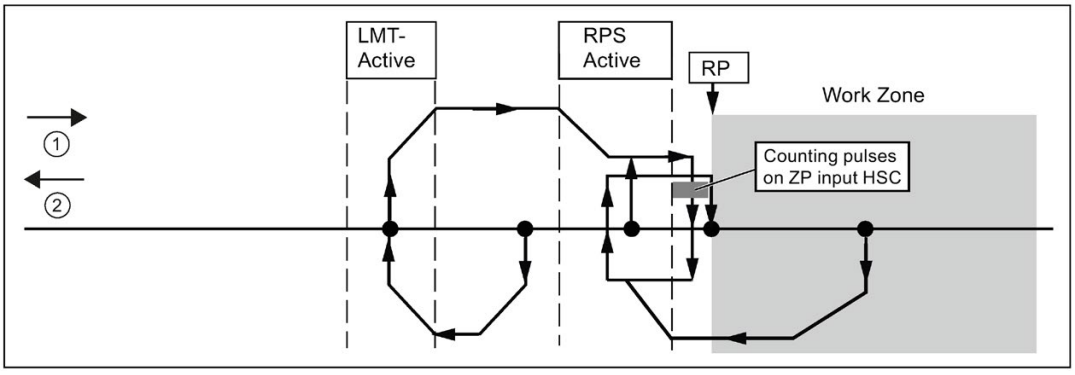
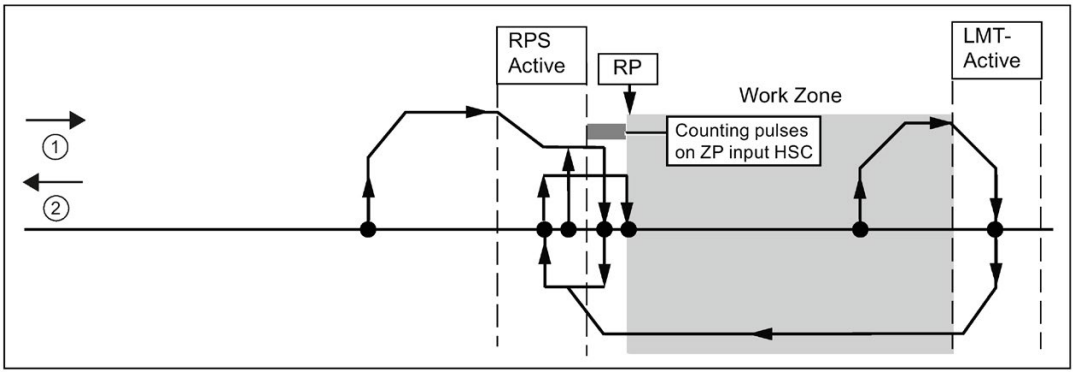


Default configuration: RP seek direction: positive and RP approach direction: positive



①: Positive motion

②: Negative motion

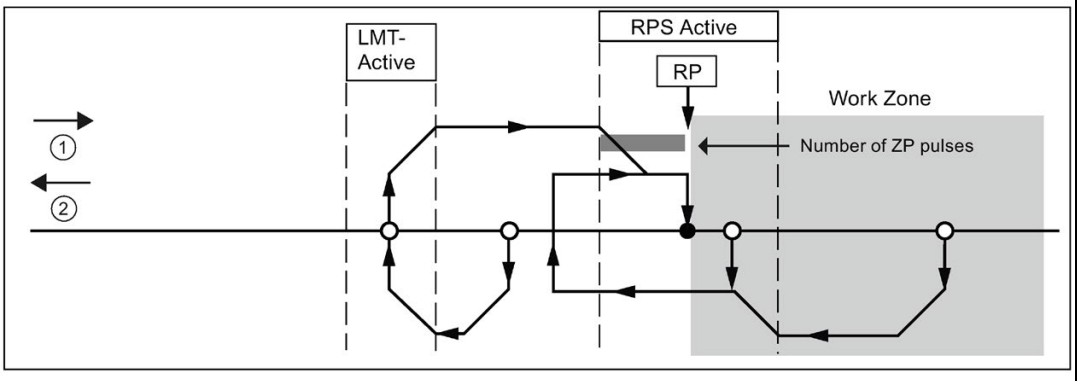
RP seek mode 3
Default configuration: RP seek direction: negative and RP approach direction: positive

Default configuration: RP seek direction: positive and RP approach direction: positive


①: Positive motion

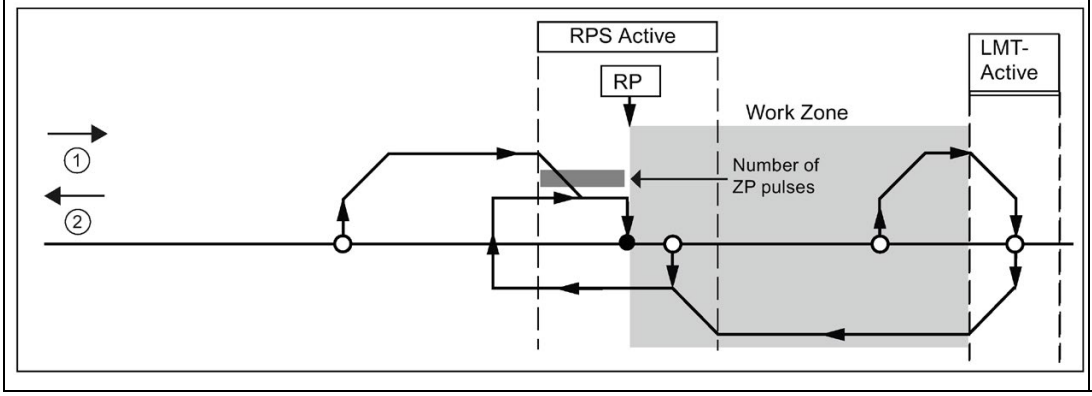
②: Negative motion

RP seek mode 4

Default configuration: RP seek direction: negative and RP approach direction: positive



Default configuration: RP seek direction: positive and RP approach direction: positive



①: Positive motion

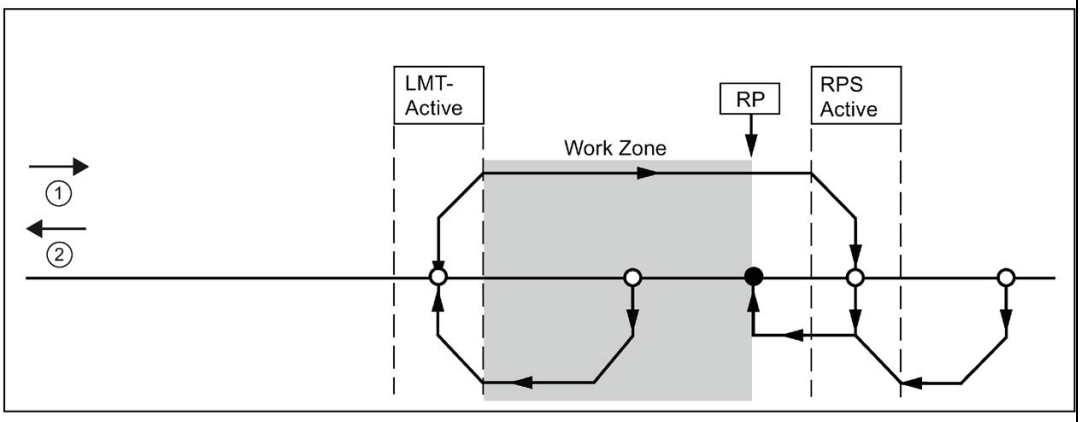
②: Negative motion

12.11.1 Selecting the work zone location to eliminate backlash

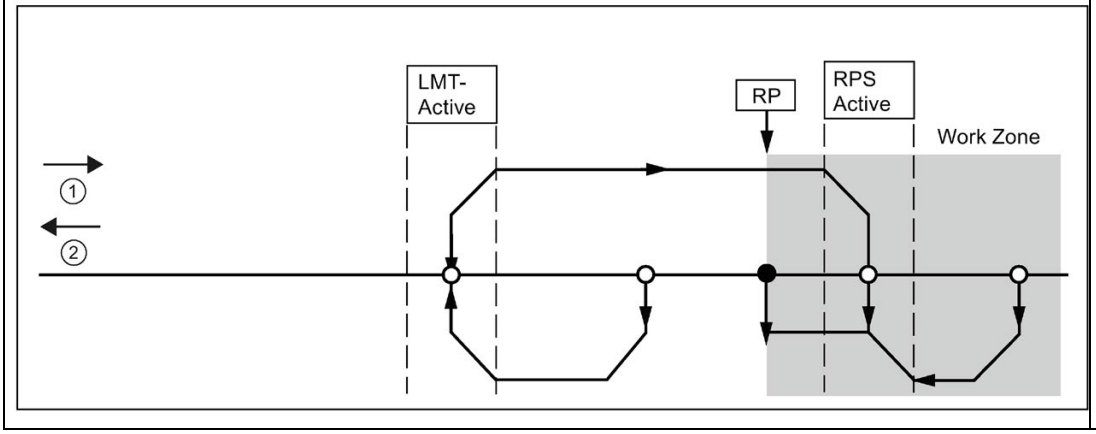
The following figure shows the work zone in relationship to the reference point (RP), the RPS Active zone, and the limit switches (LMT+ and LMT-) for an approach direction that eliminates the backlash. The second part of the illustration places the work zone so that the backlash is not eliminated. The following figure shows RP seek mode 3. A similar placement of the work zone is possible, although not recommended, for each of the search sequences for each of the other RP seek modes.

Selecting the work zone location to eliminate backlash

Backlash is eliminated: RP seek direction: negative and RP approach direction: negative



Backlash is not eliminated: RP seek direction: negative and RP approach direction: negative



①: Positive motion

② Negative motion

Technical specifications

A

A.1 General specifications

A.1.1 General technical specifications

Standards compliance

The S7-200 SMART automation system complies with the following standards and test specifications. The test criteria for the S7-200 SMART automation system are based on these standards and test specifications.

CE approval

The S7-200 SMART Automation System satisfies requirements and safety related objectives according to the EC directives listed below, and conforms to the harmonized European standards (EN) for the programmable controllers listed in the Official Journals of the European Community.

- EC Directive 2006/95/EC (Low Voltage Directive) "Electrical Equipment Designed for Use within Certain Voltage Limits"
  - EN 61131-2: Programmable controllers - Equipment requirements and tests
- EC Directive 2004/108/EC (EMC Directive) "Electromagnetic Compatibility"
  - Emission standard
    EN 61000-6-4: AI: Industrial Environment
  - Immunity standard
    EN 61000-6-2: Industrial Environment

The CE Declaration of Conformity is held on file available to competent authorities at:

Siemens AG
Sector Industry
DF FA AS DH AMB
Postfach 1963
D-92209 Amberg
Germany

Industrial environments

The S7-200 SMART automation system is designed for use in industrial environments.

Table A- 1 Industrial environments

Application field	Noise emission requirements	Noise immunity requirements
Industrial	EN 61000-6-4:	EN 61000-6-2:

Electromagnetic compatibility

Electromagnetic Compatibility (EMC) is the ability of an electrical device to operate as intended in an electromagnetic environment and to operate without emitting levels of electromagnetic interference (EMI) that may disturb other electrical devices in the vicinity.

Table A- 2 Immunity per EN 61000-6-2

Electromagnetic compatibility - Immunity per EN 61000-6-2	
EN 61000-4-2 Electrostatic discharge	±8 kV air discharge to all surfaces ±4 kV contact discharge to exposed conductive surfaces
EN 61000-4-3 Radiated, radio-frequency, electromagnetic field immunity test	80 to 1000 MHz, 10 V/m, 80% AM at 1 kHz 1.4 to 2.0 GHz, 3 V/m, 80% AM a 1 kHz 2.0 to 2.7 GHz, 1 V/m, 80% AM at 1 kHz
EN 61000-4-4 Fast transient bursts	2 kV, 5 kHz with coupling network to AC and DC system power 2 kV, 5 kHz with coupling clamp to I/O
EN 6100-4-5 Surge immunity	AC systems - 2 kV common mode, 1kV differential mode DC systems - 2 kV common mode, 1kV differential mode For DC systems (I/O signals, DC power systems) external protection is required.
EN 61000-4-6 Immunity to conducted disturbances	150 kHz to 80 MHz, 10 V RMS, 80% AM at 1 kHz
EN 61000-4-11 Voltage dips	AC systems 0% for 1 cycle, 40% for 12 cycles and 70% for 30 cycles at 60 Hz

Table A- 3 Conducted and radiated emissions per EN 61000-6-4

Electromagnetic compatibility - Conducted and radiated emissions per EN 61000-6-4		
Conducted Emissions	0.15 MHz to 0.5 MHz	<79dB (µV) quasi-peak; <66 dB (µV) average
EN 55011, Class A, Group 1	0.5 MHz to 30 MHz	<73dB (µV) quasi-peak; <60 dB (µV) average
Radiated Emissions	30 MHz to 230 MHz	<40dB (µV/m) quasi-peak; measured at 10 m
EN 55011, Class A, Group 1	230 MHz to 1 GHz	<47dB (µV/m) quasi-peak; measured at 10 m

Environmental conditions

Table A- 4 Transport and storage

Environmental conditions - Transport and storage	
EN 60068-2-2, Test Bb, Dry heat and EN 60068-2-1, Test Ab, Cold	-40 °C to +70 °C
EN 60068-2-30, Test Db, Damp heat	25 °C to 55 °C, 95% humidity
EN 60068-2-14, Test Na, temperature shock	-40 °C to +70 °C, dwell time 3 hours, 2 cycles
EN 60068-2-32, Free fall	0.3 m, 5 times, product packaging
Atmospheric pressure	1139 to 660 h Pa (corresponding to an altitude of -1000 to 3500 m)

Table A- 5 Operating conditions

Environmental conditions - Operating	
Ambient temperature range (Inlet Air 25 mm below unit)	0 °C to 55 °C horizontal mounting 0 °C to 45 °C vertical mounting 95% non-condensing humidity
Atmospheric pressure	1139 to 795 hPa (corresponding to an altitude of -1000 to 2000 m)
Concentration of contaminants	$S0_2$: < 0.5 ppm; H_2S: < 0.1 ppm; RH < 60% non-condensing
EN 60068-2-14, Test Nb, temperature change	5 °C to 55 °C, 3 K /minute
EN 60068-2-27 Mechanical shock	15 g, 11 ms pulse, 6 shocks in each of 3 axis
EN 60068-2-6 Sinusoidal vibration	DIN rail mount: 3.5 mm from 5-8.4 Hz, 1G from 8.4 - 150 Hz

Table A- 6 High potential isolation test

High potential isolation test	
24 V DC/ 5 V DC nominal circuits	707 V DC (type test of optical isolation boundaries)
230 V AC circuits to ground and 24 V DC / 5 V DC circuits	2300 V AC or 3250 V DC
Ethernet port to 24 V DC / 5 V DC circuits and ground[1]	1500 V AC (type test only)

[1] Ethernet port isolation is designed to limit hazard during short term network faults to hazardous voltages. It does not conform to safety requirements for routine AC line voltage isolation.

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

Insulation

The insulation is designed in accordance with the requirements of EN 61131-2.

Note

For modules with 24 V DC supply voltage, the electrical isolation is designed for max. 60 V AC / 75 V DC and basic insulation is designed according to EN 61131-2.

Contamination level/overvoltage category according to IEC 61131-2

- Pollution degree 2
- Overvoltage category: II

Protection class in accordance with IEC 61131-2

- Protection Class II according to EN 61131-2 (Protective conductor not required)

Degree of protection IP20

- IP20 Mechanical Protection, EN 60529
- Protects against finger contact with high voltage as tested by standard probe. External protection required for dust, dirt, water and foreign objects of < 12.5mm in diameter.

Rated voltages

Table A- 7 Rated voltages

Rated voltage	Tolerance
24 V DC	20.4 V DC to 28.8 V DC
120/240 V AC	85 V AC to 264 V AC, 47 to 63 Hz

Note

When a mechanical contact turns on output power to the S7-200 SMART CPU, or any digital expansion module, it sends a "1" signal to the digital outputs for approximately 150 microseconds. This could cause unexpected machine or process operation which could result in death or serious injury to personnel and/or damage to equipment. You must plan for this, especially if you are using devices which respond to short duration pulses.

WARNING

Time duration for a mechanical contact to turn on output power

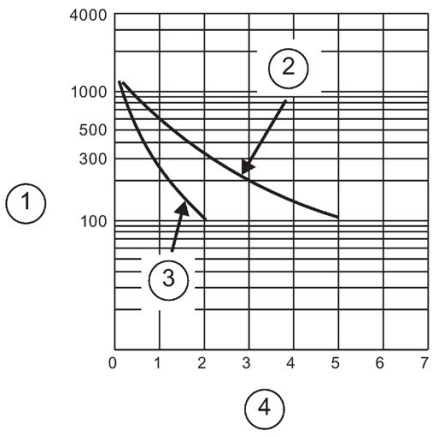
When a mechanical contact turns on output power to the S7-200 SMART CPU, or any digital expansion module, it sends a "1" signal to the digital outputs for approximately 150 microseconds.

This can cause unexpected machine or process operation which can result in death or serious injury to personnel and/or damage to equipment.

You must plan for this situation, especially, if you use devices which respond to short duration pulses.

Relay electrical service life

The typical performance data supplied by relay vendors is shown below. Actual performance may vary depending upon your specific application. An external protection circuit that is adapted to the load will enhance the service life of the contacts.



① Service life (x 10^3 operations)

② 250 V AC resistive load
30 V DC resistive load

③ 250 V AC inductive load (p.f=0.4)
30 V DC inductive load (L/R=7ms)

④ Rated Operating Current (A)

A.2 S7-200 SMART CPUs

A.2.1 CPU ST20, CPU SR20, and CPU CR20s

A.2.1.1 General specifications and features

General specifications and features of the CPU ST20, CPU SR20, and CPU CR20s

Table A- 8 General specifications

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Article number	6ES7288-1ST20-0AA0	6ES7288-1SR20-0AA0	6ES7288-1CR20-0AA1
Dimensions W x H x D (mm)	90 x 100 x 81	90 x 100 x 81	90 x 100 x 81
Weight	320 grams	367.3 grams	363 grams
Power dissipation	20 W	14 W	6 W
Current available (EM bus)	1400 mA max. (5 V DC)	1400 mA max. (5 V DC)	Not available
Current available (24 V DC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 V DC)	4 mA/input used	4 mA/input used	4 mA/ input used

Table A- 9 CPU features

Technical data		CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
User memory[1]	Program	12 Kbytes	12 Kbytes	12 Kbytes
	User data (V)	8 Kbytes	8 Kbytes	8 Kbytes
	Retentive	10 Kbytes max.[1]	10 Kbytes max.[1]	2 Kbytes max.[1]
On-board digital I/O		12 inputs/8 outputs	12 inputs/8 outputs	12 inputs/8 outputs
Process image		256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)
Analog image		56 words of inputs (AI) / 56 words of outputs (AQ)	56 words of inputs (AI) / 56 words of outputs (AQ)	Not available
Bit memory (M)		256 bits	256 bits	256 bits
Temporary (local) memory (L)		64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)
Sequential control relays (S)		256 bits	256 bits	256 bits
Expansion modules expansion		6	6	Not available
Signal board expansion		1 max.	1 max.	Not available
High-speed counters	Total	6	6	4
	Single phase	4 at 200 kHz 2 at 30 kHz	4 at 200 kHz 2 at 30 kHz	4 at 100 kHz
	A/B phase	2 at 100 Khz 2 at 20 kHz	2 at 100 Khz 2 at 20 kHz	2 at 50 kHz
Pulse outputs [2]		2 at 100 kHz[2]	2 at 100 kHz[2]	Not available
Pulse catch inputs		12	12	Not available
Cyclic interrupts		2 at 1 ms resolution	2 at 1 ms resolution	2 at 1 ms resolution
Edge interrupts		4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling
Memory card		microSDHC Card (optional)	microSDHC Card (optional)	Not available
Real time clock accuracy		+/- 120 seconds/month	+/- 120 seconds/month	Not available
Real time clock retention time		7 days typ./6 days min. at 25 °C (maintenance-free Super Capacitor)	7 days typ./6 days min. at 25 °C (maintenance-free Super Capacitor)	Not available

1 You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive times) to be retentive, up to the specified maximum amount.

2 The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 10 PROFINET features

Description	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay
Support Controller function for PROFINET communication	Yes	
Support I-Device function for PROFINET communication	Yes	
Maximum number of PROFINET device	8	
Device number of PROFINET device	1 to 8	
Maximum input size of each PROFINET device	128 Bytes	
Maximum output size of each PROFINET device	128 Bytes	
Maximum input size of PROFINET I-Device	128 Bytes	
Maximum output size of PROFINET I-Device	128 Bytes	
Maximum number of modules	64	
Minimum cyclic update time of PROFINET device	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	
CPU address range of PROFINET process image input register	I128.0 to I1279.7	
CPU address range of PROFINET process image output register	Q128.0 to Q1279.7	
CPU address of PROFINET process image input and output register for device #1	I128.0 to I255.7 Q128.0 to Q255.7	
CPU address of PROFINET process image input and output register for device #2	I256.0 to I383.7 Q256.0 to Q383.7	
CPU address of PROFINET process image input and output register for device #3	I384.0 to I511.7 Q384.0 to Q511.7	
CPU address of PROFINET process image input and output register for device #4	I512.0 to I639.7 Q512.0 to Q639.7	
CPU address of PROFINET process image input and output register for device #5	I640.0 to I767.7 Q640.0 to Q767.7	
CPU address of PROFINET process image input and output register for device #6	I768.0 to I895.7 Q768.0 to Q895.7	
CPU address of PROFINET process image input and output register for device #7	I896.0 to I1023.7 Q896.0 to Q1023.7	
CPU address of PROFINET process image input and output register for device #8	I1024.0 to I1151.7 Q1024.0 to Q1151.7	
CPU address of PROFINET process image input and output register for I-Device	I1152.0 to I1279.7 Q1152.0 to Q1279.7	

Table A- 11 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 12 User program elements supported

Element		Description
POUs	Type/quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4
Timers	Type/quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
Counters	Quantity	256

Table A- 13 Communication

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Number of ports	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 0 Serial ports: 1 (RS485) Add-on serial ports: 0
HMI device	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): Not available Serial ports: 4 connections per port
Programming device (PG)	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): Not available Serial ports: 1 connection
CPUs (PUT/GET)	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): Not available
Open user communication	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): Not available
Data rates	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): Not available RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Not available RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Cable type	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): Not available RS485: PROFIBUS network cable
PROFINET Communication			
PROFINET controller	Yes	Yes	No
PROFINET device	Yes	Yes	No
PROFINET controller			
Services			
PG/OP communication	Yes	Yes	No
S7 routing	Yes	Yes	No
Isochronous mode	No	No	No
Open IE communication	Yes	Yes	No
IRT	No	No	No
MRP	No	No	No
PROFIenergy	No	No	No
Max. number of PROFINET devices that you can connect for RT	8	8	--
Max. number of module	64	64	--
Update times	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of IO devices, and the quantity of configured user data.	No
With RT			
Send clock of 1 ms	1 ms to 512 ms	1 ms to 512 ms	--

Table A- 14 Power supply

Technical data		CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Voltage range		20.4 to 28.8 V DC	85 to 264 V AC	85 to 264 V AC
Line frequency		--	47 to 63 Hz	47 to 63 Hz
Input current	CPU only at max. load	160 mA at 24 V DC (without driving 300 mA sensor power) 430 mA at 24 V DC (with driving 300 mA sensor power)	210 mA at 120 V AC (with 300 mA power sensor output) 90 mA at 120 V AC (without 300 mA power sensor output) 120 mA at 240 V AC (with 300 mA power sensor output) 60 mA at 240 V AC (without 300 mA power sensor output)	90 mA at 120 V AC 60 mA at 240 V AC
	CPU with all expansion accessories at max. load	720 mA at 24 V DC	290 mA at 120 V AC 170 mA at 240 V AC	--

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Hold up time (loss of power)	20 ms at 24 V DC	30 ms at 120 V AC 200 ms at 240 V AC	30 ms at 120 V AC 200 ms at 240 V AC
Inrush current (max.)	11.7 A at 28.8 V DC	9.3 A at 264 V AC	16.3 A at 264 V AC
Isolation (input power to logic)	--	1500 V AC	1500 V AC
Ground leakage, AC line to functional earth	--	0.5 mA max.	0.5 mA max.
Internal fuse, not user replaceable	3 A, 250 V slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow

Table A- 15 Sensor power

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Voltage range	20.4 to 28.8 V DC	20.4 to 28.8 V DC	20.4 to 28.8 V DC
Output current rating (max.)	300 mA (short circuit protected)	300 mA (short circuit protected)	300 mA (short circuit protected)
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	< 1 V peak to peak	< 1 V peak to peak
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated

A.2.1.2 Digital inputs and outputs

Table A- 16 Digital inputs

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Number of inputs	12	12	12
Type	SinkSource (IEC Type 1 sink, except I0.0 to I0.3, I0.6 to I0.7)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal
Continuous permissible voltage	30 V DC, max.	30 V DC, max.	30 V DC, max.
Surge voltage	35 V DC for 0.5 sec	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.
Logic 1 signal (min.)	I0.0 to I0.3, I0.6 to I0.7: 4 V DC at 8 mA Other inputs: 15 V DC at 2.5 mA	15 V DC at 2.5 mA	15 V DC at 2.5 mA
Logic 0 signal (max.)	I0.0 to I0.3, I0.6 to I0.7: 1 V DC at 1 mA Other inputs: 5 V DC at 1 mA	5 V DC at 1 mA	5 V DC at 1 mA
Isolation (field side to logic)	500 V AC for 1 minute	500 V AC for 1 minute	500 V AC for 1 minute
Isolation groups	1	1	1

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Filter times	Individually selectable on each channel (points I0.0 to I1.3): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.3): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.3): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8
HSC clock input rates (max.)	Single phase: 4 HSCs at 200 kHz; 2 HSCs at 30 kHz A/B phase: 2 HSCs at 100 kHz; 2 HSCs at 20 kHz	Single phase: 4 HSCs at 200 kHz; 2 HSCs at 30 kHz A/B phase: 2 HSCs at 100 kHz; 2 HSCs at 20 kHz	Single phase: 4 HSCs at 100 kHz A/B phase: 2 HSCs at 50 kHz
Number of inputs on simultaneously	12	12	12
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): • 500 m normal (low-speed) inputs • 50 m HSC (high-speed) inputs	All inputs: Shielded: • 500 m normal inputs • 50 m HSC inputs Unshielded: • 300 m normal inputs	All inputs: Shielded: • 500 m normal inputs, • 50 m HSC inputs Unshielded: • 300 m normal inputs
	I0.6 to I0.7: • Shielded (only): 500 m normal inputs		
	All other inputs: • Shielded: 500 m normal inputs • Unshielded: 300 m normal inputs		

Table A- 17 Digital outputs

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
Number of outputs	8	8	8
Type	Solid state - MOSFET (sourcing)	Relay, dry contact	Relay, dry contact
Voltage range	20.4 to 28.8 V DC	5 to 30 V DC or 5 to 250 V AC	5 to 30 V DC or 5 to 250 V AC
Logic 1 signal at max. current	20 V DC min.	--	--
Logic 0 signal with 10 KΩ load	0.1 V DC max.	--	--
Rated current per point (max.)	0.5 A	2.0 A	2.0 A
Rated current per common (max.)	6 A	10.0 A	10.0 A
Lamp load	5 W	30 W DC/200 W AC	30 W DC/200 W AC

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay	CPU CR20s AC/DC/Relay
ON state resistance	0.6 Ω max.	0.2 Ω max. when new	0.2 Ω max. when new
Leakage current per point	10 µA max.	--	--
Surge current	8 A for 100 ms max.	7 A with contacts closed	7 A with contacts closed
Overload protection	No	No	No
Isolation (field side to logic)	500 V AC for 1 minute	1500 V AC for 1 minute (coil to contact) None (coil to logic)	1500 V AC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 M Ω min. when new	100 M Ω min. when new
Isolation between open contacts	--	750 V AC for 1 minute	750 V AC for 1 minute
Isolation groups	2	1	1
Inductive clamp voltage	L+ minus 48 V DC, 1 W dissipation	Not recommended	Not recommended
Switching delay (Qa.0 to Qa.3)	1.0 µs max., off to on 3.0 µs max., on to off	10 ms max.	10 ms max.
Switching delay (Qa.4 to Qa.7)	50 µs max., off to on 200 µs max., on to off	10 ms max.	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles	100,000 open/close cycles
Output state in STOP mode	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8	8	8
Cable length (max.), in meters	Shielded: 500 m Unshielded: 300 m	Shielded: 500 m Unshielded: 300 m	Shielded: 500 m Unshielded: 300 m

A.2.1.3 Wiring diagrams

Table A- 18 Wiring diagram for the CPU ST20 DC/DC/DC (6ES7288-1ST20-0AA0)

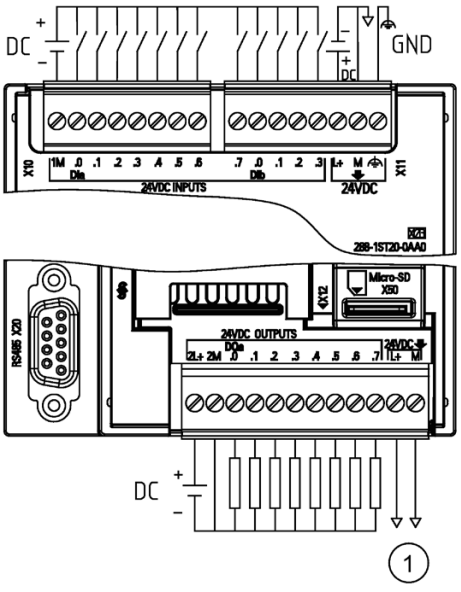
CPU ST20 DC/DC/DC (6ES7288-1ST20-0AA0)		
	①	24 V DC Sensor Power Out

Table A- 19 Connector pin locations for CPU ST20 DC/DC/DC (6ES7288-1ST20-0AA0)

Pin	X10	X11	X12
1	1M	DI a.7	2L+
2	DI a.0	DI b.0	2M
3	DI a.1	DI b.1	DQ a.0
4	DI a.2	DI b.2	DQ a.1
5	DI a.3	DI b.3	DQ a.2
6	DI a.4	L+ / 24 V DC	DQ a.3
7	DI a.5	M / 24 V DC	DQ a.4
8	DI a.6	Functional Earth	DQ a.5
9	--	--	DQ a.6
10	--	--	DQ a.7
11	--	--	L+ / 24 V DC
12	--	--	M/ 24 V DC

Table A- 20 Wiring diagram for the CPU SR20 AC/DC/Relay (6ES7288-1SR20-0AA0)

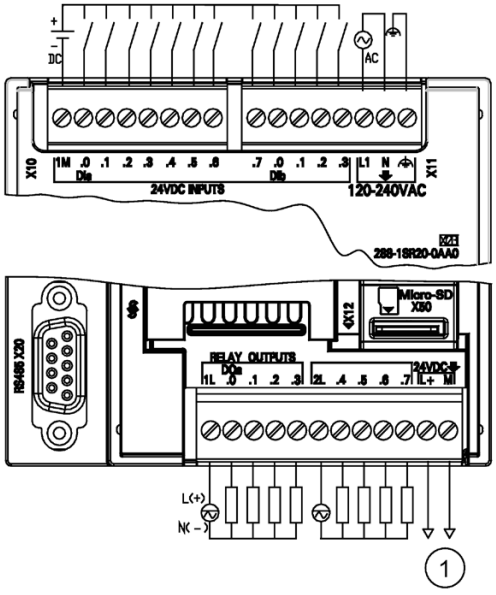
CPU SR20 AC/DC/Relay (6ES7288-1SR20-0AA0)		
	①	24 V DC sensor power output

Table A- 21 Connector pin locations for CPU SR20 AC/DC/Relay (6ES7288-1SR20-0AA0)

Pin	X10	X11	X12
1	1M	DI a.7	1L
2	DI a.0	DI b.0	DQ a.0
3	DI a.1	DI b.1	DQ a.1
4	DI a.2	DI b.2	DQ a.2
5	DI a.3	DI b.3	DQ a.3
6	DI a.4	L1 / 120 - 240 V AC	2L
7	DI a.5	N / 120 - 240 V AC	DQ a.4
8	DI a.6	Functional Earth	DQ a.5
9	--	--	DQ a.6
10	--	--	DQ a.7
11	--	--	L+ / 24 V DC Out
12	--	--	M / 24 V DC Out

Table A- 22 Wiring diagram for the CPU CR20s AC/DC/Relay (6ES7288-1CR20-0AA1)

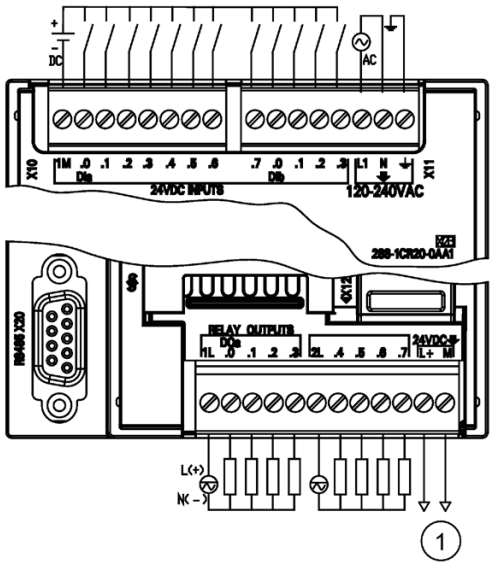
CPU CR20s AC/DC/Relay (6ES7288-1CR20-0AA1)		
	①	24 V DC sensor power output

Table A- 23 Connector pin locations for CPU CR20s AC/DC/Relay (6ES7288-1CR20-0AA1)

Pin	X10	X11	X12
1	1M	DI a.7	1L
2	DI a.0	DI b.0	DQ a.0
3	DI a.1	DI b.1	DQ a.1
4	DI a.2	DI b.2	DQ a.2
5	DI a.3	DI b.3	DQ a.3
6	DI a.4	L1 / 120 - 240 V AC	2L
7	DI a.5	N / 120 - 240 V AC	DQ a.4
8	DI a.6	Functional Earth	DQ a.5
9	--	--	DQ a.6
10	--	--	DQ a.7
11	--	--	L+ / 24 V DC Out
12	--	--	M / 24 V DC Out

A.2.2 CPU ST30, CPU SR30, and CPU CR30s

A.2.2.1 General specifications and features

General specifications and features of the CPU ST30, CPU SR30 and CPU CR30s

Table A- 24 General specifications

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
Article number	6ES7288-1ST30-0AA0	6ES7288-1SR30-0AA0	6E88288-1CR30-0AA1
Dimensions W x H x D (mm)	110 x 100 x 81	110 x 100 x 81	110 x 100 x 81
Weight	375 g	435 g	424 g
Power dissipation	12 W	14 W	7 w
Current available (EM bus)	1400 mA max. (5 V DC)	1400 mA max. (5 V DC)	Not available
Current available (24 V DC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 V DC)	4 mA/ input used	4 mA/ input used	4 mA/ input used

Table A- 25 CPU features

Technical data		CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
User memory[1]	Program	18 Kbytes	18 Kbytes	12 Kbytes
	User data (V)	12 Kbytes	12 Kbytes	8 Kbytes
	Retentive	10 Kbytes max.[1]	10 Kbytes max.[1]	2 Kbytes max.[1]
On-board digital I/O		18 inputs/12 outputs	18 inputs/12 outputs	18 inputs/12 outputs
Process image		256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)
Analog image		56 words of inputs (AI) / 56 words of outputs (AQ)	56 words of inputs (AI) / 56 words of outputs (AQ)	Not available
Bit memory (M)		256 bits	256 bits	256 bits
Temporary (local) memory (L)		64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)
Sequential control relays (S)		256 bits	256 bits	256 bits
Expansion modules expansion		6	6	Not available
Signal board expansion		1 max.	1 max.	Not available

Technical data		CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
High-speed counters	Total	6	6	4
	Single phase	5 at 200 kHz 1 at 30 kHz	5 at 200 kHz 1 at 30 kHz	4 at 100 kHz
	A/B phase	3 at 100 kHz 1 at 20 kHz	3 at 100 kHz 1 at 20 kHz	2 at 50 kHz
Pulse outputs [2]		3 at 100 kHz	--	Not available
Pulse catch inputs		12	12	Not available
Cyclic interrupts		2 at 1 ms resolution	2 at 1 ms resolution	2 at 1 ms resolution
Edge interrupts		4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling
Memory card		microSDHC Card (optional)	microSDHC Card (optional)	Not available
Real time clock accuracy		+/- 120 seconds/month	+/- 120 seconds/month	Not available
Real time clock retention time		7 days typ./6 days min. at 25 °C (maintenance-free Super Capacitor)	7 days typ./6 days min. at 25 °C (maintenance-free Super Capacitor)	Not available

[1] You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive times) to be retentive, up to the specified maximum amount.

[2] The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 26 PROFINET features

Description	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
Support Controller function for PROFINET communication	Yes	
Support I-Device function for PROFINET communication	Yes	
Maximum number of PROFINET device	8	
Device number of PROFINET device	1 to 8	
Maximum input size of each PROFINET device	128 Bytes	
Maximum output size of each PROFINET device	128 Bytes	
Maximum input size of PROFINET I-Device	128 Bytes	
Maximum output size of PROFINET I-Device	128 Bytes	
Maximum number of modules	64	
Minimum cyclic update time of PROFINET device	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	
CPU address range of PROFINET process image input register	I128.0 to I1279.7	
CPU address range of PROFINET process image output register	Q128.0 to Q1279.7	
CPU address of PROFINET process image input and output register for device #1	I128.0 to I255.7 Q128.0 to Q255.7	

Description	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
CPU address of PROFINET process image input and output register for device #2	I256.0 to I383.7 Q256.0 to Q383.7	
CPU address of PROFINET process image input and output register for device #3	I384.0 to I511.7 Q384.0 to Q511.7	
CPU address of PROFINET process image input and output register for device #4	I512.0 to I639.7 Q512.0 to Q639.7	
CPU address of PROFINET process image input and output register for device #5	I640.0 to I767.7 Q640.0 to Q767.7	
CPU address of PROFINET process image input and output register for device #6	I768.0 to I895.7 Q768.0 to Q895.7	
CPU address of PROFINET process image input and output register for device #7	I896.0 to I1023.7 Q896.0 to Q1023.7	
CPU address of PROFINET process image input and output register for device #8	I1024.0 to I1151.7 Q1024.0 to Q1151.7	
CPU address of PROFINET process image input and output register for I-Device	I1152.0 to I1279.7 Q1152.0 to Q1279.7	

Table A- 27 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 28 User program elements supported

Element		Description
POUs	Type/quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4
Timers	Type/quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
Counters	Quantity	256

Table A- 29 Communication

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
Number of ports	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 0 Serial ports: 1 (RS485) Add-on serial ports: 0
HMI device	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): Not available Serial ports: 4 connections per port
Programming device (PG)	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): Not available Serial ports: 1 connection
CPUs (PUT/GET)	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): Not available
Open user communication	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): Not available
Data rates	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): Not available RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Not available RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;
Cable type	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): Not available RS485: PROFIBUS network cable
PROFINET Communication			
PROFINET controller	Yes	Yes	No
PROFINET device	Yes	Yes	No
PROFINET controller			
Services			
PG/OP communication	Yes	Yes	No
S7 routing	Yes	Yes	No
Isochronous mode	No	No	No
Open IE communication	Yes	Yes	No
IRT	No	No	No

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
MRP	No	No	No
PROFIenergy	No	No	No
Max. number of PROFINET devices that you can connect for RT	8	8	--
Max. number of module	64	64	--
Update times	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	No
With RT			
Send clock of 1 ms	1 ms to 512 ms	1 ms to 512 ms	--

Table A- 30 Power supply

Technical data		CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
Voltage range		20.4 to 28.8 V DC	85 to 264 V AC	85 to 264 V AC
Line frequency		--	47 to 63 Hz	47 to 63 Hz
Input current	CPU only at max. load	64 mA at 24 V DC (without driving 300 mA sensor power) 365 mA at 24 V DC (with driving 300 mA sensor power)	92 mA at 120 V AC (with power sensor) 40 mA at 120 V AC (without power sensor) 52 mA at 240 V AC (with power sensor) 27 mA at 240 V AC (without power sensor)	90 mA at 120 V AC 70 mA at 240 V AC
	CPU with all expansion accessories at max. load	624 mA at 24 V DC	136 mA at 120 V AC 72 mA at 240 V AC	--
Hold up time (loss of power)		20 ms at 24 V DC	30 ms at 120 V AC 200 ms at 240 V AC	30 ms at 120 V AC 200 ms at 240 V AC
Inrush current (max.)		6A at 28.8 V DC	8.9 A at 264 V AC	16.3 A at 264 V AC
Isolation (input power to logic)		--	1500 V AC	1500 V AC
Ground leakage, AC line to functional earth		--	0.5 mA max.	0.5 mA max.
Internal fuse, not user replaceable		3 A, 250 V, slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow

Table A- 31 Sensor power

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
Voltage range	20.4 to 28.8 V DC	20.4 to 28.8 V DC	20.4 to 28.8 V DC
Output current rating (max.)	300 mA (short circuit protected)	300 mA (short circuit protected)	300 mA (short circuit protect-ed)
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	< 1 V peak to peak	< 1 V peak to peak
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated

A.2.2.2 Digital inputs and outputs

Table A- 32 Digital inputs

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
Number of inputs	18	18	18
Type	Sink/Source (IEC Type 1 sink, except I0.0 to I0.3, I0.6 to I0.7)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal
Continuous permissible voltage	30 V DC, max	30 V DC, max.	30 V DC, max.
Surge voltage	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.
Logic 1 signal (min.)	I0.0 to I0.3, I0.6 to I0.7: 4 V DC at 8 mA Other inputs: 15 V DC at 2.5 mA	15 V DC at 2.5 mA	15 V DC at 2.5 mA
Logic 0 signal (max.)	I0.0 to I0.3, I0.6 to I0.7: 1 V DC at 1 mA Other inputs: 5 V DC at 1 mA	5 V DC at 1 mA	5 V DC at 1 mA
Isolation (field side to logic)	500 V AC for 1 minute	500 V AC for 1 minute	500 V AC for 1 minute
Isolation groups	1	1	1
Filter times	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8
	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 V DC)	Single phase: 5 HSCs at 200 kHz; 1 HSC at 30 kHz A/B phase: 3 HSCs at 100 kHz; 1 HSC at 20 kHz	Single phase: 5 HSCs at 200 kHz; 1 HSCs at 30 kHz A/B phase: 3 HSCs at 100 kHz; 1 HSC at 20 kHz	Single phase: 4 HSCs at 100 kHz A/B phase: 2 HSCs at 50 kHz
Number of inputs on simultaneously	18	18	18
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): • 500 m normal (low-speed) inputs • 50 m HSC (high-speed) inputs	All inputs: • Shielded: 500 m normal inputs, 50 m HSC inputs • Unshielded: 300 m normal inputs	All inputs: • Shielded: 500 m normal inputs, 50 m HSC inputs • Unshielded: 300 m normal inputs
	I0.6 to I0.7: • Shielded (only): 500 m normal inputs		
	All other inputs: • Shielded: 500 m normal inputs • Unshielded: 300 m normal inputs[1]		

[1] When I0.0 to I0.3 are used at high-speed counter inputs, all other inputs must use shielded cable.

Table A- 33 Digital outputs

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
Number of outputs	12	12	12
Type	Solid state - MOSFET (sourcing)	Relay, dry contact	Relay, dry contact
Voltage range	20.4 to 28.8 V DC	5 to 30 V DC or 5 to 250 V AC	5 to 30 V DC or 5 to 250 V AC
Logic 1 signal at max. current	20 V DC min.	--	--
Logic 0 signal with 10 KΩ load	0.1 V DC max.	--	--
Rated current per point (max.)	0.5 A	2.0 A	2.0 A
Rated current per common (max.)	6 A	10.0 A	10.0 A
Lamp load	5 W	30 W DC/200 W AC	30 W DC/200 W AC
ON state resistance	0.6 Ω max.	0.2 Ω max. when new	0.2 Ω max. when new
Leakage current per point	10 µA max.	--	--
Surge current	8 A for 100 ms max.	7 A with contacts closed	7 A with contacts closed
Overload protection	No	No	No

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay	CPU CR30s AC/DC/Relay
Isolation (field side to logic)	500 V AC for 1 minute	1500 V AC for 1 minute (coil to contact) None (coil to logic)	1500 V AC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 M Ω min. when new	100 M Ω min. when new
Isolation between open contacts	--	750 V AC for 1 minute	750 V AC for 1 minute
Isolation groups	1	1	1
Inductive clamp voltage	L+ minus 48 V DC, 1 W dissipation	Not recommended	Not recommended
Switching delay (Qa.0 to Qa.3)	1.0 µs max., off to on 3.0 µs max., on to off	10 ms max.	10 ms max.
Switching delay (Qa.4 to Qb.7)	50 µs max., off to on, 200 µs max., on to off	10 ms max.	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles	100,000 open/close cycles
Output state in STOP mode	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	12	12	12
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

A.2.2.3 Wiring diagrams

Table A- 34 Wiring diagram for the CPU ST30 DC/DC/DC (6ES7288-1ST30-0AA0)

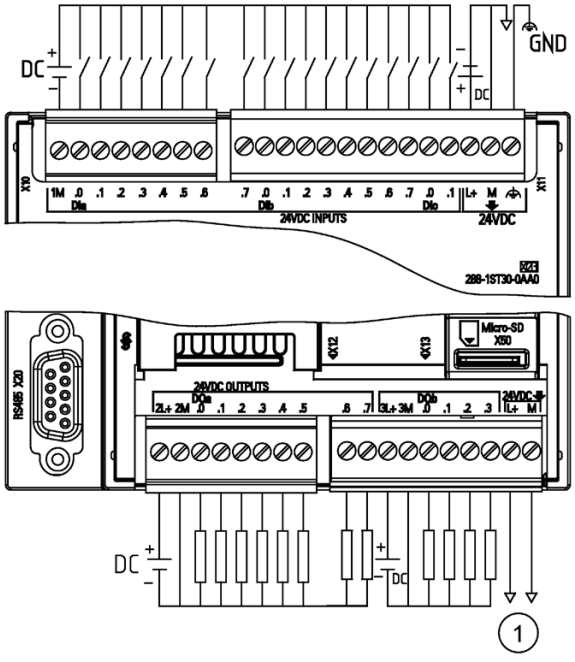
CPU ST30 DC/DC/DC (6ES7288-1ST30-0AA0)		
	①	24 V DC Sensor Power Out

Table A- 35 Connector pin locations for CPU ST30 DC/DC/DC (6ES7288-1ST30-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	2L+	DQ a.6
2	DI a.0	DI b.0	2M	DQ a.7
3	DI a.1	DI b.1	DQ a.0	3L+
4	DI a.2	DI b.2	DQ a.1	3M
5	DI a.3	DI b.3	DQ a.2	DQb.0
6	DI a.4	DIb.4	DQa.3	DQb.1
7	DI a.5	DIb.5	DQ a.4	DQb.2
8	DI a.6	DIb.6	DQ a.5	DQb.3
9	--	DIb.7	--	L+ / 24 V DC
10	--	DIc.0	--	M / 24 V DC
11	--	DIc.1	--	--
12	--	L+24 V DC	--	--
13	--	M / 24 V DC	--	--
14	--	Functional Earth	--	--

Table A- 36 Wiring diagram for the CPU SR30 AC/DC/Relay (6ES7288-1SR30-0AA0)

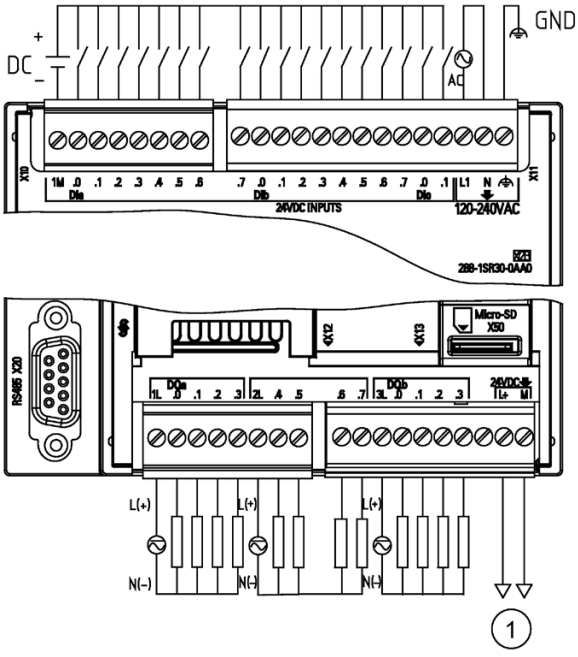
CPU SR30 AC/DC/Relay (6ES7288-1SR30-0AA0)		
	①	24 V DC Sensor Power Out

Table A- 37 Connector pin locations for CPU SR30 AC/DC/Relay (6ES7288-1SR30-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	1L	DQ a.6
2	DI a.0	DI b.0	DQ a.0	DQ a.7
3	DI a.1	DI b.1	DQ a.1	3L
4	DI a.2	DI b.2	DQ a.2	DQ b.0
5	DI a.3	DI b.3	DQ a.3	DQ b.1
6	DI a.4	DIb.4	2L	DQ b.2
7	DI a.5	DIb.5	DQ a.4	DQ b.3
8	DI a.6	DIb.6	DQ a.5	--
9	--	DIb.7	--	L+ / 24 V DC Out
10	--	DIc.0	--	M / 24 V DC Out
11	--	DIc.1	--	--
12	--	L1 / 120 - 240 V AC	--	--
13	--	N / 120 - 240 V AC	--	--
14	--	Functional Earth	--	--

Table A- 38 Wiring diagram for the CPU CR30s AC/DC/Relay (6ES7288-1CR30-0AA1)

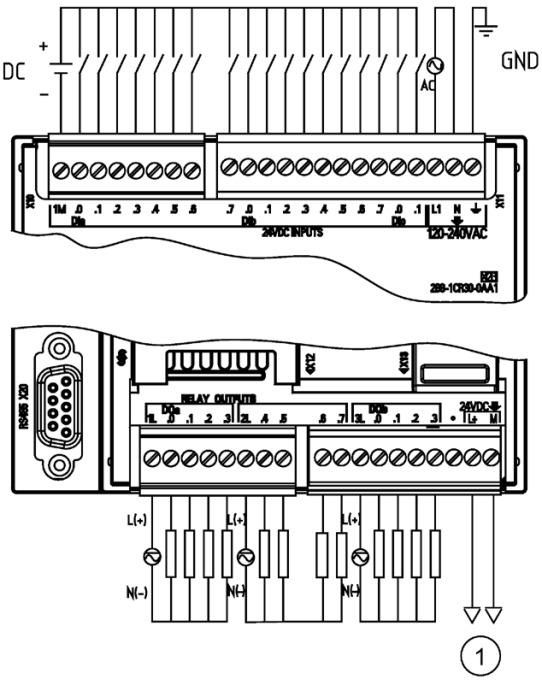
CPU CR30s AC/DC/Relay (6ES7288-1CR30-0AA1)		
	①	24 V DC Sensor Power Out

Table A- 39 Connector pin locations for CPU CR30s AC/DC/Relay (6ES7288-1CR30-0AA1)

Pin	X10	X11	X12	X13
1	1M	DI a.7	1L	DQ a.6
2	DI a.0	DI b.0	DQ a.0	DQ a.7
3	DI a.1	DI b.1	DQ a.1	3L
4	DI a.2	DI b.2	DQ a.2	DQ b.0
5	DI a.3	DI b.3	DQ a.3	DQ b.1
6	DI a.4	DIb.4	2L	DQ b.2
7	DI a.5	DIb.5	DQ a.4	DQ b.3
8	DI a.6	DIb.6	DQ a.5	--
9	--	DIb.7	--	L+ / 24 V DC Out
10	--	DIc.0	--	M / 24 V DC Out
11	--	DIc.1	--	--
12	--	L1 / 120 - 240 V AC	--	--
13	--	N / 120 - 240 V AC	--	--
14	--	Functional Earth	--	--

A.2.3 CPU ST40, CPU SR40, CPU CR40s, and CPU CR40

A.2.3.1 General specifications and features

General specifications and features of the CPU ST40, CPU SR40, and CPU CR40s

Table A- 40 General specifications

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Article number	6ES7288-1ST40-0AA0	6ES7288-1SR40-0AA0	6ES7288-1CR40-0AA1	6ES7288-1CR40-0AA0
Dimensions W x H x D (mm)	125 x 100 x 81	125 x 100 x 81	125 x 100 x 81	125 x 100 x 81
Weight	410.3 grams	441.3 grams	475 grams	486.2 grams
Power dissipation	18 W	23 W	8 W	8 W
Current available (EM bus)	1400 mA max. (5 V DC)	1400 mA max. (5 V DC)	Not available	Not available
Current available (24 V DC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 V DC)	4 mA/input used	4 mA/input used	4 mA/input used	4 mA/input used

Table A- 41 CPU features

Technical data		CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
User memory	Program	24 Kbytes	24 Kbytes	12 Kbytes	
	User data (V)	16 Kbytes	16 Kbytes	8 Kbytes	
	Retentive	10 Kbytes max.[1]	10 Kbytes max.[1]	2 Kbytes max.[1]	
On-board digital I/O		24 inputs/16 outputs	24 inputs/16 outputs	24 inputs/16 outputs	
Process image		256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)	
Analog image		56 words of inputs (AI) / 56 words of outputs (AQ)	56 words of inputs (AI) / 56 words of outputs (AQ)	Not available	
Bit memory (M)		256 bits	256 bits	256 bits	
Temporary (local) memory (L)		64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	

Technical data		CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Sequential control relays (S)		256 bits	256 bits	256 bits	
Expansion modules expansion		6 max.	6 max.	Not available	
Signal board expansion		1 max.	1 max.	Not available	
High-speed counters	Total	6	6	4	
	Single phase	4 at 200 kHz 2 at 30 kHz	4 at 200 kHz 2 at 30 kHz	4 at 100 kHz	
	A/B phase	2 at 100 kHz 2 at 20 kHz	2 at 100 kHz 2 at 20 kHz	2 at 50 kHz	
Pulse outputs [2]		3 at 100 kHz	3 at 100 kHz	Not available	
Pulse catch inputs		14	14	Not available	
Cyclic interrupts		2 at 1 ms resolution	2 at 1 ms resolution	2 at 1 ms resolution	
Edge interrupts		4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling	
Memory card		microSDHC Card (optional)	microSDHC Card (optional)	Not available	
Real time clock accuracy		120 seconds/month	120 seconds/month	Not available	
Real time clock retention time		7 days typ./6 days min. at 25 °C	7 days typ./6 days min. at 25 °C	Not available	

1 You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values) on retentive, up to the specified maximum amount.

2 The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 42 PROFINET features

Description	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay
Support Controller function for PROFINET communication	Yes	
Support I-Device function for PROFINET communication	Yes	
Maximum number of PROFINET device	8	
Device number of PROFINET device	1 to 8	
Maximum input size of each PROFINET device	128 Bytes	
Maximum output size of each PROFINET device	128 Bytes	
Maximum input size of PROFINET I-Device	128 Bytes	
Maximum output size of PROFINET I-Device	128 Bytes	
Maximum number of modules	64	
Minimum cyclic update time of PROFINET device	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	

Description	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay
CPU address range of PROFINET process image input register	I128.0 to I1279.7	
CPU address range of PROFINET process image output register	Q128.0 to Q1279.7	
CPU address of PROFINET process image input and output register for device #1	I128.0 to I255.7 Q128.0 to Q255.7	
CPU address of PROFINET process image input and output register for device #2	I256.0 to I383.7 Q256.0 to Q383.7	
CPU address of PROFINET process image input and output register for device #3	I384.0 to I511.7 Q384.0 to Q511.7	
CPU address of PROFINET process image input and output register for device #4	I512.0 to I639.7 Q512.0 to Q639.7	
CPU address of PROFINET process image input and output register for device #5	I640.0 to I767.7 Q640.0 to Q767.7	
CPU address of PROFINET process image input and output register for device #6	I768.0 to I895.7 Q768.0 to Q895.7	
CPU address of PROFINET process image input and output register for device #7	I896.0 to I1023.7 Q896.0 to Q1023.7	
CPU address of PROFINET process image input and output register for device #8	I1024.0 to I1151.7 Q1024.0 to Q1151.7	
CPU address of PROFINET process image input and output register for I-Device	I1152.0 to I1279.7 Q1152.0 to Q1279.7	

Table A- 43 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 44 User program elements supported

Element		Description
POUs	Type/quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4
Timers	Type/quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
Counters	Quantity	256

Table A- 45 Communication

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Number of ports	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 0 Serial ports: 1 (RS485) Add-on serial ports: 0	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 0
HMI device	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): Not available Serial ports: 4 connections per port	PROFINET (LAN): 8 connections Serial ports: 4 connections per port
Programming device (PG)	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): Not available Serial ports: 1 connection	PROFINET (LAN): 1 connection Serial ports: 1 connection
CPUs (PUT/GET)	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): Not available	PROFINET (LAN): 8 client and 8 server connections
Open user communication	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): Not available	PROFINET (LAN): 8 active and 8 passive connections
Data rates	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): Not available RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Not available RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;
Cable type	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): Not available RS485: PROFIBUS network cable	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable
PROFINET Communication				
PROFINET controller	Yes	Yes	No	No
PROFINET device	Yes	Yes	No	No

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
PROFINET controller				
Services				
PG/OP communication	Yes	Yes	No	No
S7 routing	Yes	Yes	No	No
Isochronous mode	No	No	No	No
Open IE communication	Yes	Yes	No	No
IRT	No	No	No	No
MRP	No	No	No	No
PROFIenergy	No	No	No	No
Max. number of PROFINET devices that you can connect for RT	8	8	--	--
Max. number of module	64	64	--	--
Update times	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	No	No
With RT				
Send clock of 1 ms	1 ms to 512 ms	1 ms to 512 ms	--	--

Table A- 46 Power supply

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Voltage range	20.4 to 28.8 V DC	85 to 264 V AC	85 to 264 V AC	
Line frequency	--	47 to 63 Hz	47 to 63 Hz	

Technical data		CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Input current (max. load)	CPU only	190 mA at 24 V DC (without driving 300 mA sensor power) 470 mA at 24 V DC (with driving 300 mA sensor power)	130 mA at 120 V AC (without driving 300 mA sensor power) 250 mA at 120 V (with driving 300 mA sensor power) 80 mA at 240 V AC (without driving 300 mA sensor power) 150 mA at 240 V AC (with driving 300 mA sensor power)	150 mA at 120 V AC 80 mA at 240 V AC	
	CPU with all expansion accessories	680 mA at 24 V DC	300 mA at 120 V AC 190 mA at 240 V AC	--	
Hold up time (loss of power)		20 ms at 24 V DC	30 ms at 120 V AC 200 ms at 240 V AC	30 ms at 120 V AC 200 ms at 240 V AC	
Inrush current (max.)		11.7 A at 28.8 V DC	16.3 A at 264 V AC	16.3 A at 264 V AC	
Isolation (input power to logic)		--	1500 V AC	1500 V AC	
Ground leakage, AC line to functional earth		--	0.5 mA	0.5 mA	
Internal fuse, not user replaceable		3 A, 250 V, slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow	

Table A- 47 Sensor power

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Voltage range	20.4 to 28.8 V DC	20.4 to 28.8 V DC	20.4 to 28.8 V DC	
Output current rating (max.)	300 mA	300 mA	300 mA (short circuit protected)	
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	< 1 V peak to peak	< 1 V peak to peak	
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated	

A.2.3.2 Digital inputs and outputs

Table A- 48 Digital inputs

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Number of inputs	24	24	24	
Type	Sink/Source (IEC Type 1 sink, except I0.0 to I0.3)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)	

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Rated voltage	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	
Continuous permissible voltage	30 V DC, max.	30 V DC, max.	30 V DC, max.	
Surge voltage	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.	
Logic 1 signal (min.)	I0.0 to I0.3: 4 V DC at 8 mA Other inputs: 15 V DC at 2.5 mA	15 V DC at 2.5 mA	15 V DC at 2.5 mA	
Logic 0 signal (max.)	I0.0 to I0.3: 1 V DC at 1 mA Other inputs: 5 V DC at 1 mA	5 V DC at 1 mA	5 V DC at 1 mA	
Isolation (field side to logic)	500 V AC for 1 minute	500 V AC for 1 minute	500 V AC for 1 minute	
Isolation group	1	1	1	
Filter times	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	
	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 V DC)	Single phase: 4 HSCs at 200 kHz; 2 HSCs at 30 kHz A/B phase: 2 HSCs at 100 kHz; 2 HSCs at 20 kHz	Single phase: 4 HSCs at 200 kHz; 2 HSCs at 30 kHz A/B phase: 2 HSCs at 100 kHz; 2 HSCs at 20 kHz	Single phase: 4 HSCs at 100 kHz A/B phase: 2 HSCs at 50 kHz	
Number of inputs on simultaneously	24	24	24	
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): • 500 m normal (low-speed) inputs, • 50 m HSC (high-speed) inputs	All inputs: Shielded: • 500 m normal inputs • 50 m HSC inputs Unshielded: • 300 m normal inputs	All inputs: Shielded: • 500 m normal inputs • 50 m HSC inputs Unshielded: • 300 m normal inputs	
	All other inputs: Shielded: • 500 m normal inputs Unshielded: • 300 m normal inputs			

Table A- 49 Digital outputs

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40s AC/DC/Relay	CPU CR40 AC/DC/Relay
Number of outputs	16	16	16	
Type	Solid state - MOSFET (sourcing)	Relay, dry contact	Relay, dry contact	
Voltage range	20.4 to 28.8 V DC	5 to 30 V DC or 5 to 250 V AC	5 to 30 V DC or 5 to 250 V AC	
Logic 1 signal at max. current	20 V DC min.	--	--	
Logic 0 signal with 10 KΩ load	0.1 V DC max.	--	--	
Rated current per point (max.)	0.5 A	2 A	2 A	
Rated current per common (max.)	6 A	10 A	10 A	
Lamp load	5 W	30 W DC / 200 W AC	30 W DC / 200 W AC	
ON state resistance	0.6 Ω max.	0.2 Ω max. when new	0.2 Ω max. when new	
Leakage current per point	10 µA max.	--	--	
Surge current	8 A for 100 ms max.	7 A with contacts closed	7 A with contacts closed	
Overload protection	No	No	No	
Isolation (field side to logic)	500 V AC for 1 minute	1500 V AC for 1 minute (coil to contact) None (coil to logic)	1500 V AC for 1 minute (coil to contact) None (coil to logic)	
Isolation resistance	--	100 MΩ min. when new	100 MΩ min. when new	
Isolation between open contact	--	750 V AC for 1 minute	750 V AC for 1 minute	
Isolation groups	2	4	4	
Inductive clamp voltage	L+ minus 48 V DC, 1 W dissipation	--	--	
Switching delay (Qa.0 to Qa.3)	1.0 µs max., off to on 3.0 µs max., on to off	10 ms max.	10 ms max.	
Switching delay (Qa.4 to Qb.7)	50 µs max., off to on 200 µs max., on to off	10 ms max.	10 ms max.	
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	10,000,000 open/close cycles	
Lifetime contacts at rated load	--	100,000 open/close cycles	100,000 open/close cycles	
Output state in STOP mode	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	
Number of outputs on simultaneously	16	16	16	
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	

A.2.3.3 Wiring diagrams

Table A- 50 Wiring diagram for the CPU ST40 DC/DC/DC (6ES7288-1ST40-0AA0)

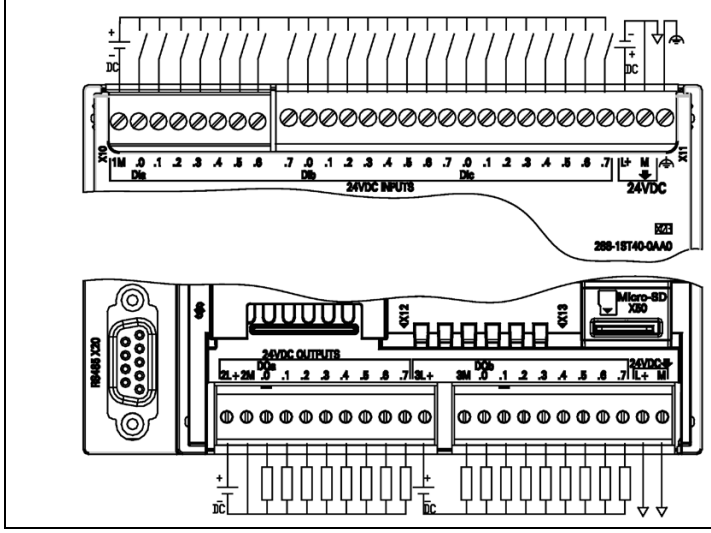
	①	24 V DC Sensor Power Out

Table A- 51 Connector pin locations for CPU ST40 DC/DC/DC (6ES7288-1ST40-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	2L+	3M
2	DI a.0	DI b.0	2M	DQ b.0
3	DI a.1	DI b.1	DQ a.0	DQ b.1
4	DI a.2	DI b.2	DQ a.1	DQ b.2
5	DI a.3	DI b.3	DQ a.2	DQ b.3
6	DI a.4	DI b.4	DQ a.3	DQ b.4
7	DI a.5	DI b.5	DQ a.4	DQ b.5
8	DI a.6	DI b.6	DQ a.5	DQ b.6
9	--	DI b.7	DQ a.6	DQ b.7
10	--	DI c.0	DQ a.7	L+ / 24 V DC Out
11	--	DI c.1	3L+	M / 24 V DC Out
12	--	DI c.2	--	--
13	--	DI c.3	--	--
14	--	DI c.4	--	--
15	--	DI c.5	--	--
16	--	DI c.6	--	--
17	--	DI c.7	--	--
18	--	L+ / 24 V DC	--	--
19	--	M / 24 V DC	--	--
20	--	Functional Earth	--	--

Table A- 52 Wiring diagram for the CPU SR40 AC/DC/Relay (6ES7288-1SR40-0AA0)

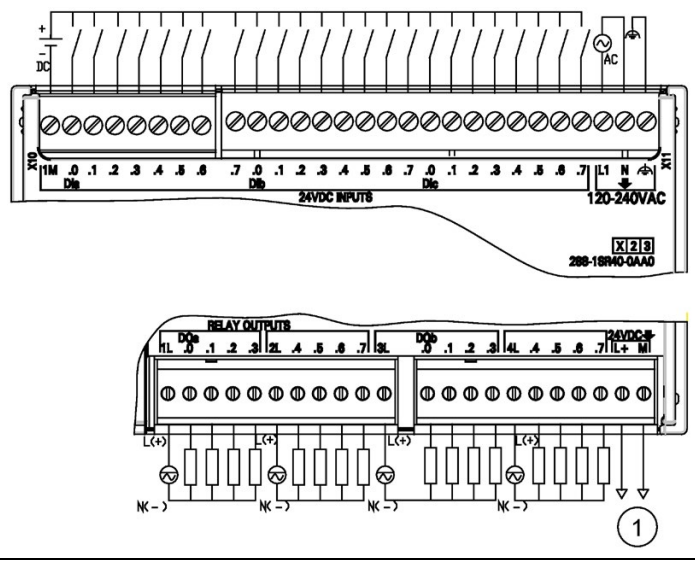
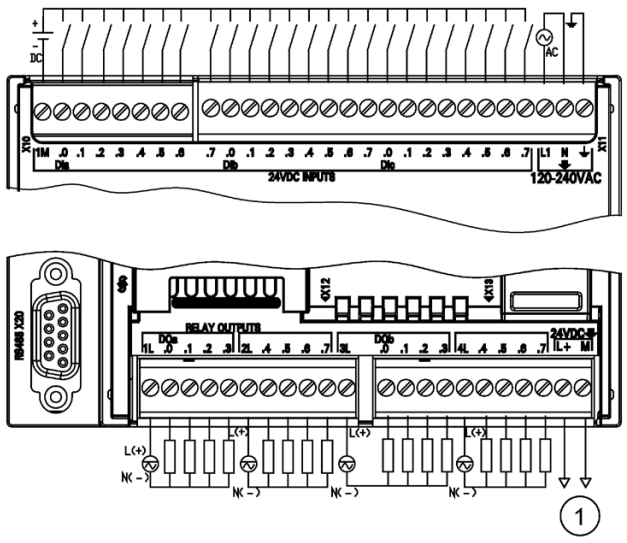
	①	24 V DC Sensor Power Out

Table A- 53 Connector pin locations for CPU SR40 AC/DC/Relay (6ES7288-1SR40-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	1L	DQ b.0
2	DI a.0	DI b.0	DQ a.0	DQ b.1
3	DI a.1	DI b.1	DQ a.1	DQ b.2
4	DI a.2	DI b.2	DQ a.2	DQ b.3
5	DI a.3	DI b.3	DQ a.3	4L
6	DI a.4	DI b.4	2L	DQ b.4
7	DI a.5	DI b.5	DQ a.4	DQ b.5
8	DI a.6	DI b.6	DQ a.5	DQ b.6
9	--	DI b.7	DQ a.6	DQ b.7
10	--	DI c.0	DQ a.7	L+ / 24 V DC Out
11	--	DI c.1	3L	M / 24 V DC Out
12	--	DI c.2	--	--
13	--	DI c.3	--	--
14	--	DI c.4	--	--
15	--	DI c.5	--	--
16	--	DI c.6	--	--
17	--	DI c.7	--	--
18	--	L1 / 120 - 240 V AC	--	--

Pin	X10	X11	X12	X13
19	--	N / 120 - 240 V AC	--	--
20	--	Functional Earth	--	--

Table A- 54 Wiring diagram for the CPU CR40s AC/DC/Relay (6ES7288-1CR40s-0AA1) and CPU CR40 AC/DC/Relay (6ES7288-1CR40s-0AA0)



① 24 V DC Sensor Power Out

Table A- 55 Connector pin locations for CPU CR40s AC/DC/Relay (6ES7288-1CR40-0AA1) and CPU CR40 AC/DC/Relay (6ES7288-1CR40s-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	1L	DQ b.0
2	DI a.0	DI b.0	DQ a.0	DQ b.1
3	DI a.1	DI b.1	DQ a.1	DQ b.2
4	DI a.2	DI b.2	DQ a.2	DQ b.3
5	DI a.3	DI b.3	DQ a.3	4L
6	DI a.4	DI b.4	2L	DQ b.4
7	DI a.5	DI b.5	DQ a.4	DQ b.5
8	DI a.6	DI b.6	DQ a.5	DQ b.6
9	--	DI b.7	DQ a.6	DQ b.7
10	--	DI c.0	DQ a.7	L+ / 24 V DC Out
11	--	DI c.1	3L+	M / 24 V DC Out
12	--	DI c.2	--	--
13	--	DI c.3	--	--
14	--	DI c.4	--	--
15	--	DI c.5	--	--

Pin	X10	X11	X12	X13
16	--	DI c.6	--	--
17	--	DI c.7	--	--
18	--	L+ / 24 V DC	--	--
19	--	M / 24 V DC	--	--
20	--	Functional Earth	--	--

A.2.4 CPU ST60, CPU SR60, CPU CR60s, and CPU CR60

A.2.4.1 General specifications and features

Table A- 56 General specifications

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Article number	6ES7288-1ST60-0AA0	6ES7288-1SR60-0AA0	6ES7288-1CR60-0AA1	6ES7288-1CR60-0AA0
Dimensions W x H x D (mm)	175 x 100 x 81	175 x 100 x 81	175 x 100 x 81	175 x 100 x 81
Weight	528.2 grams	611.5 grams	605 grams	621.9 grams
Power dissipation	20 W	25 W	10 W	10 W
Current available (EM bus)	1400 mA max. (5 V DC)	1400 mA max. (5 V DC)	Not available	Not available
Current available (24 V DC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 V DC)	4 mA/input used	4 mA/input used	4 mA/input used	4 mA/input used

Table A- 57 CPU features

Technical data		CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
User memory	Program	30 Kbytes	30 Kbytes	12 Kbytes	
	User data (V)	20 Kbytes	20 Kbytes	8 Kbytes	
	Retentive	10 Kbytes max.[1]	10 Kbytes max.[1]	2 Kbytes max.[1]	
On-board digital I/O		36 inputs/24 outputs	36 inputs/24 outputs	36 inputs/24 outputs	
Process image		256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)	
Analog image		56 words of inputs (AI) / 56 words of outputs (AQ)	56 words of inputs (AI) / 56 words of outputs (AQ)	Not available	
Bit memory (M)		256 bits	256 bits	256 bits	

Technical data		CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Temporary (local) memory (L)		64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine 60 bytes when programming in LAD or FBD (STEP 7-Micro/WIN reserves 4 bytes)	
Sequential control relays (S)		256 bits	256 bits	256 bits	
Expansion modules expansion		6 max.	6 max.	Not available	
Signal board expansion		1 max.	1 max.	Not available	
High-speed counters	Total	6	6	4	
	Single phase	4 at 200 kHz 2 at 30 kHz	4 at 200 kHz 2 at 30 kHz	4 at 100 kHz	
	A/B phase	2 at 100 kHz 2 at 20 kHz	2 at 100 kHz 2 at 20 kHz	2 at 50 kHz	
Pulse outputs [2]		3 at 100 kHz	3 at 100 kHz	Not available	
Pulse catch inputs		14	14	Not available	
Cyclic interrupts		2 at 1 ms resolution	2 at 1 ms resolution	2 at 1 ms resolution	
Edge interrupts		4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling	
Memory card		microSDHC card (optional)	microSDHC card (optional)	Not available	
Real time clock accuracy		120 seconds/month	120 seconds/month	Not available	
Real time clock retention time		7 days typ./6 days min. at 25 °C	7 days typ./6 days min. at 25 °C	Not available	

[1] You can configure areas of V memory, M memory, C memory (current values) and portions of T memory (current values on retentive timers) to be retentive, up to the specified maximum amount.

[2] The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 58 PROFINET features

Description	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay
Support Controller function for PROFINET communication	Yes	
Support I-Device function for PROFINET communication	Yes	
Maximum number of PROFINET device	8	
Device number of PROFINET device	1 to 8	
Maximum input size of each PROFINET device	128 Bytes	
Maximum output size of each PROFINET device	128 Bytes	
Maximum input size of PROFINET I-Device	128 Bytes	

Description	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay
Maximum output size of PROFINET I-Device	128 Bytes	
Maximum number of modules	64	
Minimum cyclic update time of PROFINET device	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	
CPU address range of PROFINET process image input register	I128.0 to I1279.7	
CPU address range of PROFINET process image output register	Q128.0 to Q1279.7	
CPU address of PROFINET process image input and output register for device #1	I128.0 to I255.7 Q128.0 to Q255.7	
CPU address of PROFINET process image input and output register for device #2	I256.0 to I383.7 Q256.0 to Q383.7	
CPU address of PROFINET process image input and output register for device #3	I384.0 to I511.7 Q384.0 to Q511.7	
CPU address of PROFINET process image input and output register for device #4	I512.0 to I639.7 Q512.0 to Q639.7	
CPU address of PROFINET process image input and output register for device #5	I640.0 to I767.7 Q640.0 to Q767.7	
CPU address of PROFINET process image input and output register for device #6	I768.0 to I895.7 Q768.0 to Q895.7	
CPU address of PROFINET process image input and output register for device #7	I896.0 to I1023.7 Q896.0 to Q1023.7	
CPU address of PROFINET process image input and output register for device #8	I1024.0 to I1151.7 Q1024.0 to Q1151.7	
CPU address of PROFINET process image input and output register for I-Device	I1152.0 to I1279.7 Q1152.0 to Q1279.7	

Table A- 59 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 60 User program elements supported

Element		Description
POUs	Type/quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4

Element		Description
Timers	Type/quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
Counters	Quantity	256

Table A- 61 Communication

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Number of ports	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)	PROFINET (LAN): 0 Serial ports: 1 (RS485) Add-on serial ports: 0	PROFINET (LAN): 1 Serial ports: 1 (RS485) Add-on serial ports: 0
HMI device	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): 8 connections Serial ports: 4 connections per port	PROFINET (LAN): Not available Serial ports: 4 connections per port	PROFINET (LAN): 8 connections Serial ports: 4 connections per port
Programming device (PG)	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): 1 connection Serial ports: 1 connection	PROFINET (LAN): Not available Serial ports: 1 connection	PROFINET (LAN): 1 connection Serial ports: 1 connection
CPUs (PUT/GET)	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): 8 client and 8 server connections	PROFINET (LAN): Not available	PROFINET (LAN): 8 client and 8 server connections
Open user communication	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): 8 active and 8 passive connections	PROFINET (LAN): Not available	PROFINET (LAN): 8 active and 8 passive connections
Data rates	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): Not available RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s	PROFINET (LAN): 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Not available RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;	PROFINET (LAN): Transformer isolated, 1500 V AC RS485: RS485 signal to chassis ground, 500 V AC/ 707 V DC; RS485 signal to CPU logic common, 500 V AC/ 707V DC;
Cable type	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable	PROFINET (LAN): Not available RS485: PROFIBUS network cable	PROFINET (LAN): CAT5e shielded RS485: PROFIBUS network cable

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
PROFINET Communication				
PROFINET controller	Yes	Yes	No	No
PROFINET device	Yes	Yes	No	No
PROFINET controller				
Services				
PG/OP communication	Yes	Yes	No	No
S7 routing	Yes	Yes	No	No
Isochronous mode	No	No	No	No
Open IE communication	Yes	Yes	No	No
IRT	No	No	No	No
MRP	No	No	No	No
PROFIenergy	No	No	No	No
Max. number of PROFINET devices that you can connect for RT	8	8	--	--
Max. number of module	64	64	--	--
Update times	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.	No	No
With RT				
Send clock of 1 ms	1 ms to 512 ms	1 ms to 512 ms	--	--

Table A- 62 Power supply

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Voltage range	20.4 to 28.8 V DC	85 to 264 V AC	85 to 264 V AC	
Line frequency	--	47 to 63 Hz	47 to 63 Hz	

Technical data		CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Input current (max. load)	CPU only	220 mA at 24 V DC (without driving 300 mA sensor power) 500 mA at 24 V DC (with driving 300 mA sensor power)	160 mA at 120 V AC (without driving 300 mA sensor power) 280 mA at 120 V AC (with driving 300 mA sensor power) 90 mA at 240 V AC (without driving 300 mA sensor power) 160 mA at 240 V AC (with driving 300 mA sensor power)	150 mA at 120 V AC 100 mA at 240 V AC	
	CPU with all expansion accessories	710 mA at 24 V DC	370 mA at 120 V AC 220 mA at 240 V AC	Not available	
Hold up time (loss of power)		20 ms at 24 V DC	30 ms at 120 V AC 200 ms at 240 V AC	30 ms at 120 V AC 200 ms at 240 V AC	
Inrush current (max.)		11.5 A at 28.8 V DC	16.3 A at 264 V DC	16.3 A at 264 V AC	
Isolation (input power to logic)		None	1500 V AC	1500 V AC	
Ground leakage, AC line to functional earth		None	None	None	
Internal fuse, not user replaceable		3 A, 250 V, slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow	

Table A- 63 Sensor power

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Voltage range	20.4 to 28.8 V DC	20.4 to 28.8 V DC	20.4 to 28.8 V DC	
Output current rating (max.)	300 mA	300 mA	300 mA (short circuit protected)	
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	< 1 V peak to peak	< 1 V peak to peak	
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated	

A.2.4.2 Digital inputs and outputs

Table A- 64 Digital inputs

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Number of inputs	36	36	36	
Type	Sink/Source (IEC Type 1 sink, except I0.0 to I0.3)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)	
Rated voltage	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Continuous permissible voltage	30 V DC, max.	30 V DC, max.	30 V DC, max.	
Surge voltage	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.	
Logic 1 signal (min.)	I0.0 to I0.3: 4 V DC at 8 mA Other inputs: 15 V DC at 2.5 mA	15 V DC at 2.5 mA	15 V DC at 2.5 mA	
Logic 0 signal (max.)	I0.0 to I0.3: 1 V DC at 1 mA Other inputs: 5 V DC at 1 mA	5 V DC at 1 mA	5 V DC at 1 mA	
Isolation (field side to logic)	500 V AC for 1 minute	500 V AC for 1 minute	500 V AC for 1 minute	
Isolation groups	1	1	1	
Filter times	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	
	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I1.6 and greater): ms: 0, 6.4, 12.8	
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 V DC)	Single phase: 4 HSCs at 200 kHz; 2 HSCs at 30 kHz A/B phase: 2 HSCs at 100 kHz; 2 HSCs at 20 kHz	Single phase: 4 HSCs at 200 kHz; 2 HSCs at 30 kHz A/B phase: 2 HSCs at 100 kHz; 2 HSCs at 20 kHz	Single phase: 4 HSCs at 100 kHz A/B phase: 2 HSCs at 50 kHz	
Number of inputs on simultaneously	36	36	36	
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): • 500 m normal (low-speed) inputs • 50 m HSC (high-speed) inputs	All inputs: Shielded: • 500 m normal inputs • 50 m HSC inputs Unshielded: • 300 m normal inputs	All inputs: Shielded: • 500 m normal inputs • 50 m HSC inputs Unshielded: • 300 m normal inputs	
	All other inputs: Shielded: • 500 m normal inputs Unshielded: • 300 m normal inputs			

Table A- 65 Digital outputs

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60s AC/DC/Relay	CPU CR60 AC/DC/Relay
Number of outputs	24	24	24	
Type	Solid state - MOSFET (sourcing)	Relay, dry contact	Relay, dry contact	
Voltage range	20.4 to 28.8 V DC	5 to 30 V DC or 5 to 250 V AC	5 to 30 V DC or 5 to 250 V AC	
Logic 1 signal at max. current	20 V DC min.	--	--	
Logic 0 signal with 10 KΩ load	0.1 V DC max.	--	--	
Rated current per point (max.)	0.5 A	2 A	2 A	
Rated current per common (max.)	6 A	10 A	10 A	
Lamp load	5 W	30 W DC / 200 W AC	30 W DC / 200 W AC	
ON state resistance	0.6 Ω max.	0.2 Ω max. when new	0.2 Ω max. when new	
Leakage current per point	10 µA max.	--	--	
Surge current	8 A for 100 ms max.	7 A with contacts closed	7 A with contacts closed	
Overload protection	No	No	No	
Isolation (field side to logic)	500 V AC for 1 minute	1500 V AC for 1 minute (coil to contact) None (coil to logic)	1500 V AC for 1 minute (coil to contact) None (coil to logic)	
Isolation resistance	--	100 M Ω min. when new	100 M Ω min. when new	
Isolation between open contacts	--	750 V AC for 1 minute	750 V AC for 1 minute	
Isolation groups	3	6	6	
Inductive clamp voltage	L+ minus 48 V DC, 1 W dissipation	--	-	
Switching delay (Qa.0 to Qa.3)	1.0 µs max., off to on 3.0 µs max., on to off	10 ms max.	10 ms max.	
Switching delay (Qa.4 to Qc.7)	50 µs max., off to on 200 µs max., on to off	10 ms max.	10 ms max.	
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	10,000,000 open/close cycles	
Lifetime contacts at rated load	--	100,000 open/close cycles	100,000 open/close cycles	
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	
Number of outputs on simultaneously	24	24	24	
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	

A.2.4.3 Wiring diagrams

Table A- 66 Wiring diagram for the CPU ST60 DC/DC/DC (6ES7288-1ST60-0AA0)

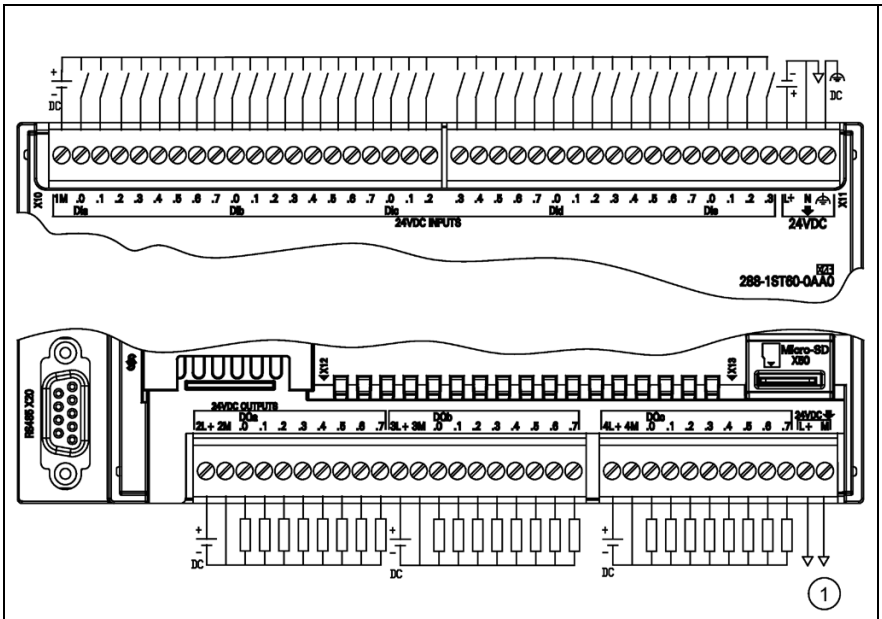
	①	24 V DC Sensor Power Output

Table A- 67 Connector pin locations for CPU ST60 DC/DC/DC (6ES7288-1ST60-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI c.3	2L+	4L+
2	DI a.0	DI c.4	2M	4M
3	DI a.1	DI c.5	DQ a.0	DQ c.0
4	DI a.2	DI c.6	DQ a.1	DQ c.1
5	DI a.3	DI c.7	DQ a.2	DQ c.2
6	DI a.4	DI d.0	DQ a.3	DQ c.3
7	DI a.5	DI d.1	DQ a.4	DQ c.4
8	DI a.6	DI d.2	DQ a.5	DQ c.5
9	DI a.7	DI d.3	DQ a.6	DQ c.6
10	DI b.0	DI d.4	DQ a.7	DQ c.7
11	DI b.1	DI d.5	3L+	L+ / 24 V DC Out
12	DI b.2	DI d.6	3M	M / 24 V DC Out
13	DI b.3	DI d.7	DQ b.0	--
14	DI b.4	DI e.0	DQ b.1	--
15	DI b.5	DI e.1	DQ b.2	--
16	DI b.6	DI e.2	DQ b.3	--
17	DI b.7	DI e.3	DQ b.4	--
18	DI c.0	L+ / 24 V DC	DQ b.5	--

Pin	X10	X11	X12	X13
19	DI c.1	M / 24 V DC	DQ b.6	--
20	DI c.2	Functional Earth	DQ b.7	--

Table A- 68 Wiring diagram for the CPU SR60 AC/DC/Relay (6ES7288-1SR60-0AA0)

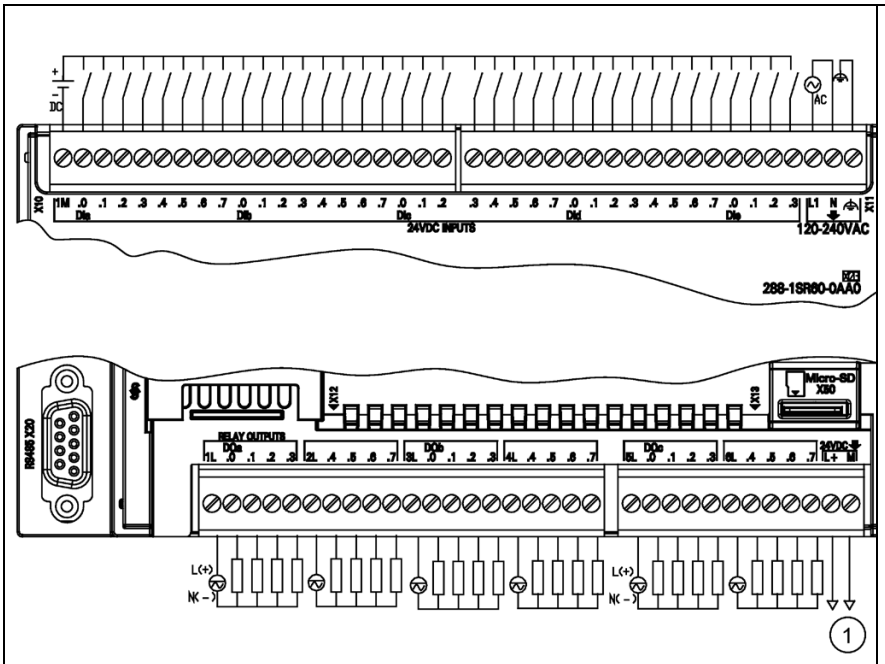
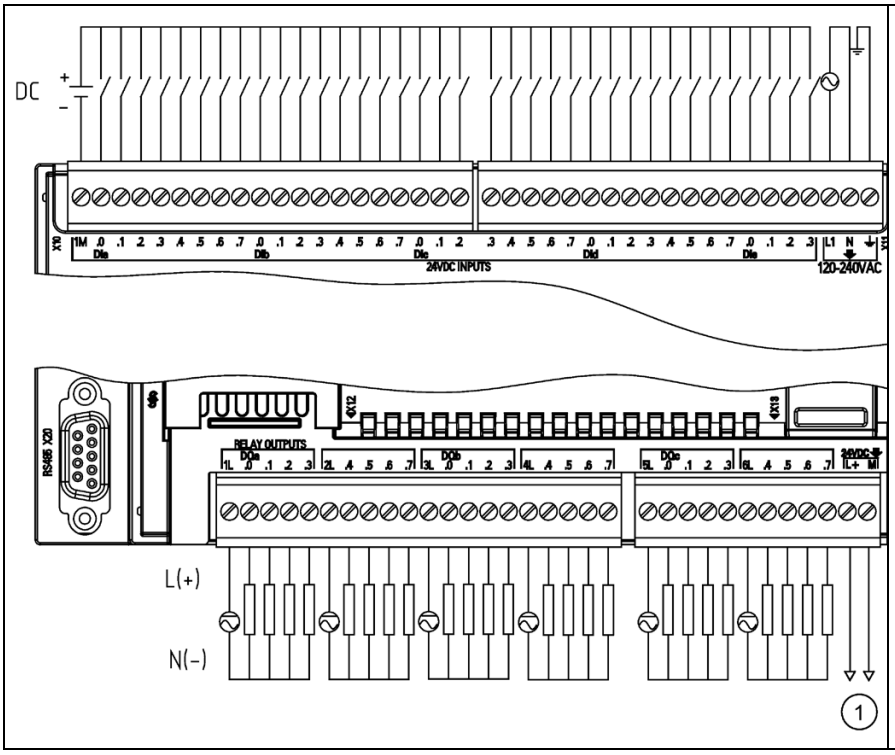


Table A- 69 Connector pin locations for CPU SR60 AC/DC/Relay (6ES7288-1SR60-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI c.3	1L	5L
2	DI a.0	DI c.4	DQ a.0	DQ c.0
3	DI a.1	DI c.5	DQ a.1	DQ c.1
4	DI a.2	DI c.6	DQ a.2	DQ c.2
5	DI a.3	DI c.7	DQ a.3	DQ c.3
6	DI a.4	DI d.0	2L	6L
7	DI a.5	DI d.1	DQ a.4	DQ c.4
8	DI a.6	DI d.2	DQ a.5	DQ c.5
9	DI a.7	DI d.3	DQ a.6	DQ c.6
10	DI b.0	DI d.4	DQ a.7	DQ c.7
11	DI b.1	DI d.5	3L	L+ / 24 V DC Out
12	DI b.2	DI d.6	DQ b.0	M / 24 V DC Out
13	DI b.3	DI d.7	DQ b.1	--
14	DI b.4	DI e.0	DQ b.2	--

Pin	X10	X11	X12	X13
15	DI b.5	DI e.1	DQ b.3	--
16	DI b.6	DI e.2	4L	--
17	DI b.7	DI e.3	DQ b.4	--
18	DI c.0	L1 / 120 - 240 V AC	DQ b.5	--
19	DI c.1	N / 120 - 240 V AC	DQ b.6	--
20	DI c.2	Functional Earth	DQ b.7	--

Table A- 70 Wiring diagram for the CPU CR60s AC/DC/Relay (6ES7288-1CR60-0AA1) and CPU CR60 AC/DC/Relay (6ES7288-1CR60-0AA0)



① 24 V DC Sensor Power Output

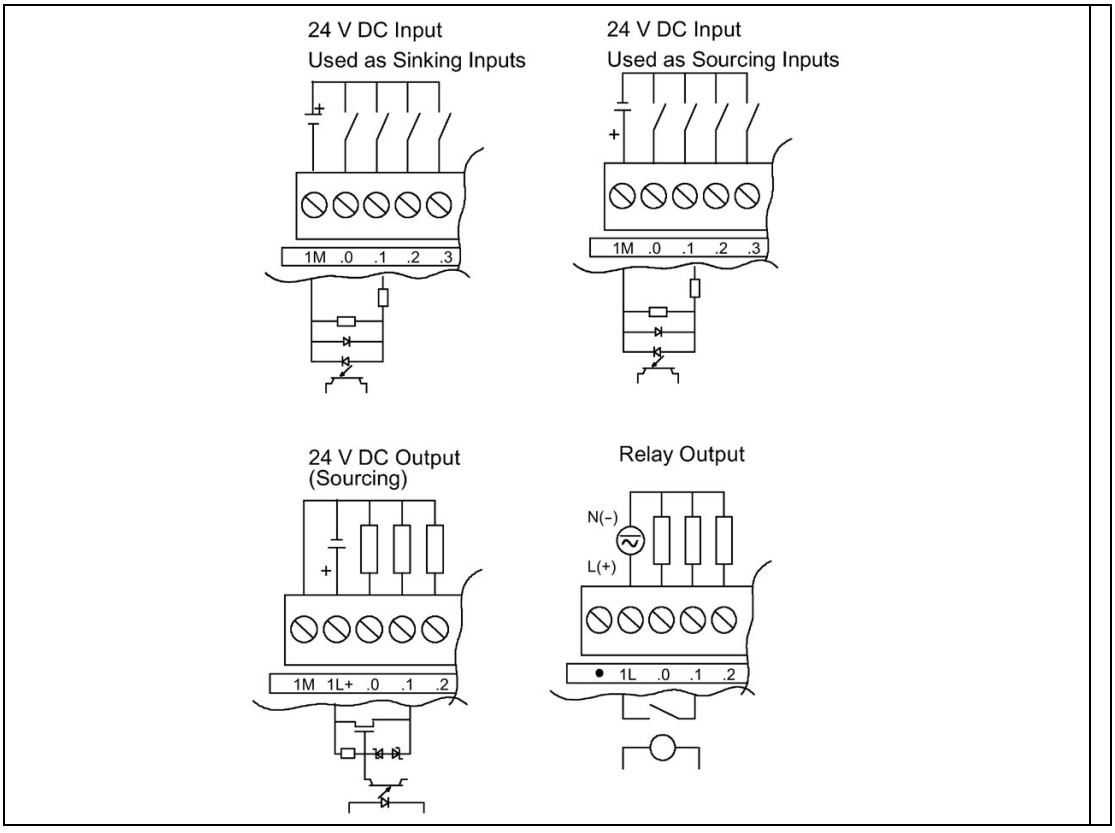
Table A- 71 Connector pin locations for CPU CR60s AC/DC/Relay (6ES7288-1CR60-0AA1) and CPU CR60 AC/DC/Relay (6ES7288-1CR60-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI c.3	1L	5L
2	DI a.0	DI c.4	DQ a.0	DQ c.0
3	DI a.1	DI c.5	DQ a.1	DQ c.1
4	DI a.2	DI c.6	DQ a.2	DQ c.2
5	DI a.3	DI c.7	DQ a.3	DQ c.3
6	DI a.4	DI d.0	2L	6L
7	DI a.5	DI d.1	DQ a.4	DQ c.4

Pin	X10	X11	X12	X13
8	DI a.6	DI d.2	DQ a.5	DQ c.5
9	DI a.7	DI d.3	DQ a.6	DQ c.6
10	DI b.0	DI d.4	DQ a.7	DQ c.7
11	DI b.1	DI d.5	3L	L+ / 24 V DC Out
12	DI b.2	DI d.6	DQ b.0	M / 24 V DC Out
13	DI b.3	DI d.7	DQ b.1	--
14	DI b.4	DI e.0	DQ b.2	--
15	DI b.5	DI e.1	DQ b.3	--
16	DI b.6	DI e.2	4L	--
17	DI b.7	DI e.3	DQ b.4	--
18	DI c.0	L1 / 120 - 240 V AC	DQ b.5	--
19	DI c.1	N / 120 - 240 V AC	DQ b.6	--
20	DI c.2	Functional Earth	DQ b.7	--

A.2.5 Wiring diagrams for sink and source input, and relay output

Table A- 72 Wiring diagrams for sink input, source input, and relay output



A.3 Digital inputs and outputs expansion modules (EMs)

A.3.1 EM DE08 and EM DE16 digital input specifications

Table A- 73 General specifications

Model	EM Digital 8 x Inputs (EM DE08)	EM Digital 16 x Inputs (EM DE16)
Article number	6ES7288-2DE08-0AA0	6ES7288-2DE16-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81
Weight	141.4 grams	176 grams
Power dissipation	1.5 W	2.3 W
Current consumption (SM Bus)	105 mA	105 mA
Current consumption (24 V DC)	4 mA / input used	4 mA / input used

Table A- 74 Digital inputs

Model	EM Digital 8 x Inputs (EM DE08)	EM Digital 16 x Inputs (EM DE16)
Number of inputs	8	16
Type	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal
Continuous permissible voltage	30 V DC, max.	30 V DC, max.
Surge voltage	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.
Logic 1 signal (min.)	15 V DC at 2.5 mA	15 V DC at 2.5 mA
Logic 0 signal (max.)	5 V DC at 1 mA	5 V DC at 1 mA
Isolation (field side to logic)	500 V AC for 1 minute	500 V AC for 1 minute
Isolation groups	2	4
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)
Number of inputs on simultaneously	8	16
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs

Table A- 75 Wiring diagram for the EM DE08 Digital 8 x Inputs (6ES7288-2DE08-0AA0) and EM DE16 Digital 16 x Input (6ES7288-2DE16-0AA0)

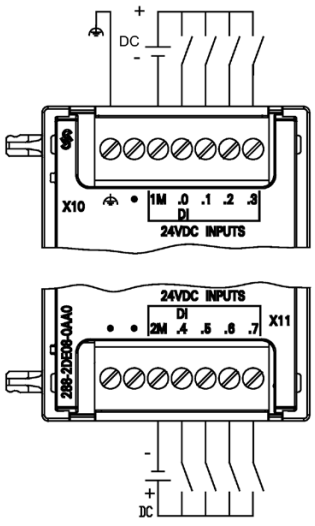
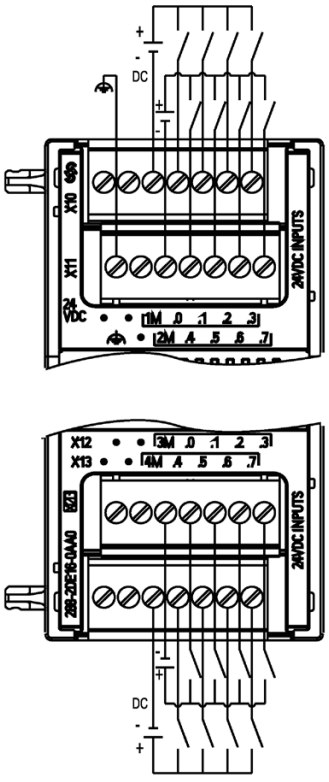
EM DE08 Digital 8 x Inputs (6ES7288-2DE08-0AA0)	EM DE16 Digital 16 x Inputs (6ES7288-2DE16-0AA0)
	

Table A- 76 Connector pin locations for EM DE08 Digital 8 x Input (6ES7288-2DE08-0AA0)

Pin	X10	X11
1	Functional Earth	No connection
2	No connection	No connection
3	1M	2M
4	DI a.0	DI a.4
5	DI a.1	DI a.5
6	DI a.2	DI a.6
7	DI a.3	DI a.7

Table A- 77 Connector pin locations for EM DE16 Digital 16 x Input (6ES7288-2DE16-0AA0)

Pin	X10	X11	X12	X13
1	No connection	Functional Earth	No connection	No connection
2	No connection	No connection	No connection	No connection
3	1M	2M	3M	4M
4	DI a.0	DI a.4	DI b.0	DI b.4
5	DI a.1	DI a.5	DI b.1	DI b.5
6	DI a.2	DI a.6	DI b.2	DI b.6
7	DI a.3	DI a.7	DI b.3	DI b.7

A.3.2 EM DT08, EM DR08, EM QR16, and EM QT16 digital output specifications

Table A- 78 General specifications

Model	EM Digital 8 x Outputs (EM DT08)	EM Digital 8 x Outputs Relay (EM DR08)	EM Digital 16 x Outputs Relay (EM QR16)	EM Digital 16 x Outputs Transistor (EM QT16)
Article number	6ES7288-2DT08-0AA0	6ES7288-2DR08-0AA0	6ES7288-2QR16-0AA0	6ES7288-2QT16-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81	45 x 100 x 81	45 x 100 x 81
Weight	147 grams	166.3 grams	221 grams	186 grams
Power dissipation	1.5 W	4.5 W	4.5 W	1.7 W
Current consumption (SM Bus)	120 mA	120 mA	110 mA, SM bus	120 mA, SM bus
Current consumption (24 V DC)	--	11 mA / relay coil used	150 mA, all relay are on	50 mA

Table A- 79 Digital outputs

Model	EM Digital 8 x Outputs (EM DT08)	EM Digital 8 x Outputs Relay (EM DR08)	EM Digital 16 x Outputs Relay (EM QR16)	EM Digital 16 x Outputs Transistor (EM QT16)
Number of outputs	8	8	16	16
Type	Solid state - MOSFET (sourcing)	Relay, dry contact	Relay, dry contact	Solid state - MOSFET (sourcing)
Voltage range	20.4 to 28.8 V DC	5 to 30 V DC or 5 to 250 V AC	5 to 30 V DC or 5 to 250 V AC	20.4 to 28.8 V DC
Logic 1 signal at max. current	20 V DC	-	-	20 V DC
Logic 0 signal with 10 K Ω load	0.1 V DC	-	-	0.1 V DC
Rated current per point (max.)	0.75 A	2.0 A	2.0 A	0.75 A
Rated current per common (max.)	3 A	8 A	8 A	3 A
Lamp load	5 W DC	30 W DC / 200 W AC	30 W DC/200 W AC	5 W

Model	EM Digital 8 x Outputs (EM DT08)	EM Digital 8 x Outputs Relay (EM DR08)	EM Digital 16 x Outputs Relay (EM QR16)	EM Digital 16 x Outputs Transistor (EM QT16)
ON state contact resistance	0.6 Ω	0.2 Ω max. when new	0.2 Ω max. when new	0.6 Ω max.
Leakage current per point	10 µA	--	--	10 µA
Surge current	8 A for 100 ms max.	7 A with contacts closed	7 A with contacts closed	8 A for 100 ms max.
Overload protection	No	No	No	No
Isolation (field side to logic)	Optical, 500 V AC for 1 minute	1500 V AC for 1 minute (coil to contact) None (coil to logic)	1500 V AC for 1 minute (coil to contact) None (coil to logic)	500 V AC for 1 minute
Isolation resistance	-	100 M Ω min. when new	100 M Ω min. when new	-
Isolation between open contacts	-	750 V AC for 1 minute	750 V AC for 1 minute	-
Isolation groups	2	2	4	4
Inductive clamp voltage	Minus 48 V DC, 1 W dissipation	-	-	L+ minus 48 V, 1 W dissipation
Switching delay	Switch on less than 50 µs and switch off less than 200 µS	10 ms max.	10 ms max.	Switch on less than 50 µs and switch off less than 200 µS
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	10,000,000 open/close cycles	-
Lifetime contacts at rated load	--	100,000 open/close cycles	100,000 open/close cycles	-
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8	8	• 8 (no adjacent points) at 60 °C horizontal or 50 °C vertical • 16 at 55 °C horizontal or 45 °C vertical	16
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

Table A- 80 Wiring diagrams for the EM DT08 Digital 8 x Outputs (6ES7288-2DT08-0AA0) and EM DR08 Digital 8 x Outputs x Relay (6ES7288-2DR08-0AA0)

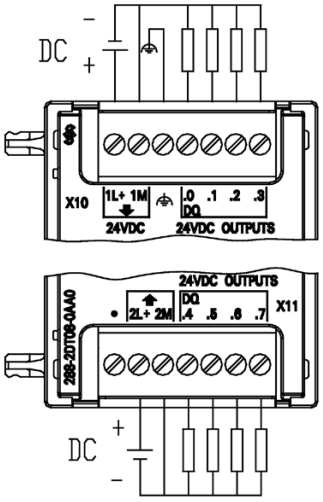
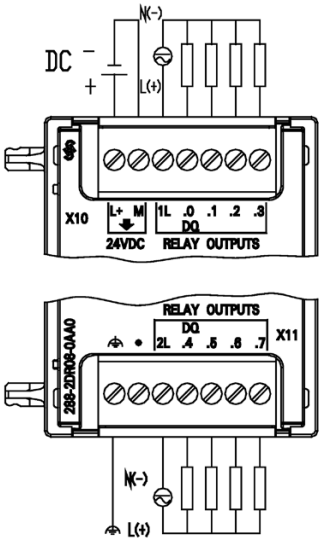
EM DT08 Digital 8 x Outputs (6ES7288-2DT08-0AA0)	EM DR08 Digital 8 x Outputs x Relay (6ES7288-2DR08-0AA0)
	

Table A- 81 Connector pin locations for EM DT08 Digital 8 x Outputs (6ES7288-2DT08-0AA0

Pin	X10	X11
1	1L+ / 24 V DC	No connection
2	1M / 24 V DC	2L+ / 24 V DC
3	Functional Earth	2M / 24 V DC
4	DQ a.0	DQ a.4
5	DQ a.1	DQ a.5
6	DQ a.2	DQ a.6
7	DQ a.3	DQ a.7

Table A- 82 Connector pin locations for EM DR08 Digital 8 x Outputs x Relay (6ES7288-2DR08-0AA0)

Pin	X10	X11
1	L+ / 24 V DC	Functional Earth
2	M / 24 V DC	No connection
3	1L	2L
4	DQ a.0	DQ a.4
5	DQ a.1	DQ a.5

Pin	X10	X11
6	DQ a.2	DQ a.6
7	DQ a.3	DQ a.7

Table A- 83 Wiring diagrams for the EM QR16 Digital 16 x Outputs Relay (6ES7288-2QR16-0AA0) and EM QT16 Digital 16 x Outputs Transition (6ES7288-2QT16-0AA0)

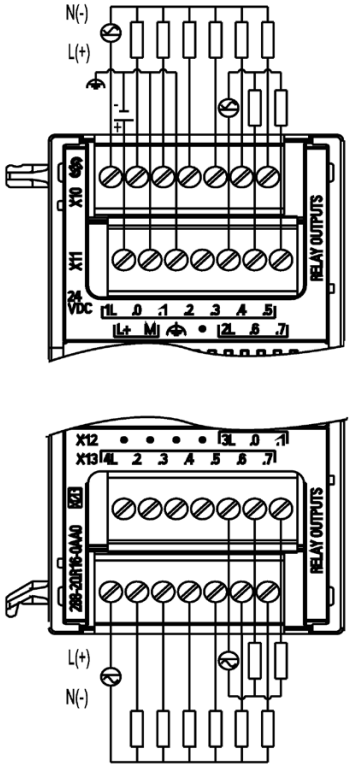
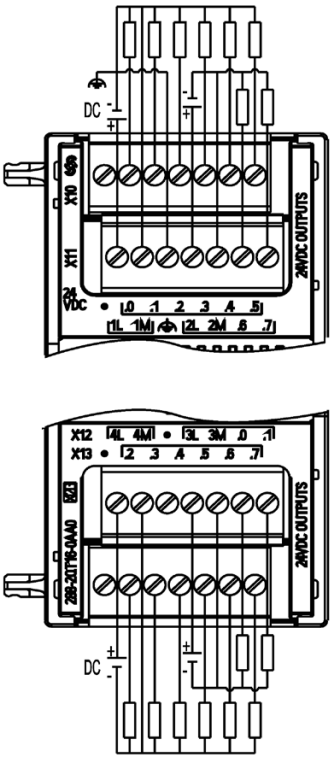
EM QR16 Digital 16 x Outputs Relay (6ES7288-2QR16-0AA0)	EM QT16 Digital 16 x Outputs Transition (6ES7288-2QT16-0AA0)
	

Table A- 84 Connector pin locations for EM QR16 Digital 16 x Outputs Relay (6ES7288-2QR16-0AA0)

Pin	X10	X11	X12	X13
1	1L	L+ / 24 V DC	No connection	4L
2	DQ a.0	M / 24 V DC	No connection	DQ b.2
3	DQ a.1	Functional Earth	No connection	DQ b.3
4	DQ a.2	No connection	No connection	DQ b.4
5	DQ a.3	2L	3L	DQ b.5
6	DQ a.4	DQ a.6	DQ b.0	DQ b.6
7	DQ a.5	DQ a.7	DQ b.1	DQ b.7

Table A- 85 Connector pin locations for EM QT16 Digital 16 x Outputs Transition (6ES7288-2QT16-0AA0)

Pin	X10	X11	X12	X13
1	No connection	1L / 24 V DC	4L / 24 V DC	No connection
2	DQ a.0	1M / 24 V DC	4M / 24 V DC	DQ b.2
3	DQ a.1	Functional Earth	No connection	DQ b.3
4	DQ a.2	2L / 24 V DC	3L / 24 V DC	DQ b.4
5	DQ a.3	2M / 24 V DC	3M / 24 V DC	DQ b.5
6	DQ a.4	DQ a.6	DQ b.0	DQ b.6
7	DQ a.5	DQ a.7	DQ b.1	DQ b.7

A.3.3 EM DT16, EM DR16, EM DT32, and EM DR32 digital input/output specifications

Table A- 86 General specifications

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs / 16 x Relay Outputs (EM DR32)
Article number	6ES7288-2DT16-0AA0	6ES7288-2DR16-0AA0	6ES7288-2DT32-0AA0	6ES7288-2DR32-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81	70 x 100 x 81	70 x 100 x 81
Weight	179.7g grams	201.9 grams	257.3 grams	295.4 grams
Power dissipation	2.5 W	5.5 W	4.5 W	10 W
Current consumption (SM Bus)	145 mA	145 mA	185 mA	180 mA
Current consumption (24 V DC)	4 mA / Input used 11 mA / Relay coil used	4 mA / Input used 11 mA / Relay coil used	4 mA / Input used	4 mA / Input used

Table A- 87 Digital inputs

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs / 16 x Relay Outputs (EM DR32)
Number of inputs	8	8	16	16
Type	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal	24 V DC at 4 mA, nominal

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs / 16 x Relay Outputs (EM DR32)
Continuous permissible voltage	30 V DC max.	30 V DC max.	30 V DC max.	30 V DC max.
Surge voltage	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.	35 V DC for 0.5 sec.
Logic 1 signal (min.)	15 V DC	15 V DC	15 V DC	15 V DC
Logic 0 signal (max.)	5 V DC	5 V DC	5 V DC	5 V DC
Isolation (field side to logic)	500 V AC for 1 minute	500 V AC for 1 minute	500 V AC for 1 minute	500 V AC for 1 minute
Isolation groups	2	2	2	2
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4
Number of inputs on simultaneously	8	8	16	16
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs

Table A- 88 Digital outputs

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs/ 16 x Relay Outputs (EM DR32)
Number of outputs	8	8	16	16
Type	Solid state-MOSFET (sourcing)	Relay, dry contact	Solid state-MOSFET (sourcing)	Relay, dry contact
Voltage range	20.4 to 28.8 V DC	5 to 30 V DC or 5 to 250 V AC	20.4 to 28.8 V DC	5 to 30 V DC or 5 to 250 V AC
Logic 1 signal at max. current	20 V DC, min.	--	20 V DC, min.	--
Logic 0 signal with 10 KΩ load	0.1 V DC, max.	--	0.1 V DC, max.	--
Rated current per point (max.)	0.75 A	2 A	0.75 A	2 A
Rated current per common (max.)	3 A	8 A	6 A	8 A
Lamp load	5 W	30 W DC/200 W AC	5 W	30 W DC/200 W AC
ON state contact resistance	0.6 Ω max.	0.2 Ω max. when new	0.6 Ω max.	0.2 Ω max. when new
Leakage current per point	10 µA max.	--	10 µA max.	--
Surge current	8 A for 100 ms max.	7 A with contacts closed	8 A for 100 ms max.	7 A with contacts closed

Model	**EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)**	**EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)**	**EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)**	**EM Digital 16 x Inputs/ 16 x Relay Outputs (EM DR32)**
Overload protection	No	No	No	No
Isolation (field side to logic)	500 V AC for 1 minute	1500 V AC for 1 minute (coil to contact) None (coil to logic)	500 V AC for 1 minute	1500 V AC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 M Ω min. when new	--	100 M Ω min. when new
Isolation between open contacts	--	750 V AC for 1 minute	--	750 V AC for 1 minute
Isolation groups	2	2	3	4
Inductive clamp voltage	Minus 48 V	--	Minus 48 V	--
Switching delay	50 µs max., off to on 200 µs max., on to off	10 ms max.	50 µs max., off to on 200 µs max., on to off	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	--	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles	--	100,000 open/close cycles
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8	8	16	16
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

Table A- 89 Wiring diagrams for the EM DT16 Digital 8 x Inputs / Digital 8 x Outputs (6ES7288-2DT16-0AA0) and EM DR16 Digital 8 x Inputs / 8 x Relay Outputs (6ES7288-2DR16-0AA0)

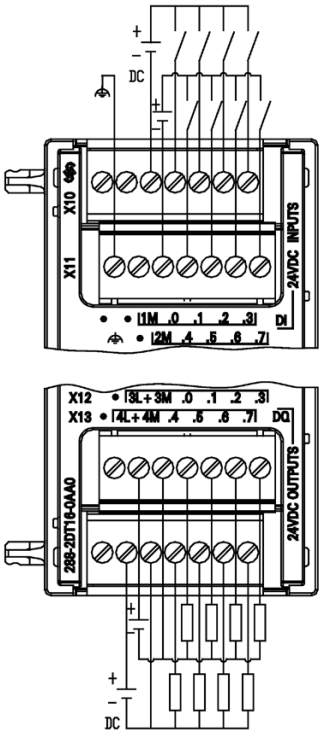
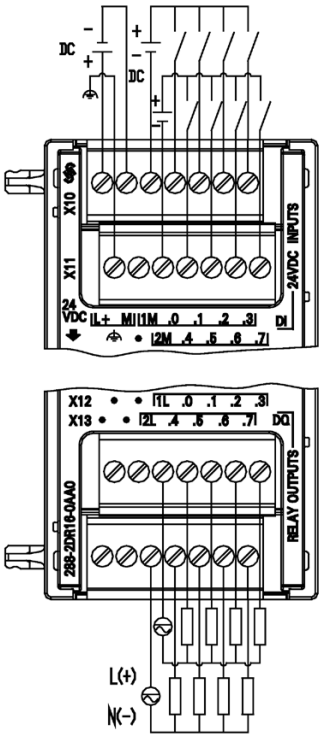
EM DT16 Digital 8 x Inputs / Digital 8 x Outputs (6ES7288-2DT16-0AA0)	EM DR16 Digital 8 x Inputs / 8 x Relay Outputs (6ES7288-2DR16-0AA0)
	

Table A- 90 Connector pin locations for EM DT16 Digital 8 x Inputs / Digital 8 x Outputs (6ES7288-2DT16-0AA0)

Pin	X10	X11	X12	X13
1	No connection	Functional Earth	No connection	No connection
2	No connection	No connection	3L+ / 24 V DC	4L+ / 24 V DC
3	1M	2M	3M / 24 V DC	4M / 24 V DC
4	DI a.0	DI a.4	DQ a.0	DQ a.4
5	DI a.1	DI a.5	DQ a.1	DQ a.5
6	DI a.2	DI a.6	DQ a.2	DQ a.6
7	DI a.3	DI a.7	DQ a.3	DQ a.7

Table A- 91 Connector pin locations for EM DR16 Digital 8 x Inputs / 8 x Relay Outputs (6ES7288-2DR16-0AA0)

Pin	X10	X11	X12	X13
1	L+ / 24 V DC	Functional Earth	No connection	No connection
2	M / 24 V DC	No connection	No connection	No connection
3	1M	2M	1L	2L
4	DI a.0	DI a.4	DQ a.0	DQ a.4
5	DI a.1	DI a.5	DQ a.1	DQ a.5
6	DI a.2	DI a.6	DQ a.2	DQ a.6
7	DI a.3	DI a.7	DQ a.3	DQ a.7

Table A- 92 Wiring diagrams for the EM DT32 Digital 16 x Inputs / Digital 16 x Outputs (6ES7288-2DT32-0AA0 and EM DR32 Digital 16 x Inputs / 16 x Relay (6ES7288-2DR32-0AA0)

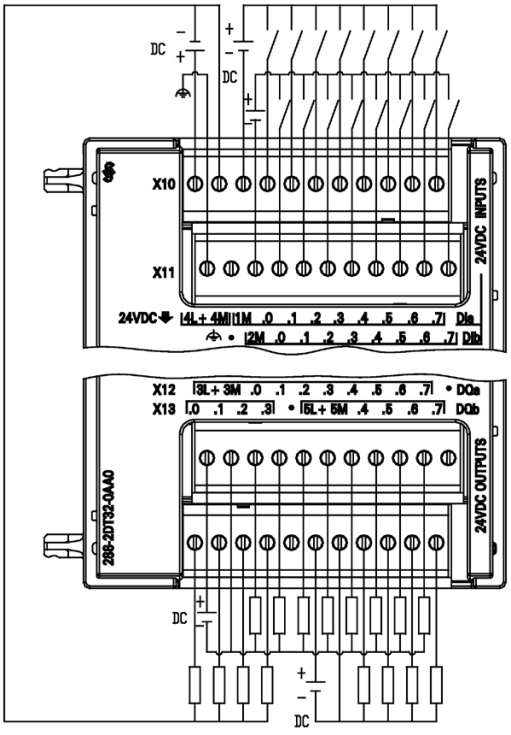
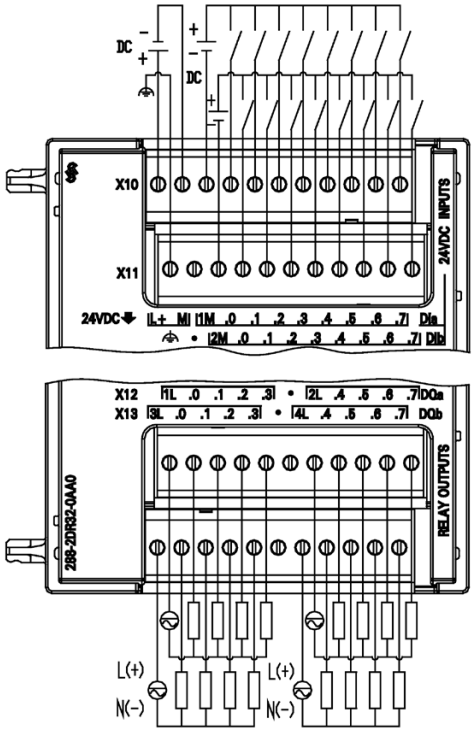
EM DT32 Digital 16 x Inputs / Digital 16 x Outputs (6ES7288-2DT32-0AA0)	EM DR32 Digital 16 x Inputs / 16 x Relay (6ES7288-2DR32-0AA0)
	

Table A- 93 Connector pin locations for EM DT32 Digital 16 x Inputs / Digital 16 x Outputs (6ES7288-2DT32-0AA0)

Pin	X10	X11	X12	X13
1	4L+ / 24 V DC[1]	Functional Earth	3L+ / 24 V DC	DQ b.0[1]
2	4M / 24 V DC[1]	No connection	3M / 24 V DC	DQ b.1[1]
3	1M	2M	DQ a.0	DQ b.2[1]
4	DI a.0	DI b.0	DQ a.1	DQ b.3[1]
5	DI a.1	DI b.1	DQ a.2	No connection
6	DI a.2	DI b.2	DQ a.3	5L+ / 24 V DC
7	DI a.3	DI b.3	DQ a.4	5M / 24 V DC
8	DI a.4	DI b.4	DQ a.5	DQ b.4
9	DI a.5	DI b.5	DQ a.6	DQ b.5
10	DI a.6	DI b.6	DQ a.7	DQ b.6
11	DI a.7	DI b.7	No connection	DQ b.7

[1] In same isolation group.

Table A- 94 Connector pin locations for EM DR32 Digital 16 x Inputs / 16 x Relay (6ES7288-2DR32-0AA0)

Pin	X10	X11	X12	X13
1	L+ / 24 V DC	Functional Earth	1L	3L
2	M / 24 V DC	No connection	DQ a.0	DQ b.0
3	1M	2M	DQ a.1	DQ b.1
4	DI a.0	DI b.0	DQ a.2	DQ b.2
5	DI a.1	DI b.1	DQ a.3	DQ b.3
6	DI a.2	DI b.2	No connection	No connection
7	DI a.3	DI b.3	2L	4L
8	DI a.4	DI b.4	DQ a.4	DQ b.4
8	DI a.5	DI b.5	DQ a.5	DQ b.5
10	DI a.6	DI b.6	DQ a.6	DQ b.6
11	DI a.7	DI b.7	DQ a.7	DQ b.7

A.4 Analog inputs and outputs expansion modules (EMs)

A.4.1 EM AE04 and EM AE08 analog input specifications

Table A- 95 General specifications

Model	EM Analog 4 x inputs (EM AE04)	EM Analog 8 x inputs (EM AE08)
Article number	6ES7288-3AE04-0AA0	6ES7288-3AE08-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81
Weight	147 grams	186 grams
Power dissipation	1.5 W (no load)	2.0 W (no load)
Current consumption (SM Bus)	80 mA	80 mA
Current consumption (24 V DC)	40 mA (no load)	70 mA (no load)

Table A- 96 Analog inputs

Model	EM Analog 4 x inputs (EM AE04)	EM Analog 8 x inputs (EM AE08)
Number of inputs	4	8
Type	Voltage or current (differential), selectable in groups of 2	Voltage or current (differential), selectable in groups of 2
Range	±10 V, ±5 V, ±2.5 V, or 0 to 20 mA	±10 V, ±5 V, ±2.5 V, or 0 to 20 mA
Full scale range (data word)	-27,648 to 27,648	-27,648 to 27,648
Overshoot/undershoot range (data word)	Voltage: 27,649 to 32,511 / -27,649 to -32,512 Current: 27,649 to 32,511 / -4864 to 0	Voltage: 27,649 to 32,511 / -27,649 to -32,512 Current: 27,649 to 32,511 / -4864 to 0 (Refer to Analog input representation for voltage and Analog input representation for current (Page 823).)
Overflow/underflow (data word)	Voltage: 32,512 to 32,767 / -32,513 to -32,768 Current: 32,512 to 32,767 / -4,865 to -32,768	Voltage: 32,512 to 32,767 / -32,513 to -32,768 Current: 32,512 to 32,767 / -4,865 to -32,768 (Refer to Analog input representation for voltage and Analog input representation for current (Page 823).)
Resolution	Voltage mode: 12 bits + sign bit Current mode: 12 bits	Voltage mode: 12 bits + sign bit Current mode: 12 bits
Maximum withstand voltage/current	±35 V / ±40 mA	±35 V / ±40 mA
Smoothing	None, weak, medium or strong	None, weak, medium or strong (Refer to Analog input response times for step response time.) (Page 822)
Noise rejection	400, 60, 50, or 10 Hz	400, 60, 50, or 10 Hz
Input impedance	≥9 M Ω (voltage) / 250 Ω (current)	≥9 M Ω (voltage) / 250 Ω (current)
Isolation (field side to logic)	None	None

Model	EM Analog 4 x inputs (EM AE04)	EM Analog 8 x inputs (EM AE08)
Accuracy (25 °C / 0 to 55 °C)	Voltage mode: ±0.1% / ±0.2% of full scale Current mode: ±0.2% / ±0.3% of full scale	Voltage mode: ±0.1% / ±0.2% of full scale Current mode: ±0.2% / ±0.3% of full scale
Measuring principle	Actual value conversion	Actual value conversion
Common mode rejection	40 dB, DC to 60 Hz	40 dB, DC to 60 Hz
Operational signal range	Signal plus common mode voltage must be less than +12 V and greater than -12 V	Signal plus common mode voltage must be less than +12 V and greater than -12 V
Cable length (max.), in meters	100 m twisted and shielded	100 m twisted and shielded

Table A- 97 Diagnostics

Model	EM Analog 4 x inputs (EM AE04)	EM Analog 8 x inputs (EM AE08)
Overflow/underflow	Yes	Yes
24 V DC low voltage	Yes	Yes

EM AE04 and EM AE08 wiring current transducers

Wiring current transducers are available as 2-wire transducers and 4-wire transducers as shown below.

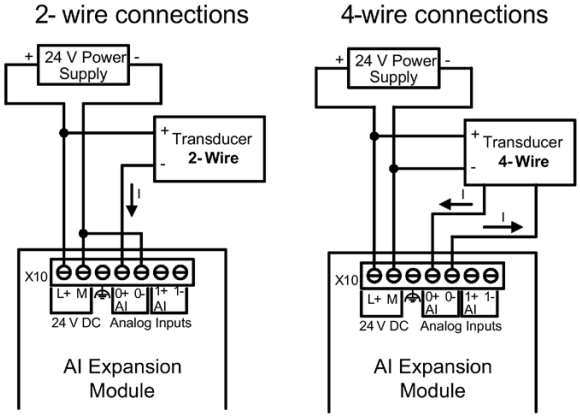


Table A- 98 Wiring diagram EM AE04 Analog 4 x Inputs (6ES7288-3AE04-0AA0) and EM AE08 Analog 8 x Inputs (6ES7288-3AE08-0AA0

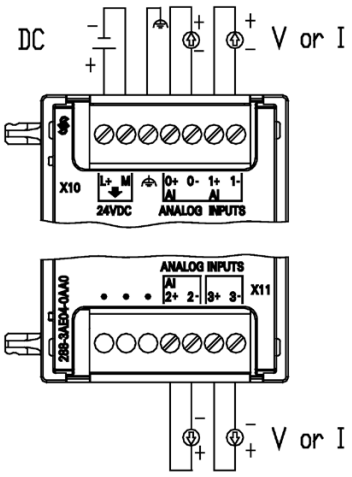
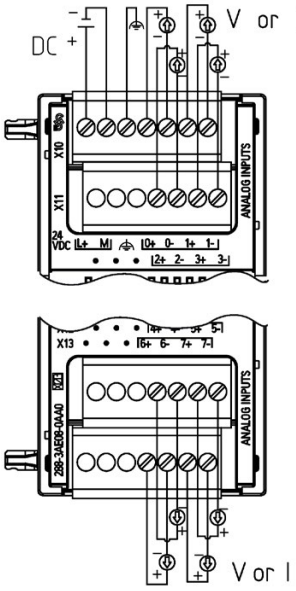
EM AE04 Analog 4 x Inputs (6ES7288-3AE04-0AA0)	EM AE08 Analog 8 x Inputs (6ES7288-3AE08-0AA0)
	
Note: Connectors must be gold. See Appendix F, Spare parts and other hardware, for article number.	

Table A- 99 Connector pin locations for EM AE04 Analog 4 x Inputs (6ES7288-3AE04-0AA0)

Pin	X10 (gold)	X11 (gold)
1	L+ / 24 V DC	No connection
2	M / 24 V DC	No connection
3	Functional Earth	No connection
4	AI 0+	AI 2+
5	AI 0-	AI 2-
6	AI 1+	AI 3+
7	AI 1-	AI 3-

Table A- 100 Connector pin locations for EM AE08 Analog 8 x Inputs (6ES7288-3AE08-0AA0)

Pin	X10 (gold)	X11 (gold)	X12 (gold)	X13 (gold)
1	L+ / 24 V DC	No connection	No connection	No connection
2	M / 24 V DC	No connection	No connection	No connection
3	Functional Earth	No connection	No connection	No connection
4	AI 0+	AI 2+	AI 4+	AI 6+
5	AI 0-	AI 2-	AI 4-	AI 6-

Pin	X10 (gold)	X11 (gold)	X12 (gold)	X13 (gold)
6	AI 1+	AI 3+	AI 5+	AI 7+
7	AI 1-	AI 3-	AI 5-	AI 7-

A.4.2 EM AQ02 and EM AQ04 analog output module specifications

Table A- 101 General specifications

Technical data	EM Analog 2 x outputs (EM AQ02)	EM Analog 4 x outputs (EM AQ04)
Article number	6ES7288-3AQ02-0AA0	6ES7288-3AQ04-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81
Weight	147.1 grams	170.5 grams
Power dissipation	1.5 W (no load)	2.1 W (no load)
Current consumption (SM Bus)	60 mA	60 mA
Current consumption (24 V DC)	50 mA (no load)	75 mA (no load)
	90 mA (20 mA load per channel)	155 mA (20 mA load per channel)

Table A- 102 Analog outputs

Technical data	EM Analog 2 x outputs (EM AQ02)	EM Analog 4 x outputs (EM AQ04)
Number of outputs	2	4
Type	Voltage or current	Voltage or current
Range	±10 V or 0 to 20 mA	±10 V or 0 to 20 mA
Resolution	Voltage mode: 11 bits + sign bit Current mode: 11 bits	Voltage mode: 11 bits + sign bit Current mode: 11 bits
Full scale range (data word)	Voltage: -27,648 to 27,648 Current: 0 to 27,648	Voltage: -27,648 to 27,648 Current: 0 to 27,648 (Refer to the output ranges for voltage and current (Page 824).)
Accuracy (25 °C / 0 to 55 °C)	±0.5% / ±1.0% of full scale	±0.5% / ±1.0% of full scale
Settling time (95% of new value)	Voltage: 300 µs (R), 750 µ (R), 750 µs (1 µF) Current: 600 µs (1 mH), 2 ms (10 mH)	Voltage: 300 µs (R), 750 µ (R), 750 µs (1 µF) Current: 600 µs (1 mH), 2 ms (10 mH)
Load impedance	Voltage: ≥ 1000 Ω Current: ≤ 500 Ω	Voltage: ≥ 1000 Ω Current: ≤ 600 Ω
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Isolation (field side to logic)	None	None
Cable length (max.), in meters	100 m twisted and shielded	100 m twisted and shielded

Table A- 103 Diagnostics

Technical data	EM Analog 2 x outputs (EM AQ02)	EM Analog 4 x outputs (EM AQ04)
Overflow/underflow	Yes	Yes
Short to ground (voltage mode only)	Yes	Yes
Wire break (current mode only)	Yes	Yes
24 V DC low voltage	Yes	Yes

Table A- 104 Wiring diagram for the EM AQ02 Analog 2 x Outputs (6ES7288-3AQ02-0AA0) and EM AQ04 Analog 4 x Outputs (6ES7288-3AQ04-0AA0)

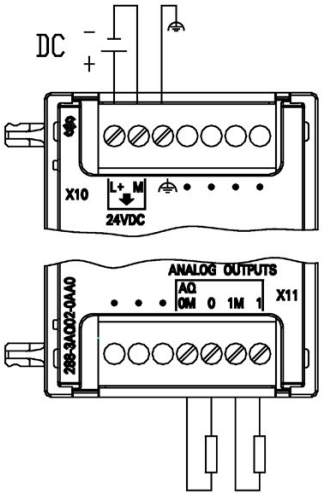
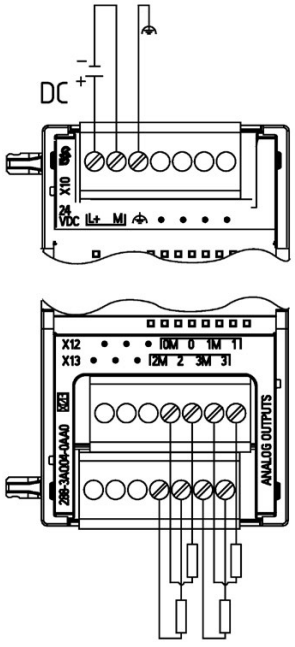
EM AQ02 Analog 2 x Outputs (6ES7288-3AQ02-0AA0)	EM AQ04 Analog 4 x Outputs (6ES7288-3AQ04-0AA0)
	
Note: Connectors must be gold. See Appendix F, Spare parts and other hardware, for article number.	

Table A- 105 Connector pin locations for EM AQ02 Analog 2 x Outputs (6ES7288-3AQ02-0AA0)

Pin	X10 (gold)	X11 (gold)
1	L+ / 24 V DC	No connection
2	M / 24 V DC	No connection
3	Functional Earth	No connection
4	No connection	AQ 0M
5	No connection	AQ 0

Pin	X10 (gold)	X11 (gold)
6	No connection	AQ 1M
7	No connection	AQ 1

Table A- 106 Connector pin locations for EM AQ04 Analog 4 x Outputs (6ES7288-3AQ04-0AA0)

Pin	X10 (gold)	X11 (gold)	X12 (gold)	X13 (gold)
1	L+ / 24 V DC	No connection	No connection	No connection
2	M / 24 V DC	No connection	No connection	No connection
3	Functional Earth	No connection	No connection	No connection
4	No connection	No connection	AQ 0M	AQ 2M
5	No connection	No connection	AQ 0	AQ 2
6	No connection	No connection	AQ 1M	AQ 3M
7	No connection	No connection	AQ 1	AQ 3

A.4.3 EM AM03 and EM AM06 analog input/output module specifications

Table A- 107 General specifications

Technical data	EM 2 x Analog Inputs / 1 x Analog Outputs (AM03)	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Article number	6ES7288-3AM03-0AA0	6ES7288-3AM06-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81
Weight	172 grams	173.4 grams
Power dissipation	1.1 W (no load)	2.0 W (no load)
Current consumption (SM Bus)	60 mA	80 mA
Current consumption (24 V DC)	30 mA (no load)	60 mA (no load)
	50 mA (20 mA load per channel)	100 mA (20 mA load per channel)

Table A- 108 Analog inputs

Model	EM 2 x Analog Inputs / 1 x Analog Outputs (AM03)	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Number of inputs	2	4
Type	Voltage or current (differential): Selectable in groups of 2	Voltage or current (differential): Selectable in groups of 2
Range	±10 V, ±5 V, ±2.5 V, or 0 to 20 mA	±10 V, ±5 V, ±2.5 V, or 0 to 20 mA
Full scale range (data word)	-27,648 to 27,648	-27,648 to 27,648
Overshoot/undershoot range (data word)	Voltage: 27,649 to 32,511 / -27,649 to -32,512 Current: 27,649 to 32,511 / -4,864 to 0	Voltage: 27,649 to 32,511 / -27,649 to -32,512 Current: 27,649 to 32,511 / -4,864 to 0

Model	EM 2 x Analog Inputs / 1 x Analog Outputs (AM03)	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Overflow/underflow (data word)	Voltage: 32,512 to 32,767 / -32,513 to -32,768 Current: 32,512 to 32,767 / -4,865 to -32,768	Voltage: 32,512 to 32,767 / -32,513 to -32,768 Current: 32,512 to 32,767 / -4,865 to -32,768
Resolution	Voltage mode: 12 bits + sign Current mode: 12 bits	Voltage mode: 12 bits + sign Current mode: 12 bits
Maximum withstand voltage/current	±35 V / ±40 mA	±35 V / ±40 mA
Smoothing	None, weak, medium, or strong	None, weak, medium, or strong
Noise rejection	400, 60, 50, or 10 Hz	400, 60, 50, or 10 Hz
Input impedance	≥9 M Ω	≥9 M Ω
Isolation (field side to logic)	None	None
Accuracy (25 °C / 0 to 55 °C)	Voltage mode: ±0.1% /±0.2% of full scale Current mode: ±0.2% /±0.3% of full scale	Voltage mode: ±0.1% /±0.2% of full scale Current mode: ±0.2% /±0.3% of full scale
Analog to digital conversion time	625 µs (400 Hz rejection)	625 µs (400 Hz rejection)
Common mode rejection	40 dB, DC to 60 Hz	40 dB, DC to 60 Hz
Operational signal range	Signal plus common mode voltage must be less than +12 V and greater than -12 V	Signal plus common mode voltage must be less than +12 V and greater than -12 V
Cable length (max.), in meters	100 m twisted and shielded	100 m twisted and shielded

Table A- 109 Analog outputs

Technical data	EM 2 x Analog Inputs / 1 x Analog Outputs (AM03)	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Number of outputs	1	2
Type	Voltage or current	Voltage or current
Range	±10 V or 0 to 20 mA	±10 V or 0 to 20 mA
Resolution	Voltage mode: 11 bits + sign Current mode: 11 bits	Voltage mode: 11 bits + sign Current mode: 11 bits
Full scale range (data word)	Voltage: -27,648 to 27,648 Current: 0 to 27,648	Voltage: -27,648 to 27,648 Current: 0 to 27,648
Accuracy (25 °C / 0 to 55 °C)	±0.5% / ±1.0% of full scale	±0.5% / ±1.0% of full scale
Settling time (95% of new value)	Voltage: 300 µs (R), 750 µs (1 µF) Current: 600 µs (1 mH), 2 ms (10 mH)	Voltage: 300 µs (R), 750 µs (1 µF) Current: 600 µs (1 mH), 2 ms (10 mH)
Load impedance	Voltage: ≥ 1000 Ω Current: ≤ 500 Ω	Voltage: ≥ 1000 Ω Current: ≤ 500 Ω
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Isolation (field side to logic)	None	None
Cable length (max.), in meters	100 m twisted and shielded	100 m twisted and shielded

Table A- 110 Diagnostics

Model	EM 2 x Analog Inputs / 1 x Analog Outputs (AM03)	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Overflow/underflow	Yes	Yes
Short to ground (voltage mode only)	Yes	Yes
Wire break (current mode only)	Yes	Yes
24 V DC low voltage	Yes	Yes

EM AM03 wiring current transducers

Wiring current transducers are available as 2-wire transducers and 4-wire transducers as shown below.

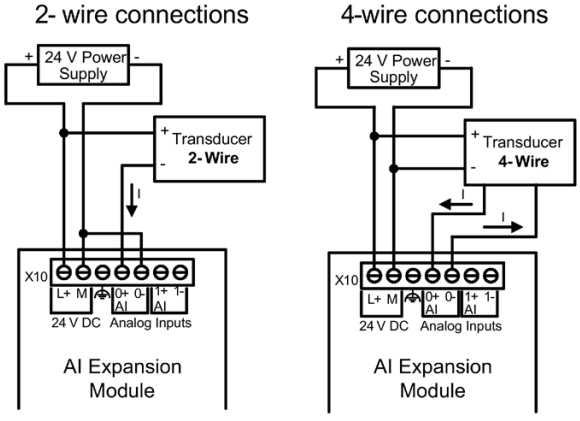


Table A- 111 Wiring diagrams for the EM AM03 2 x Analog Inputs / 1 x Analog Outputs (6ES7288-3AM03-0AA and the EM AM06 4 x Analog Inputs / 2 x Analog Outputs (6ES7288-3AM06-0AA0)

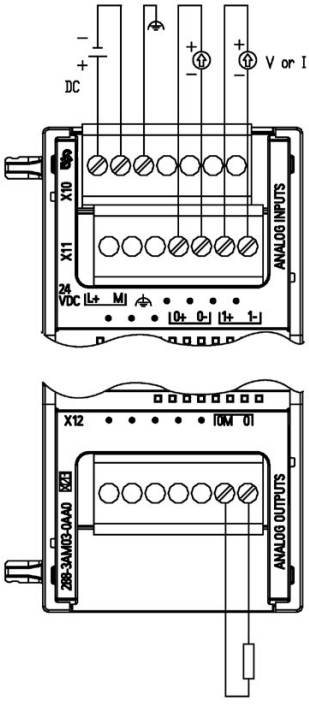
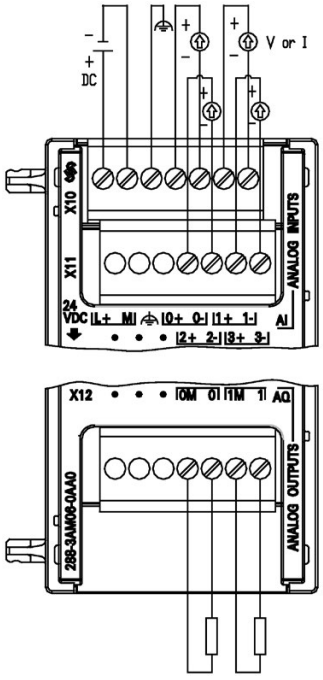
EM AM03 2 x Analog Inputs / 1 x Analog Outputs (6ES7288-3AM03-0AA0)	EM AM06 4 x Analog Inputs / 2 x Analog Outputs (6ES7288-3AM06-0AA0)
	
Note: Connectors must be gold. See Appendix F, Spare parts and other hardware, for article number.	

Table A- 112 Connector pin locations for AM03 2 x Analog Inputs / 1 x Analog Outputs (6ES7288-3AM03-0AA0)

Pin	X10 (gold)	X11 (gold)	X12 (gold)
1	L+ / 24 V DC	No connection	No connection
2	M / 24 V DC	No connection	No connection
3	Functional Earth	No connection	No connection
4	No connection	AI 0+	No connection
5	No connection	AI 0-	No connection
6	No connection	AI 1+	AQ 0M
7	No connection	AI 1-	AQ 0

Table A- 113 Connector pin locations for AM06 4 x Analog Inputs / 2 x Analog Outputs (6ES7288-3AM06-0AA0)

Pin	X10 (gold)	X11 (gold)	X12 (gold)
1	L+ / 24 V DC	No connection	No connection
2	M / 24 V DC	No connection	No connection
3	Functional Earth	No connection	No connection
4	AI 0+	AI 2+	AQ 0M
5	AI 0-	AI 2-	AQ 0
6	AI 1+	A1 3+	AQ 1M
7	AI 1-	A1 3-	AQ 1

A.4.4 Step response of the analog inputs

Table A- 114 Step response (ms), 0 to full-scale measured at 95%

Smoothing selection (sample averaging)	Noise reduction/rejection frequency (Integration time selection)			
	400 Hz (2.5 ms)	60 Hz (16.6 ms)	50 Hz (20 ms)	10 Hz (100 ms)
None (1 cycle): No averaging	4 ms	18 ms	22 ms	100 ms
Weak (4 cycles): 4 samples	9 ms	52 ms	63 ms	320 ms
Medium (16 cycles): 16 samples	32 ms	203 ms	241 ms	1200 ms
Strong (32 cycles): 32 samples	61 ms	400 ms	483 ms	2410 ms
Sample time				
• 4 AI x 13 bits	• 0.625 ms	• 4.17 ms	• 5 ms	• 25 ms
• 8 AI x 13 bits	• 1.25 ms	• 4.17 ms	• 5 ms	• 25 ms

A.4.5 Sample time and update times for the analog inputs

Table A- 115 Sample time and update time

Rejection frequency (Integration time)	Sample time	Module update time for all channels	
		4-channel SM	8-channel SM
400 Hz (2.5 ms)	• 4-channel SM: 0.625 ms • 8-channel SM: 1.250 ms	0.625 ms	1.250 ms
60 Hz (16.6 ms)	4.170 ms	4.17 ms	4.17 ms
50 Hz (20 ms)	5.000 ms	5 ms	5 ms
10 Hz (100 ms)	25.000 ms	25 ms	25 ms

A.4.6 Measurement ranges of the analog inputs for voltage and current (SB and EM)

Table A- 116 Analog input representation for voltage (SB and EM)

System		Voltage Measuring Range				
Decimal	Hexadecimal	±10 V	±5 V	±2.5 V	±1.25 V	
32767	7FFF[1]	11.851 V	5.926 V	2.963 V	1.481 V	Overflow
32512	7F00					
32511	7EFF	11.759 V	5.879 V	2.940 V	1.470 V	Overshoot range
27649	6C01					
27648	6C00	10 V	5 V	2.5 V	1.250 V	Rated range
20736	5100	7.5 V	3.75 V	1.875 V	0.938 V	
1	1	361.7 µV	180.8 µV	90.4 µV	45.2 µV	
0	0	0 V	0 V	0 V	0 V	
-1	FFFF					
-20736	AF00	-7.5 V	-3.75 V	-1.875 V	-0.938 V	
-27648	9400	-10 V	-5 V	-2.5 V	-1.250 V	
-27649	93FF					Undershoot range
-32512	8100	-11.759 V	-5.879 V	-2.940 V	-1.470 V	
-32513	80FF					Underflow
-32768	8000	-11.851 V	-5.926 V	-2.963 V	-1.481 V	

[1] 7FFF can be returned for one of the following reasons: overflow (as noted in this table), before valid values are available (for example immediately upon a power up), or if a wire break is detected.

Table A- 117 Analog input representation for current (SB and EM)

System		Current measuring range		
Decimal	Hexadecimal	0 mA to 20 mA	4 mA to 20 mA	
32767	7FFF	23.70 mA	22.96 mA	Overflow
32512	7F00			
32511	7EFF	23.52 mA	22.81 mA	Overshoot range
27649	6C01			
27648	6C00	20 mA	20 mA	Nominal range
20736	5100	15 mA	16 mA	
1	1	723.4 nA	4 mA + 578.7 nA	
0	0	0 mA	4 mA	
-1	FFFF			Undershoot range
-4864	ED00	-3.52 mA	1.185 mA	
-4865	ECFF			Underflow
-32768	8000			

A.4.7 Measurement ranges of the analog outputs for voltage and current (SB and EM)

Table A- 118 Analog output representation for voltage (SB and EM)

System		Voltage Output Range	
Decimal	**Hexadecimal**	**± 10 V**	
32767	7FFF	See note 1	Overflow
32512	7F00	See note 1	
32511	7EFF	11.76 V	Overshoot range
27649	6C01		
27648	6C00	10 V	Rated range
20736	5100	7.5 V	
1	1	361.7 µ V	
0	0	0 V	
-1	FFFF	-361.7 µ V	
-20736	AF00	-7.5 V	
-27648	9400	-10 V	
-27649	93FF		Undershoot range
-32512	8100	-11.76 V	
-32513	80FF	See note 1	Underflow
-32768	8000	See note 1	

[1] In an overflow or underflow condition, analog outputs will take on the substitute value of the STOP mode.

Table A- 119 Analog output representation for current (SB and EM)

System		Current output range		
Decimal	**Hexadecimal**	**0 mA to 20 mA**	**4 mA to 20 mA**	
32767	7FFF	See note 1	See note 1	Overflow
32512	7F00	See note 1	See note 1	
32511	7EFF	23.52 mA	22.81 mA	Overshoot range
27649	6C01			
27648	6C00	20 mA	20 mA	Rated range
20736	5100	15 mA	16 mA	
1	1	723.4 nA	4 mA + 578.7 nA	
0	0	0 mA	4mA	
-1	FFFF		4 mA to 578.7 nA	Undershoot range
-6912	E500		0 mA	
-6913	E4FF			Not possible. Output value limited to 0 mA.
-32512	8100			
-32513	80FF	See note 1	See note 1	Underflow
-32768	8000	See note 1	See note 1	

[1] In an overflow or underflow condition, analog outputs will take on the substitute value of the STOP mode.

A.5 Thermocouple and RTD expansion modules (EMs)

A.5.1 Thermocouple expansion modules (EMs)

A.5.1.1 EM AT04 thermocouple specifications

Table A- 120 General specifications

Model	EM AT04 AI 4 x 16 bit TC
Article number	6ES7288-3AT04-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81
Weight	125 grams
Power dissipation	1.5 W
Current consumption (SM Bus)	80 mA
Current consumption (24 V DC) [1]	40 mA

[1] 20.4 to 28.8 V DC (Class 2, Limited Power, or sensor power from PLC)

Table A- 121 Analog inputs

Model		EM AT04 AI 4 x 16 bit TC
Number of inputs		4
Range Nominal range (data word) Overrange/underrange (data word) Overflow/underflow (data word)		See Thermocouple selection table.
Resolution	Temperature	0.1 °C / 0.1 °F
	Voltage	15 bits plus sign
Maximum withstand voltage		± 35
Noise rejection		85 dB for selected filter setting (10 Hz, 50 Hz, 60 Hz or 400 Hz)
Common mode rejection		> 120 dB at 120 V AC
Impedance		≥ 10 MΩ
Isolation	Field to logic	500 V AC
	Field to 24 V DC	500 V AC
	24 V DC to logic	500 V AC
Channel to channel isolation		--
Accuracy		See Thermocouple selection table.
Repeatability		±0.05% FS
Measuring principle		Integrating

Model	EM AT04 AI 4 x 16 bit TC
Module update time	See Filter selection table.
Cold junction error	±1.5 °C
Cable length (meters)	100 m to sensor max.
Wire resistance	100 Ω max.

Table A- 122 Diagnostics

Model	EM AT04 AI 4 x 16 bit TC
Overflow/underflow [1]	Yes
Wire break (current mode only) [2]	Yes
24 V DC low voltage [1]	Yes

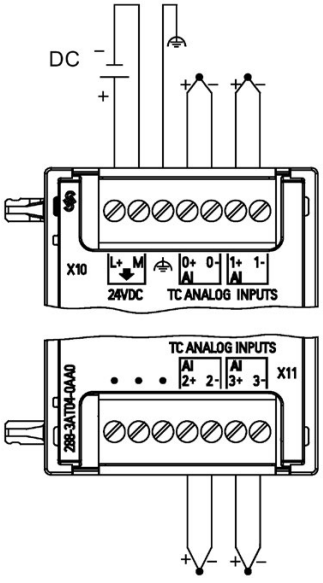
[1] The overflow, underflow and low voltage diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

[2] When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The EM AT04 Thermocouple (TC) analog expansion module measures the value of voltage connected to the module inputs. The temperature measurement type can be either "Thermocouple" or "Voltage".

- "Thermocouple": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).
- "Voltage": The nominal range full scale value will be decimal 27648.

Table A- 123 Wiring diagram for the EM AT04 Thermocouple 4 x 16 bit (6ES7288-3AT04-0AA0)

EM AT04 4 x 16 bit (6ES7288-3AT04-0AA0)	
	
Note: Connectors must be gold. See Appendix F, Spare Parts and other hardware for article number.	

① TC 2, 3, 4, and 5 not shown connected for clarity.

Table A- 124 Connector pin locations for EM AT04 4 x 16 bit (6ES7288-3AT04-0AA0)

Pin	X10 (gold)	X11 (gold)
1	L+ / 24 V DC	No connection
2	M / 24 V DC	No connection
3	Functional Earth	No connection
4	AI 0+ /TC	AI 2+ /TC
5	AI 0- /TC	AI 2- /TC
6	AI 1+ /TC	AI 3+ /TC
7	AI 1- /TC	AI 3- /TC

Note

Unused analog inputs should be shorted.

The thermocouple unused channels can be deactivated. No error will occur if an unused channel is deactivated.

Thermocouples are formed whenever two dissimilar metals are electrically bonded to each other. A voltage is generated that is proportional to the junction temperature. This voltage is small; one microvolt could represent many degrees. Measuring the voltage from a thermocouple, compensating for extra junctions, and then linearizing the result forms the basis of temperature measurement using thermocouples.

When you connect a thermocouple to the EM AT04 Thermocouple module, the two dissimilar metal wires are attached to the module at the module signal connector. The place where the two dissimilar wires are attached to each other forms the sensor thermocouple.

Two more thermocouples are formed where the two dissimilar wires are attached to the signal connector. The connector temperature causes a voltage that adds to the voltage from the sensor thermocouple. If this voltage is not corrected, then the temperature reported will deviate from the sensor temperature.

Cold junction compensation is used to compensate for the connector thermocouple. Thermocouple tables are based on a reference junction temperature, usually zero degrees Celsius. The cold junction compensation compensates the connector to zero degrees Celsius. The cold junction compensation restores the voltage added by the connector thermocouples. The temperature of the module is measured internally, and then converted to a value to be added to the sensor conversion. The corrected sensor conversion is then linearized using the thermocouple tables.

For optimum operation of the cold junction compensation, the thermocouple module must be located in a thermally stable environment. Slow variation (less than 0.1 °C/minute) in ambient module temperature is correctly compensated within the module specifications. Air movement across the module will also cause cold junction compensation errors.

If better cold junction error compensation is needed, an external iso-thermal terminal block may be used. The thermocouple module provides for use of a 0 °C referenced or 50 °C referenced terminal block.

The ranges and accuracy for the different thermocouple types supported by the EM AT04 Thermocouple expansion module are shown in the table below.

Table A- 125 EM AT04 Thermocouple selection table

Type	Under-range minimum[1]	Nominal range low limit	Nominal range high limit	Over-range maximum[2]	Normal range [3, 4] accuracy @ 25 °C	Normal range [1, 2] accuracy -20 °C to 55 °C
J	-210.0 °C	-150.0 °C	1200.0 °C	1450.0 °C	±0.3 °C	±0.6 °C
K	-270.0 °C	-200.0 °C	1372.0 °C	1622.0 °C	±0.4 °C	±1.0 °C
T	-270.0 °C	-200.0 °C	400.0 °C	540.0 °C	±0.5 °C	±1.0 °C
E	-270.0 °C	-200.0 °C	1000.0 °C	1200.0 °C	±0.3 °C	±0.6 °C
R & S	-50.0 °C	100.0 °C	1768.0 °C	2019.0 °C	±1.0 °C	±2.5 °C
B	0.0 °C	200.0 °C	800.0 °C	--	±2.0 °C	±2.5 °C
	--	800.0 °C	1820.0 °C	1820 °C	±1.0 °C	±2.3 °C
N	-270.0 °C	-200.0 °C	1300.0 °C	1550.0 °C	±1.0 °C	±1.6 °C
C	0.0 °C	100.0 °C	2315.0 °C	2500.0 °C	±0.7 °C	±2.7 °C

Type	Under-range minimum[1]	Nominal range low limit	Nominal range high limit	Over-range maximum[2]	Normal range [3, 4] accuracy @ 25 °C	Normal range [1, 2] accuracy -20 °C to 55 °C
TXK/XK(L)	-200.0 °C	-150.0 °C	800.0 °C	1050 °C	±0.6 °C	±1.2 °C
Voltage	-32512	-27648 -80mV	27648 80mV	32511	±0.05%	±0.1%

1 Thermocouple values below the under-range minimum value are reported as -32768.

2 Thermocouple values above the over-range maximum value are reported as 32767.

3 Internal cold junction error is ±1.5 °C for all ranges. This adds to the error in this table. The module requires at least 30 minutes of warm-up time to meet this specification.

4 In the presence of radiated radio frequency of 970 MHz to 990 MHz, the accuracy of the EM AT04 AI 4 x 16 bit TC may be degraded.

Note

Thermocouple channel

Each channel on the Thermocouple expansion module can be configured with a different thermocouple type (selectable in the software during configuration of the module).

Table A- 126 Noise reduction and update times for the EM AT04 Thermocouple

Rejection frequency selection	Integration time	4 Channel module update time (seconds)
400 Hz (2.5 ms)	10 ms [1]	0.143
60 Hz (16.6 ms)	16.67 ms	0.223
50 Hz (20 ms)	20 ms	0.263
10 Hz (100 ms)	100 ms	1.225

1 To maintain module resolution and accuracy when 400 Hz rejection is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

It is recommended for measuring thermocouples that a 100 ms integration time be used. The use of smaller integration times will increase the repeatability error of the temperature readings.

Note

After power is applied, the module performs internal calibration for the analog-to-digital converter. During this time the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time. Because the configuration of the module can vary the length of the initialization time, you should verify the behavior of the module in your configuration. If required, you can include logic in your user program to accommodate the initialization time of the module.

Representation of analog values for Thermocouple Type J

A representation of the analog values of thermocouples type J is shown in the table below.

Table A- 127 Representation of analog values of thermocouples type J

Type J in °C	Units		Type J in °F	Units		Range
	Decimal	Hexadecimal		Decimal	Hexadecimal	
> 1450.0	32767	7FFF	> 2642.0	32767	7FFF	Overflow
1450.0 : 1200.1	14500 : 12001	38A4 : 2EE1	2642.0 : 2192.2	26420 : 21922	6734 : 55A2	Overrange
1200.0 : -150.0	12000 : -1500	2EE0 : FA24	2192.0 : -238.0	21920 : -2380	55A0 : F6B4	Rated range
-150.1 : -210.0	-1501 : -2100	FA23 : F7CC	-238.2 : -346.0	-2382 : -3460	F6B2 : F27C	Underrange
< -210.0	-32768	8000	< -346.0	-32768	8000	Underflow[1]

[1] Faulty wiring (for example, polarity reversal, or open inputs) or sensor error in the negative range (for example, wrong type of thermocouple) may cause the thermocouple module to signal underflow.

A.5.2 RTD expansion modules (EMs)

EM RTD specifications

Table A- 128 General specifications

Technical data	EM RTD 2 x 16 bit (EM AR02)	EM RTD 4 x 16 bit (EM AR04)
Article number	6ES7288-3AR02-0AA0	6ES7288-3AR04-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81
Weight	148.7 grams	150 grams
Power dissipation	1.5 W	1.5 W
Current consumption (SM Bus)	80 mA	80 mA
Current consumption (24 V DC) [1]	40 mA	40 mA

Table A- 129 Analog inputs

Technical data	EM RTD 2 x 16 bit (EM AR02)	EM RTD 4 x 16 bit (EM AR04)
Number of inputs	2	4
Type	Module referenced RTD and Ω	Module referenced RTD and Ω

Technical data		EM RTD 2 x 16 bit (EM AR02)	EM RTD 4 x 16 bit (EM AR04)
Range Nominal range (data word) Overshoot/undershoot range (data word) Overflow/underflow (data word)		See RTD Sensor selection table	See RTD Sensor selection table
Resolution	Temperature	0.1 °C / 0.1 °F	0.1 °C / 0.1 °F
	Resistance	15 bits + sign bit	15 bits + sign bit
Maximum withstand voltage		±35 V	±35 V
Noise rejection		85 dB at 10 Hz / 50 Hz /60 Hz / 400 Hz	85 dB at 10 Hz / 50 Hz /60 Hz / 400 Hz
Common mode rejection		>120 dB	>120 dB
Impedance		≥10 M Ω	≥10 M Ω
Isolation	Field side to logic	500 V AC	500 V AC
	Field to 24 V DC	500 V AC	500 V AC
	24 V DC to logic	500 V AC	500 V AC
Channel to channel isolation		0	0
Accuracy		See RTD Sensor selection table	See RTD Sensor selection table
Repeatability		±0.05% FS	±0.05% FS
Maximum sensor dissipation		0.5 m W	0.5 m W
Measuring principle		Sigma-delta	Sigma-delta
Module update time		See Noise reduction selection table	See Noise reduction selection table
Cable length (max.), in meters		100 m to sensor max.	100 m to sensor max.
Wire resistance (max.)	except 10 Ω RTD	20 Ω	20 Ω
	10 Ω RTD	2.7 Ω	2.7 Ω

Table A- 130 Diagnostics

Technical data	EM RTD 2 x 16 bit (EM AR02)	EM RTD 4 x 16 bit (EM AR04)
Overflow/underflow [1,2]	Yes	Yes
Wire break [3]	Yes	Yes
24 V DC low voltage [1]	Yes	Yes

1 The overflow, underflow and low voltage diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

2 For resistance ranges underflow detection is never enabled.

3 When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The EM RTD analog expansion module measures the value of resistance connected to the module inputs. The measurement type can be selected as either "Resistor" or "Thermal resistor".

- "Resistor": The nominal range full scale value will be decimal 27648.
- "Thermal resistor": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).

The EM RTD module supports measurements with 2-wire, 3-wire and 4-wire connections to the sensor resistor.

Table A- 131 Ranges and accuracy for the different sensors supported by the RTD expansion module

Temperature coefficient	RTD type	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range accuracy @ 25 °C	Normal range accuracy -20 ° C to 60 °C
Pt 0.003850	Pt 10	-243.0 °C	-200.0 °C	850.0 °C	1000.0 °C	±1.0 °C	±2.0 °C
ITS90 DIN EN 60751	Pt 50	-243.0 °C	-200.0 °C	850.0 °C	1000.0 °C	±0.5 °C	±1.0 °C
	Pt 100						
	Pt 200						
	Pt 500						
	Pt 1000						
Pt 0.003902 Pt 0.003916 Pt 0.003920	Pt 100	-243.0 °C	-200.0 °C	850.0 °C	1000.0 °C	± 0.5 °C	±1.0 °C
	Pt 200	-243.0 °C	-200.0 °C	850. 0°C	1000. 0°C	± 0.5 °C	±1.0 °C
	Pt 500						
	Pt 1000						
Pt 0.003910	Pt 10	-273.2 °C	-240.0 °C	1100. 0°C	1295 °C	±1.0 °C	±2.0 °C
	Pt 50	-273.2 °C	-240.0 °C	1100.0 °C	1295 °C	±0.8 °C	±1.6 °C
	Pt 100						
	Pt 500						
Ni 0.006720 Ni 0.006180	Ni 100	-105.0 °C	-60.0 °C	250.0 °C	295.0 °C	±0.5 °C	±1.0 °C
	Ni 120						
	Ni 200						
	Ni 500						
	Ni 1000						
LG-Ni 0.005000	LG-Ni 1000	-105.0 °C	-60.0 °C	250.0 °C	295.0 °C	±0.5 °C	±1.0 °C
Ni 0.006170	Ni 100	-105.0 °C	-60.0 °C	180.0 °C	212.4 °C	±0.5 °C	±1.0 °C
Cu 0.004270	Cu 10	-240.0 °C	-200.0 °C	260.0 °C	312.0 °C	±1.0 °C	±2.0 °C
Cu 0.004260	Cu 10	-60.0 °C	-50.0 °C	200.0 °C	240.0 °C	±1.0 °C	±2.0 °C
	Cu 50	-60.0 °C	-50.0 °C	200.0 °C	240.0 °C	±0.6 °C	±1.2 °C
	Cu 100						
Cu 0.004280	Cu 10	-240.0 °C	-200.0 °C	200.0 °C	240.0 °C	±1.0 °C	±2.0 °C
	Cu 50	-240.0 °C	-200.0 °C	200.0 °C	240.0 °C	±0.7 °C	±1.4 °C
	Cu 100						

Temperature coefficient	RTD type	Under range minimum[1]	Nominal range low limit	Nominal range high limit	Over range maximum[2]	Normal range accuracy @ 25 °C	Normal range accuracy -20 ° C to 60 °C

[1] RTD values below the under-range minimum value report -32768.

[2] RTD values above the over-range maximum value report +32767.

Table A- 132 Resistance

Range	Under range minimum	Nominal range low limit	Nominal range high limit	Over range maximum[1]	Normal range accuracy @ 25 °C	Normal range accuracy -20 °C to 60°C
150 Ω	n/a	0 (0 Ω)	27648 (150 Ω)	176.383 Ω	±0.05%	±0.1%
300 Ω	n/a	0 (0 Ω)	27648 (300 Ω)	352.767 Ω	±0.05%	±0.1%
600 Ω	n/a	0 (0 Ω)	27648 (600 Ω)	705.534 Ω	±0.05%	±0.1%

[1] Resistance values above the over-range minimum value are reported as +32767.

Note

The module reports 32767 on any activated channel with no sensor connected. If open wire detection is also enabled, the module flashes the appropriate red LEDs.

When 500 Ω and 1000 Ω RTD ranges are used with other lower value resistors, the error may increase to two times the specified error.

Best accuracy will be achieved for the 10 Ω RTD ranges if 4 wire connections are used.

The resistance of the connection wires in 2 wire mode will cause an error in the sensor reading and therefore accuracy is not guaranteed.

Table A- 133 Noise reduction and update times for the RTD module

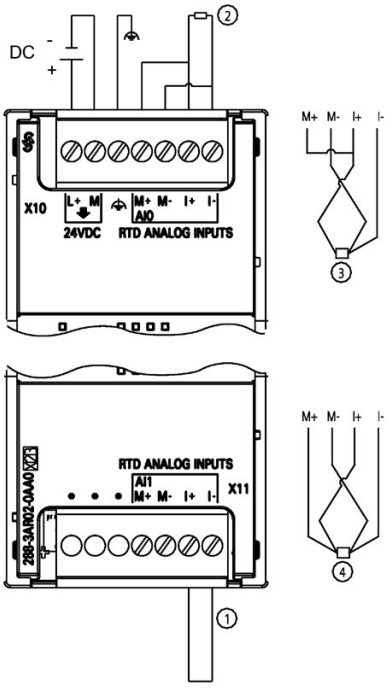
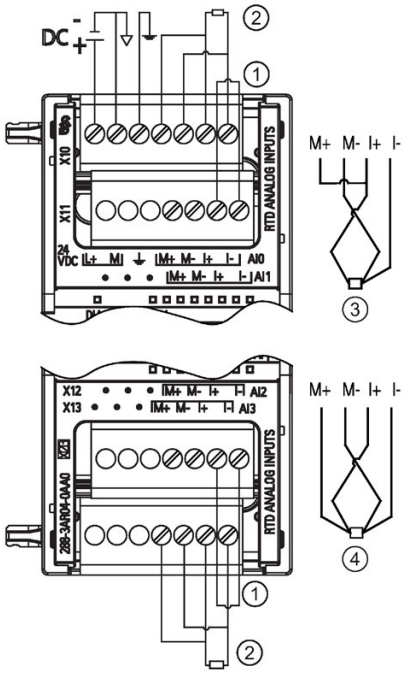
Rejection frequency selection	Integration time	Update time (seconds)
400 Hz (2.5 ms)	10 ms [1]	4-/2-wire: 0.142 3-wire: 0.285
60 Hz (16.6 ms)	16.67 ms	4-/2-wire: 0.222 3-wire: 0.445
50 Hz (20 ms)	20 ms	4-/2-wire: 0.262 3-wire: .505
10 Hz (100 ms)	100 ms	4-/2-wire: 1.222 3-wire: 2.445

[1] To maintain module resolution and accuracy when the 400 Hz filter is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

Note

After power is applied, the module performs internal calibration for the analog-to-digital converter. During this time the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time. Because the configuration of the module can vary the length of the initialization time, you should verify the behavior or the module in your configuration. If required, you can include logic in your user program to accommodate the initialization time of the module.

Table A- 134 Wiring diagrams for the EM AR02 RTD 2 x 16 bit (6ES7288-3AR02-0AA0) and EM AR04 RTD 4 x 16 bit (6ES7288-3AR04-0AA0)

EM AR02 RTD 2 x 16 bit (6ES7288-3AR02-0AA0)	EM AR04 RTD 4 x 16 bit (6ES7288-3AR04-0AA0)
	

Note: Connectors must be gold. See Appendix F, Spare parts and other hardware, for article number.

① Loop-back unused RTD inputs

② 2-wire RTD ③ 3-wire RTD ④ 4-wire RTD

Note: Connectors must be gold. See Appendix F, Spare parts and other hardware for article number.

Table A- 135 Connector pin locations for EM AR02 RTD 2 x 16 bit (6ES7288-3AR02-0AA0)

Pin	X10 (gold)	X11 (gold)
1	L+ / 24 V DC	No connection
2	M / 24 V DC	No connection
3	Functional Earth	No connection
4	AI 0 M+/RTD	AI 1 M+/RTD
5	AI 0 M-/RTD	AI 1 M-/RTD
6	AI 0 I+/RTD	AI 1 I+/RTD
7	AI 0 I-/RTD	AI 1 I-/RTD

Table A- 136 Connector pin locations for EM AR04 RTD 4 x 16 bit (6ES7288-3AR04-0AA0)

Pin	X10 (gold)	X11 (gold)	X12 (gold)	X13 (gold)
1	L+ / 24 V DC	No connection	No connection	No connection
2	M / 24 V DC	No connection	No connection	No connection
3	Functional Earth	No connection	No connection	No connection
4	AI 0 M+/RTD	AI 1 M+/RTD	AI 2 M+/RTD	AI 3 M+/RTD
5	AI 0 M-/RTD	AI 1 M-/RTD	AI 2 M-/RTD	AI 3 M-/RTD
6	AI 0 I+/RTD	AI 1 I+/RTD	AI 2 I+.RTD	AI 3 I+/RTD
7	AI 0 I-/RTD	AI 1 I-/RTD	AI 2 I-/RTD	AI 3 I/-/RTD

A.6 Digital signal boards

A.6.1 SB DT04 digital input/output specifications

Table A- 137 General specifications

Technical data	SB Digital 2 x Inputs / 2 x Digital Outputs (DT04)
Article number	6ES7288-5DT04-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	18.1 grams
Power dissipation	1.0 W
Current consumption (5 V DC)	50 mA
Current consumption (24 V DC)	4 mA / Input used

Table A- 138 Digital inputs

Technical data	SB Digital 2 x Inputs / 2 x Digital Outputs (DT04)
Number of inputs	2
Type	Sink (IEC Type 1 sink)
Rated voltage	24 V DC at 4 mA, nominal
Continuous permissible voltage	30 V DC max.
Surge voltage	35 V DC for 0.5 sec.
Logic 1 signal (min.)	15 V DC at 2.5 mA
Logic 0 signal (max.)	5 V DC at 1 mA
Isolation (field side to logic)	500 V AC for 1 minute
Isolation groups	1
Filter times	Individually selectable on each channel: µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8
Number of inputs on simultaneously	2
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs

Table A- 139 Digital outputs

Technical data	SB Digital 2 x Inputs / 2 x Digital Outputs (DT04)
Number of outputs	2
Output type	Solid state - MOSFET (sourcing)
Voltage range	20.4 to 28.8 V DC
Logic 1 signal at max. current	20 V DC min.
Logic 0 signal at max. current	0.1 V DC max.
Rated current per point (max.)	0.5 A
Rated current per common (max.)	1 A
Lamp load	5 W
On state contact resistance	0.6 Ω max.
Leakage current per point	10 µA max.
Surge current	5 A for 100 ms max.
Overload protection	No
Isolation (field side to logic)	500 V AC for 1 minute
Isolation groups	1
Inductive clamp voltage	L+ minus 48 V, 1 W dissipation
Switching delay	2 µs max. off to on 10 µs max. on to off
Output behavior in STOP	Last value or substitute value (default value 0)

Technical data	SB Digital 2 x Inputs / 2 x Digital Outputs (DT04)
Number of outputs on simultaneously	2
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 150 m normal inputs

Table A- 140 Wiring diagram for the SB DT04 2 Digital Input/2 Digital Output (6ES7288-5DT04-0AA0)

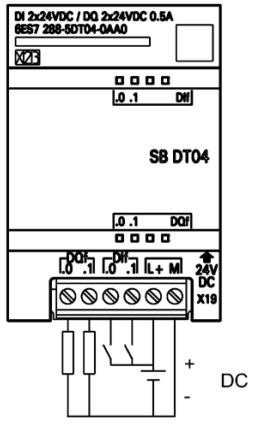
SB DT04 2 Digital Input/2 Digital Output (6ES7288-5DT04-0AA0)	
	

Table A- 141 Connector pin locations for SB DT04 2 Digital Input/2 Digital Output (6ES7288-5DT04-0AA0)

Pin	X19
1	DQ f.0
2	DQ f.1
3	DI f.0
4	DI f.1
5	L+ / 24 V DC
6	M / 24 V DC

A.7 Analog signal boards

A.7.1 SB AE01 analog input specifications

Table A- 142 General specifications

Technical data	SB Analog 1 x Input (SB AE01)
Article number	6ES7288-5AE01-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	20 grams
Power dissipation	0.4 W
Current consumption (5 V DC)	50 mA (5 V and 3.3 combined)
Current consumption (24 V DC)	None

Table A- 143 Analog inputs

Technical data		SB Analog 1 x input (SB AE01)
Number of inputs		1
Type		Voltage or current (differential)
Range		±10 V, ±5 V, ±2.5 V, or 0 to 20 mA
Resolution		11 bits + sign bit (voltage mode) 11 bits (current mode)
Full scale range (data word)		-27,648 to 27,648
Accuracy (25 °C / 0 to 50 °C)		Voltage mode: ±0.3 % / ±0.6 % of full scale Current mode: ±0.3 % / ±0.6 % of full scale
Overshoot/undershoot range (data word)		Voltage: 27,649 to 32,511 / -27,649 to -32,512 Current: 27,649 to 32,511 / -4864 to 0 (Refer to Analog input representation for voltage and Analog input representation for current (Page 823).)
Overflow/underflow (data word)		Voltage: 32,512 to 32,767 / -32,513 to -32,768 Current: 32,512 to 32,767 / -4,865 to -32,768 (Refer to Analog input representation for voltage and Analog input representation for current (Page 823).)
Maximum withstand voltage / current		±35 V / ±40 mA
Smoothing		None, weak, medium, or strong (Refer to Analog input response times for step response time (Page 822).)
Noise rejection		400, 60, 50, or 10 Hz
Measuring principle common mode rejection		40 dB, DC to 60 Hz rejection
Operational signal range (signal plus common mode voltage)		Signal plus common mode voltage must be less than +35 V and greater than -35 V
Input impedance		
	Differential mode	220 KΩ (voltage) / 250 Ω (current)
	Common mode	55 KΩ (voltage) / 55 Ω (current)

Technical data	SB Analog 1 x input (SB AE01)
Isolation (field side to logic)	None
Cable length (meters)	100 m twisted and shielded

Table A- 144 Diagnostics

Model	SB Analog 1 x input (SB AE01)
Overflow/underflow	Yes
24 V DC low voltage	None

SB AE01 wiring current transducers

Wiring current transducers are available as 2-wire transducers and 4-wire transducers as shown below.

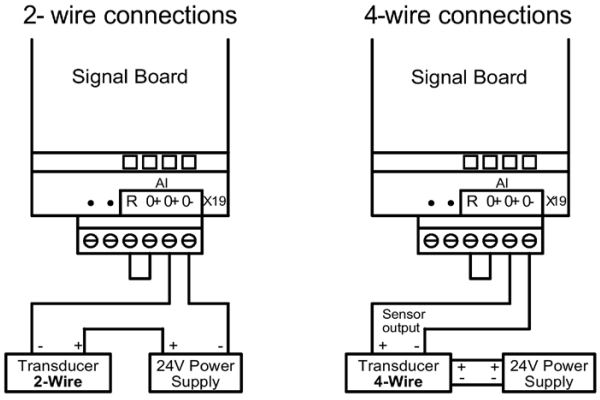


Table A- 145 Wiring diagram for the SB AE01 Analog 1 x Input (6ES7288-5AE01-0AA0)

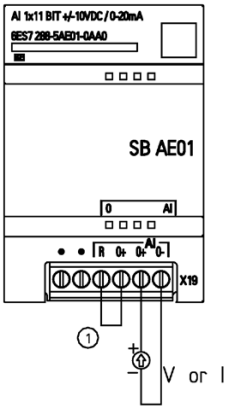
SB AE01 - SB Analog Input 1 x Input (6ES7288-5AE01-0AA0)	
	① Connect "R" and "0+" for current applications. Note: Connectors must be gold. See Appendix F, Spare parts and other hardware for article number.

Table A- 146 Connector pin locations for SB AE01 Analog Input 1 x Input (6ES7288-5AE01-0AA0)

Pin	X19
1	No connection
2	No connection
3	AI R
4	AI 0+
5	AI 0+
6	AI 0-

A.7.2 SB AQ01 analog output specifications

Table A- 147 General specifications

Technical data	SB Analog 1 x Output (SB AQ01)
Article number	6ES7288-5AQ01-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	17.4 grams
Power dissipation	1.5 W
Current consumption (5 V DC)	15 mA
Current consumption (24 V DC)	40 mA (no load)

Table A- 148 Analog outputs

Technical data	SB Analog 1 x Output (SB AQ01)
Number of outputs	1
Type	Voltage or current
Range	±10 V, 0 to 20 mA
Resolution	Voltage: 11 bits + sign Current: 11 bits
Full scale range (data word) Refer to the output ranges for voltage and current.	-27,648 to 27,648 (-10 V to 10 V) 0 to 27,648 (0 to 20 mA)
Accuracy (25 °C / 0 to 55 °C)	±0.5% / ±1%
Settling time (95% of new value)	Voltage: 300 µs (R), 750 µs (1 µF) Current: 600 µs (1mH), 2 ms (10 mH)
Load impedance	Voltage: ≥ 1000 ohms Current: ≤ 600 ohms
Output behavior in STOP	Last value, substitute value (default value 0)
Isolation (field side to logic)	None
Cable length (max.), in meters	10 m twisted and shielded

Table A- 149 Diagnostics

Technical data	SB Analog 1 x Output (SB AQ01)
Overflow/underflow	Yes
Short to ground (voltage mode only)	Yes
Wire break (current mode only)	Yes

Table A- 150 Wiring diagram for the SB AQ01 Analog 1 x Output (6ES7288-5AQ01-0AA0)

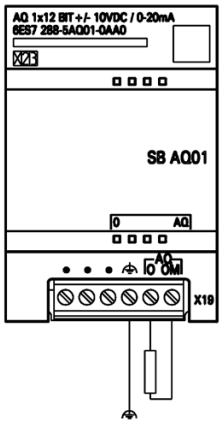
SB AQ01 Analog 1 x Output (6ES7288-5AQ01-0AA0)	
	

Table A- 151 Connector pin locations for SB AQ01 Analog 1 x Output (6ES7288-5AQ01-0AA0)

Pin	X19
1	No connection
2	No connection
3	No connection
4	Functional Earth
5	AQ 0
6	AQ 0M

A.8 RS485/RS232 signal boards

A.8.1 SB RS485/RS232 specifications

Table A- 152 General specifications

Technical data	SB RS485/RS232
Article number	6ES7288-5CM01-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	18.2 grams
Power dissipation	0.5 W
Current consumption (5 V DC)	50 mA
Current consumption (24 V DC)	N/A

Table A- 153 RS485 Transmitter and receiver

Technical data	SB RS485/RS232
Common mode voltage range	-7 V to +12 V, 1 second, 3 VRMS continuous
Transmitter differential output voltage	2 V min. at RL = 100 Ω 1.5 V min. at RL = 54 Ω
Termination and bias	4.7 K Ω to +5 V on TXD 4.7 K Ω to GND on RXD
Receiver input impedance	12 K Ω min.
Receiver threshold/sensitivity	+/- 0.2 V min. 60 mV typical hysteresis
Isolation RS 485 signal to chassis ground RS485 signal to CPU logic common	None
Cable length, shielded	With isolated repeater: 1000 m up to 187.5 Kbps Without isolated repeater: 50 m

Table A- 154 RS232 Transmitter and receiver

Technical data	SB RS485/RS232
Transmitter output voltage	+/-5 V min. at RL = 3K Ω
Transmit output voltage	+/- 15 V DC max.
Receiver input impedance	3 K Ω min.
Receiver threshold/sensitivity	0.8 V min. low, 2.4 max. high 0.5 V typical hysteresis
Receiver input voltage	+/- 30 V DC max.

Technical data	SB RS485/RS232
Isolation RS232 signal to chassis ground RS232 signal to CPU logic common	None
Cable length, shielded	10 m max.

Table A- 155 Wiring diagram for the SB CM01 RS485/RS232 (6ES7288-5CM01-0AA0)

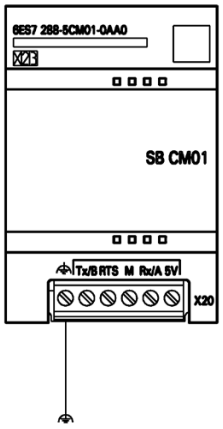
SB CM01 RS485/RS232 (6ES7288-5CM01-0AA0)	
	

Table A- 156 Connector pin locations for SB CM01 RS485/RS232 (6ES7288-5CM01-0AA0)

Pin	X20
1	Functional Earth
2	Tx/B
3	RTS
4	M
5	Rx/A
6	5 V Out (Bias Voltage)

A.9 Battery board signal boards (SBs)

A.9.1 SB BA01 Battery board

SB BA01 Battery board

The S7-200 SMART SB BA01 battery board provides for long-term backup of the real-time clock. The battery board plugs into the signal board slot of the S7-200 SMART CPU (firmware V2.0 and later versions). You must add the SB BA01 to the device configuration and download the hardware configuration to the CPU for the SB BA01 to gain access to the additional battery health reporting options.

When you purchase the SB BA01 battery board, it does not include the battery (type CR1025). You must purchase the battery separately.

Note

The SB BA01 is mechanically designed to fit the CPUs with the firmware V2.0 and later versions.

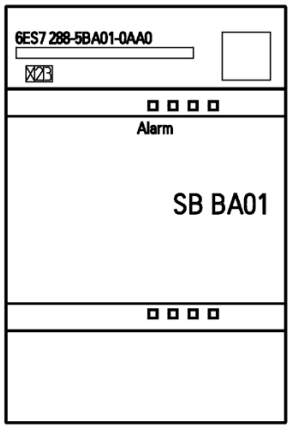
Table A- 157 General specifications

Technical data	SB BA01 Battery Board
Article number	6ES7288-5BA01-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	20 grams
Power dissipation	0.6 W
Current consumption (5 V DC)	18 mA
Current consumption (24 V DC)	None

Battery (not included)	SB BA01 Battery Board
Hold up time	Approximately 1 year
Battery type	CR1025 Refer to Installing or replacing a battery in the SB BA01 battery board (Page 50)
Nominal voltage	3 V
Nominal capacity	30 mAH

Diagnostics	SB BA01 Battery Board
Critical battery level	< 2.5 V
Battery diagnostic	Low voltage indicator: • Low battery voltage causes the LED on the BA01 panel to illuminate with the red light continuously ON. • Diagnostic alarm and/or digital output status of battery low condition available
Battery status	Battery status bit provided 0 = Battery OK 1 = Battery low
Battery status update	Battery status is updated at power up and then once per day while CPU is in RUN mode.

Table A- 158 Wiring diagram for the SB BA01 Battery board (6ES7288-5BA01-0AA0)

SB BA01 Battery board (6ES7288-5BA01-0AA0)	
	

A.10 EM DP01 PROFIBUS DP module

Table A- 159 General specifications

Technical data		EM DP01 PROFIBUS DP
Article number		6ES7288-7DP01-0AA0
Dimensions W x H x D (mm)		70 x 100 x 81
Weight		176.2 g
Power dissipation		1.5 W (no load)
V DC requirements		
	+5 V DC (SM Bus)	150 mA (no load)
	+24 V DC	See below

Table A- 160 EM features

Technical data	EM DP01 PROFIBUS DP module
Number of ports (limited power)	1
Electrical interface	RS485
PROFIBUS DP / MPI baud rates (set automatically)	9.6, 19.2, 45.45, 93.75, 187.5, and 500 Kbps; 1.5, 3, 6, and 12 Mbps
Protocols	PROFIBUS DP slave and MPI slave
Cable length	
Up to 93.7 Kbps	1200 m
187.5 Kbps	1000 m
500 Kbps	400 m
1 to 1.5 Mbps	200 m
3 to 12 Mbps	100 m
Network capabilities	
Station address settings	0 to 99 (set by rotary switches)
Maximum stations per segment	32
Maximum stations per network	126, up to 99 EM DP01 stations
MPI connections	6 total (1 reserved for an OP)

Table A- 161 Power supply

Technical data	EM DP01 PROFIBUS DP
24 V DC input power requirements	
Voltage range	20.4 to 28.8 V DC (Class 2, Limited Power, or sensor power from PLC)
Maximum current: Module only with port active Add 90 mA of 5 V port load Add 120 mA of 24 V port load	 30 mA 60 mA 180 mA
Ripple noise (< 10 MHz)	< 1 V peak to peak (maximum)
Isolation (field to logic) [1]	500 V AC for 1 minute
5 V DC power on communications port	
Maximum current per port	900 mA @ nominal 5 V
Current limit	2.7 A @ 5 V
Isolation (5 V DC to logic)	500 V AC for 1 minute
24 V DC power on communications port	
Voltage range	20.4 to 28.8 V DC
Maximum current per port	120 mA @ nominal 24 V

Technical data	EM DP01 PROFIBUS DP
24 V DC input power requirements	
Current limit	0.7 to 2.4 A
Isolation	Not isolated, same circuit as input 24 V DC

[1] No power is supplied to module logic by the 24 V DC supply. 24 V DC supplies power for the communications port.

A.10.1 S7-200 SMART CPUs that support the EM DP01 PROFIBUS DP module

The S7-200 SMART EM DP01 PROFIBUS DP module is an intelligent expansion module designed to work with the S7-200 SMART CPUs, firmware version 2.1 or later, shown in the following table.

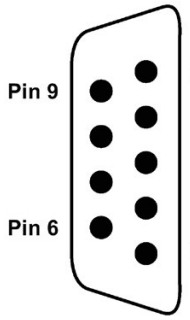
Table A- 162 EM DP01 PROFIBUS DP module compatibility with S7-200 SMART CPUs, firmware version 2.1 or later

CPU	Description
ST20	CPU ST20 (DC/DC/DC)
SR20	CPU SR20 (AC/DC/Relay)
ST30	CPU ST30 (DC/DC/DC)
SR30	CPU SR30 (AC/DC/Relay)
ST40	CPU ST40 (DC/DC/DC)
SR40	CPU SR40 (AC/DC/Relay)
ST60	CPU ST60 (DC/DC/DC)
SR60	CPU SR60 (AC/DC/Relay)

A.10.2 Connector pin assignments for EM DP01

The RS485 communication serial port on the EM DP01 is RS485-compatible on a nine-pin subminiature D female connector, in accordance with the PROFIBUS standard as defined in the European Standard EN 50170. The following table shows the connector that provides physical connection for the communication port and describes the communication port pin assignments.

Table A- 163 Pin assignments for the S7-200 SMART EM DP01

Pin Number	Connector	PROFIBUS
1		Shield
2		24 V Return
3		RS485 Signal B
4		Request-to-Send
5		5 V Return
6		+5 V (isolated)
7		+24 V
8		RS485 Signal A
9		NC

A.10.3 EM DP01 PROFIBUS DP module wiring diagram

Table A- 164 Wiring diagram for the EM DP01 PROFIBUS DP module (6ES7288-7DP01-0AA0

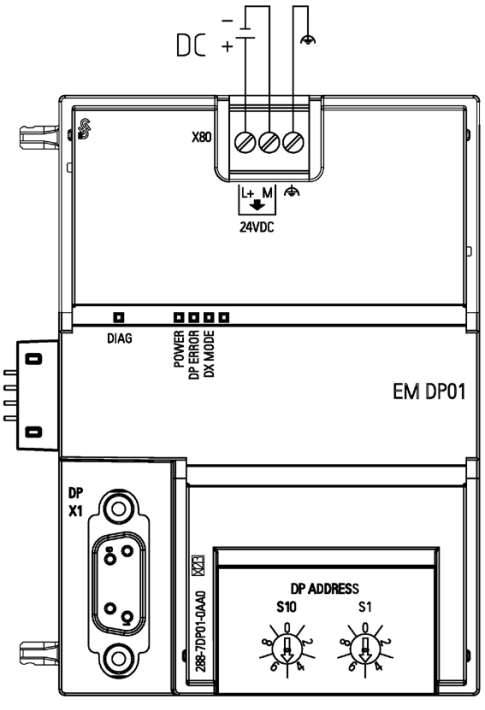
EM DP01 PROFIBUS DP module (6ES7288-7DP01-0AA0)	
	

Table A- 165 Connector pin locations for EM DP01 PROFIBUS DP module (6ES7288-7DP01-0AA0)

Pin	X80
1	L+ / 24 V DC
2	M / 24 V DC
3	Functional Earth

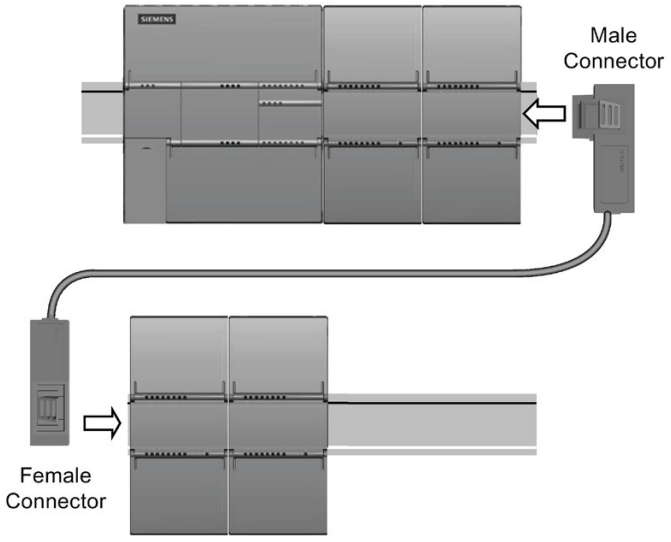
A.11 S7-200 SMART cables

A.11.1 S7-200 SMART I/O expansion cable

Table A- 166 S7-200 SMART Expansion cable

Technical Data	
Article number	6ES7288-6EC01-0AA0
Cable length	1 m
Weight	80 g

Refer to the installation section for information about installing and removing the S7-200 SMART expansion cable.



A.11.2 RS-232/PPI Multi-Master Cable and USB/PPI Multi-Master Cable

A.11.2.1 Overview

RS-232/PPI Multi-Master Cable and USB/PPI Multi-Master Cable Specifications

Table A- 167 General Specifications

Technical Data	RS-232/PPI Multi-Master cable	USB/PPI Multi-Master cable
Article number	6ES7901-3CB30-0XA0	6ES7901-3DB30-0XA0
Supply voltage	14.4 to 28.8 VDC	14.4 to 28.8 VDC
Supply current at 24 V nominal supply	60 mA RMS max.	50 mA RMS max.
Isolation	RS-485 to RS-232: 500 VDC	RS-485 to USB: 500 VDC

RS-485 Side Electrical Characteristics		
Common mode voltage range	-7 V to +12 V, 1 second, 3 V RMS continuous	-7 V to +12 V, 1 second, 3 V RMS continuous
Receiver input impedance	5.4 kΩ min. including termination	5.4 kΩ min. including termination
Termination/bias	10 kΩ to +5 V on B, PROFIBUS pin 3 10 kΩ to GND on A, PROFIBUS pin 8	10 kΩ to +5 V on B, PROFIBUS pin 3 10 kΩ to GND on A, PROFIBUS pin 8
Receiver threshold/sensitivity	±0.2 V, 60 mV typical hysteresis	±0.2 V, 60 mV typical hysteresis
Transmitter differential output voltage	2 V min. at R_L = 100 Ω, 1.5 V min. at R_L = 54 Ω	2 V min. at R_L = 100 Ω, 1.5 V min. at R_L =54 Ω

RS-232 Side Electrical Characteristics		
Receiver input impedance	3 kΩ min.	-
Receiver threshold/sensitivity	0.8 V min. low, 2.4 V max. high 0.5 V typical hysteresis	-
Transmitter output voltage	± 5 V min. at R_L = 3 kΩ	-

USB Side Electrical Characteristics		
Full speed (12 MB/s), Human Interface Device (HID)		
Supply current at 5 V	-	50 mA max.
Power down current	-	400 µA max.

Features

You can use the RS232/PPI Multi-Master cable with the S7-200 SMART CPUs in Freeport mode. See the Freeport mode for more information.

The USB cable requires the STEP 7-Micro/WIN SMART V2.3 (or later) programming software for operation.

A.11.2.2 RS-232/PPI Multi-Master Cable

Table A- 168 RS-232/PPI Multi-Master Cable - Pin-outs for RS-485 to RS-232 Local Mode Connector

RS-485 Connector Pin-out		RS-232 Local Connector Pin-out	
Pin Number	Signal description	Pin number	Signal Description
1	No connect	1	Data Carrier Detect (DCD) (not used)
2	24 V Return (RS-485 logic ground)	2	Receive Data (RD) (output from PC/PPI cable)
3	Signal B (RxD/TxD+)	3	Transmit Data (TD) (input to PC/PPI cable)
4	RTS (TTL level)	4	Data Terminal Ready (DTR)[1]
5	No connect	5	Ground (RS-232 logic ground)
6	No connect	6	Data Set Ready (DSR) [1]
7	24 V Supply	7	Request To Send (RTS) (not used)
8	Signal A (RxD/TxD-)	8	Clear To Send (CTS) (not used)
9	Protocol select	9	Ring Indicator (RI) (not used)

[1] Pins 4 and 6 are connected internally.

Table A- 169 RS-232/PPI Multi-Master Cable - Pin-outs for RS-485 to RS-232 Remote Mode Connector

RS-485 Connector Pin-out		RS-232 Remote Connector Pin-out [1]	
Pin Number	Signal Description	Pin Number	Signal Description
1	No connect	1	Data Carrier Detect (DCD) (not used)
2	24 V Return (RS-485 logic ground)	2	Receive Data (RD) (input to PC/PPI cable)
3	Signal B (RxD/TxD+)	3	Transmit Data (TD) (output from PC/PPI cable)
4	RTS (TTL level)	4	Data Terminal Ready (DTR) [2]
5	No connect	5	Ground (RS-232 logic ground)
6	No connect	6	Data Set Ready (DSR) [2]
7	24 V Supply	7	Request To Send (RTS) (output from PC/PPI cable)
8	Signal A (RxD/TxD-)	8	Clear To Send (CTS) (not used)
9	Protocol select	9	Ring Indicator (RI) (not used)

[1] A conversion from female to male, and a conversion from 9-pin to 25-pin is required for modems.

[2] Pins 4 and 6 are connected internally.

Use the RS-232/PPI Multi-Master Cable for Freeport operation

For connection directly to your personal computer:

- Set the PPI/Freeport mode (Switch 5=0)
- Set the baud rate (Switches 1, 2, and 3)
- Set Local (Switch 6=0). The Local setting is the same as setting the PC/PPI cable to DCE.
- Set the Protocol select to 10 Bit (Switch 7=1). See Note below.

For connection to a modem:

- Set the PPI/Freeport mode (Switch 5=0)
- Set the baud rate (Switches 1, 2, and 3)
- Set Remote (Switch 6=1). The Remote setting is the same as setting the PC/PPI cable to DTE.
- Set the Protocol select to 10 Bit (Switch 7 = 1). See Note below.

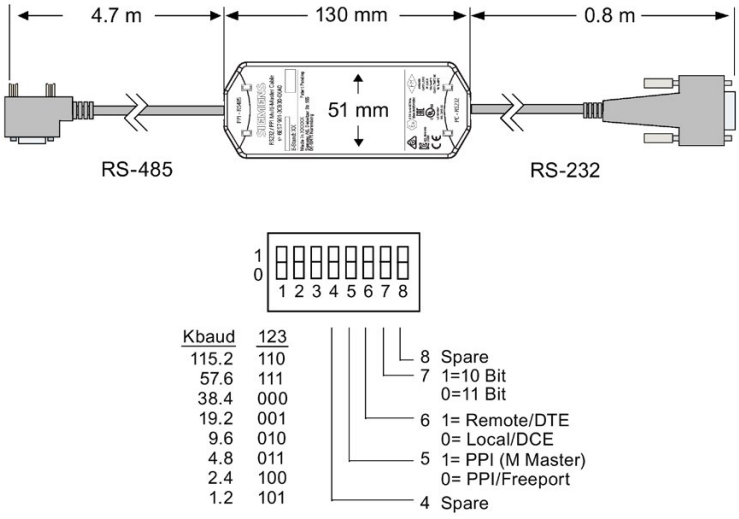
Note

The CRxxs CPUs require the protocol select switch 7 to be set to 1 to allow Freeport mode. Setting switch 7 to 0 disables Freeport mode in the CRxxs CPUs.

The SR and ST CPUs ignore the select switch 7 setting.

RS-232/PPI Multi-Master cable dimensions, label and LEDs

The following figure shows the RS-232/PPI Multi-Master Cable dimensions, label and LEDs.



LED	Color	Description
Tx	Green	RS-232 transmit indicator
Rx	Green	RS-232 receive indicator
PPI	Green	RS-485 transmit indicator

A.11.2.3 USB/PPI Multi-Master Cable

To use the USB cable, you must have STEP 7-Micro/WIN SMART V2.3 (or later) installed. It is recommended that you use the USB cable only with S7-200 SMART CPUs firmware version V2.3 (or later).

The USB cable does not support Freeport communications.

Note

If you use this cable with S7-200 SMART CPU versions prior to V2.3, certain operations are not possible (for example, downloading the user program).

Note

Attaching a USB-PPI cable to the CPU's RS485 port forces the CPU to exit Freeport mode and enable PPI mode. This allows STEP 7-Micro/WIN SMART V2.3 to regain control of the the CPU.

Table A- 170 USB/PPI Multi-Master cable - Pin-outs for the RS-485 to USB Series "A" Connector

RS-485 Connector Pin-out		USB Connector Pin-out	
Pin Number	**Signal Description**	**Pin Number**	**Signal Description**
1	No connect	1	USB -DataP
2	24 V Return (RS-485 logic ground)	2	USB -DataM
3	Signal B (RxD/TxD+)	3	USB 5 V
4	RTS (TTL level)	4	USB logic ground
5	No connect	-	-
6	No connect	-	-
7	24 V Supply	-	-
8	Signal A (RxD/TxD-)	-	-
9	Protocol select	-	-

The following figure shows theUSB/PPI Multi-Master Cable dimensions and LEDs.

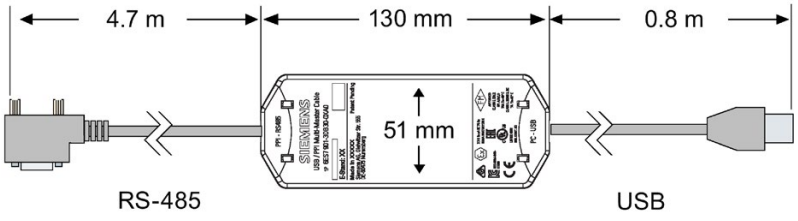


Figure A-1 USB/PPI Multi-Master Cable Dimensions and LEDs

LED	Color	Description
Tx	Green	USB transmit indicator
Rx	Green	USB receive indicator
PPI	Green	RS-485 transmit indicator

Calculating a power budget

B

B.1 Power budget

Your CPU has an internal power supply that provides power for the CPU, the expansion modules, signal boards, and other 24 V DC user power requirements. Use the following information as a guide for determining how much power (or current) the CPU can provide for your configuration. The new compact CPUs (CRs) do not support expansion modules or signal boards.

Refer to the technical specifications for your particular CPU to determine the 24 V DC sensor supply power budget, the 5 V DC logic budget supplied by your CPU and the 5 V DC power requirements of the expansion modules and signal boards. Refer to the Calculating a power budget to determine how much power (or current) the CPU can provide for your configuration.

The standard CPU provides the 5 V DC logic power needed for any expansion in your system. Pay careful attention to your system configuration to ensure that the CPU can supply the 5 V DC power required by your selected expansion modules. If your configuration requires more power than the CPU can supply, you must remove a module.

Note

If the CPU power budget is exceeded, you may not be able to connect the maximum number of modules allowed for your CPU.

The standard CPU also provides a 24 V DC sensor supply that can supply 24 V DC for input points, for relay coil power on the expansion modules, or for other requirements. If your power requirements exceed the budget of the sensor supply, then you must add an external 24 V DC power supply to your system. You must manually connect the 24 V DC supply to the input points or relay coils.

If you require an external 24 V DC power supply, ensure that the power supply is not connected in parallel with the sensor supply of the CPU. For improved electrical noise protection, it is recommended that the commons (M) of the different power supplies be connected.

WARNING

Connecting power supplies safely

Connecting an external 24 V DC power supply in parallel with the 24 V DC sensor supply of the CPU can result in a conflict between the two supplies as each seeks to establish its own preferred output voltage level.

The result of this conflict can be shortened lifetime or immediate failure of one or both power supplies, with consequent unpredictable operation of the PLC system. Unpredictable operation could result in death or serious injury to personnel, and/or damage to equipment.

The DC sensor supply of the CPU and any external power supply should provide power to different points. A single connection of the commons is allowed.

Some of the 24 V DC power input ports in the S7-200 SMART system are interconnected, with a common logic circuit connecting multiple M terminals. For example, the following circuits are interconnected when designated as "not isolated" in the data sheets: the 24 V DC power supply of the CPU, the power input for the relay coil of an EM, or the power supply for a non-isolated analog input. All non-isolated M terminals must connect to the same external reference potential.

WARNING

Avoiding unwanted current flow

Connecting non-isolated M terminals to different reference potentials will cause unintended current flows that may cause damage or unpredictable operation in the PLC and any connected equipment.

Failure to comply with these guidelines could cause damage or unpredictable operation which could result in death or severe personal injury and/or property damage.

Always ensure that all non-isolated M terminals in an S7-200 SMART system are connected to the same reference potential.

Refer to the technical specifications for your particular CPU (Page 756) to determine the 24 V DC sensor supply power budget, the 5 V DC logic budget supplied by your CPU and the 5 V DC power requirements of the expansion modules and signal boards.

B.2 Calculating a sample power requirement

Calculating a sample power requirement

The following table shows a sample calculation of the power requirements for a CPU that includes the following:

- CPU SR40 AC/DC/Relay
- 3 each EM Digital Output 8 x Relay (EM DR08)
- 1 each EM Digital 8 x Inputs (EM DE08)

This installation has a total of 32 inputs and 40 outputs.

Note

The CPU has already allocated the power required to drive the CPUs internal relay coils. You do not need to include the internal relay coil power requirements in a power budget calculation.

The CPU in this example provides sufficient 5 V DC current, but does not provide enough 24 V DC current from the sensor supply for all of the inputs and expansion relay coils. The I/O requires 392 mA and the CPU provides 300 mA. This installation requires an additional source of at least 92 mA of 24 V DC power to operate all the included 24 V DC inputs and outputs.

Table B- 1 Calculation of the power budget for a sample configuration

CPU power budget	**5 V DC**	**24 V DC**
CPU SR40 AC/DC/Relay	1400 mA	300 mA
minus		
System requirements	**5 V DC**	**24 V DC**
CPU SR40, 24 inputs		24 * 4 mA = 96 mA
Slot 0: EM DR08	120 mA	8 * 11 mA = 88 mA
Slot 1: EM DR08	120 mA	8 * 11 mA = 88 mA
Slot 2: EM DR08	120 mA	8 * 11 mA = 88 mA
Slot 3: EM DE08	105 mA	8 * 4 mA = 32 mA
Total requirements	465 mA	392 mA
equals		
Current balance	**5 V DC**	**24 V DC**
Current balance total	275 mA	(92 mA)

B.3 Calculating your power requirement

Calculating your power requirement

Use the table below to determine how much power (or current) the CPU can provide for your configuration. Refer to the technical specifications (Page 751) for the power budgets of your CPU model and the power requirements of your digital modules, analog modules or signal boards.

Table B- 2 Power budget

Power budget	**5 V DC**	**24 V DC**
minus		
System requirements	**5 V DC**	**24 V DC**
Total requirements		
equals		
Current balance	**5 V DC**	**24 V DC**
Current balance total		

Error codes

C

C.1 Timestamp mismatch

This warning message indicates that the timestamps for the project do not match the timestamps for the program in the PLC. This may indicate that the programs are different, in which case it would be dangerous to continue the current operation. However, the programs might be functionally identical and still have different timestamps.

What actions modify the program timestamps?

Each program contains two distinct timestamps; the "Created" timestamp and the "Last Modified" timestamp. The created timestamp is set when the project is created by the New Project option. The Created timestamp is not affected by any user edits or program compilation.

The Last Modified timestamp is used to indicate when the user last modified the program. There are many conditions that cause the Last Modified timestamp to be set:

1. An edit of instructions or operands in the program block editor.
2. Adding, deleting, or modifying a variable or global symbol.
3. Adding or deleting a POU.
4. Compiling the program block.
5. Downloading the program block (this automatically compiles the program block and therefore sets the last modified timestamp).

Note that although all of these actions will cause the last modified timestamp to be set, this does not necessarily mean that the programs are different. For this reason, STEP 7-Micro/WIN SMART provides the "Compare" option, to allow you to determine whether the programs are really different

How do I tell if the programs are really different?

You can choose to compare the program block in the PLC with the project's program block by clicking the "Compare" button. The results of this comparison allow you to determine whether to continue the status operation.

How do I synchronize timestamps?

Downloading a new project to the PLC synchronizes the timestamps, which allows you to run status.

C.2 PLC non-fatal error codes

PLC compiler and run-time errors are non-fatal errors. Non-fatal errors can degrade some aspect of the performance of your PLC, but do not render the PLC incapable of executing the user program or updating the I/O.

- **Run-time programming errors** are non-fatal error conditions created by your program while the program is being executed. An example of this is an indirect-address pointer, which was valid when the program compiled, and then modified by program execution to point to an out-of-range address. Access the PLC Information from the PLC menu ribbon strip to determine what type of error has occurred.

  You can correct run-time programming errors only by modifying the user program. The CPU clears the run-time programming errors at the next transition from STOP to RUN mode.

- **PLC compiler errors** (or program-compile errors) prevent you from downloading the program to the PLC. STEP 7-Micro/WIN SMART detects compile errors when you compile or download (Page 40) the program, and shows errors in the output window. If there is a compile error, the PLC retains the current program that is resident in the PLC.

I/O errors are also non-fatal errors. When problems occur with the I/O of the CPU, signal board, and expansion modules, the PLC records the error information in special memory (SM) bits that your program can monitor and evaluate.

Non-fatal error codes

Hexadecimal error code	Non-fatal PLC program compiler errors
0080	The program is too large for the CPU; please reduce the program size
0081	Logic stack underflow; split the network into multiple networks
0082	Illegal instruction; check instruction mnemonics
0083	Illegal instruction before end of main program; remove incorrect instruction
0085	Illegal combination of FOR/NEXT; add FOR instruction or delete NEXT instruction
0086	Illegal combination of FOR/NEXT; add NEXT instruction or delete FOR instruction
0087	Missing label or POU; add the appropriate label
0088	Illegal instruction before end of subroutine; add RET to the end of the subroutine or remove incorrect instruction
0089	Illegal instruction before end of interrupt routine; add RETI to the end of the interrupt routine or remove incorrect instruction
008B	Illegal jump in or out of a SCR segment
008C	Duplicate label or POU name
008D	Exceeded maximum label or POU number; ensure that the number of labels allowed has not been exceeded
0090	Illegal operand
0091	Memory range error; check the operand ranges
0092	Illegal count operand; verify the maximum count size
0093	FOR/NEXT nesting level exceeded

Hexadecimal error code	Non-fatal PLC program compiler errors
0095	Missing LSCR instruction
0096	Missing SCRE instruction or illegal instruction before SCRE
0099	Too many password-protected POUs
009B	Illegal index for a string operation
009D	Illegal parameter detected in system block
009F	Illegal program organization

Hexadecimal error code	Transition to RUN mode prevented (run inhibit conditions)
0070	Run inhibit due to memory card inserted
0071	Run inhibit due to missing configured device
0072	Run inhibit due to mismatched device configuration (note: this error also includes device parameterization errors)
0073	Run inhibit due to attempted firmware update
0074	Run inhibit due to serious HW error on expansion module or signal board

Hexadecimal error code	Non-fatal run-time programming problem
0000	No non-fatal errors present
0001	HSC instruction enabled before executing HDEF instruction
0002	Input interrupt point already assigned to an HSC
0003	HSC input point already assigned to an input interrupt or other HSC
0004	Instruction not allowed in an interrupt routine
0005	Simultaneous HSC/PLS/motion instructions
0006	Indirect addressing error
0007	Time of day instructions data error
0008	Maximum user subroutine nesting level exceeded
0009	Simultaneous XMT/RCV instructions on Port 0
000A	Execution of an HDEF instruction of a previously configured HSC
000B	Simultaneous execution of XMT/RCV instructions on Port 1
000D	Attempt to redefine pulse output while it is active
000E	Number of PTO profile segment was set to 0
000F	Illegal numeric value encountered in compare contact instruction
0013	Illegal PID loop table

Hexadecimal error code	Non-fatal run-time programming problem
0014	Data log error: • There are too many DATx_WRITE subroutine executions in one program scan. Only 10 to 15 data log writes per second can be sustained. When there are too many DATx-WRITE executions per second, then the allotted memory will be full and for a short period of time no new data log records are stored. • Executing a data log write subroutine without first configuring a data log with the data log wizard
0016	HSC or interrupt input point already assigned to motion
0017	PTO/PWM output point already assigned to motion
0019	Signal Board not present or not configured
001A	Scan watchdog timeout.
001B	Attempt to change time base on enabled PWM
001C	Serious hardware error on expansion module or signal board
0090	Illegal operand
0091	Operand range error; check the operand ranges
0092	Illegal count operand; verify the maximum count size
0098	Illegal program edit in RUN mode
009A	Attempt to switch into freeport mode inside a user interrupt routine
009B	Illegal index for a string operation (user requested index=0)

See also

Special memory (SM) and system symbol names (Page 865)

C.3 PLC non-fatal error SM flags

Overview

Non-fatal errors are those that may degrade some aspect of PLC performance, but do not render the PLC incapable of executing the user program and updating I/O. To help you in debugging your program, information associated with error conditions is stored in Special Memory (SM) locations (Page 865), which can be accessed by the user program. For example, if you do not want to continue in RUN mode with certain non-fatal error conditions, you can have the user program force a transition to STOP mode when the undesirable condition occurs.

The following table lists and describes the Special Memory non-fatal error information.

SM bit	Non-fatal error description	SM byte	Non-fatal error description
SM0.2	Retentive Data Lost	SMB9	Module 0 I/O Error Byte
SM0.7	RTC_Lost	SMB11	Module 1 I/O Error Byte

SM bit	Non-fatal error description	SM byte	Non-fatal error description
SM1.3	Divide by Zero Error	SMB13	Module 2 I/O Error Byte
SM3.0	Parity Error	SMB15	Module 3 I/O Error Byte
SM4.0	Comm. Interrupt Queue Overflow	SMB17	Module 4 I/O Error Byte
SM4.1	Input Interrupt Queue Overflow	SMB19	Module 5 I/O Error Byte
SM4.2	Timed Interrupt Queue Overflow	SMB29	Signal Board I/O Error Byte
SM4.3	Run-Time Programming Problem		
SM5.0	I/O Error (any I/O error bit set)		

C.4 PLC fatal error codes

Overview

Fatal errors cause the PLC to stop the execution of your program. Depending on the severity of the error, a fatal error can render the PLC incapable of performing any or all functions. The objective for handling fatal errors is to bring the PLC to a safe state from which the PLC can respond to interrogations about the existing error conditions.

The PLC performs the following tasks when a fatal error is detected.

- Changes to STOP mode
- Turns on both the System Fault LED and the STOP LED
- Turns off the outputs

The PLC remains in this condition until the fatal error is corrected. The table shown below provides a list with descriptions for the fatal error codes that can be read from the PLC.

STEP 7-Micro/WIN SMART displays the error codes generated by the PLC, along with a brief description, in the PLC Information dialog. To access PLC information, click the PLC button

from the Information area of the PLC menu ribbon strip.

Once you have corrected the conditions that caused the fatal error, power-cycle the PLC or perform a warm restart from STEP 7-Micro/WIN SMART. To perform a warm start, click the Warm Start button

in the Modify area of the PLC menu ribbon strip.

Restarting the PLC clears the fatal error condition and causes power-up diagnostic testing. If another fatal error condition occurs, the PLC sets the System Fault LED again; otherwise, the PLC begins normal operation.

There are several possible error conditions that can render the PLC incapable of communication, in which case you cannot view the PLC error code. This type of error indicates a hardware failure requiring the PLC module to be repaired; it cannot be fixed by changes to the program or by clearing the PLC memory.

Fatal error codes

Hexadecimal error code	Description
0000	No fatal errors present
0001	System firmware checksum error
0002	Compiled user program checksum error
0004	Permanent memory failed
0005	Permanent memory error on user program
0006	Permanent memory error on system block
0007	Permanent memory error on force data
0009	Permanent memory error on user data, DB1
000A	Memory card failed
000B	Memory card error on user program
000C	Memory card error on system block
000D	Memory card error on force data
000F	Memory card error on user data, DB1
0010	Internal firmware error
0015	User program has compile error on power-up
0016	User data has compile error on power-up
0017	System block has compile error on power-up
0018	CPU HW identification data not available or corrupted
0019	HW watchdog timeout error
0x0020	PN SDB checksum error in internal flash

Special memory (SM) and system symbol names

D

D.1 SM (Special Memory) overview

The S7-200 SMART CPU provides special memory that contains system data. SMW is the prefix that indicates a special memory word. SMB is the prefix that indicates a special memory byte. You address individual bits as SM<byte number>.<bit number>. The System Symbol table in STEP 7-Micro/WIN SMART displays the special memory.

SMB0 to SMB29, SMB480 to SMB515, SMB1000 to SMB1699, and SMB1800 to SMB1999 (S7-200 SMART read-only special memory)

The S7-200 SMART CPU writes new changes to the system data stored in special memory.

Read system status from the CPU

SMB0 to SMB29, SMB480 to SMB515, SMB1000 to SMB1699, and SMB1800 to SMB1999 are read-only from your program. If a program includes logic to write to a read-only SM address, STEP 7-Micro/WIN SMART compiles the program without error. The CPU program compiler, however, will reject the program and display "Operand range error, Download failed".

Your program can read data stored in special memory addresses, evaluate the current system status, and use conditional logic to decide how to respond. In run mode, the continuous scanning of your program logic provides continuous monitoring of system data.

- SMB0 (Page 867) System status bits
- SMB1 (Page 868) Instruction execution status bits
- SMB2 (Page 869) Freeport receive character
- SMB3 (Page 870) Freeport parity error
- SMB4 (Page 870) Interrupt queue overflow, run-time program error, interrupts enabled, freeport transmitter idle, and forced value
- SMB5 (Page 871) I/O error status bits
- SMB6-SMB7 (Page 871) CPU ID, error status, and digital I/O points
- SMB8-SMB19 (Page 872) I/O module ID and errors
- SMW22-SMW26 (Page 873) Scan times
- SMB28-SMB29 (Page 873) Signal board ID and errors
- SMB480-SMB515 (Page 887) Data log status (read only)
- SMB1000-SMB1049 (Page 889) CPU hardware/firmware ID
- SMB1050-SMB1099 (Page 890) SB (signal board) hardware/firmware ID

- SMB1100-SMB1399 (Page 890) EM (expansion module) hardware/firmware ID
- SMB1400-SMB1699 (Page 893) EM (expansion module) module-specific data
- SMB1800-SMB1999 (Page 893): PROFINET device status.

Note

If your program uses the range SMB1800 to SMB1999 and is created in STEP 7-Micro/WIN SMART V2.3 or previous versions, the program will be cleared in V2.4 and later versions, and you must modify your program to use the other read/write SM addresses.

SMB30 to SMB194 and SMB566 to SMB749 (S7-200 SMART read/write special memory)

The S7-200 SMART CPU performs these actions:

- Reads configuration/control data from special memory
- Writes new changes to the system data in special memory.

Read system status from CPU

Write (send) control commands to the CPU

Your program can read and write all SM addresses in this range. The normal usage of SM data varies according to the function of each address.

SM addresses provide a means to access system status data, configure system options, and control system functions. In run mode, continuous scanning of your program provides for continuous access to special system features.

- SMB30 (Port 0) and SMB130 (Port 1) (Page 873) Port configuration for the integrated RS485 port (Port 0) and the CM01 Signal Board (SB) RS232/RS485 port (Port 1)
- SMB34-SMB35 (Page 874) Time intervals for timed interrupts
- SMB36-45 (HSC0), SMB46-55 (HSC1), SMB56-65 (HSC2), SMB136-145 (HSC3), SMB146-SMB155 (HSC4), SMB156-SMB165 (HSC5) (Page 875) High-speed counter configuration and operation
- SMB66-SMB85 (Page 880) PLS0 and PLS1 high-speed outputs
- SMB86-SMB94 and SMB186-SMB194 (Page 883) Receive message control
- SMW98 (Page 885) I/O expansion bus communication errors
- SMW100-SMW114 (Page 886) System alarms
- SMB130 (Page 873) Port configuration for the CM01 Signal Board (SB) RS232/RS485 port (Port 1) (See SMB30)
- SMB146-SMB155 (HSC4) and SMB156-SMB165 (HSC5) (Page 887) High-speed counter configuration and operation (See SMB36)

- SMB166-SMB169 (Page 880) PTO0 profile definition table
- SMB176-SMB179 (Page 880) PTO1 profile definition table
- SMB186-SMB194 (Page 883) Receive message control (See SMB86-SMB94)
- SMB566-SMB575 (Page 880) PLS2 high-speed output
- SMB576-SMB579 (Page 880) PTO2 profile definition table
- SMB600-SMB649 (Page 888) Axis 0 open loop motion control
- SMB650-SMB699 (Page 889) Axis 1 open loop motion control
- SMB700-SMB749 (Page 889) Axis 2 open loop motion control

⚠ WARNING

Risks with STEP 7-Micro/WIN Version 4.0 or greater (.mwp files) with absolute special memory (SM) addressing

You can open a program (.mwp file) from an earlier version of STEP 7-Micro/WIN in STEP 7-Micro/WIN SMART. If that program uses symbolic special memory (SM) addressing, then insert the System Symbol table (Page 108) in your project. The symbols map correctly to the current SM addresses. If, however, the program uses absolute SM addressing, those absolute SM addresses might no longer exist.

Programs based on inconsistent definitions of SM addresses can result in unexpected machine or process operation. Unexpected machine or process operation can cause death or serious injury to personnel, and/or damage to equipment.

If you open an .mwp file in STEP 7-Micro/WIN SMART, delete the "S7-200 Symbols" table and insert the "System Symbols" table. The symbols in the former .mwp program map to the current SM address scheme. Convert any absolute SM addresses to use the corresponding symbol name.

D.2 SMB0: System status

Special Memory Byte 0 (SM0.0 - SM0.7) provides eight bits the S7-200 SMART CPU updates at the end of each scan cycle.

Table D- 1 SMB0 system status bits

S7-200 SMART symbol name	SM address	Description
Always_On	SM0.0	This bit is always TRUE.
First_Scan_On	SM0.1	The CPU sets this bit to TRUE, for the first scan cycle, and sets it to FALSE thereafter. One use for this bit is to call an initialization subroutine.

S7-200 SMART symbol name	SM address	Description
Retentive_Lost	SM0.2	The CPU sets this bit TRUE for one scan cycle after: • Reset to factory communication command • Reset to factory memory card evaluation • Evaluation of program transfer card in which a new system block was loaded from the card. • Problem with retentive record that the CPU stored on the last power down. This bit can be used as either an error memory bit or as a mechanism to invoke a special start-up sequence.
RUN_Power_Up	SM0.3	The CPU sets this bit to TRUE for one scan cycle when RUN mode is entered from a power-up or warm restart condition. This bit can be used to provide machine warm-up time before starting an operation.
Clock_60s	SM0.4	This bit provides a clock pulse. The bit is FALSE for 30 seconds and TRUE for 30 seconds, for a cycle time of one minute. This bit provides an easy-to-use delay or a one-minute clock pulse.
Clock_1s	SM0.5	This bit provides a clock pulse. The bit is FALSE for 0.5 seconds and then TRUE for 0.5 seconds for a cycle time of one second. This bit provides an easy-to-use delay or a one-second clock pulse.
Clock_Scan	SM0.6	This bit is a scan cycle clock that is TRUE for one scan and then FALSE for the next scan. On subsequent scans the bit alternates between TRUE and FALSE. You can use this bit as a scan counter input.
RTC_Lost	SM0.7	This bit applies to CPU models that have a real-time clock. The CPU sets this bit to TRUE for one scan cycle if the time on the real time clock device was reset or lost at power-up. The program can use this bit as either an error memory bit or to invoke a special start-up sequence.

D.3 SMB1: Instruction execution status

Special memory byte 1 (SM1.0 - SM1.7) provides execution status for various instructions, such as table and math operations. These bits are set and reset by instructions at execution time.

Table D- 2 SMB1 instruction execution status bits

S7-200 SMART symbol name	SM address	Description
Result_0	SM1.0	Certain instructions set this bit to TRUE when the result of the operation is zero.
Overflow_Illegal	SM1.1	Certain instructions set this bit to TRUE when either an overflow results or when the instruction detects an illegal number value.
Neg_Result	SM1.2	Math operations set this bit TRUE when the operation produces a negative result.
Divide_By_0	SM1.3	The CPU sets this bit TRUE when the program attempts a division by zero.
Table_Overflow	SM1.4	The Add to Table (ATT) instruction sets this bit TRUE when the referenced data table is full.
Table_Empty	SM1.5	The CPU sets this bit TRUE when either LIFO or FIFO instructions attempt to read from an empty table.

S7-200 SMART symbol name	SM address	Description
Not_BCD	SM1.6	The CPU sets this bit TRUE for an illegal value (non-BCD) in a BCD to binary conversion.
Not_Hex	SM1.7	The CPU sets this bit TRUE for an illegal value (non-hex ASCII digit) during ASCII to Hex (ATH) conversion.

D.4 SMB2: Freeport receive character

Special memory byte 2 is the Freeport receive character buffer. While in Freeport mode, the CPU stores each character that the CPU receives in this byte for easy access by your program.

Table D- 3 SMB2 Freeport received character

S7-200 SMART symbol name	SM address	Description
Receive_Char	SMB2	This byte contains each character that the CPU receives from Port 0 or Port 1 during Freeport communication.

Note

Port 0 and Port 1 share SMB2 and SMB3

When the CPU receives a character on Port 0, note the following:

- The CPU executes the interrupt routine attached to that event (interrupt event 8).
- SMB2 contains the character received on Port 0.
- SMB3 contains the parity status of the received character.

Likewise, when the CPU receives a character on Port 1, note the following:

- The CPU executes the interrupt routine attached to that event (interrupt event 25).
- SMB2 contains the character received on Port 1.
- SMB3 contains the parity status of the received character.

D.5 SMB3: Freeport character error

The CPU sets SM3.0 TRUE when Freeport communication detects a parity, framing, break, or overrun error on a received character. Use this bit to discard the message.

Table D- 4 SMB3 Freeport character error

S7-200 SMART symbol name	SM address	Description
Parity_Err	SM3.0	This bit indicates that the CPU received a parity, framing, break, or overrun error from Port 0 or Port 1: • FALSE: no error • TRUE: error

D.6 SMB4: Interrupt queue overflow, run-time program error, interrupts enabled, freeport transmitter idle, and value forced

Special memory byte 4 (SM4.0 - SM4.7) contains the interrupt queue overflow bits. These bits indicate either that interrupts are occurring at a rate greater than the CPU can process or that the global interrupt disable (DISI) instruction (Page 307) has disabled interrupts.

Other bits indicate:

- Enabled or disabled status of interrupts
- A run-time program error
- Freeport transmitter status
- One or more forced PLC memory values

Table D- 5 SMB4 system status

S7-200 SMART symbol name	SM address	Description
Comm_Int_Ovr	** SM4.0	TRUE: Communication interrupt queue has overflowed.
Input_Int_Ovr	** SM4.1	TRUE: Input interrupt queue has overflowed.
Timed_Int_Ovr	** SM4.2	TRUE: Timed interrupt queue has overflowed.
RUN_Err	SM4.3	TRUE: CPU has detected a run-time programming non-fatal error.
Int_Enable	SM4.4	TRUE: Enabled interrupts exists
Xmit0_Idle	SM4.5	TRUE: Port 0 transmitter is idle (FALSE: Transmission in progress).
Xmit1_Idle	SM4.6	TRUE: Port 1 transmitter is idle (FALSE: Transmission in progress).
Force_On	SM4.7	TRUE: Existence of forced PLC memory

** Use status bits SM4.0, SM4.1, and SM4.2 only inside an interrupt routine. The CPU resets these status bits when the CPU empties the interrupt queue and returns control to the main program.

D.7 SMB5: I/O error status

Special Memory Byte 5 (SM5.0 - SM5.7) contains a status bit that indicates error conditions in the I/O system.

Table D- 6 SMB5 I/O error status

S7-200 SMART symbol name	SM address	Description
IO_Err	SM5.0	This bit is set ON if any I/O errors are present.

D.8 SMB6-SMB7: CPU ID, error status, and digital I/O points

Special memory bytes 6 and 7 provide CPU information.

S7-200 SMART symbol name	SM address	Read-only SMB6 and SMB7 (CPU ID, error status, and digital I/O point)								
CPU_ID	SMB6	MSB							LSB	
		7							0	
		1	x	x	x	c	d	0	0	
	SM6.4 to SM6.6		0	0	0	= CPU CR20s				
			0	0	1	= CPU CR40s				
			0	1	0	= CPU CR60s				
			0	1	1	= CPU SR20 / ST20				
			1	0	0	= CPU SR40 / ST40				
			1	0	1	= CPU SR60 / ST60				
			1	1	0	= CPU CR30s				
			1	1	1	= CPU SR30 / ST30				
	SM6.2 to SM6.3					c				Configuration / parameterization error 0 = no error, 1 = error
							d			Diagnostic alarm (See SMW100 for alarm codes) 0 = no error, 1 = error
CPU_IO	SMB7	MSB							LSB	
		7							0	
		i	i	i	i	q	q	q	q	
	SM7.0 to SM7.7	i	i	i	i					0 to 15 bytes of digital input points
						q	q	q	q	0 to 15 bytes of digital output points

See also SMW100-SMW114 System alarm codes (Page 886)

D.9 SMB8-SMB19: I/O module ID and errors

SMB8 through SMB19 are organized in byte pairs for expansion modules 0 to 5.

The even-numbered byte of each pair is the module-identification register. These bytes identify the module type, the I/O type, and the number of inputs and outputs.

The odd-numbered byte of each pair is the module error register. These bytes provide an indication of any errors detected in the I/O for that module.

S7-200 SMART symbol name	SM address	Read-only SMB8-SMB21 module ID and error data
EM0_ID	SMB8	Expansion module 0 ID register
EM0_Err	SMB9	Expansion module 0 error register (See SMW104 for diagnostic alarm code)
EM1_ID	SMB10	Expansion module 1 ID register
EM1_Err	SMB11	Expansion module 1 error register (See SMW106 for diagnostic alarm code)
EM2_ID	SMB12	Expansion module 2 ID register
EM2_Err	SMB13	Expansion module 2 error register (See SMW108 for diagnostic alarm code)
EM3_ID	SMB14	Expansion module 3 ID register
EM3_Err	SMB15	Expansion module 3 error register (See SMW110 for diagnostic alarm code)
EM4_ID	SMB16	Expansion module 4 ID register
EM4_Err	SMB17	Expansion module 4 error register (See SMW112 for diagnostic alarm code)
EM5_ID	SMB18	Expansion module 5 ID register
EM5_Err	SMB19	Expansion module 5 error register (See SMW114 for diagnostic alarm code)

Even numbered byte-I/O module ID register

MSB							LSB
7							0
m	0	0	a	i	i	q	q

- m: Module present — 0 = Present; 1 = Not present
- a: I/O type — 0 = Digital; 1 = Analog
- ii: Inputs — 00 = No inputs; 01 = 2 AI or 8 DI; 10 = 4 AI or 16 DI; 11 = 8 AI or 32 DI
- qq: Outputs — 00 = No outputs; 01 = 2 AQ or 8 DQ; 10 = 4 AQ or 16 DQ; 11 = 8 AQ or 32 DQ

Odd numbered byte-I/O module error register

MSB							LSB
7							0
c	d	0	b	0	0	0	m

- c: 0 = No error; 1 = Configuration / parameterization error
- d: 0 = No error; 1 = Diagnostic alarm
- b: 0 = No error; 1 = Bus access error
- m: 0 = OK; 1 = Configured module missing

See also SMW100-SMW114 System alarm codes (Page 886)

D.10 SMW22-SMW26: Scan times

SMW22, SMW24, and SMW26 contain information on the scan time. You can read the last scan time, minimum scan time, and maximum scan time (millisecond values).

Table D- 7 SMW22-SMW26 PLC scan times

S7-200 SMART symbol name	SM address	Description
Last_Scan	SMW22	Scan time of the last scan.
Minimum_Scan	SMW24	Minimum scan time recorded since entering the RUN mode or since resetting these values from the PLC Information dialog.
Maximum_Scan	SMW26	Maximum scan time recorded since entering the RUN mode or since resetting these values from the PLC Information dialog.

D.11 SMB28-SMB29: Signal board ID and errors

SMB28-SMB29 byte addresses store the signal board type and error status.

S7-200 SMART symbol name	SM address	Read-only SMB28-SMB29 signal board ID and error data
SB_ID	SMB28	Signal board ID register
SB_Err	SMB29	Signal board error register (See SMW102 for diagnostic alarm code)

SMB28 signal board ID register

MSB							LSB
7							0
m	0	0	a	i	i	q	q

- m: Module present — 0 = Present; 1 = Not present
- a: I/O type — 0 = Digital; 1 = Analog
- ii: Inputs — 00 = No inputs; 01 = 2 AI or 8 DI; 10 = 4 AI or 16 DI; 11 = 8 AI or 32 DI
- qq: Outputs — 00 = No outputs; 01 = 2 AQ or 8 DQ; 10 = 4 AQ or 16 DQ; 11 = 8 AQ or 32 DQ

SMB29 signal board error register

MSB							LSB
	7						0
	c	d	0	b	0	0	0	m

- c: 0 = No error; 1 = Configuration / parameterization error
- d: 0 = No error; 1 = Diagnostic alarm
- b: 0 = No error; 1 = Bus access error
- m: 0 = OK; 1 = Configured signal board missing

See also SMW100-SMW110 System alarm codes (Page 886)

D.12 SMB30: (Port 0) and SMB130: (Port 1)

SMB30 configures Port 0 (onboard RS485 port).

SMB130 configures Port 1 (optional CM01 signal board).

You can read and write to SMB30 and SMB130. These bytes configure the respective communication port for Freeport operation and provide selection of either Freeport or system protocol support.

S7-200 SMART symbol name	SM address		Bit Format	Bit format MSB LSB								
	Port 0	Port 1		7							0	
P0_Config	SMB30			p	p	d	b	b	b	m	m	
P1_Config		SMB130										
	SM30.6 – SM30.7	SM130.6 – SM130.7	pp:	0	0	= No parity						
				0	1	= Even parity						
				1	0	= No parity						
				1	1	= Odd parity						
	SM30.5	SM130.5	d:	0	= 8 data bits per character							
				1	= 7 data bits per character							
	SM30.2 – SM30.4	SM130.2 – SM130.4	bbb:	0	0	0	= 38,400 bps					
				0	0	1	= 19,200 bps					
				0	1	0	= 9,600 bps					
				0	1	1	= 4,800 bps					
				1	0	0	= 2,400 bps					
				1	0	1	= 1,200 bps					
				1	1	0	= 115,200 bps					
				1	1	1	= 57,600					
P0_Config_0	SM30.0		mm:	0	0	= PPI slave mode						
P1_Config_0		SM130.0		0	1	= Freeport protocol						
	SM30.1	SM130.1		1	0	= Reserved (defaults to PPI slave mode)						
				1	1	= Reserved (defaults to PPI slave mode)						
				Note: Bits 2 through 7 are ignored in PPI modes								

D.13 SMB34-SMB35: Time intervals for timed interrupts

Special memory bytes 34 and 35 control the time interval of timed interrupts 0 and 1 (Page 309). You can assign time intervals in 1-ms increments from 1 ms to 255 ms. The CPU captures the time-interval value at the time that the CPU attaches the interrupt routine to corresponding timed interrupt event. To change the time interval, you must reattach the timed interrupt event to the same interrupt routine or to a different interrupt routine. You can terminate a timed interrupt event by detaching the event.

Table D- 8 SMB34-SMB35 timed interrupt intervals

S7-200 SMART symbol name	SM address	Description
Time_0_Intrvl	SMB34	Timed interrupt 0: Time interval value (in 1 ms increments from 1 ms to 255 ms).
Time_1_Intrvl	SMB35	Timed interrupt 1: Time interval value (in 1 ms increments from 1 ms to 255 ms).

D.14 SMB36-SMB45 (HSC0), SMB46-SMB55 (HSC1), SMB56-SM65 (HSC2), SMB136-SMB145 (HSC3), SMB146-SMB155 (HSC4), SMB156-SMB165 (HSC5): high-speed counters

These bytes provide configuration and operation information for the high-speed counters:

- HSC0
- HSC1
- HSC2
- HSC3
- HSC4
- HSC5

Table D- 9 HSC0 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC0_Status	**SMB36**	**HSC0 counter status** **Note**: Counter status bits are valid only while the CPU is executing an interrupt routine that a high-speed counter event triggered.
	SM36.0–SM36.4	Reserved
HSC0_Status_5	SM36.5	HSC0 current counting direction status bit: TRUE: Counting up
HSC0_Status_6	SM36.6	HSC0 current value equals preset value status bit: TRUE: Equal
HSC0_Status_7	SM36.7	HSC0 current value is greater than preset value status bit: TRUE: Greater than
HSC0_Ctrl	**SMB37**	**HSC0 counter control**
HSC0_Reset_Level	SM37.0	HSC0 active level control bit for Reset: FALSE: Reset is active high, TRUE: Reset is active low
	SM37.1	Reserved
HSC0_Rate	SM37.2	HSC0 counting rate selection for Quadrature counters: FALSE: 4x counting rate; TRUE: 1x counting rate
HSC0_Dir	SM37.3	HSC0 direction control bit: TRUE: Count up
HSC0_Dir_Update	SM37.4	HSC0 update direction: TRUE: update direction
HSC0_PV_Update	SM37.5	HSC0 update preset value: TRUE: Write new preset value to HSC0 preset
HSC0_CV_Update	SM37.6	HSC0 update current value: TRUE: Write new current value to HSC0 current
HSC0_Enable	SM37.7	HSC0 enable bit: TRUE: enable
HSC0_CV	**SMD38**	**HSC0 new current value** You use SMD38 to set HSC0 current value to any value you choose. To update the current value, write the new current value to SMD38; write 1 to SM37.6; and execute the HSC instruction. The instruction then writes the new current value to HSC0's current count register.
HSC0_PV	**SMD42**	**HSC0 new preset value** You use SMD42 to set HSC0 preset value to any value you choose. To update the preset value, write the new preset value to SMD42; write 1 to SM37.5; and execute the HSC instruction. The instruction then writes the new preset value to HSC0's preset register.

Table D- 10 HSC1 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC1_Status	**SMB46**	**HSC1 counter status** **Note**: Counter status bits are valid only while the CPU is executing an interrupt routine that a high-speed counter event triggered.
	SM46.0–SM46.4	Reserved
HSC1_Status_5	SM46.5	HSC1 current counting direction status bit: TRUE: Counting up
HSC1_Status_6	SM46.6	HSC1 current value equals preset value status bit: TRUE: Equal
HSC1_Status_7	SM46.7	HSC1 current value is greater than preset value status bit: TRUE: Greater than
HSC1_Ctrl	**SMB47**	**HSC1 control**
	SM47.0-SM47.2	Reserved
HSC1_Dir	SM47.3	HSC1 direction control bit: TRUE: Count up FALSE: Count down
HSC1_Dir_Update	SM47.4	HSC1 update direction: TRUE: update direction
HSC1_PV_Update	SM47.5	HSC1 update preset value: TRUE: Write new preset value to HSC1 preset
HSC1_CV_Update	SM47.6	HSC1 update current value: TRUE: Write new current value to HSC1 current
HSC1_Enable	SM47.7	HSC1 enable bit: TRUE: enable HSC FALSE: disable HSC
HSC1_CV	SMD48	**HSC1 new current value** You use SMD48 to set HSC1 current value to any value you choose. To update the current value, write the new current value to SMD48; write 1 to SM47.6; and execute the HSC instruction. The instruction then writes the new current value to HSC1's current count register.
HSC1_PV	SMD52	**HSC1 new preset value** You use SMD52 to set HSC1 preset value to any value you choose. To update the preset value, write the new preset value to SMD52; write 1 to SM47.5; and execute the HSC instruction. The instruction then writes the new preset value to HSC1's preset register.

Table D- 11 HSC2 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC2_Status	**SMB56**	**HSC2 counter status** **Note:** Counter status bits are valid only while the CPU is executing an interrupt routine that a high-speed counter event triggered.
	SM56.0–SM56.4	Reserved
HSC2_Status_5	SM56.5	HSC2 current counting direction status bit: TRUE: Counting up
HSC2_Status_6	SM56.6	HSC2 current value equals preset value status bit: TRUE: Equal
HSC2_Status_7	SM56.7	HSC2 current value is greater than preset value status bit: TRUE: Greater than
HSC2_Ctrl	**SMB57**	**HSC2 control**

S7-200 SMART symbol name	SM address	Description
HSC2_Reset_Level	SM57.0	HSC2 active level control bit for Reset: FALSE: Reset is active high; TRUE: Reset is active low
	SM57.1	Reserved
HSC2_Rate	SM57.2	HSC2 counting rate selection for Quadrature counters: FALSE: 4x counting rate; TRUE: 1x counting rate
HSC2_Dir	SM57.3	HSC2 direction control bit: TRUE: Count up
HSC2_Dir_Update	SM57.4	HSC2 update direction: TRUE: update direction
HSC2_PV_Update	SM57.5	HSC2 update preset value: TRUE: Write new preset value to HSC2 preset
HSC2_CV_Update	SM57.6	HSC2 update current value: TRUE: Write new current value to HSC2 current
HSC2_Enable	SM57.7	HSC2 enable bit: TRUE: enable
HSC2_CV	**SMD58**	**HSC2 new current value** You use SMD58 to set HSC2 current value to any value you choose. To update the current value, write the new current value to SMD58; write 1 to SM57.6; and execute the HSC instruction. The instruction then writes the new current value to HSC2's current count register.
HSC2_PV	**SMD62**	**HSC2 new preset value** You use SMD62 to set HSC2 preset value to any value you choose. To update the preset value, write the new preset value to SMD62; write 1 to SM57.5; and execute the HSC instruction. The instruction then writes the new preset value to HSC2's preset register.

Table D- 12 HSC3 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC3_Status	**SMB136**	**HSC3 counter status** **Note**: Counter status bits are valid only while the CPU is executing an interrupt routine that a high-speed counter event triggered.
	SM136.0–SM136.4	Reserved
HSC3_Status_5	SM136.5	HSC3 current counting direction status bit: TRUE: Counting up
HSC3_Status_6	SM136.6	HSC3 current value equals preset value status bit: TRUE: Equal
HSC3_Status_7	SM136.7	HSC3 current value is greater than preset value status bit: TRUE: Greater than
HSC3_Ctrl	**SMB137**	**HSC3 counter control**
	SM137.0–SM137.2	Reserved
HSC3Dir	SM137.3	HSC3 direction control bit: TRUE: Count up
HSC3_Dir_Update	SM137.4	HSC3 update direction: TRUE: Update direction
HSC3_PV_Update	SM137.5	HSC3 update preset value: TRUE: Write new preset value to HSC3 preset
HSC3_CV_Update	SM137.6	HSC3 update current value: TRUE: Write new current value to HSC3 current
HSC3_Enable	SM137.7	HSC3 enable bit: TRUE: enable

S7-200 SMART symbol name	SM address	Description
HSC3_CV	**SMD138**	**HSC3 new current value** You use SMD138 to set HSC3 current value to any value you choose. To update the current value, write the new current value to SMD138; write 1 to SM137.6; and execute the HSC instruction. The instruction then writes the new current value to HSC3's current count register.
HSC3_PV	**SMD142**	**HSC3 new preset value** You use SMD142 to set HSC3 preset value to any value you choose. To update the preset value, write the new preset value to SMD142; write 1 to SM147.5; and execute the HSC instruction. The instruction then writes the new preset value to HSC3's preset register.

Table D- 13 HSC4 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC4_Status	**SMB146**	**HSC4 counter status** **Note**: Counter status bits are valid only while the CPU is executing an interrupt routine that a high-speed counter event triggered.
	SM146.0–SM146.4	Reserved
HSC4_Status_5	SM146.5	HSC4 current counting direction status bit: TRUE: Counting up
HSC4_Status_6	SM146.6	HSC4 current value equals preset value status bit: TRUE: Equal
HSC4_Status_7	SM146.7	HSC4 current value is greater than preset value status bit: TRUE: Greater than
HSC4_Ctrl	**SMB147**	**HSC4 counter control**
HSC4_Reset_Level	SM147.0	HSC4 active level control bit for Reset: FALSE: Reset is active high, TRUE: Reset is active low
	SM147.1	Reserved
HSC4_Rate	SM147.2	HSC4 Counting rate selection for Quadrature counters: FALSE: 4x counting rate; TRUE: 1x counting rate
HSC4Dir	SM147.3	HSC4 direction control bit: TRUE: Count up
HSC4_Dir_Update	SM147.4	HSC4 update direction: TRUE: update direction
HSC4_PV_Update	SM147.5	HSC4 update preset value: TRUE: Write new preset value to HSC4 preset
HSC4_CV_Update	SM147.6	HSC4 update current value: TRUE: Write new current value to HSC4 current
HSC4_Enable	SM147.7	HSC4 enable bit: TRUE: enable
HSC4_CV	**SMD148**	**HSC4 new current value** You use SMD148 to set HSC4 current value to any value you choose. To update the current value, write the new current value to SMD148; write 1 to SM147.6; and execute the HSC instruction. The instruction then writes the new current value to HSC4's current count register.
HSC4_PV	**SMD152**	**HSC4 new preset value** You use SMD152 to set HSC4 preset value to any value you choose. To update the preset value, write the new preset value to SMD152; write 1 to SM147.5; and execute the HSC instruction. The instruction then writes the new preset value to HSC4's preset register.

Table D- 14 HSC5 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC5_Status	**SMB156**	**HSC5 counter status** **Note**: Counter status bits are valid only while the CPU is executing an interrupt routine that a high-speed counter event triggered.
	SM156.0–SM156.4	Reserved
HSC5_Status_5	SM156.5	HSC5 current counting direction status bit: TRUE: Counting up
HSC5_Status_6	SM156.6	HSC5 current value equals preset value status bit: TRUE: Equal
HSC5_Status_7	SM156.7	HSC5 current value is greater than preset value status bit: TRUE: Greater than
HSC5_Ctrl	**SMB157**	**HSC5 counter control**
HSC5_Reset_Level	SM157.0	HSC5 active level control bit for Reset: FALSE: Reset is active high, TRUE: Reset is active low
	SM157.1	Reserved
HSC5_Rate	SM157.2	HSC5 Counting rate selection for Quadrature counters: FALSE: 4x counting rate; TRUE: 1x counting rate
HSC5Dir	SM157.3	HSC5 direction control bit: TRUE: Count up
HSC5_Dir_Update	SM157.4	HSC5 update direction: TRUE: Update direction
HSC5_PV_Update	SM157.5	HSC5 update preset value: TRUE: Write new preset value to HSC5 preset
HSC5_CV_Update	SM157.6	HSC5 update current value: TRUE: Write new current value to HSC5 current
HSC5_Enable	SM157.7	HSC5 enable bit: TRUE: Enable
HSC5_CV	**SMD158**	**HSC5 new current value** You use SMD158 to set HSC5 current value to any value you choose. To update the current value, write the new current value to SMD158; write 1 to SM157.6; and execute the HSC instruction. The instruction then writes the new current value to HSC5's current count register.
HSC5_PV	**SMD162**	**HSC5 new preset value** You use SMD162 to set HSC5 preset value to any value you choose. To update the preset value, write the new preset value to SMD162; write 1 to SM157.5; and execute the HSC instruction. The instruction then writes the new preset value to HSC5's preset register.

D.15 SMB66-SMB85 (PTO0/PWM0, PTO1/PWM1), SMB166-SMB169 (PTO0), SMB176-SMB179 (PTO1), and SMB566-SMB579 (PTO2/PWM2): high-speed outputs

The S7-200 SMART CPU uses the following special memory to monitor and control the pulse train outputs (PTO0 and PTO1) and pulse width modulation outputs (PWM0 and PWM1) for the PLS (Pulse) instruction:

- SMB66-SMB85
- SMB166-SMB169
- SMB176-SMB179

The CPU uses SMB566-SMB579 to monitor and control pulse train output PTO2 and pulse width modulation output PWM2.

Table D- 15 High-speed output 0 configuration and control

S7-200 SMART symbol name	SM address	Function
PTO0_Status	**SMB66**	**PTO0 Status**
PLS0_Abort_AE	SM66.4	PTO0 profile was aborted due to an add error: FALSE: No abort; TRUE: aborted
PLS0_Disable_UC	SM66.5	PTO0 user manually disabled a PTO profile while it was running: FALSE: Not disabled; TRUE: Manually disabled
PLS0_Ovr	SM66.6	PTO0 pipeline overflow/underflow, loading pipeline while full or transferring an empty pipeline: FALSE: No overflow; TRUE: Pipeline overflow/underflow
PLS0_Idle	SM66.7	PTO0 idle: FALSE: PTO in progress; TRUE: PTO is idle
PLS0_Ctrl	**SMB67**	**Monitor and control PTO0 (Pulse Train Output) and PWM0 (Pulse Width Modulation) for Q0.0**
PLS0_Cycle_Update	SM67.0	PTO0/PWM0 update the cycle time or frequency value: FALSE: No update; TRUE: Write new cycle time/frequency
PWM0_PW_Update	SM67.1	PWM0 update the pulse width value: FALSE: No update; TRUE: Write new pulse width
PTO0_PC_Update	SM67.2	PTO0 update the pulse count value: FALSE: No update; TRUE: Write new pulse count
PWM0_TimeBase	SM67.3	PWM0 time base: FALSE: 1 µs/tick, TRUE: 1 ms/tick
	SM67.4	Reserved
PTO0_Operation	SM67.5	PTO0 select single/multiple segment operation: FALSE: Single; TRUE:Multiple
PLS0_Select	SM67.6	PTO0/PWM0 mode select: FALSE: PWM; TRUE: PTO
PLS0_Enable	SM67.7	PTO0/PWM0 enable: FALSE: Disable; TRUE: Enable
PLS0_Cycle	**SMW68**	**Word data type: PWM0 cycle time value (2 to 65,535 units of time base); PTO0 frequency value (1 to 65,535 Hz)**
PWM0_PW	**SMW70**	**Word data type: PWM0 pulse width value (0 to 65,535 units of time base)**
PTO0_PC	**SMD72**	**Double Word data type: PTO0 pulse count value (1 to 2,147,483,647)**

S7-200 SMART symbol name	SM address	Function
PTO0_Seg_Num	**SMB166**	**Byte data type: The currently executing segment number for PTO0's profile**
PTO0_Profile_Offset	**SMW168**	**Word data type: Starting location of PTO0's profile table (byte offset from V0)**

Table D- 16 High-speed output 1 configuration and control

S7-200 SMART symbol name	SM address	Function
PTO1_Status	**SMB76**	**PTO1 Status**
PLS1_Abort_AE	SM76.4	PTO1 profile was aborted due to an add error: FALSE: No abort; TRUE:Aborted
PLS1_Disable_UC	SM76.5	PTO1 user manually disabled a PTO profile while it was running: FALSE: Not disabled; TRUE: Manually disabled
PLS1_Ovr	SM76.6	PTO1 pipeline overflow/underflow, loading pipeline while full or transferring an empty pipeline: FALSE: No overflow; TRUE: Pipeline overflow/underflow
PLS1_Idle	SM76.7	PTO1 idle: FALSE: PTO in progress; TRUE: PTO is idle
PLS1_Ctrl	**SMB77**	**Monitor and control PTO1 (Pulse Train Output) and PWM1 (Pulse Width Modulation) for Q0.1**
PLS1_Cycle_Update	SM77.0	PTO1/PWM1 update the cycle time or frequency value: FALSE: No update; TRUE: Write new cycle time/frequency
PWM1_PW_Update	SM77.1	PWM1 update the pulse width value: FALSE: No update; TRUE: Write new pulse width
PTO1_PC_Update	SM77.2	PTO1 update the pulse count value: FALSE: No update; TRUE: Write new pulse count
PWM1_TimeBase	SM77.3	PWM1 time base: FALSE: 1 µs/tick, TRUE: 1 ms/tick
	SM77.4	Reserved
PTO1_Operation	SM77.5	PTO1 select single/multiple segment operation: FALSE: Single; TRUE: Multiple
PLS1_Select	SM77.6	PTO1/PWM1 mode select: FALSE: PWM; TRUE: PTO
PLS1_Enable	SM77.7	PTO1/PWM1 enable: FALSE: Disable; TRUE: Enable
PLS1_Cycle	**SMW78**	**Word data type: PWM1 cycle time value (2 to 65,535 units of time base); PTO1 frequency value (1 to 65,535 Hz)**
PWM1_PW	**SMW80**	**Word data type: PWM1 pulse width value (0 to 65,535 units of time base)**
PTO1_PC	**SMD82**	**Double Word data type: PTO1 pulse count value (1 to 2,147,483,647)**
PTO1_Seg_Num	**SMB176**	**Byte data type: The currently executing segment number for PTO1's profile**
PTO1_Profile_Offset	**SMW178**	**Word data type: Starting location of PTO1's profile table (byte offset from V0)**

Table D- 17 High-speed output 2 configuration and control

S7-200 SMART symbol name	SM address	Description
PTO2_Status	**SMB566**	**PTO2 Status**
PLS2_Abort_AE	SM566.4	PTO2 profile was aborted due to an add error: FALSE: No abort; TRUE:Aborted
PLS2_Disable_UC	SM566.5	PTO2 user manually disabled a PTO profile while it was running: FALSE: Not disabled; TRUE: Manually disabled
PLS2_Ovr	SM566.6	PTO2 pipeline overflow/underflow, loading pipeline while full or transferring an empty pipeline: FALSE: No overflow; TRUE: Pipeline overflow/underflow
PLS2_Idle	SM566.7	PTO2 idle: FALSE: PTO in progress; TRUE: PTO is idle
PLS2_Ctrl	**SMB567**	**Monitor and control PTO2 (Pulse Train Output) and PWM2 (Pulse Width Modulation) for Q0.3**
PLS2_Cycle_Update	SM567.0	PTO2/PWM2 update the cycle time or frequency value: FALSE: No update; TRUE: Write new cycle time/frequency
PWM2_PW_Update	SM567.1	PWM2 update the pulse width value: FALSE: No update; TRUE: Write new pulse width
PTO2_PC_Update	SM567.2	PTO2 update the pulse count value: FALSE: No update; TRUE: Write new pulse count
PLS2_TimeBase	SM567.3	PWM2 time base: FALSE: 1 µs/tick, TRUE: 1 ms/tick
	SM567.4	Reserved
PTO2_Operation	SM567.5	PTO2 select single/multiple segment operation: FALSE: Single; TRUE: Multiple
PLS2_Select	SM567.6	PTO2/PWM2 mode select: FALSE: PWM; TRUE: PTO
PLS2_Enable	SM567.7	PTO2/PWM2 enable: FALSE: disable; TRUE: enable
PLS2_Cycle	**SMW568**	**Word data type: PWM2 cycle time value (2 to 65,535 units of time base); PTO2 frequency value (1 to 65,535 Hz)**
PWM2_PW	**SMW570**	**Word data type: PWM2 pulse width value (0 to 65,535 units of time base)**
PTO2_PC	**SMD572**	**Double Word data type: PTO2 pulse count value (1 to 2,147,483,647)**
PTO2_Seg_Num	**SMB576**	**Byte data type: The currently executing segment number for PTO2's profile**
PTO2_Profile_Offset	**SMW578**	**Word data type: Starting location of PTO2's profile table (byte offset from V0)**

D.16 SMB86-SMB94 and SMB186-SMB194: Receive message control

You use SMB86-SMB94 and SMB186-SMB194 to control and read status of the RCV (Receive message) instruction (Page 191).

S7-200 SMART symbol name	SM address		Bit Format	Receive message status byte MSB LSB								
	Port 0	**Port 1**		7							0	
P0_Stat_Rcv	**SMB86**			n	r	e	0	0	t	c	p	
P1_Stat_Rcv		**SMB186**										
P0_Stat_Rcv_7	SM86.7		n:	1	= Receive message was terminated by user disable command, else n = 0							
P1_Stat_Rcv_7		SM186.7										
P0_Stat_Rcv_6	SM86.6		r:	1	= Receive message terminated: (No start condition defined, character count of 0, or execution of receive message with transmit active) else, r = 0							
P1_Stat_Rcv_6		SM186.6										
P0_Stat_Rcv_5	SM86.5		e:	1	= End character received else, e = 0							
P1_Stat_Rcv_5		SM186.5										
P0_Stat_Rcv_2	SM86.2		t:	1	= Receive message terminated: timer expired else, t = 0							
P1_Stat_Rcv_2		SM186.2										
P0_Stat_Rcv_1	SM86.1		c:	1	= Receive message terminated: maximum character count reached else, c = 0							
P1_Stat_Rcv_1		SM186.1										
P0_Stat_Rcv_0	SM86.0		p:	1	= Receive message was terminated because of a parity, framing, break, or overrun error else, p = 0							
P1_Stat_Rcv_0		SM186.0										

			Bit Format	Receive message control byte MSB LSB								
	Port 0	**Port 1**		7							0	
P0_Ctrl_Rcv	**SMB87**			en	sc	ec	il	c/m	Tmr	bk	0	
P1_Ctrl_Rcv		**SMB187**										
P0_Ctrl_Rcv_7	SM87.7		en:	0	= Receive message function is disabled							
P1_Ctrl_Rcv_7		SM187.7		1	= Receive message function is enabled							
P0_Ctrl_Rcv_6	SM87.6		sc:	0	= Ignore SMB88 or SMB188							
P1_Ctrl_Rcv_6		SM187.6		1	= Use the value of SMB88 or SMB188 to detect start of message							
P0_Ctrl_Rcv_5	SM87.5		ec:	0	= Ignore SMB89 or SMB189							
P1_Ctrl_Rcv_5		SM187.5		1	= Use the value of SMB89 or SMB189 to detect end of message							
P0_Ctrl_Rcv_4	SM87.4		il:	0	= Ignore SMW90 or SMW190							
P1_Ctrl_Rcv_4		SM187.4		1	= Use the value of SMW90 or SMW190 to detect an idle line condition							
P0_Ctrl_Rcv_3	SM87.3		c/m:	0	= Timer is an inter-character timer							
P1_Ctrl_Rcv_3		SM187.3		1	= Timer is a message timer							
P0_Ctrl_Rcv_2	SM87.2		tmr:	0	= Ignore SMW92 or SMW192							
P1_Ctrl_Rcv_2		SM187.2		1	= Terminate receive if the time period in SMW92 or SMW192 is exceeded							
P0_Ctrl_Rcv_1	SM87.1		bk:	0	= Ignore break conditions							
P1_Ctrl_Rcv_1		SM187.1		1	= Use break condition as start of message detection							

You use the bits of the message control byte define the criteria for identifying the message. The message control byte also includes the start of message and end of message criteria. To determine the start of a message, either of two sets of logically ANDed start-of-message criteria must be true and must occur in sequence. "In sequence" means one of the following:

- Start character follows idle line
- Start character follows break

A logical OR of the end-of-message criteria determines the end of a message. The equations for start and end criteria are as follows:

Start of message = (il AND sc) OR (bk AND sc)

End of message = ec OR tmr OR maximum character count reached

S7-200 SMART symbol name	**SM address**		Bit Format	**Programming the start of a message criteria** MSB LSB								
	Port 0	**Port 1**		7							0	
P0_Ctrl_Rcv	SMB87			en	**sc**	ec	**il**	c/m	Tmr	**bk**	0	
P1_Ctrl_Rcv		SMB187										
Idle line detection					0		1			0		SMW90 > 0
Start character detection					1		0			0		SMW90 = don't care
Break detection					0		0			1		SMW90 = don't care
Any response to a request					0		1			0		SMW90 = 0 (Message timer can be used to terminate receive, if there is no response)
Break and a start character					1		0			1		SMW90 = don't care
Idle line and start character					1		1			0		SMW90 > 0
Idle line and start character (illegal)					1		1			0		SMW90 = 0
Note: Receive will automatically be terminated by a parity, framing, overrun or break error.												

S7-200 SMART symbol name	**SM address**		
	Port 0	**Port 1**	
P0_Start_Char	SMB88		Start of message character.
P1_Start_Char		SMB188	
P0_End_Char	SMB89		End of message character.
P1_End_Char		SMB189	
P0_Idle_Time	SMW90		Word data: Idle line time period given in milliseconds. The first character received after the idle line time has expired is the start of a new message.
P1_Idle_Time		SMW190	
P0_Timeout	SMW92		Word data: Inter-character/message timer timeout value given in milliseconds. If the time period is exceeded, the receive message is terminated.
P1_Timeout		SMW192	
P0_Max_Char	SMB94		Maximum number of characters to be received (1 to 255 bytes). **Note:** This range must be set to the expected maximum buffer size, even if the character count message termination is not used.
P1_Max_Char		SMB194	

D.17 SMW98: Expansion I/O bus communication errors

SMW98 gives you information about errors on the expansion I/O bus.

Table D- 18 SMW98 Expansion I/O bus communication error counter

S7-200 SMART symbol name	SM address (Read/Write)	Description
EM_Parity_Err	SMW98	The CPU increments this word each time the CPU detects a parity error on the expansion I/O bus. Power cycling the CPU clears the word or writing a zero to the word.

D.18 SMW100-SMW114 System alarms

Special memory words SMW100-SMW114 provide alarm and diagnostic error codes for CPU, SB (signal board), and EM (expansion modules).

S7-200 SMART symbol name	SM address	Current diagnostic alarm source
CPU_Alarm	SMW100	CPU diagnostic alarm code
SB_Alarm	SMW102	Signal board diagnostic alarm code
EM0_Alarm	SMW104	Expansion module bus slot 0 diagnostic alarm code
EM1_Alarm	SMW106	Expansion module bus slot 1 diagnostic alarm code
EM2_Alarm	SMW108	Expansion module bus slot 2 diagnostic alarm code
EM3_Alarm	SMW110	Expansion module bus slot 3 diagnostic alarm code
EM4_Alarm	SMW112	Expansion module bus slot 4 diagnostic alarm code
EM5_Alarm	SMW114	Expansion module bus slot 5 diagnostic alarm code

Alarm code format	MSB															LSB	
	15	14	13					8	7							0	
	d	s	c	c	c	c	c	c	a	a	a	a	a	a	a	a	
d: Alarm direction	0	Input channel or not applicable															
	1	Output channel															
s: Alarm scope		0	On individual channel														
		1	On entire module														
c: Channel number			c	c	c	c	c	c	Number of the affected channel, if alarm scope = “On individual channel” or 0, if alarm scope = “On entire module”								
a: Alarm type									0	0	0	0	0	0	0	0	00H: No alarm
									0	0	0	0	0	0	0	1	01H: Short circuit
									0	0	0	0	0	x	x	x	02H to 05H: Reserved
									0	0	0	0	0	1	1	0	06H: Wire break
									0	0	0	0	0	1	1	1	07H: Upper limit exceeded
									0	0	0	0	1	0	0	0	08H: Lower limit exceeded
									0	0	0	0	x	x	x	x	09H to 0FH: Reserved
									0	0	0	1	0	0	0	0	10H: Parameterization error
									0	0	0	1	0	0	0	1	11H: Sensor or load voltage missing
									0	0	0	x	x	x	x	x	12H to 1FH: Reserved
									0	0	1	0	0	0	0	0	20H: Internal error (MID problem)
									0	0	1	0	0	0	0	1	21H: Internal error (IID problem)
									0	0	1	0	0	0	1	0	22H: Reserved
									0	0	1	0	0	0	1	1	23H: Configuration error
									0	0	1	0	0	1	0	0	24H: Reserved
									0	0	1	0	0	1	0	1	25H: Bad or missing firmware
									0	0	1	0	x	x	x	x	26H to 2AH: Reserved
									0	0	1	0	1	0	1	1	2BH: Battery voltage low
									x	x	x	x	x	x	x	x	2CH to FFH: Reserved

D.19 SMB130: Freeport control for port 1 (See SMB30)

Refer to "SMB30: (Port 0) and SMB130: (Port 1)" (Page 873) for details.

D.20 SMB146-SMB155 (HSC4) and SMB156-SMB165 (HSC5)

Refer to "SMB36-45 (HSC0), SMB46-55 (HSC1), SMB56-65 (HSC2), SMB136-145 (HSC3), SMB146-SMB155 (HSC4), SMB156-SMB165 (HSC5): high-speed counters" (Page 875) for details.

D.21 SMB186-SMB194: Receive message control (See SMB86-SMB94)

Refer to "SMB86-SMB94 and SMB186-SMB194" (Page 883) for details.

D.22 SMB480-SMB515: Data log status

SMB480 through SMB515 are read-only special memory addresses that indicate the status of Data log operations.

S7-200 SMART symbol name	SM address	Function
DL0_InitResult	SMB480	Initialization result code for Data Log 0: After power-up and after a down-load of the System block, the CPU performs the data log analysis. • 00H: data log OK • 01H: initialization in progress • 02H: data log file not found • 03H: data log initialization error • 04H to FEH: reserved • FFH: data log not configured
DL1_InitResult	SMB481	Initialization result code for Data Log 1 (See SMB 480 for result codes)
DL2_InitResult	SMB482	Initialization result code for Data Log 2 (See SMB 480 for result codes)
DL3_InitResult	SMB483	Initialization result code for Data Log 3 (See SMB 480 for result codes)
DL0_Maximum	SMW500	Data log 0: Configured maximum allowed number of records
DL0_Current	SMW502	Data log 0: Actual maximum allowed number of records
DL1_Maximum	SMW504	Data log 1: Configured maximum allowed number of records
DL1_Current	SMW506	Data log 1: Actual maximum allowed number of records
DL2_Maximum	SMW508	Data log 2: Configured maximum allowed number of records
DL2_Current	SMW510	Data log 2: Actual maximum allowed number of records
DL3_Maximum	SMW512	Data log 3: Configured maximum allowed number of records
DL3_Current	SMW514	Data log 3: Actual maximum allowed number of records

D.23 SMB600-SMB749: Axis (0, 1, and 2) open loop motion control

Axis configuration and control SM addresses

Wizard-generated program code reads and writes the axis special memory data.

Axis data SM address			Axis function
Axis 0	**Axis 1**	**Axis 2**	
SMB600-SMB615	SMB650-SMB665	SMB600-SMB615	Axis name (16 ASCII characters). The first character is the lowest numbered byte in the sequence.
SMB616-SMB619	SMB616-SMB619	SMB616-SMB619	Reserved
SMW620	SMW670	SMW720	Error code - See Axis of Motion error codes (Page 731)

Axis 0 SMB622	**Axis 1 SMB672**	**Axis 2 SMB722**	MSB 7	6	5	4	3	2	1	LSB 0
			DIS	0	**TRIG**	**STP**	**LMT-**	**LM +**	**RPS**	**ZP**
SM622.7	SM672.7	SM722.7	**DIS:** disable outputs	0 = No current flow; 1 = Current flow						
SM622.5	SM672.5	SM722.5	**TRIG:** Trigger input	0 = Not active; 1 = Active						
SM622.4	SM672.4	SM722.4	**STP :** Stop input	0 = Not active; 1 = Active						
SM622.3	SM672.3	SM722.3	**LMT - :** Negative travel limit input	0 = Not active; 1 = Active						
SM622.2	SM672.2	SM722.2	**LMT + :** Positive travel limit input	0 = Not active; 1 = Active						
SM622.1	SM672.1	SM722.1	**RPS :** Reference point switch input	0 = Not active; 1 = Active						
SM622.0	SM672.0	SM722.0	**ZP :** Zero pulse input	0 = Not active; 1 = Active						

Axis 0 SMB623	**Axis 1 SMB673**	**Axis 2 SMB723**	MSB 7	6	5	4	3	2	1	LSB 0
			0	0	0	0	0	**OR**	**R**	**CFG**
SM623.2	SM673.2	SM723.2	**OR**:Target speed out of range	0 = In range; 1 = Out of range						
SM623.1	SM673.1	SM723.1	**R:** Direction of rotation	0 = Positive rotation; 1 = Negative rotation						
SM623.0	SM673.0	SM723.0	**CFG**: Axis configured	0 = Not configured; 1 = Configured						

Axis data SM address			Axis function
Axis 0	**Axis 1**	**Axis 2**	
SMB624	SMB674	SMB724	**CUR_PF** is a byte that indicates which profile the Axis of Motion is currently executing.
SMB625	SMB675	SMB725	**CUR_STP** is a byte that indicates which step in the profile the Axis of Motion is currently executing.
SMD626	SMD676	SMD726	**CUR_POS** is a double-word value that indicates the current position of the axis.
SMD630	SMD680	SMD730	**CUR_SPD** is a double-word value that indicates the current speed of the axis.

Axis 0 SMB634	**Axis 1 SMB684**	**Axis 2 SMB734**		MSB 7	6	5	4	3	2	1	LSB 0	
				D	**Error**:							
SM634.7	SM684.7	SM734.7	**D**: Done bit	0								
				1	Operation complete (set by the axis during initialization)							
SM634.0-SM634.6	SM684.0-SM684.6	SM734.0-SM734.6	**Error**:	See motion instruction error codes								

Axis data SM address			Axis function
Axis 0	**Axis 1**	**Axis 2**	
SMB635-SMB645	SMB685-SMB695	SMB735-SMB745	Reserved
SMD646	SMD646	SMD746	V memory pointer to the configuration/profile table for the axis. A pointer value to an area other than V memory is invalid.

D.24 SMB650-SMB699: Axis 1 open loop motion control (See SMB600-SMB740)

Refer to "SMB600-SMB749: Axis (0, 1, and 2) open loop motion control (Page 888)" for details.

D.25 SMB700-SMB749: Axis 2 open loop motion control (See SMB600-SMB740)

Refer to "SMB600-SMB749: Axis (0, 1, and 2) open loop motion control (Page 888)" for details.

D.26 SMB1000-SMB1049: CPU hardware/firmware ID

This CPU writes the information to special memory after a power-up or warm restart transition. The SMB1000-SMB1049 section of special memory is read-only.

SM address	Description
SMW1000	CPU vendor ID: (always 0x002A)
SMB1002 to SMB1021	CPU order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1022 to SMB1037	CPU serial number: ASCII characters, left-justified in field, padded with spaces
SMW1038	CPU hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.

SM address	Description
SMD1040	CPU firmware version: • Byte 0: ASCII 'V' • Byte 1: Functional version • Byte 2: Minor change version • Byte 3: Bug fix version
SMW1044	CPU firmware version counter (range 0x0000 to 0x00FF)
SMW1046	Reserved: Always 0x0000
SMW1048	CPU device type: Always 0x0001

D.27 SMB1050-SMB1099: SB (signal board) hardware/firmware ID

The CPU writes the signal board information to special memory after a power-up or warm restart transition. The SMB1050-SMB1099 section of special memory is read-only.

SM address	Description
SMW1050	Signal board vendor ID: 0x002A if a Siemens SB is present; 0x0000 if no SB is present
SMB1052 to SMB1071	Signal board order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1072 to SMB1087	Signal board serial number: ASCII characters, left-justified in field, padded with spaces
SMW1088	Signal board hardware version: Represents the hardware E-stand; range=0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.
SMD1090	Signal board firmware version: • Byte 0: ASCII 'V' • Byte 1: Functional version • Byte 2: Minor change version • Byte 3: Bug fix version
SMW1094	Signal board firmware version counter (range 0x0000 to 0x00FF)
SMW1096	Reserved, always 0x0000
SMW1098	Signal board device type: I/O=0x0003, communications=0x0004, all other values reserved

D.28 SMB1100-SMB1399: EM (expansion module) hardware/firmware ID

The CPU writes expansion module information to special memory after a power-up or warm restart transition. The SMB1100-SMB1399 section of special memory is read-only.

SM addresses for slot 0	Description
SMW1100	EM bus Slot 0 vendor ID: 0x002A if a Siemens EM is present; 0x0000 if no EM is present
SMB1102 to SMB1121	EM bus Slot 0 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1122 to SMB1137	EM bus slot 0 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1138	EM bus slot 0 hardware version: represents the hardware E-stand; range = 0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.

SM addresses for slot 0	Description
SMD1140	EM bus slot 0 firmware version: • Byte 0: ASCII 'V'; • Byte 1: Functional version, • Byte 2: Minor change version, • Byte 3: Bug fix version
SMW1144	EM bus slot 0 firmware version counter (range 0x0000 to 0x00FF)
SMW1146	Reserved, always 0x0000
SMW1148	EM bus slot 0 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 1	Description
SMW1150	EM bus slot 1 vendor ID: 0x002A if a Siemens EM is present; 0x0000 if no EM is present
SMB1152 to SMB1171	EM bus slot 1 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1172 to SMB1187	EM bus slot 1: serial number: ASCII characters, left-justified in field, padded with spaces
SMW1188	EM bus slot 1 hardware version: Represents the hardware E-stand; range=0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.
SMD1190	EM bus slot 1 firmware version: • Byte 0: ASCII 'V' • Byte 1: Functional version • Byte 2: Minor change version • Byte 3: Bug fix version
SMW1194	EM bus slot 1 firmware version counter (range 0x0000 to 0x00FF)
SMW1196	Reserved, always 0x0000
SMW1198	EM bus slot 1 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 2	Description
SMW1200	EM bus slot 2 vendor ID: 0x002A if a Siemens EM is present; 0x0000 if no EM is present
SMB1202 to SMB1221	EM bus slot 2 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1222 to SMB1237	EM bus slot 2 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1238	EM bus slot 2 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.
SMD1240	EM bus slot 2 firmware version: • Byte 0: ASCII 'V' • Byte 1: Functional version • Byte 2: Minor change version • Byte 3: Bug fix version
SMW1244	EM bus slot 2 firmware version counter (range 0x0000 to 0x00FF)
SMW1246	Reserved, always 0x0000
SMW1248	EM bus slot 2 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 3	Description
SMW1250	EM bus slot 3 vendor ID: 0x002A if a Siemens EM is present; 0x0000 if no EM is present
SMB1252 to SMB1271	EM bus slot 3 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1272 to SMB1287	EM bus slot 3 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1288	EM bus slot 3 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.
SMD1290	EM bus slot 3 firmware version: • Byte 0: ASCII 'V' • Byte 1: Functional version • Byte 2: Minor change version • Byte 3: Bug fix version
SMW1294	EM bus slot 3 firmware version counter (range 0x0000 to 0x00FF)
SMW1296	Reserved, always 0x0000
SMW1298	EM bus slot 3 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 4	Description
SMW1300	EM bus slot 4 vendor ID: 0x002A if a Siemens EM is present; 0x0000 if no EM is present
SMB1302 to SMB1321	EM bus slot 4 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1322 to SMB1327	EM bus slot 4 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1338	EM bus slot 4 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.
SMD1340	EM bus slot 4 firmware version: • Byte 0: ASCII 'V' • Byte 1: Functional version • Byte 2: Minor change version • Byte 3: Bug fix version
SMW1344	EM bus slot 4 firmware version counter (range 0x0000 to 0x00FF)
SMW1346	Reserved, always 0x0000
SMW1348	EM bus slot 4 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 5	Description
SMW1350	EM bus slot 5 vendor ID: 0x002A if a Siemens EM is present; 0x0000 if no EM is present
SMB1352 to SMB1371	EM bus slot 5 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1372 to SMB1387	EM bus slot 5 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1388	EM bus slot 5 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFD 0x0000, 0xFFFE, and 0xFFFF are reserved values.

SM addresses for slot 5	Description
SMD1390	EM bus slot 5 firmware version: • Byte 0: ASCII 'V' • Byte 1: Functional version • Byte 2: Minor change version • Byte 3: Bug fix version
SMW1394	EM bus slot 5 firmware version counter (range 0x0000 to 0x00FF)
SMW1396	Reserved, always 0x0000
SMW1398	EM bus slot 5 device type: I/O=0x0003, communications=0x0004, all other values reserved

D.29 SMB1400-SMB1699: EM (expansion module) module-specific data

The CPU reserves an additional 50 bytes for each expansion module for read-only module-specific data:

SM addresses	Description
SMB1400 to SMB1449	EM bus Slot 0: Module specific information
SMB1450 to SMB1499	EM bus Slot 1: Module specific information
SMB1500 to SMB1549	EM bus Slot 2: Module specific information
SMB1550 to SMB1599	EM bus Slot 3: Module specific information
SMB1600 to SMB1649	EM bus Slot 4: Module specific information
SMB1650 to SMB1699	EM bus Slot 5: Module specific information

D.30 SMB1800-SMB1999: PROFINET device status

The SMB1800-SMB1935 section indicates the status of IO device or cyclic data if a S7-200 SMART CPU works as a controller.

SM address	Description
SMB1800 to SMB1807	Every byte indicates the status of each PROFINET device. • 0x00: Not configured. • 0x80: OK. • 0×81: Not connected. (The controller cannot connect with the device.) • 0×82: Diagnosis. (An alarm is reported.)
SMB1808 to SMB1871	Alarm status of each module. SMB1808~SMB1815 has 64 bits, which shows the alarm status of the device #1 (Device Number =1). 0 is "OK", 1 is "Error". For example, SM1808.0 indicates the first module status of the first device. SM1816.0 indicates the first module status of the second device.
SMB1872 to SMB1935	IO data status of IO device module. SMB1872~SMB1879 has 64 bits, which shows the alarm status of IO data in the device #1 (Device Number =1). 0 is "OK", 1 is "IO data error". For example, SM1872.0 indicates the first module IO data status of the first device. SM1880.2 indicates the third module IO data status of the second device.

The SM1936.0- SM1937.0 section shows the I-Device's connection status with its higher-level controller and the I-Device IO data status if a S7-200 SMART CPU works as an I-Device.

SM address	Description
SMB1936	This byte indicates the PROFINET I-Device's connection status with its higher-level controller. • 0x00: Not configured. • 0x80: OK. • 0×81: Not connected. (The I-Device is not connected with the higher-level controller.) • 0×82: Diagnosis. (The I-Device is connected with the higher-level controller, but the I-Device configuration doesn't match with the higher-level controller configuration.)
SM1937.0	I-Device IO data status. 0 is "OK", 1 is "IO data error".
SM1937.1 to SMB1999	Reserved.

References

E

E.1 Often-used special memory bits

The complete list of pre-defined special memory program symbols is available to your project, in the System Symbol table.

Table E- 1 Often-used special memory bits

SM address	System symbol name	Description
SM0.0	Always_On	This bit is always TRUE.
SM0.1	First_Scan_On	The CPU sets this bit to TRUE, for the first scan cycle, and sets it to FALSE thereafter. One use for this bit is to call an initialization subroutine.
SM0.2	Retentive_Lost	TRUE for one scan cycle if retentive data is lost
SM0.3	RUN_Power_Up	TRUE for one scan cycle when RUN mode is entered from a power-up condition
SM0.4	Clock_60s	Clock pulse that is TRUE for 30 s, OFF for 30 s, for a cycle time of 1 min.
SM0.5	Clock_1s	Clock pulse that is TRUE for 0.5 s, OFF for 0.5 s, for a cycle time of 1 s.
SM0.6	Clock_Scan	Scan cycle clock that is TRUE for one scan cycle and OFF for the next scan cycle
SM0.7	RTC_Lost	For CPU models that have a real-time clock, the CPU sets this bit TRUE for one scan cycle if the time on the real-time clock device was reset or lost at power up. The program can use this bit as either an error memory bit or to invoke a special start-up sequence.
SM1.0	Result_0	Set to TRUE by the execution of certain instructions when the result of the operation = 0
SM1.1	Overflow_Illegal	Set to TRUE by the execution of certain instructions on overflow or illegal numeric value
SM1.2	Neg_Result	Set to TRUE when a math operation produces a negative result
SM1.3	Divide_By_0	Set to TRUE when an attempt is made to divide by zero
SM1.4	Table_Overflow	Set to TRUE when the Add to Table instruction attempts to overfill the table
SM1.5	Table_Empty	Set to TRUE when a LIFO or FIFO instruction attempts to read from an empty table
SM1.6	Not_BCD	Set to TRUE when an attempt is made to convert a non-BCD value to a binary value
SM1.7	Not_Hex	Set to TRUE when an ASCII value cannot be converted to a valid hexadecimal value

E.2 Interrupt events in priority order

Table E- 2 Priority order for interrupt events

Priority group	Event	Description
Communications Highest Priority	8	Port 0 Receive character
	9	Port 0 Transmit complete
	23	Port 0 Receive message complete
	24	Port 1 Receive message complete
	25	Port 1 Receive character
	26	Port 1 Transmit complete
Discrete Medium Priority	19	PTO0 Pulse count complete
	20	PTO1 Pulse count complete
	34	PTO2 Pulse count complete
	0	I0.0 Rising edge
	2	I0.1 Rising edge
	4	I0.2 Rising edge
	6	I0.3 Rising edge
	35	I7.0 Rising edge (signal board)
	37	I7.1 Rising edge (signal board)
	1	I0.0 Falling edge
	3	I0.1 Falling edge
	5	I0.2 Falling edge
	7	I0.3 Falling edge
	36	I7.0 Falling edge (signal board)
	38	I7.1 Falling edge (signal board)
	12	HSC0 CV=PV (current value = preset value)
	27	HSC0 Direction changed
	28	HSC0 External reset
	13	HSC1 CV=PV (current value = preset value)
	16	HSC2 CV=PV (current value = preset value)
	17	HSC2 Direction changed
	18	HSC2 External reset
	32	HSC3 CV=PV (current value = preset value)
	29	HSC4 CV=PV
	30	HSC4 direction changed
	31	HSC4 external reset
	33	HSC5 CV=PV
	43	HSC5 direction changed
	44	HSC5 external reset
Timed Lowest Priority	10	Timed interrupt 0 SMB34
	11	Timed interrupt 1 SMB35
	21	Timer T32 CT=PT interrupt

Priority group	Event	Description
	22	Timer T96 CT=PT interrupt

E.3 High-speed counter summary

	Clock A	Direction / Clock B	Reset	Single phase / Dual phase maximum clock / input rate	AB quadrature phase maximum clock / input rate
HSC0	I0.0	I0.1	I0.4	S model CPUs:[1] • 200 kHz	S model CPUs: • 100 kHz = Maximum 1x count rate • 400 kHz = Maximum 4x count rate
				C model CPUs:[2] • 100 kHz	C model CPUs: • 50 kHz = Maximum 1x count rate • 200 kHz = Maximum 4x count rate
HSC1	I0.1			S model CPUs: • 200 kHz	
				C model CPUs: • 100 kHz	
HSC2	I0.2	I0.3	I0.5	S model CPUs: • 200 kHz	S model CPUs: • 100 kHz = Maximum 1x count rate • 400 kHz = Maximum 4x count rate
				C model CPUs: • 100 kHz	C model CPUs: • 50 kHz = Maximum 1x count rate • 200 kHz = Maximum 4x count rate
HSC3	I0.3			S model CPUs: • 200 kHz	
				C model CPUs: • 100 kHz	
HSC4	I0.6	I0.7	I1.2	SR30 and ST30 model CPUs: • 200 kHz	SR30 and ST30 model CPUs: • 100 kHz = Maximum 1x count rate • 400 kHz = Maximum 4x count rate
				SR20, ST20, SR40, ST40, SR60, and ST60 model CPUs: • 30 kHz	SR20, ST20, SR40, ST40, SR60, and ST60 model CPUs: • 20 kHz = Maximum 1x count rate • 80 kHz = Maximum 4x count rate
				C model CPUs: • n/a	C model CPUs: • n/a

	Clock A	Direction / Clock B	Reset	Single phase / Dual phase maximum clock / input rate	AB quadrature phase maximum clock / input rate
HSC5	I1.0	I1.1	I1.3	S model CPUs: • 30 kHz	S model CPUs • 20 kHz = Maximum 1x count rate • 80 kHz = Maximum 4x count rate
				C model CPUs: • n/a	C model CPUs: • n/a

[1] S model CPUs: SR20, ST20, SR30, ST30, SR40, ST40, SR60, and ST60

[2] C model CPUs: CR20s, CR30s, CR40s, and CR60s

E.4 STL instructions

Instructions

The STL instruction names and descriptions are shown in the tables below. See the chapter on program instructions (Page 163) for the LAD and FBD instructions.

Boolean instructions	
STL	**Description**
LD bit LDI bit LDN bit LDNI bit	Load Load Immediate Load Not Load Not Immediate
A bit AI bit AN bit ANI bit	AND AND Immediate AND Not AND Not Immediate
O bit OI bit ON bit ONI bit	OR OR Immediate OR Not OR Not Immediate
LDBx IN1, IN2	Load result of Byte Compare IN1 (x:<, <=,=, >=, >, <>I) IN2
ABx IN1, IN2	AND result of Byte Compare IN1 (x:<, <=,=, >=, >, <>) IN2
OBx IN1, IN2	OR result of Byte Compare IN1 (x:<, <=,=, >=, >, <>) IN2
LDWx IN1, IN2	Load result of Word Compare IN1 (x:<, <=,=, >=, >, <>) IN2
AWx IN1, IN2	AND result of Word Compare IN1 (x:<, <=,=, >=, >, <>)I N2

Boolean instructions	
STL	**Description**
OWxIN1, IN2	OR result of Word Compare IN1 (x:<, <=,=, >=, >, <>) IN2
LDDx IN1, IN2	Load result of DWord Compare IN1 (x:<, <=,=, >=, >, <>) IN2
ADx IN1, IN2	AND result of DWord Compare IN1 (x:<, <=,=, >=, >, <>)IN2
ODx IN1, IN2	OR result of DWord Compare IN1 (x:<, <=,=, >=, >, <>) IN2
LDRx IN1, IN2	Load result of Real Compare IN1 (x:<, <=,=, >=, >, <>) IN2
ARx IN1, IN2	AND result of Real Compare IN1 (x:<, <=,=, >=, >, <>) IN2
ORx IN1, IN2	OR result of Real Compare IN1 (x:<, <=,=, >=, >, <>) IN2
NOT	Stack Negation
EU ED	Detection of Rising Edge Detection of Falling Edge
= bit =I bit	Assign Value Assign Value Immediate
S bit, N R bit, N SI bit, N RI bit, N	Set bit Range Reset bit Range Set bit Range Immediate Reset bit Range Immediate
Not available in STL	SR (Set dominate bistable) RS (Reset dominate bistable)
LDSx IN1, IN2 ASx IN1, IN2 OSx IN1, IN2	Load result of String Compare IN1 (x: =, <>) IN2 AND result of String Compare IN1 (x: =, <>) IN2 OR result of String Compare IN1 (x: =, <>) IN2
ALD OLD	AND Load OR Load
LPS LRD LPP LDS n	Logic Push (stack control) Logic Read (stack control) Logic Pop (stack control) Load Stack (stack control), n= 0 to 8
AENO	AND ENO

Math, increment, and decrement instructions	
STL	**Description**
+I IN1, OUT +D IN1, OUT +R IN1, OUT	Add Integer, Double Integer or Real IN1+OUT=OUT
-I IN1, OUT -D IN1, OUT -R IN1, OUT	Subtract Integer, Double Integer, or Real OUT-IN1=OUT
MUL IN1, OUT	Multiply Integer (16*16->32)
*I IN1, OUT *D IN1, OUT *R IN1, IN2	Multiply Integer, Double Integer, or Real IN1 * OUT = OUT
DIV IN1, OUT	Divide Integer (16/16->32)
/I IN1, OUT /D,IN1, OUT /R IN1, OUT	Divide Integer, Double Integer, or Real OUT / IN1 = OUT
SQRT IN, OUT	Square Root
LN IN, OUT	Natural Logarithm
EXP IN, OUT	Natural Exponential
SIN IN, OUT	Sine
COS IN, OUT	Cosine
TAN IN, OUT	Tangent
INCB OUT INCW OUT INCD OUT	Increment Byte, Word or DWord
DECB OUT DECW OUT DECD OUT	Decrement Byte, Word, or DWord
PID TBL, LOOP	PID Loop

Timer and counter instructions	
STL	**Description**
TON Txxx, PT	On-Delay Timer
TOF Txxx, PT	Off-Delay Timer
TONR Txxx, PT	Retentive On-Delay Timer
BITIM OUT	Beginning Interval Timer
CITIM IN, OUT	Calculate Interval Timer
CTU Cxxx, PV	Count Up
CTD Cxxx, PV	Count Down
CTUD Cxxx, PV	Count Up/Down

High-speed instructions	
STL	**Description**
HDEF HSC, MODE	Define High-Speed Counter mode
HSC n	Activate High-Speed Counter
PLS n	Pulse Output:

Real time clock instructions	
STL	**Description**
TODR T	Read Time of Day clock
TODW T	Write Time of Day clock
TODRX T	Read Real Time Clock Extended
TODWX T	Set Real Time Clock Extended

Program control instructions	
STL	**Description**
END	Conditional End of Program
STOP	Transition to STOP Mode
WDR	WatchDog Reset (500 ms)
JMP N	Jump to defined Label
LBL N	Define a Label
CALL N [N1,...]	Call a Subroutine [N1, ... up to 16 optional parameters]
CRET	Conditional Return from SBR
FOR INDX,INIT,FINAL	For/Next Loop
NEXT	
LSCR N	Load, Transition, Conditional End, and End Sequence Control Relay segment
SCRT N	
CSCRE	
SCRE	
GERR ECODE	Get Error code

Move, Shift, and Rotate instructions	
STL	**Description**
MOVB IN, OUT	Move Byte, Word, DWord, Real
MOVW IN, OUT	
MOVD IN, OUT	
MOVR IN, OUT	
BIR IN, OUT	Move Byte Immediate Read
BIW IN, OUT	Move Byte Immediate Write

Move, Shift, and Rotate instructions	
STL	**Description**
BMB IN, OUT, N BMW IN, OUT, N BMD IN, OUT, N	Block Move Byte, Word, DWord
SWAP IN	Swap Bytes
SHRB DATA, S_BIT, N	Shift Register Bit
SRB OUT, N SRW OUT, N SRD OUT, N	Shift Right Byte, Word, DWord
SLB OUT, N SLW OUT, N SLD OUT, N	Shift Left Byte, Word, DWord
RRB OUT, N RRW OUT, N RRD OUT, N	Rotate Right Byte, Word, DWord
RLB OUT, N RLW OUT, N RLD OUT, N	Rotate Left Byte, Word, DWord

Logical instructions	
STL	**Description**
ANDB IN1, OUT ANDW IN1, OUT ANDD IN1, OUT	Logical AND of Byte, Word, and DWord
ORB IN1, OUT ORW IN1, OUT ORD IN1, OUT	Logical OR of Byte, Word, and DWord
XORB IN1, OUT XORW IN1, OUT XORD IN1, OUT	Logical XOR of Byte, Word, and DWord
INVB OUT INVW OUT INVD OUT	Invert Byte, Word and DWord (1's complement)

String instructions	
STL	**Description**
SLEN IN, OUT	String Length
SCAT N, OUT	Concatenate String
SCPY IN, OUT	Copy String
SSCPY IN, INDX, N, OUT	Copy Substring from String
CFND IN1, IN2, OUT	Find First Character within String
SFND IN1, IN2, OUT	Find String within String

Table, Find, and Conversion instructions	
STL	**Description**
ATT DATA, TBL	Add data to table
LIFO TBL, DATA FIFO TBL, DATA	Get data from table
FND= TBL, PTN, INDX FND<> TBL, PTN, INDX FND< TBL, PTN, INDX FND> TBL, PTN, INDX	Find data value in table that matches comparison
FILL IN, OUT, N	Fill memory space with pattern
BCDI OUT IBCD OUT	Convert BCD to Integer Convert Integer to BCD
BTI IN, OUT ITB IN, OUT ITD IN, OUT DTI IN, OUT	Convert Byte to Integer Convert Integer to Byte Convert Integer to Double Integer Convert Double Integer to Integer
DTR IN, OUT TRUNC IN, OUT ROUND IN, OUT	Convert DWord to Real Convert Real to Double Integer Convert Real to Double Integer
ATH IN, OUT, LEN HTA IN, OUT, LEN ITA IN, OUT, FMT DTA IN, OUT, FM RTA IN, OUT, FM	Convert ASCII to Hex Convert Hex to ASCII Convert Integer to ASCII Convert Double Integer to ASCII Convert Real to ASCII
DECO IN, OUT ENCO IN, OUT	Decode Encode
SEG IN, OUT	Generate 7-segment pattern
ITS IN, FMT, OUT DTS IN, FMT, OUT RTS IN, FMT, OUT	Convert Integer to String Convert Double Integer to String Convert Real to String
STI STR, INDX, OUT STD STR, INDX, OUT STR STR, INDX, OUT	Convert Substring to Integer Convert Substring to Double Integer Convert Substring to Real

Interrupt instructions	
STL	**Description**
CRETI	Conditional Return from Interrupt
ENI	Enable Interrupts
DISI	Disable Interrupts
ATCH INT, EVNT	Attach Interrupt routine to event
DTCH EVNT	Detach event
CEVENT EVNT	Clear all interrupt events of type EVNT

Communications instructions	
STL	**LAD/FBD**
GET	Reads remote station data
PUT	Writes data to a remote station
XMT TBL, PORT	Freeport transmission
RCV TBL, PORT	Freeport receive message
GIP ADDR, MASK, GATE	Get CPU address, subnet mask, and gateway
SIP ADDR, MASK, GATE	Set CPU address, subnet mask, and gateway
GPA ADDR, PORT	Get Port Address
SPA ADDR, PORT	Set Port Address

Note

The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port and do not support any functions related to the use of Ethernet communications.

E.5 Memory ranges and features

The following table provides the memory ranges and features for CPUs of firmware V2.5. For CPUs of firmware versions prior to V2.5, refer to the *S7-200 SMART System Manual* for your specific CPU model and version.

Description	CPU CR20s, CPU CR30s, CPU CR40s, CPU CR40, CPU CR60s, CPU CR60	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU SR60, CPU ST60
User program size	12 KB	12 KB	18 KB	24 KB	30 KB
User data size	8 KB	8 KB	12 KB	16 KB	20 KB

Description		CPU CR20s, CPU CR30s, CPU CR40s, CPU CR40, CPU CR60s, CPU CR60	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU SR60, CPU ST60
Process image input register		I0.0 to I31.7	I0.0 to I31.7	I0.0 to I31.7	I0.0 to I31.7	I0.0 to I31.7
Process image output register		Q0.0 to Q31.7	Q0.0 to Q31.7	Q0.0 to Q31.7	Q0.0 to Q31.7	Q0.0 to Q31.7
Analog inputs (read only)		--	AIW0 to AIW110	AIW0 to AIW110	AIW0 to AIW110	AIW0 to AIW110
Analog outputs (write only)		--	AQW0 to AQW110	AQW0 to AQW110	AQW0 to AQW110	AQW0 to AQW110
Variable memory (V)		VB0 to VB8191	VB0 to VB8191	VB0 to VB12287	VB0 to VB16383	VB0 to VB20479
Local memory (L)[1]		LB0 to LB63	LB0 to LB63	LB0 to LB63	LB0 to LB63	LB0 to LB63
Bit memory (M)		M0.0 to M31.7	M0.0 to M31.7	M0 to M31.7	M0.0 to M31.7	M0.0 to M31.7
Special Memory (SM)	Total	SM0.0 to SM2047.7	SM0.0 to SM2047.7	SM0.0 to SM2047.7	SM0.0 to SM2047.7	SM0.0 to SM2047.7
	Read only	SM0.0 to SM29.7 SMB480.0 to SM515.7 SM1000.0 to SM1699.7	SM0.0 to SM29.7 SMB480.0 to SM515.7 SM1000.0 to SM1699.7	SM0.0 to SM29.7 SMB480.0 to SM515.7 SM1000.0 to SM1699.7	SM0.0 to SM29.7 SMB480.0 to SM515.7 SM1000.0 to SM1699.7	SM0.0 to SM29.7 SMB480.0 to SM515.7 SM1000.0 to SM1699.7
Timers		256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)
Retentive on-delay	1 ms	T0, T64	T0, T64	T0, T64	T0, T64	T0, T64
	10 ms	T1 to T4, and T65 to T68	T1 to T4, and T65 to T68	T1 to T4, and T65 to T68	T1 to T4, and T65 to T68	T1 to T4, and T65 to T68
	100 ms	T5 to T31, and T69 to T95	T5 to T31, and T69 to T95	T5 to T31, and T69 to T95	T5 to T31, and T69 to T95	T5 to T31, and T69 to T95
On/Off delay	1 ms	T32, T96	T32, T96	T32, T96	T32, T96	T32, T96
	10 ms	T33 to T36, and T97 to T100	T33 to T36, and T97 to T100	T33 to T36, and T97 to T100	T33 to T36, and T97 to T100	T33 to T36, and T97 to T100
	100 ms	T37 to T63, and T101 to T255	T37 to T63, and T101 to T255	T37 to T63, and T101 to T255	T37 to T63, and T101 to T255	T37 to T63, and T101 to T255
Counters		256 (C0 to C255)	256 (C0 to C255)	256 (C0 to C255)	256 (C0 to C255)	256 (C0 to C255)
High-speed counters		HC0 to HC3	HC0 to HC5	HC0 to HC5	HC0 to HC5	HC0 to HC5
Sequential control relays (S)		S0.0 to S31.7	S0.0 to S31.7	S0.0 to S31.7	S0.0 to S31.7	S0.0 to S31.7
Accumulator registers		AC0 to AC3	AC0 to AC3	AC0 to AC3	AC0 to AC3	AC0 to AC3
Jumps/Labels		0 to 255	0 to 255	0 to 255	0 to 255	0 to 255
Call/Subroutine		0 to 127	0 to 127	0 to 127	0 to 127	0 to 127
Interrupt routines		0 to 127	0 to 127	0 to 127	0 to 127	0 to 127
Positive/negative transitions		1024	1024	1024	1024	1024

Description	CPU CR20s, CPU CR30s, CPU CR40s, CPU CR40, CPU CR60s, CPU CR60	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU SR60, CPU ST60
PID control loops	0 to 7	0 to 7	0 to 7	0 to 7	0 to 7
Ports	Integrated RS485 port (Port 0) [2]	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)

[1] LB60 to LB63 are reserved by STEP 7-Micro/WIN SMART when programming in LAD or FBD.

[2] The CPU models CPU CR20s, CPU CR30s, CPU CR40s, and CPU CR60s have no Ethernet port. These CPUs do not support any functions related to the use of Ethernet communications.

The S7-200 SMART CPU firmware release V2.4 and later versions supports the PROFINET communication on the following eight CPU models. For the detailed parameters and PROFINET process image information, refer to the following table.

Description	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU SR60, CPU ST60
Maximum number of PROFINET device	8			
Device Number of PROFINET device	1 to 8			
Maximum input size of each PROFINET device	128 Bytes			
Maximum output size of each PROFINET device	128 Bytes			
Maximum number of modules	64			
Minimum cyclic update time of PROFINET device	The minimum value of the update time also depends on the communication component set for PROFINET, on the number of PROFINET devices, and the quantity of configured user data.			
CPU address range of PROFINET process image input register	I128.0 to I1151.7			
CPU address range of PROFINET process image output register	Q128.0 to Q1151.7			
CPU address of PROFINET process image input register for device #1	I128.0 to I255.7			
CPU address of PROFINET process image input register for device #2	I256.0 to I383.7			
CPU address of PROFINET process image input register for device #3	I384.0 to I511.7			
CPU address of PROFINET process image input register for device #4	I512.0 to I639.7			
CPU address of PROFINET process image input register for device #5	I640.0 to I767.7			

Description	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU SR60, CPU ST60
CPU address of PROFINET process image input register for device #6	I768.0 to I895.7			
CPU address of PROFINET process image input register for device #7	I896.0 to I1023.7			
CPU address of PROFINET process image input register for device #8	I1024.0 to I1151.7			
CPU address of PROFINET process image input register for I-Device	I1152.0 to I1279.7			
CPU address of PROFINET process image output register for device #1	Q128.0 to Q255.7			
CPU address of PROFINET process image output register for device #2	Q256.0 to Q383.7			
CPU address of PROFINET process image output register for device #3	Q384.0 to Q511.7			
CPU address of PROFINET process image output register for device #4	Q512.0 to Q639.7			
CPU address of PROFINET process image output register for device #5	Q640.0 to Q767.7			
CPU address of PROFINET process image output register for device #6	Q768.0 to Q895.7			
CPU address of PROFINET process image output register for device #7	Q896.0 to Q1023.7			
CPU address of PROFINET process image output register for device #8	Q1024.0 to Q1151.7			
CPU address of PROFINET process image output register for I-Device	Q1152.0 to Q1279.7			

Ordering information

F

F.1 CPU modules

CPU models	Article number
CPU SR20, AC/DC/Relay	6ES7288-1SR20-0AA0
CPU ST20, DC/DC/DC	6ES7288-1ST20-0AA0
CPU CR20s, AC/DC/Relay	6ES7288-1CR20-0AA1
CPU SR30, AC/DC/Relay	6ES7288-1SR30-0AA0
CPU ST30, DC/DC/DC	6ES7288-1ST30-0AA0
CPU CR30s, AC/DC/Relay	6ES7288-1CR30-0AA1
CPU ST40, DC/DC/DC	6ES7288-1ST40-0AA0
CPU SR40, AC/DC/Relay	6ES7288-1SR40-0AA0
CPU CR40s, AC/DC/Relay	6ES7288-1CR40-0AA1
CPU SR60, AC/DC/Relay	6ES7288-1SR60-0AA0
CPU ST60, DC/DC/DC	6ES7288-1ST60-0AA0
CPU CR60s, AC/DC/Relay	6ES7288-1CR60-0AA1
CPU CR40, AC/DC/Relay	6ES7288-1CR40-0AA0[1]
CPU CR60, AC/DC/Relay	6ES7288-1CR60-0AA0[1]

[1] For CPUs using firmware versions prior to V2.3, refer to the S7-200 SMART System Manual for your specific CPU model and version.

F.2 Expansion modules (EMs) and signal boards (SBs)

Expansion modules and signal boards	Article number
EM Digital 8 x Inputs (EM DE08)	6ES7288-2DE08-0AA0
EM Digital 16 x Inputs (EM DE16)	6ES7288-2DE16-0AA0
EM Digital 8 x Outputs Transistor (EM DT08)	6ES7288-2DT08-0AA0
EM Digital 8 x Outputs Relay (EM DR08)	6ES7288-2DR08-0AA0
EM Digital 16 x Outputs Relay (EM QR16)	6ES7288-2QR16-0AA0
EM Digital 16 x Outputs Transistor (EM QT16)	6ES7288-2QT16-0AA0
EM Digital 8 x Inputs / Digital 8 x Outputs Transistor (EM DT16)	6ES7288-2DT16-0AA0
EM Digital 8 x Inputs/ Relay 8 x Outputs (EM DR16)	6ES7288-2DR16-0AA0
EM Digital 16 x Inputs / Digital 16 x Outputs Transistor (EM DT32)	6ES7288-2DT32-0AA0
EM Digital 16 x Inputs / Relay 16 x Outputs (EM DR32)	6ES7288-2DR32-0AA0
EM Analog 4 x Inputs (EM AE04)	6ES7288-3AE04-0AA0

Expansion modules and signal boards	Article number
EM Analog 2 x Outputs (EM AQ02)	6ES7288-3AQ02-0AA0
EM Analog 4 x Outputs (EM AQ04)	6ES7288-3AQ04-0AA0
EM Analog 8 x Inputs (EM AE08)	6ES7288-3AE08-0AA0
EM Analog 2 x Inputs / Analog 1 x Outputs (EM AM03)	6ES7288-3AM03-0AA0
EM Analog 4 x Inputs / Analog 2 x Outputs (EM AM06)	6ES7288-3AM06-0AA0
EM RTD 2 x 16 bit (EM AR02)	6ES 288-3AR02-0AA0
EM RTD 4 x 16 bit (EM AR04)	6ES7288-3AR04-0AA0
EM TC 4 x 16 bit (EM AT04)	6ES7288-3AT04-0AA0
EM DP01 PROFIBUS DP SMART (EM DP01)	6ES7288-7PD01-0AA0
SB Digital 2 x Inputs / Digital 2 x Outputs (SB DT04)	6ES7288-5DT04-0AA0
SB Analog 1 x Output (SB AQ01)	6ES7288-5AQ01-0AA0
SB Analog 1 x Input (SB AE01)	6ES7288-5AE01-0AA0
SB RS485/RS232 (SB CM01)	6ES7288-5CM01-0AA0
SB Battery (SB BA01)	6ES7288-5BA01-0AA0

F.3 Programming software

Programming software	Article number
STEP 7-Micro/WIN SMART Individual License (CD-ROM)	6ES7288-8SW01-0AA0
Drives V90 (PC tools) software	Can be downloaded from the Siemens Services and Support website

F.4 Communication

Communications cards	Article number
CP 5411: Short AT ISA	6GK1541-1AA00
CP 5512: PCMCIA Type II	6GK1551-2AA00
CP 5611: PCI card (version 3.0 or greater)	6GK1561-1AA00

F.5 Spare parts and other hardware

Cables, network connectors, repeaters, and end retainers	Article number
I/O Expansion cable, 1 m	6ES7288-6EC01-0AA0
MPI Cable	6ES7901-0BF00-0AA0
RS-232/PPI Multi-Master cable	6ES7901-3CB30-0XA0
USB/PPI Multi-Master cable	6ES7901-3BD30-0XA0
PROFIBUS Network cable	6XV1830-0EH30
Network Bus Connector with Programming Port Connector, Vertical Cable Outlet	6ES7972-0BB12-0XA0
Network Bus Connector (no programming port connector), Vertical Cable Outlet	6ES7972-0BA12-0XA0
RS485 Bus Connector with 35° Cable Outlet (no programming port connector)	6ES7972-0BA42-0XA0
RS485 Bus Connector with 35° Cable Outlet (with programming port connector)	6ES7972-0BB42-0XA0
RS485 IP 20 Repeater, Isolated	6ES7972-0AA02-0XA0
TD/CPU Connecting cable	6ES7901-3EB10-0XA0
End Retainer Thermoplast, 10 MM	8WA1808
End Retainer, Steel	8WA1805

Table F- 1 Terminal block spare kits

If you have	Use this terminal block spare kit (4/pk)	
S7-200 SMART module (article number)	Terminal block article number	Terminal block description
CPU SR20, AC/DC/Relay (6ES7288-1SR20-0AA0)	6ES7292-1AH30-0XA0	8 pin, tin-plated
	6ES7292-1AH40-0XA0	8 pin, tin-plated, keyed
	6ES7292-1AM30-0XA0	12 pin, tin-plated
CPU ST20, DC/DC/DC (6ES7288-1ST20-0AA0)	6ES7292-1AH30-0XA0	8 pin, tin-plated
	6ES7292-1AM30-0XA0	12 pin, tin-plated
CPU SR30, AC/DC/Relay (6ES7288-1SR30-0AA0)	6ES7292-1AH30-0XA0	8 pin, tin-plated
	6ES7292-1AH40-0XA0	8 pin, tin-plated, keyed
	6ES7292-1AP30-0XA0	14 pin, tin-plated
	6ES7292-1AK30-0XA0	10 pin, tin-plated
CPU ST30, DC/DC/DC (6ES7288-1ST30-0AA0)	6ES7292-1AH30-0XA0	8 pin, tin-plated
	6ES7292-1AP30-0XA0	14 pin, tin-plated
	6ES7292-1AK30-0XA0	10 pin, tin-plated
CPU ST40, DC/DC/DC (6ES7288-1ST40-0AA0)	6ES7292-1AH30-0XA0	8 pin, tin-plated
	6ES7292-1AV30-0XA0	20 pin, tin-plated
	6ES7 292-1AL30-0XA0	11 pin, tin-plated
CPU SR40, AC/DC/Relay (6ES7288-1SR40-0AA0)	6ES7292-1AV30-0XA0	20 pin, tin-plated
	6ES7292-1AV40-0XA0	20 pin, tin-plated, keyed
	6ES7292-1AM30-0XA0	12 pin, tin-plated
CPU CR40, AC/DC/Relay (6ES7288-1CR40-0AA0)	6ES7292-1AH30-0XA0	8 pin, tin-plated
	6ES7292-1AV30-0XA0	20 pin, tin-plated
	6ES7292-1AL30-0XA0	11 pin, tin-plated

If you have S7-200 SMART module (article number)	Use this terminal block spare kit (4/pk)	
	Terminal block article number	Terminal block description
	6ES7292-1AL40-0XA0	11 pin, tin-plated, keyed
CPU ST60, DC/DC/DC (6ES7288-1ST60-0AA0)	6ES7292-1AV30-0XA0	20 pin, tin-plated
	6ES7292-1AM30-0XA0	12 pin, tin-plated
CPU SR60, AC/DC/Relay (6ES7288-1SR60-0AA0)	6ES7292-1AV30-0XA0	20 pin, tin-plated
	6ES7292-1AV40-0XA0	20 pin, tin-plated, keyed
	6ES7292-1AM30-0XA0	12 pin, tin-plated
CPU CR60, AC/DC/Relay (6ES7288-1CR60-0AA0)	6ES7292-1AV30-0XA0	20 pin, tin-plated
	6ES7292-1AV40-0XA0	20 pin, tin-plated, keyed
	6ES7292-1AM30-0XA0	12 pin, tin-plated 0
EM Digital 8 x Inputs (EM DE08) (6ES7288-2DE08-0AA0)	6ES7292-1AG30-0XA0	7 pin, tin-plated
EM Digital 8 x Outputs (EM DT08) (6ES7288-2DT08-0AA0)	6ES7292-1AG30-0XA0	7 pin, tin-plated
EM Digital 8 x Outputs Relay (EM DR08) (6ES7288-2DR08-0AA0)	6ES7292-1AG30-0XA0	7 pin, tin-plated
	6ES7292-1AG40-0XA0	7 pin, tin-plated, keyed-right
EM Digital 8 x Inputs / Digital 8 x Outputs (EM DT16) (6ES7288-2DT16-0AA0)	6ES7292-1AG30-0XA0	7 pin, tin-plated
EM Digital 8 x Inputs/ Relay 8 x Outputs (EM DR16) (6ES7288-2DR16-0AA0)	6ES7292-1AG30-0XA0	7 pin, tin-plated
	6ES7292-1AG40-0XA0	7 pin, tin-plated, keyed-right
EM Digital 16 x Inputs / Digital 16 x Outputs (EM DT32) (6ES7288-2DT32-0AA0)	6ES7292-1AL30-0XA0	11 pin, tin-plated
EM Digital 16 x Inputs / Relay 16 x Outputs (EM DR32) (6ES7288-2DR32-0AA0)	6ES7292-1AL30-0XA0	11 pin, tin-plated
	6ES7292-1AL40-0XA0	11 pin, tin-plated, keyed
EM Analog 4 x Inputs (EM AE04) (6ES7288-3AE04-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM Analog 8 x Inputs (EM AE08) (6ES7288-3AE08-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM Analog 2 x Outputs (EM AQ02) (6ES7288-3AQ02-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM Analog 4 x Outputs (EM AQ04) (6ES7288-3AQ04-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM Analog 2 x Inputs / Analog 1 x Outputs (EM AM03) (6ES7288-3AM03-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM Analog 4 x Inputs / Analog 2 x Outputs (EM AM06) (6ES7288-3AM06-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM RTD 2 x 16 bit (EM AR02) (6ES7288-3AR02-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM RTD 4 x 16 bit (EM AR04) (6ES7288-3AR04-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM TC 4 x 16 bit (EM AT04) (6ES7288-3AT04-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated
EM Profibus DP SMART (EM DP01) (6ES7288-7DP01-0AA0)	6ES7292-1BG30-0XA0	7 pin, gold-plated

F.6 Human Machine Interface devices

Human Machine Interface device	Article number
SMART LINE HMIs	
SMART LINE 700 IE	6AV6648-0CC11-3AX0
SMART LINE 1000 IE	6AV6648-0CE11-3AX0
Micro HMIs	
TD 400C TEXT DISPLAY, 4 LINES[1]	6AV6640-0AA00-0AX1
TD400C Blank faceplate material, A4 size (10 sheets/package)	6AV6671-0AP00-0AX0

[1] : Includes one blank faceplate overlay for customization. For additional blank faceplate overlays, order the blank faceplate material for your TD device.

Index

*****

*D (STL-Multiply double integer), 284
*I (STL-Multiply integer), 284
*R (STL-Multiply real), 284

.

.mwp files, 99
.smart files, 99

/

/D (STL-Divide double integer), 284
/I (STL-Divide integer), 284
/R (STL-Divide real), 284

+

+D (STL-Add double integer), 284
+I (STL-Add integer), 284
+R (STL-Add real), 284

<

<=B, 220
<=R, 220
<=W, 220
<>B, 220
<>R, 220
<>S, 224
<>W, 220
<B, 220
<R, 220
<W, 220

=

= (STL-output), 172
==B, 220
==R, 220
==S, 224
==W, 220
=I (STL-output immediate), 172

>

>=B, 220
>=R, 220
>=W, 220
>B, 220
>R, 220
>W, 220

A

A (STL-AND), 163
AB< (STL-AND compare byte less than), 220
AB<= (STL-AND compare byte less than or equal), 220
AB<> (STL-AND compare byte not equal), 220
AB= (STL-AND compare byte equal), 220
AB> (STL-AND compare byte greater than), 220
AB>= (STL-AND compare byte greater than or equal), 220
Absolute positioning mode, 605
AC
    isolation guidelines, 57
    wiring guidelines, 58
AC inductive loads, 60
Access rights
    CPU security, 138
    password privilege levels, 138
Active/Passive communication partners, 386
AD< (STL-AND compare double word less than), 220
AD<= (STL-AND compare double word less than or equal), 220
AD<> (STL-AND compare double word not equal), 220
AD= (STL-AND compare double word equal), 220
AD> (STL-AND compare double word greater than), 220
AD>= (STL-AND compare double word greater than or equal), 220
ADD_DI, 284
ADD_I, 284
ADD_R, 284
Addressing
    accumulators, 71
    analog inputs, 73
    analog outputs, 74
    counter memory, 70
    creating pointers and using indirect address, 76

example of pointer offset to access data, 80
example of pointer to access data in a table, 79
flag memory, 69
high-speed counters, 70
local and expansion I/O, 76
local memory, 72
memory areas, 68
process image output register, 68
sequence control relay (SCR) memory, 74
special memory (SM) bits, 72
symbol table, 109
timer memory, 69
variable memory, 69
AENO (STL stack-AND ENO bit)
instruction, 168
logic stack overview, 167
AI (STL-AND immediate), 165
Air flow, 44
Alarms
analog input configuration, 145
from system (SMW100-SMW114), 886
ALD (STL stack-AND Load), 168
AN (STL-AND NOT), 163
Analog I/O
input representation (current), 823
input representation (voltage), 823
output representation (current), 824
output representation (voltage), 824
status indicators, 93
step response times (SM), 822
Analog inputs
analog type configuration, 143
rejection, 143
smoothing, 143
system block configuration, 143
Analog outputs
analog type configuration, 146
states at RUN/ STOP transition, 146
ANDB (STL-AND byte), 320
ANDD (STL-AND dword), 320
ANDW (STL-AND word), 320
ANI (STL-AND NOT immediate), 165
AR< (STL-AND compare real less than), 220
AR<= (STL-AND compare real less than or equal), 220
AR<> (STL-AND compare real not equal), 220
AR= (STL-AND compare real equal), 220
AR> (STL-AND compare real greater than), 220
AR>= (STL-AND compare real greater than or equal), 220
Article numbers, 909, 909, 910, 913
AS<> (STL-AND string compare not equal), 224
AS= (STL-AND string compare equal), 224
ASCII array conversion instructions, 229
Assigning
global symbols, 109
local variables, 114
variables (local), 112
ATCH, 307
ATH, 229
ATT, 350
Auto-tuning PID, 661
AW< (STL-AND compare word less than), 220
AW<= (STL-AND compare word less than or equal), 220
AW<> (STL-AND compare word not equal), 220
AW= (STL-AND compare word equal), 220
AW> (STL-AND compare word greater than), 220
AW>= (STL-AND compare word greater than or equal), 220
Axis 0 motion control (SMB600-SMB649), 888
Axis of Motion
ACCEL_TIME, 677
AXISx_ABSPOS, 709
AXISx_CACHE, 707
AXISx_CFG, 706
AXISx_CTRL, 697
AXISx_DIS, 706
AXISx_GOTO, 700
AXISx_LDOFF, 703
AXISx_LDPOS, 704
AXISx_MAN, 698
AXISx_RDPOS, 708
AXISx_RSEEK, 702
AXISx_RUN, 701
AXISx_SRATE, 705
configuring, 683
configuring reference point and seek parameters, 692
configuring the Backlash compensation, 691
configuring the input pin locations, 684
defining the motion profile, 694
displaying and controlling the operation of axes, 724
displaying and modifying the configuration of axes, 730
displaying the profile configuration for the axes, 730
eliminating backlash, 750
entering acceleration time, 690
entering jerk time, 691
entering jog parameters, 690
entering maximum start and stop speed, 689
error codes, 731
mapping the I/O, 685
Motion control panel, 724
phasing, 687

polarity, 688
programming, 682
RP Seek modes, 745
SM locations, 742
subroutine guidelines, 696
subroutines, 695

B

B_I, 226
BA01 battery signal board, 156
Baud rate
communications, 131
setting, 466
switch selections:RS232/PPI Multi-Master cable, 484
BCD_I, 226
BCDI (STL-BCD to integer), 226
BGN_ITIME, 367
Biasing and terminating
CM01 signal board, 479
network cable, 478
Biasing PID loop, 294
BIR (STL-byte immediate read), 325
Bit logic instructions
AENO (STL-AND ENO), 167
contacts, 163
contacts (Immediate), 165
edge detectors, 171
input examples, 176
NOP (No operation), 175
NOT, 170
output coils, 172
output examples, 177
set and reset bits, 173
set and reset dominant bistable, 174
STL logic stack instructions for inputs, 167
STL logic stack operations, 168
BITIM (STL-Begin interval timer), 367
BIW (STL-byte immediate write), 325
BLKMOV_B, 323
BLKMOV_D, 323
BLKMOV_W, 323
BMB (STL-block move byte), 323
BMD (STL-block move double word), 323
BMW (STL- block move word), 323
Bookmarks in programs, 647
BTI (STL-Byte to integer), 226
Buffer consistency
PROFIBUS, 447
Building status charts, 652
Building your communication network, 475
Byte consistency
PROFIBUS, 447

C

Cable
expansion, 849
Cables
USB/PPI Multi-Master, 850
CAL_ITIME, 367
CALL, 368
Card (memory), 88, 89
CE approval, 751
CFND (STL-Find character), 347
Character error received from Freeport (SMB3), 870
Charts
building, 652
creating, 652
opening, 652
CHR_FIND, 347
CITIM (STL-Calculate interval time), 367
Clearance for airflow and cooling, 44
Cleared memory card, 160
Clearing PLC memory, 158
Clock instructions
READ_RTC, 178
READ_RTCX, 181
SET_RTC, 178
SET_RTCX, 181
CLR_EVNT, 307
CM01 signal board
biasing and terminating, 479
connector pin assignments, 478
Coils
output, 172
output (immediate), 172
reset bits, 173
reset bits immediate, 173
set bits, 173
set bits immediate, 173
Cold junction compensation
thermocouple, 155
Thermocouple, 828
Communication drivers, 384
Communication ports, 383
connector pin assignments, 477
Freeport mode, 481
system block configuration, 131
Communications
article numbers for modules, 910
choices, 464
dialog, 391, 470

Ethernet hardware connection, 29
Ethernet, RS485, and RS232, 25, 381
IP address, 391
locating MAC address on CPU, 398
network, 28
number of connections (Ethernet), 382
number of connections (RS232), 382
number of connections (RS485), 382
point-to-point interface (PPI) protocol, 465
restricting writes, 138
RS485 configuration, 464
RS485 hardware connection, 32
RS485 network address, 470
Communications instructions
receive, 191
transmit, 191
Communications protocol
PROFIBUS, 442
Compare instructions
compare character strings, 224
compare number value, 220
Compatibility
EM DP01 PROFIBUS DP module, 847
Compiling programs
downloading, 80
PLC non-fatal program errors, 860
Configuration
CPU, system block, 129
dynamic IP information, 391
dynamic PROFINET device name, 411
EM DP01 PROFIBUS DP, 446
Ethernet, 391
IP address, 391
MAC address, 398
profile table (Axis of Motion), 734
RS485 network, 470
RS485 network address, 470
static IP information, 393
Configuration drawings, 95
Connections
number of connections (Ethernet), 382
number of connections (RS232), 382
number of connections (RS485), 382
types of communication, 25, 381
Connector, 52
Contact information, 3
Contacts
negative edge detector, 171
normally closed, 163
normally closed (immediate), 165
normally open, 163
normally open (immediate), 165
NOT, 170
positive edge detector, 171
Contamination level, 754
Convert instructions
ASCII array conversions, 229
ASCII sub-string to number value, 239
encode and decode, 242
number value to string, 235
standard conversion, 226
Cooling, 44
COS (cosine), 290
Counter instructions
high-speed counters, 248
standard counters, 245
CPU
accessing data, 67
clearing memory, 158
configuring communication to HMI, 400
connecting power, 28
CPU CR20s specifications, 756
CR20s specifications, 756
CR30s specifications, 767
CR30s wiring diagram, 777
CR40 specifications, 778
CR40 wiring diagram, 786
CR40s specifications, 778
CR40s wiring diagram, 788
CR60 specifications, 789
CR60 wiring diagram, 797
CR60s specifications, 789
CR60s wiring diagram, 799
dimensions, 19, 46
DIN rail, 49
Ethernet communication, 386
Ethernet port, 391
expansion cable, 55
expansion modules supported, 23
fatal errors, 863
features, 19
installation, 47
installation on a panel, 48
IP address, 391
isolation guidelines, 57
LED indicators, 92
LEDs, 19
MAC address, 398
memory card, 88
non-fatal error memory locations, 862
number of communication connections, 382
number of PPI connections, 465
process image register, 65
RS485 network address, 470

RS485 network port, 470
setting the type, 39
SR20 specifications, 756
SR20 wiring diagram, 765
SR30 specifications, 767
SR40 specifications, 778
SR40 wiring diagram, 786
SR60 specifications, 789
SR60 wiring diagram, 797
ST20 wiring diagram, 765
ST30 specifications, 767
ST30 wiring diagram, 775
ST40 specifications, 778
ST40 wiring diagram, 786
ST60 specifications, 789
ST60 wiring diagram, 797
system block, 129
types of communication, 25, 381
wiring guidelines, 58
CPU
SR30 wiring diagram, 775
CPU hardware/firmware ID (SMB1000-SMB1049), 889
CPU ID register (SMB6-SMB7), 871
CPU role
Controller, 415
Controller and I-Device, 425
I-Device, 422
CRET (STL-Conditional return from subroutine), 368
Cross reference, 647
CTD (count down), 245
CTU (count up), 245
CTUD (count up/down), 245
Customer support, 3

D

-D (STL-Subtract double integer), 284
Data
receiving, 194
retention, 137
Data block (DB), 97
Data consistency
PROFIBUS, 447
Data log
status (SMB480-SMB515), 887
DC
isolation guidelines, 57
wiring guidelines, 58
DC inductive loads, 60
Debugging and monitoring
forcing values, 654
program editor status, 649
DEC_B, 292
DEC_DW, 292
DEC_W, 292
DECB (STL-Decrement byte), 292
DECD (STL-Decrement double word), 292
DECO, 242
DECW (STL-Decrement word), 292
Default gateway IP address, 390
Defining
local variables, 114
Deleting GSDML files, 410
Device configuration of CPU and modules, 129
DI_I, 226
DI_R, 226
DI_S, 235
Diagnostics
LED indicators, 92
Differential term, PID algorithm, 302
Digital input filter time, 134
Digital input filters, 134
Digital inputs, 134
Digital outputs, 136
Dimensions
CPU, 19
mounting, 46
DIN rail, 47, 49
DISI, 307
Displaying status, 649
DIV, 288
DIV_DI, 284
DIV_I, 284
DIV_R, 284
Downloading
programs, 80
sample program, 40
DP device
EM DP01 PROFIBUS DP, 444
Drive communication
calculating time requirement, 576
Drives, 681, 709, 910
DTA, 229
DTCH, 307
DTI (STL-Double integer to integer), 226
DTR (STL-Double integer to real), 226
DTS (STL-Double integer to string), 235
Dynamic IP information, 391

E

ED (STL-Edge Down), 171
Edge detectors, 171
Electromagnetic compatibility (EMC), 752

Element usage, 647
EM (expansion module) hardware/firmware ID (SMB1100-SMB1299), 890
EM DE16 specifications, 801
EM DP01 PROFIBUS DP module
  additional configuration features, 457
  configuration options, 448
  data exchange mode, 454
  DP protocol, 444
  GSD device database file, 458
  on PROFIBUS network, 445
  status LEDs, 848
  wiring diagram, 848
ENCO, 242
END, 337
ENI, 307
Environmental
  industrial environments, 752
  operating conditions, 753
  transport and storage conditions, 753
Errors
  Axis of Motion, 731
  character error received from Freeport communication (SMB3), 870
  compile and run-time errors (PLC program), 860
  data retention, 137
  fatal (PLC), 863
  fatal error effect on run-time execution, 124
  GET_TABLE and PUT_TABLE instructions, 185
  I/O error status, 871
  I/O module ID and error registers (SMB8-SMB19), 872
  memory locations (PLC non-fatal errors), 862
  Modbus RTU master execution, 501
  Modbus RTU slave execution, 506
  Modbus TCP client execution, 516
  Modbus TCP general exception codes, 528
  Modbus TCP server execution, 519
  Motion instruction, 732
  non-fatal error effect on run-time execution, 123
  PID auto-tune, 668
  PLC error handling, 117
  PWMx_RUN subroutine, 676
  signal board ID and error registers (SMB28-SMB29), 873
  timestamp mismatch (PC/PLC program difference), 859
Errors
  Modbus TCP general communication exception codes, 528
Ethernet
  configuring communication between CPU and HMI device, 400
  GET, 184
  IP address, 390
  ISO-on-TCP protocol, 401
  MAC address, 398
  networks, 385
  number of communication connections, 382
  Ports, 402
  TCP protocol, 401
  TSAPs, 403
  types of communication, 25, 381
  UDP protocol, 401
Ethernet communications
  STEP 7-Micro/WIN SMART settings, 30
Ethernet network
  configuring the IP address for a CPU, 391
  searching for CPU, 397
EU STL-Edge Up), 171
Events, interrupts, 309
Example of creating a PROFINET network, 426
Examples
  Axis of Motion AXISx_CTRL, AXISx_RUN, AXISx_SEEK, and AXISx_MAN subroutines application, 719
  Axis of Motion simple relative move (cut-to-length application), 717
  bit logic input, 176
  bit logic output, 177
  configuring the PROFIBUS DP EM DP01 I/O, 450
  conversion instructions, 228
  count up/down counter instruction, 247
  DISCONNECT instruction, 551
  GET and PUT instructions, 189
  high-speed counter initialization sequences, 265
  high-speed counter programming, 254
  installing the PROFIBUS DP EM DP01 GSD file, 448
  ISO_CONNECT instruction, 537
  Modbus RTU slave protocol, programming, 506
  Open user communication (OUC) library, 554, 562, 564
  PROFIBUS DP communications to a CPU, 462
  Shift register bit (SHRB) instruction, 343
  subroutine for sampling the value of an analog input, 98
  Table Find (TBL_FIND) instruction, 356
  tables, 356
  TCP_CONNECT instruction, 534
  TCP_RECV instruction, 545
  UDP_CONNECT instruction, 539

UDP_RECV instruction, 550
UDP_SEND instruction, 547
Using the AXISx_ABSPOS subroutine to read the absolute position from a SINAMICS V90 servo drive, 711
Executing
program, 65
single or multiple scans, 656
EXP (natural exponential), 290
Expansion and local I/O addressing, 76
Expansion cable, 849
installation, 55
removal, 55
Expansion I/O bus - communication errors (SMW98), 885
Expansion module
EM QR16, 803
Expansion module (EM)
EM AE04 specifications, 813
EM AE04 wiring diagram, 815
EM AM06 specifications, 818
EM AM06 wiring diagram, 821
EM AQ02 specifications, 816
EM AQ02, wiring diagram, 817
EM AR02 (RTD) specifications, 830
EM AR02 (RTD) wiring diagram, 835
EM AT04 specifications, 825
EM DE08 specifications, 801
EM DE08 wiring diagram, 802
EM DR08 specifications, 803
EM DR08 wiring diagram, 805
EM DR16 specifications, 807
EM DR16 wiring diagram, 810
EM DR32 specifications, 807
EM DR32 wiring diagram, 811
EM DT08 specifications, 803
EM DT08 wiring diagram, 805
EM DT16 specifications, 807
EM DT16 wiring diagram, 810
EM DT32 specifications, 807
EM DT32 wiring diagram, 811
installing and removing, 54
wiring diagram, 825
Expansion module (SB)
SB AE01 specifications, 838
Expansion module EM)
EM Q04, wiring diagram, 817
expansion modules
EM DE08, 801
Expansion modules, 23, 871
dimensions, 46
module error status (SMB5), 871
module ID and error registers (SMB8-SMB19), 872
Expansion modules (EM)
analog input representation (current), 823
analog input representation (voltage), 823
analog output representation (current), 824
analog output representation (voltage), 824

F

Factory defaults memory card, 160
Fatal error effect on run-time execution, 124
Fatal errors (PLC), 863
FBD editor, 105
Features
CPU, 19
expansion modules supported, 23
Features of the new version, 22
FIFO, 351
File menu
Download, 80
GSDML Management, 408
Upload, 83
FILL (STL-table fill), 353
FILL_N, 353
Filter time, 134
Filters, digital input configuration, 134
Find PROFINET Devices, 411
Firmware update, 86
First scan flag (SMB0), 867
First scan, executing, 656
Flag memory, 69
Floating point values, 303
FND=, <>, <, > (STL-table find), 354
FOR, 326
Force
writing and forcing outputs in STOP mode, 655
Forced value indicator (SMB4), 870
Forgotten password, 138, 159
Freeport mode
character interrupt control, 202
enabling, 192
example, 482
Freeport character error (SMB3), 870
Freeport configuration (SMB30-Port 0 and SMB130-Port 1), 874
Freeport receive character (SMB2), 869
Freeport transmitter idle (SMB4), 870
interrupts, 312
SMB86-SMB94 and SMB186-SMB194 receive message control, 883

Freeze outputs
analog output configuration, 146
digital output configuration, 136

G

General technical specifications, 751
GERR (STL-Get non-fatal error code), 338
GET, 184
GET_ADDR, 204
GET_ERROR, 338
GIP_ADDR, 205
Global symbols, 109
Grounding, 58
GSD file
EM DP01 PROFIBUS DP module, 458
GSDML, 407
Guidelines
grounding and circuit, 56
installation on a panel, 48
installation procedures, 47
isolation, 57
wiring guidelines, 58

H

Hardware troubleshooting, 658
HDEF (high-speed counter definition), 248
Heat, high voltage, and electrical noise, 43
High-speed counter registers, 875
High-speed counters, 254
High-vibration environment, 49
HMI
configuring Ethernet communication, 400
devices, 480
general guidelines for networks, 480
multi-master and multi-slave PPI networks, 469
single-master PPI networks, 468
supported devices, 24, 384
Hotline, 3
HSC (high-speed counter), 161, 248
HSC0, HSC1, HSC2, HSC3, HSC4, and HSC5 high-speed counter registers (SMB36-65, SMB136-145, SMB146-155, SMB156-165), 875
HTA, 229

I

-I (STL-Subtract integer), 284
I/O
analog input representation (current), 823
analog input representation (voltage), 823
analog output representation (current), 824
analog output representation (voltage), 824
analog status indicators, 93
step response times (SM), 822
I/O addressing, 76
I/O error status
PLC non-fatal error codes, 860
PLC non-fatal error SM flags, 862
SMB5, 871
I/O expansion bus - communication errors (SMW98), 885
I/O Module ID and error registers (SMB8-SMB19), 872
I_B, 226
I_BCD, 226
I_DI, 226
I_S, 235
IBCD (STL-Integer to BCD), 226
I-Device, 405
I-Device with lower-level PROFINET IO system, 406
I-Device without lower-level PROFINET IO system, 406
I-Device with lower-level PROFINET IO system, 425
I-Device without lower-level PROFINET IO system, 422
Illegal syntax
symbol table, 109
Immediate I/O read/writes, 65
Immediate instructions
LAD, FBD, and STL, 165
Importing GSDML files, 408
INC_B, 292
INC_DW, 292
INC_W, 292
INCB (STL-Increment byte), 292
INCD (STL-Increment double word), 292
Incremental jog mode, 624
INCW (STL-Increment word), 292
Indirect addressing
creating pointers and using indirect address, 76
example of pointer offset to access data, 80
example of pointer to access data in a table, 79
symbol table, 111
Inductive loads, 60
Input filter time, 134
Input process image register, 64
Inputs
edge detectors, 171
example bit logic, 176
NOT contact, 170
physical and in program, 63
pulse catch bits (system block), 134

reading, 64
standard contacts, 163, 165
STL logic stack, 167
Installation
clearance for airflow and cooling, 44
dimensions, 46
DIN rail, 49
expansion cable, 55
expansion module (EM), 54
grounding, 58
grounding and circuit, 56
guidelines, 43
high-vibration environment, 49
inductive loads, 60
isolation, 57
isolation guidelines, 57
lamp loads, 60
overview, 43, 47
panel, 48
separate the devices from heat, high voltage, and electrical noise, 43
signal board (SB), 51
terminal block connector, 52
wiring guidelines, 58
Instruction execution status bits (SMB1), 868
Instruction libraries, 644
Instructions
GET, 184
GET_ADDR, 204
GIP_ADDR, 205
loop control (PID), 293
MBUS_CLIENT, 513
MBUS_SERVER, 517
PUT, 184
quick reference guide, 898
SET_ADDR, 204
SIP_ADDR, 205
Insulation, 754
Integral term, PID algorithm, 301
Interrupt routines, 65
elements of a user program, 97
Interrupts
attach/detach, enable/disable, conditional return, and clear event instructions, 307
event support by CPU model, 309
example programs, 313
global interrupt enable state (SMB4), 870
interrupt queue overflow (SMB4), 870
overview, 309
priority and queuing, 313
programming guidelines, 310
time interval values for timed interrupts (SMB34-SMB35), 874
types of interrupt events, 312
INV_B, 319
INV_DW, 319
INV_W, 319
INVB (STL-invert byte), 319
INVD (STL-invert double word), 319
INVW (STL-invert word), 319
IP address, 390
assigning, 387, 396
configuring, 391
MAC address, 398
IP router, 391
Isolation, 57
Isolation guidelines, 57
ITA, 229
ITB (STL-Integer to byte), 226
ITD (STL-Integer to double integer), 226
ITS (STL-Integer to string), 235

J

JMP, 327
Jog mode, 621

L

L memory, 112
LAD editor, 104
Lamp loads, 60
LBL, 327
LD (STL stack-Load NOT immediate), 167
LD (STL stack-Load), 167
LD (STL-Load), 163
LDB< (STL-Load compare byte less than), 220
LDB<= (STL-Load compare byte less than or equal), 220
LDB<> (STL-Load compare byte not equal), 220
LDB= (STL-Load compare byte equal), 220
LDB> (STL-Load compare byte greater than), 220
LDB>= (STL-Load compare byte greater than or equal), 220
LDD< (STL-Load compare double word less than), 220
LDD<= (STL-Load compare double word less than or equal), 220
LDD<> (STL-Load compare double word not equal), 220
LDD= (STL-Load compare double word equal), 220
LDD> (STL-Load compare double word greater than), 220

LDD>= (STL-Load compare double word greater than or equal), 220
LDI (STL stack-Load immediate), 167
LDI (STL-Load NOT immediate), 165
LDN (STL stack-Load NOT), 167
LDN (STL-Load NOT), 163
LDNI (STL-Load NOT immediate), 165
LDR< (STL-Load compare real less than), 220
LDR<= (STL-Load compare real less than or equal), 220
LDR<> (STL-Load compare real not equal), 220
LDR= (STL-Load compare real equal), 220
LDR> (STL-Load compare real greater than), 220
LDR>= (STL-Load compare real greater than or equal), 220
LDS (STL stack-Load), 168
LDS<> (STL-Load string compare not equal), 224
LDS= (STL-Load string compare equal), 224
LDW< (STL-Load compare word less than), 220
LDW<= (STL-Load compare word less than or equal), 220
LDW<> (STL-Load compare word not equal), 220
LDW= (STL-Load compare word equal), 220
LDW> (STL-Load compare word greater than), 220
LDW>= (STL-Load compare word greater than or equal), 220
LED indicators
  CPU status, 92
Library
  creating, 644
  types, 487
LIFO, 351
LN (natural logarithm), 290
Local and expansion I/O addressing, 76
Local variables, 112
Local/Partner connection, 386
Logic stack
  STL inputs, 167
  STL stack operations, 168
Logic, control, 63
Logical operation instructions
  AND, OR, XOR (byte, word, and dword), 320
  invert, 319
Loop control (PID)
  adjusting bias, 305
  converting inputs, 303
  converting outputs, 304
  error conditions, 306
  forward/reverse, 304
  loop definition table, 662
  modes, 305
Loop table, 306
Lost password, 159
LPP (STL stack-Logic Pop), 168
LPS (STL stack-Logic Push), 168
LRD (STL stack-Logic Read), 168
LSCR (STL-Load SCR), 329

M

MAC address, 398
Main entry, 802
Main program, 96
Math instructions
  add, subtract, multiply and divide, 284
  divide integer with remainder, 288
  increment and decrement, 292
  multiply integers to double integer and divide integer with remainder, 288
  numeric functions, 290
MBUS_CLIENT, 513
MBUS_CTRL (initialize Modbus master communication), 496
MBUS_INIT (initialize slave communication), 504
MBUS_MSG / MB_MSG2 (send message from Modbus master), 498
MBUS_SERVER, 517
MBUS_SLAVE (slave response to master message), 505
Memory, 862
  addresses for non-fatal error indicators, 862
  clearing PLC, 158
  retentive range configuration, 137
Memory card
  program transfer card, 88, 89
  reset to factory defaults, 160
  types of, 84
  using, 85
Modbus general
  addressing, 490
  advanced user information, 509, 525
  initialization and execution time for Modbus protocol, 494
  library features, 492
  requirements for using Modbus instructions, 493
Modbus RTU library overview, 487
Modbus RTU master
  example program, 507
  execution error codes, 501
  MBUS_CTRL (initialize master communication), 496
  MBUS_MSG / MB_MSG2 (send message from master), 498
  using the instructions, 495

Modbus RTU slave
execution error codes, 506
MBUS_INIT (initialize slave communication), 504
MBUS_SLAVE (slave response to master message), 505
using the instructions, 502
Modbus TCP
general exception codes, 528, 528
MODBUS TCP
MBUS_CLIENT, 513
MBUS_SERVER, 517
Modbus TCP client
execution error codes, 516
Modbus TCP library
example, 520
Modbus TCP library features, 511
Modbus TCP library overview, 487
Modbus TCP server
execution error codes, 519
Modules
CPU CR20s, 756
CPU CR30s, 767
CPU CR40, 778
CPU CR40s, 778
CPU CR60, 789
CPU CR60s, 789
CPU SR20, 756
CPU SR30, 767
CPU SR40, 778
CPU SR60, 789
CPU ST20, 756
CPU ST30, 767
CPU ST40, 778
CPU ST60, 789
dimensions, 46
EM AE04, 813
EM AM06, 818
EM AQ02, 816
EM AR02 (RTD), 830
EM DE08, 801
EM DE16 specifications, 801
EM DP01 PROFIBUS DP, 845
EM DR08, 803
EM DR16, 807
EM DR32, 807
EM DT08, 803
EM DT16, 807
EM DT32, 807
EM QR16, 803
EM QT16, 803
SB AE01, 838
SB AQ01, 840, 841
SB CM01, 842
SB DT04, 836, 837
Motion control
configuring reference point and seek parameters, 692
configuring the Backlash compensation, 691
configuring the input pin locations, 684
defining the motion profile, 694
entering acceleration time, 690
entering jerk time, 691
entering jog parameters, 690
entering maximum start and stop speed, 689
mapping the I/O, 685
Motion features, 680
phasing, 687
polarity, 688
Motion control panel, 723
displaying and controlling the operation of axes, 724
displaying and modifying the configuration of axes, 730
displaying the profile configuration for the axes, 730
Motion inputs and outputs
CPU, 681
Motion instruction, error codes, 732
Motion profile
configuring, 678
creating steps, 680
defining, 678
mode of operation, 679
Motion wizard
configuration/profile table, 734
maximum and start/stop speeds, 676
open loop motion control, 673
SM locations, 742
Mounting
DIN rail, 49
expansion cable, 55
isolation, 57
overview, 47
panel, 48
wiring guidelines, 58
MOV_B, 322
MOV_BIR, 325
MOV_BIW, 325
MOV_DW, 322
MOV_R, 322
MOV_W, 322
MOVB (STL-move byte), 322
MOVD (STL-move double word), 322
Move instructions
block move (byte, word, dword), 323
move (byte, word, dword, real), 322

move byte immediate (read and write), 325
SWAP (exchange byte data in a word), 324
MOVR (STL-move real), 322
MOVW (STL-move word), 322
MUL, 288
MUL_DI, 284
MUL_I, 284
MUL_R, 284
Multiple scans, 656

N

Network communications, 28
Networks (communication)
addresses, 466
biasing and terminating the network cable, 478
biasing cable, 479
calculating network distances, 475
general guidelines for building, 475
network configurations, 464
safety concerns, 475
sample RS485 network configurations, 468
selecting the network cable, 477
single-master PPI, 469
types of communication, 25, 381
New features, 22
NEXT, 326
Non-fatal error effect on run-time execution, 123
Non-fatal PLC errors
compile and run-time, 860
Special Memory locations, 862
Non-volatile memory, 84
NOP, 175
Normally closed contact
immediate, 165
standard, 163
Normally open contact
immediate, 165
standard, 163
NOT (STL), 170
Number value to string conversion instructions, 235

O

O (STL-OR), 163
OB< (STL-OR compare byte less than), 220
OB<= (STL-OR compare byte less than or equal), 220
OB<> (STL-OR compare byte not equal), 220
OB= (STL-OR compare byte equal), 220
OB> (STL-OR compare byte greater than), 220
OB>= (STL-OR compare byte greater than or equal), 220
OB1, 96
OD< (STL-OR compare double word less than), 220
OD<= (STL-OR compare double word less than or equal), 220
OD<> (STL-OR compare double word not equal), 220
OD= (STL-OR compare double word equal), 220
OD> (STL-OR compare double word greater than), 220
OD>= (STL-OR compare double word greater than or equal), 220
OI (STL-OR immediate), 165
OLD (STL stack-OR Load), 168
ON (STL-OR NOT), 163
ONI (STL-OR NOT immediate), 165
Open loop control, 673
Open user communication (OUC)
connection instructions, 402
connection types, 402
Open user communication (OUC) library
common library instruction parameters, 530
DISCONNECT instruction, 550
instruction error codes, 552
ISO_CONNECT instruction, 535
overview, 487
TCP_CONNECT instruction, 532
TCP_RECV instruction, 542
TCP_SEND instruction, 540
UDP_CONNECT instruction, 538
UDP_RECV instruction, 548
UDP_SEND instruction, 545
Opening earlier STEP 7-Micro/WIN projects, 99
Operating mode
changing to RUN, 41, 92
changing to STOP, 41, 92
startup options, 142
Operator stations, 95
Options
STL status, 652
OR< (STL-OR compare real less than), 220
OR<= (STL-OR compare real less than or equal), 220
OR<> (STL-OR compare real not equal), 220
OR= (STL-OR compare real equal), 220
OR> (STL-OR compare real greater than), 220
OR>= (STL-OR compare real greater than or equal), 220
ORB (STL-OR byte), 320
ORD (STL-OR dword), 320
Ordering information, 909
ORW (STL-OR word), 320
OS<> (STL-OR string compare not equal), 224
OS= (STL-OR string compare equal), 224

OUC library example
Active partner (client) program, 554
CheckErrors subroutine program, 562
Passive partner (server) program, 564
OUC library example Active partner (client) program
symbol table, 563
OUC library example Passive partner (server) program
symbol table, 570
Output image register, 64
Outputs
coils, 172
example bit logic, 177
physical and in program, 63
set and reset bits, 173
set and reset dominant bistable, 174
writing, 65
Overvoltage, 754
OW< (STL-OR compare word less than), 220
OW<= (STL-OR compare word less than or equal), 220
OW<> (STL-OR compare word not equal), 220
OW= (STL-OR compare word equal), 220
OW> (STL-OR compare word greater than), 220
OW>= (STL-OR compare word greater than or equal), 220

P

Password
lost or forgotten, 159
privilege levels, 138
Password protection, 138
PID auto-tune
auto-deviation, 666
auto-hysteresis, 666
exception conditions, 668
prerequisites, 665
PV out-of-range, 669
sequence, 666
PID loop control
loop definition table, 662
PID Tune control panel, 669
PID loop instruction
alarm checking, 306
loop control types, 302
understanding, 300
PID Tune control panel, 669
Pin assignments for network connector, 477
Pipelining
PTO pulses, 276
PLC
clearing memory, 158
compile and run-time errors, 860
expansion cable, 55
fatal errors, 863
information (hardware/firmware, error status, run/stop event log), 117
installation, 47
installation on a panel, 48
memory card, 88
non-fatal error memory locations, 862
system block, 129
PLC menu
Download, 80
Upload, 83
PLS
instruction, 273
Special Memory to monitor and control PTO and PWM outputs, 880
PN Read Write library
PN_RD_REC instruction, 572
PN_WR_REC instruction, 572
PN Read Write Record library
overview, 487
Pointer
creating pointers and using indirect address, 76
example of pointer offset to access data, 80
example of pointer to access data in a table, 79
Power interruption (PLC), 137
Power requirements
calculating, 858
CPU, 45, 855
sample, 857
Power supply, 45, 855
PPI communication
changing to Freeport mode, 193
multi-master and multi-slave PPI networks, 469
port configuration in system block, 131
single-master networks, 468
Prerequisite
Using SINA_PARA_S instruction, 635
Using SINA_POS instruction, 592
Using SINA_SPEED instruction, 628
Previous STEP 7-Micro/WIN projects, 99
PROFIBUS
DP device, 442
S7-200 SMART EM DP01 PROFIBUS DP module, 443
Profile table values
PTO generators, 281
PROFINET
device naming and addressing, 410
PROFINET device status
SMB1800-SMB1999, 893

PROFINET instruction
BLKMOV_BIR, 378
BLKMOV_BIW, 378
BMIR, 378
BMIW, 378
RDREC, 374
WRREC, 374
Program block, 96
Program control instructions
END, STOP, and WDR, 337
FOR-NEXT loop, 326
GET_ERROR, 338
JMP-LBL, 327
SCR (Sequence control relay), 329
Program editor
bookmarks, 647
debugging and monitoring, 649
STL status options, 652
types, 103
using, 36
Program instructions
bit
logic, 163, 165, 167, 168, 170, 171, 172, 173, 174, 175, 176, 177
clock, 178, 181
compare, 220, 224
convert, 226, 229, 235, 239, 242
counters, 245, 248, 254, 265
interrupts, 307
logical operations, 319, 320
math, 284, 288, 290, 292
move, 322, 323, 324, 325
program control, 326, 327, 329, 337, 338
shift and rotate, 339, 342
string, 346, 347
subroutine, 368, 369
table, 350, 351, 353
table find, 354
timer, 358, 367
Program status
building a status chart, 652
displaying in program editor, 649
executing a limited number of scans, 656
Program transfer card, 85
Programs
elements, 96
executing limited scans, 656
interrupt routines, 97
memory card, 88, 89
status charts, 652
subroutines, 96
Projects
opening previous STEP 7-Micro/WIN projects, 99
Proportional term, PID algorithm, 301
Protection class, 754
Protocols
PROFIBUS DP, 444
PTO0, PWM0, PTO1, PWM1, PTO2, and PWM2 high-speed outputs (SMB66-SMB85, SMB166-SMB169, SMB176-SMB179, and SMB566-SMB579), 880
Pulse catch bits, digital input configuration in system block, 134
Pulse train output (PTO)
cycle time, 275
instruction, 161
pulse output instruction (PLS), 273
Pulse width modulation (PWM)
cycle time, 277
instruction, 161
output, 674, 674
pulse output instruction (PLS), 273
pulse outputs, 675
PUT, 184
PWM wizard
PWMx_RUN subroutine, 675
PWMx_RUN, 675

Q

Queue interrupt overflow (SMB4), 870
quick access toolbar, 26

R

R (STL-Reset), 173
-R (STL-Subtract real), 284
R_S, 235
Rated voltages, 754
RCV (receive message control SMB86-SMB94 and SMB186-SMB194), 883
READ_RTC, 178
READ_RTCX, 181
Real number values, 74
Receive instruction
break detection, 198
end character detection, 199
end conditions, 196
idle line detection, 196
intercharacter timer, 199
maximum character count, 201
message timer, 201
parity errors, 201

start character detection, 197
user termination, 202
Referencing (active referencing) mode, 613
Referencing (set reference point) mode), 616
Relative positioning mode, 602
Relay electrical service life, 756
Repeaters, 476
Reset-to-factory-defaults memory card, 160
Restoring data after power-on, 91
RET, 368
Retentive memory, 84
Retentive ranges, system block configuration, 137
RETI, 307
RI (STL-Reset immediate), 173
ROUND, 226
RS (LAD/FBD Reset dominant bistable), 174
RS232, 485
Freeport mode, 483
number of communication connections, 382
types of communication, 25, 381
RS232/PPI cable, 483
RS485
communication overview, 464
communication ports configuration, 131
number of communication connections, 382
sample network configurations, 468
setting baud rate and port network address, 467
types of communication, 25, 381
RS485 address
assigning, 471
RS485 communications
STEP 7-Micro/WIN SMART settings, 33
RS485 network
address, 470
configuring the RS485 network address for a CPU, 470
searching for CPU, 474
RS485 network address
configuring, 470
RTA, 229
RTD analog inputs
coefficient, 148
rejection, 148
resistor, 148
RTD type, 148
scale, 148
smoothing, 148
system block configuration, 148
RTS (STL-Real to string), 235
RUN mode, 41, 92
RUN to STOP transition
analog output states, 146
digital output states, 136
Run-time and PLC compile errors, 860

S

S (STL-Set), 173
S_DI, 239
S_I, 239
S_R, 239
S7-200 SMART
as PROFIBUS slave device, 444
Safety circuits, 95
Sample control program, 35
Sample network configurations, RS485 devices, 468
Saving project, 39
SB (signal board) hardware/firmware ID (SMB1050-SMB1099), 890
Scan cycle
executing a single scan, 656
executing multiple scans, 656
scan times (SMW22-SMW26), 873
SCR, 329
SCRE, 329
SCRT, 329
Security, 138
SEG, 226
Selecting the network cable, 477
Service and support, 3
Set and reset dominant bistable instructions, 174
Set and reset immediate instructions, 173
SET_ADDR, 204
SET_RTC, 178
SET_RTCX, 181
Setting the CPU type, 39
Setup mode, 609
SFND (STL-Find string), 347
Shift and rotate instructions
bit (SHRB), 342
byte, word, dword, 339
SHL_B, 339
SHL_DW, 339
SHL_W, 339
SHR_B, 339
SHR_DW, 339
SHR_W, 339
SHRB, 342
SI (STL-set immediate), 173
Siemens technical support, 3
Siemens-supplied libraries, 487

Signal board (SB
SB BA01 specifications, 844
Signal board (SB)
installing and removing, 51
SB AQ01 specifications, 840
SB AQ01 wiring diagram, 841
SB BA01 wiring diagram, 844
SB CM01 specifications, 842
SB CM01 wiring diagram, 843
SB DT04 specifications, 836
SB DT04 wiring diagram, 837
Signal board ID and error registers (SMB28-SMB29), 873
Signal boards (SB)
analog output representation (current), 824
analog output representation (voltage), 824
input representation (current), 823
input representation (voltage), 823
Signal modules (SM)
step response times, 822
SIN (sine), 290
SINA_PARA_S instruction
example, 642
SINA_POS instruction
Absolute positioning mode example, 607
Incremental jog mode example, 625
Jog mode example, 622
operating mode selection, 601
Referencing (active referencing) mode example, 614
Referencing (set reference point) mode example, 617
Relative positioning mode example, 603
Setup mode example, 611
Traversing blocks mode example, 619
SINA_SPEED instruction
example, 633
SINAMICS library
overview, 487
SINA_PARA_S instruction, 638
SINA_POS instruction, 595
SINA_SPEED instruction, 631
SINAMICS_Control, 591
SINAMICS_Parameter, 591
SINAMICS library features, 591
SIP_ADDR, 205
SLB (STL-Shift left byte), 339
SLD (STL-Shift left dword), 339
SLW (STL-Shift left word), 339
SM (Special memory) assignments and functions, 865
SM locations (Axis of Motion), 742
SM memory, PTO/PWM operation, 278
SMB0 system status bits, 867
SMB1 instruction execution status bits, 868
SMB1000-SMB1049 CPU hardware/firmware ID, 889
SMB1050-SMB1099 SB (signal board) hardware/firmware ID, 890
SMB1100-SMB1299 EM (expansion module) hardware/firmware ID, 890
SMB130 Port 1 configuration, 874
SMB136-145 (HSC3) high-speed counter 3, 875
SMB146-155 (HSC4) high-speed counter 4, 875
SMB156-165 (HSC5) high-speed counter 5, 875
SMB1800-SMB1999 PROFINET device status, 893
SMB186-SMB194 receive message control, 883
SMB2 Freeport receive character, 869
SMB28-SMB29 signal board ID and error registers, 873
SMB3 Freeport character error, 870
SMB30 (Port 0) and SMB130 (Port 1) configuration, 874
SMB34-SMB35 time interval values for timed interrupts, 874
SMB36-45 (HSC0) high-speed counter 0, 875
SMB4 interrupt queue overflow, run-time program error, interrupts enabled, Freeport transmitter idle, value forced, 870
SMB46-55 (HSC1) high-speed counter 1, 875
SMB480-SMB515 Data log status, 887
SMB5 I/O error status bit, 871
SMB56-65 (HSC2) high-speed counter 2, 875
SMB566-SMB579: PTO2 and PWM2 high-speed outputs,
SMB600-SMB649 Axis 0 motion control, 888
SMB66-SMB85, SMB166-SMB169, and SMB176-SMB179: PTO0, PWM0, PTO1, and PWM1 high-speed outputs,
SMB66-SMB85, SMB166-SMB169, SMB176-SMB179, and SMB566-SMB579: PTO0, PWM0, PTO1, PWM1, PTO2, and PWM2 high-speed outputs,
SMB6-SMB7 CPU ID register, 871
SMB86-SMB94 and SMB186-SMB194 receive message control, 883
SMB8-SMB19 I/O module ID and error registers, 872
SMW100-SMW114 System alarms, 886
SMW22-SMW26 scan times, 873
SMW98 Expansion I/O bus - communication errors, 885
Software debugging, 647
Special memory assignments and functions, 865
Special memory bytes
EM DP01 PROFIBUS DP, 455
Specifications, (SB BA01)
analog input representation (current), 823

analog input representation (voltage), 823
analog output representation (current), 824
analog output representation (voltage), 824
CE approval, 751
contamination level, 754
CPU CR20s, 756
CPU CR30s, 767
CPU CR40, 778
CPU CR40s, 778
CPU CR60, 789
CPU CR60s, 789
CPU SR20, 756
CPU SR30, 767
CPU SR40, 778
CPU SR60, 789
CPU ST20, 756
CPU ST30, 767
CPU ST40, 778
CPU ST60, 789
electromagnetic compatibility (EMC), 752
EM AE04, 813
EM AM06, 818
EM AQ02, 816
EM AR02 (RTD), 830
EM AT04, 825
EM DP01 PROFIBUS DP, 845
EM DR08, 803
EM DR16, 807
EM DR32, 807
EM DT08, 803
EM DT16, 807
EM DT32, 807
EM QR16, 803
EM QT16, 803
environmental conditions, 753
general technical specifications, 751
industrial environments, 752
insulation, 754
overvoltage, 754
protection, 754
rated voltages, 754
relay electrical service life, 756
SB AE01, 838
SB AQ01, 840, 841
SB CM01, 842
SB DT04, 836, 837
step response times (SM), 822
SQRT (square root), 290
SR (LAD/FBD Set dominant bistable), 174
SRB (STL-Shift right byte), 339
SRD (STL-Shift right dword), 339
SRW (STL-Shift right word), 339
SSCPY (STL-Copy substring), 346
SSTR_CPY, 346
Starting
startup options, 142
Starting Ethernet communications
STEP 7-Micro/WIN SMART, 30
Starting RS485 communications
STEP 7-Micro/WIN SMART, 33
Static IP information, 391
Status
building a status chart, 652
displaying in program editor, 649
executing a limited number of scans, 656
Status error (timestamp mismatch), 859
Status LEDs
CPU, 92
EM DP01 PROFIBUS DP, 456
Expansion modules (EMs), 92
PROFINET network, 413
STD (STL-Sub-string to double integer), 239
STEP 7-Micro/WIN (earlier versions), 99
STEP 7-Micro/WIN SMART
connecting with the CPU, 31, 33, 34, 392, 472
equipment requirements, 26
Ethernet port configuration, 391
RS485 network port configuration, 470
RUN and STOP mode, 41, 92
Steppers in motion control, 676
STI (STL-Sub-string to integer), 239
STL
logic stack operations, 168
status options, 652
STL editor, 105
STOP (instruction), 337
STOP mode, 41, 92
analog output states, 146
digital output states, 136
writing and forcing outputs, 655
STR (STL-Sub-string to real), 239
STR_FIND, 347
String instructions
copy substring, 346
Find string / character, 347
Strings
format, 75
representation, 75
SUB_DI, 284
SUB_I, 284
SUB_R, 284
Subroutine instructions
Call parameter and return examples, 369
CALL, RET, 368

Subroutines
Axis of Motion, 695
element of user program, 96
guidelines, 696
PWMx_RUN, 675
Support, 3
Suppression circuits, 60
SWAP, 324
Symbol table, 109
Symbols (symbolic addressing)
defining global symbols, 109
indirect addressing, 111
Synchronous updates (PWM instruction), 278
System alarms (SMW100-SMW114), 886
System block, 97
BA01 battery signal board, 156
CPU configuration, 129
digital input filters, 134
IP address of CPU, 393
password and security, 138
RS485 network address of CPU, 470
RS485/RS232 CM01 communications signal board, 155
RTD analog input module, 147
startup options, 142
TC analog input module, 152
System status bits (SMB0), 867

T

Table instructions
ATT, 350
FIFO/LIFO, 351
FILL_N, 353
TBL_FIND, 354
TAN (tangent), 290
TBL_FIND, 354
TC analog input module, 152
TC analog inputs
rejection, 152
scale, 152
smoothing, 152
system block configuration, 152
TC type, 152
Technical specifications, 751
Technical support, 3
Terminal block connector, 52
Thermal zone, 44
Thermocouple
basic operation, 155, 828
cold junction compensation, 155, 828
EM AT04 Thermocouple filter selection table, 828
EM AT04 Thermocouple selection table, 828
Timed interrupt configuration (SMB34-SMB35), 874
Time-of-day
clock instructions, 178
extended clock instructions, 181
protection for reads and writes, 138
Timer instructions
BITIM, CITIM, 367
interrupts, 313
programming tips and examples, 360
TON, TONR, TOF, 358
Timestamp mismatch (PC/PLC program difference), 859
TODR (STL-Read time-of-day clock), 178
TODRX (STL-Read time-of-day clock extended), 181
TODW (STL-Write time-of-day clock), 178
TODWX (STL-Write time-of-day clock extended), 181
TOF (Off-delay timer), 358
TON (On-delay timer), 358
TONR (On-delay timer retentive), 358
Tools (options)
execution status coloring, 649
STL status, 652
Tools menu
Find PROFINET Devices, 411
Motion control panel, 723
PID Tune control panel, 669
Transfer card, 85
Transmission rate, 475
Transmit instruction
example, 203
transmitting data, 193
Traversing blocks mode, 618
Troubleshooting
LED indicators, 92
Troubleshooting S7-200 SMART hardware, 658
TRUNC, 226

U

Uploading programs, 83
User-defined libraries, 644
USS protocol instructions
example program, 588
using, 577
USS_CTRL, 580
USS_INIT, 578
USS_RPM_x, 583
USS_WPM_x, 585
USS protocol library
calculating time for communications, 576

execution errors, 588
overview, 487, 575
requirements, 575
using the USS protocol instructions, 577

V

V90 drives, 681, 709, 910
Variable table, 112
Vibration, 49

W

WAND_B, 320
WAND_DW, 320
WAND_W, 320
WDR (watchdog timer reset), 337
Wiring diagram
EM AT04, 825
wiring diagrams
EM DE16, 802
Wiring diagrams
CPU CR30s, 777
CPU CR40, 786
CPU CR40s, 788
CPU CR60, 797
CPU CR60s, 799
CPU SR20, 765
CPU SR30, 775
CPU SR40, 786
CPU SR60, 797
CPU ST20, 765
CPU ST30, 775
CPU ST40, 786
CPU ST60, 797
EM AE04, 815
EM AM06, 821
EM AQ02, 817
EM AQ04, 817
EM AR02 (RTD), 835
EM DE08, 802
EM DP01 PROFIBUS DP module, 848
EM DR08, 805
EM DR16, 810
EM DR32, 811
EM DT08, 805
EM DT16, 810
EM DT32, 811
SB AQ01, 841
SB CM01, 843
Wiring diagrams
SB BA01, 844
Wiring guidelines, 58
clearance for airflow and cooling, 44
DIN rail, 49
grounding, 58
grounding and circuit, 56
inductive loads, 60
installation, 43
isolation, 57
lamp loads, 60
separate the devices from heat, high voltage, and electrical noise, 43
terminal block connector, 52
Wizards
high-speed counter (HSC), 254
Text Display, 24
WOR_B, 320
WOR_DW, 320
WOR_W, 320
Word access, 68
Word consistency
PROFIBUS, 447
Writing values
outputs, 65
writing and forcing outputs in STOP mode, 655
WXOR_B, 320
WXOR_DW, 320
WXOR_W, 320

X

XORB (STL-XOR byte), 320
XORD (STL-XOR dword), 320
XORW (STL-XOR word), 320