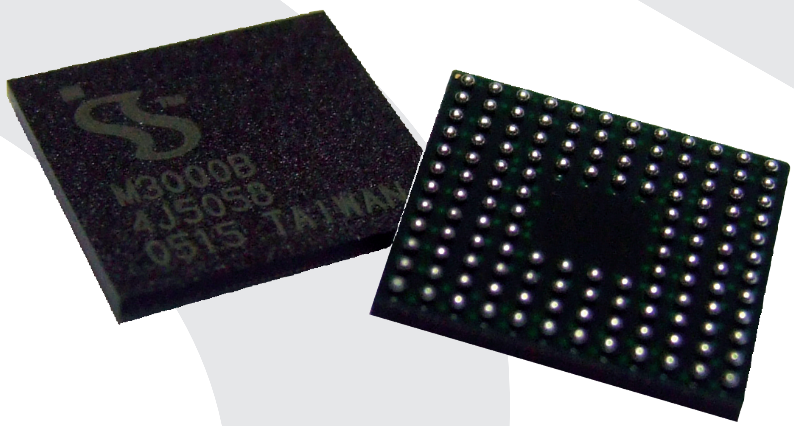
M3000

MOTOR AND MOTION CONTROLLER

USER'S GUIDE



SYSTEM SEMICONDUCTOR™

M3000 Manual Change Log		
Date	Revision	Changes
08/13/2007	R081307	Initial Release

The information in this book has been carefully checked and is believed to be accurate; however, no responsibility is assumed for inaccuracies.

System Semiconductor, reserves the right to make changes without further notice to any products herein to improve reliability, function or design. System Semiconductor does not assume any liability arising out of the application or use of any product or circuit described herein in accordance with System Semiconductor's standard warranty; neither does it convey any license under its patent rights of others. System Semiconductor SYSTEM SEMICONDUCTOR inc.™ and SSI are trademarks of System Semiconductor.

System Semiconductor's general policy does not recommend the use of its products in life support or aircraft applications wherein a failure or malfunction of the product may directly threaten life or injury. Per System Semiconductor's terms and conditions of sales, the user of System Semiconductor, products in life support or aircraft applications assumes all risks of such use and indemnifies System Semiconductor, against all damages.

M3000 Motor and Motion Controller
Revision R081307

CONTENTS

SYSTEM SEMICONDUCTOR

SYSTEM SEMICONDUCTOR

LIST OF FIGURES

LIST OF TABLES

This Page Intentionally Left Blank

SECTION 1
M3000 INTRODUCTION

The System Semiconductor M3000 Motion Controller is a highly integrated, mixed signal system-on-a-chip. The M3000 combines all the major building blocks necessary to control and position multi-phase step motors while also working as a high-speed general purpose microcontroller incorporating extensive communication, analog and system functions.

NOTE The M3001 128 FPBGA Package has 64kx16 Program memory (Code Space) internal to the device.

This is not available on the M3000 128 and 160 QFP packages.

Integration of System Semiconductor's patent-pending phase current control circuits enables motor performance to reach new limits of increased speed and smoothness while lowering audible noise and vibration. And with System Semiconductor's advanced acceleration, velocity and position control circuits virtually eliminating corresponding time critical tasks, the CPU is freed to perform other system control functions allowing system throughput rivaling high-end DSP's costing far more.

Incorporation of the M3000's extensive communication and general analog functions provides the user the capability to control a large variety of systems without additional circuits.

By integrating all major system's functions into one system-on-a-chip, performance and reliability are greatly enhanced while cost and time to market are reduced. A large temperature range also makes the M3000 ideal for commercial, industrial and automotive applications.

Integrated into the M3000:

CPU Block

- A high-speed RISC based Atmel AVR core running at 20 MHz
- 4K x 8 of data RAM
- 64K x 16 program memory internal (M3001 128 TFBGA Only)
- 64K x 16 program memory interface (M3000 128 and 160 QFP Only)
- Boot code for system initialization and in-circuit FLASH programming
- JTAG port for system debug
- Watchdog timer
- Interrupt controller with 2 external interrupts
- Internal and external counter timers
- 20 general purpose I/O lines[1]
- External 4K x 8 data interface[2]

Communication Block

- 2 SPI ports
- General purpose UART
- CAN controller

General Purpose Analog Block

- 10-bit A to D converter
- 10-bit D to A converter
- Analog MUX
- Operational amplifier

Motor Phase Current Control Block

- Sine/cosine generator capable of 20 different microstep resolutions including:
  - » Degrees: 0.01 deg / microstep[3]
  - » Metric: 0.001 mm / microstep[3] (25.4 mm/rev linear device)
  - » Arc minutes: 1 arc minute / microstep[3]
- 10-bit sine/cosine D to A converters
- 8-bit reference D to A converter
- Advanced dual H-bridge control
- Advanced phase current control with low and mid range resonance reduction

Motion Control Block

- Advanced acceleration/deceleration velocity generator capable of up to a 5 MHz Step Clock rate
- 32-bit position counter
- 32-bit position compare register
- 32-bit high-speed position capture register
- 5MHz encoder interface
- 32-bit encoder counter
- External clock interface for following (with ratio) or providing an external clock and direction that accepts/outputs step and direction, quadrature or step up/down signals

[1]12 I/O lines on 128-pin TQFP Package, 20 I/O lines on the 128 Pin TFBGA (M3001) and 160-pin LQFP package.

[2]Not available in 128 pin TQFP package.

[3]200 step / revolution motor.

M3000 Block Diagram

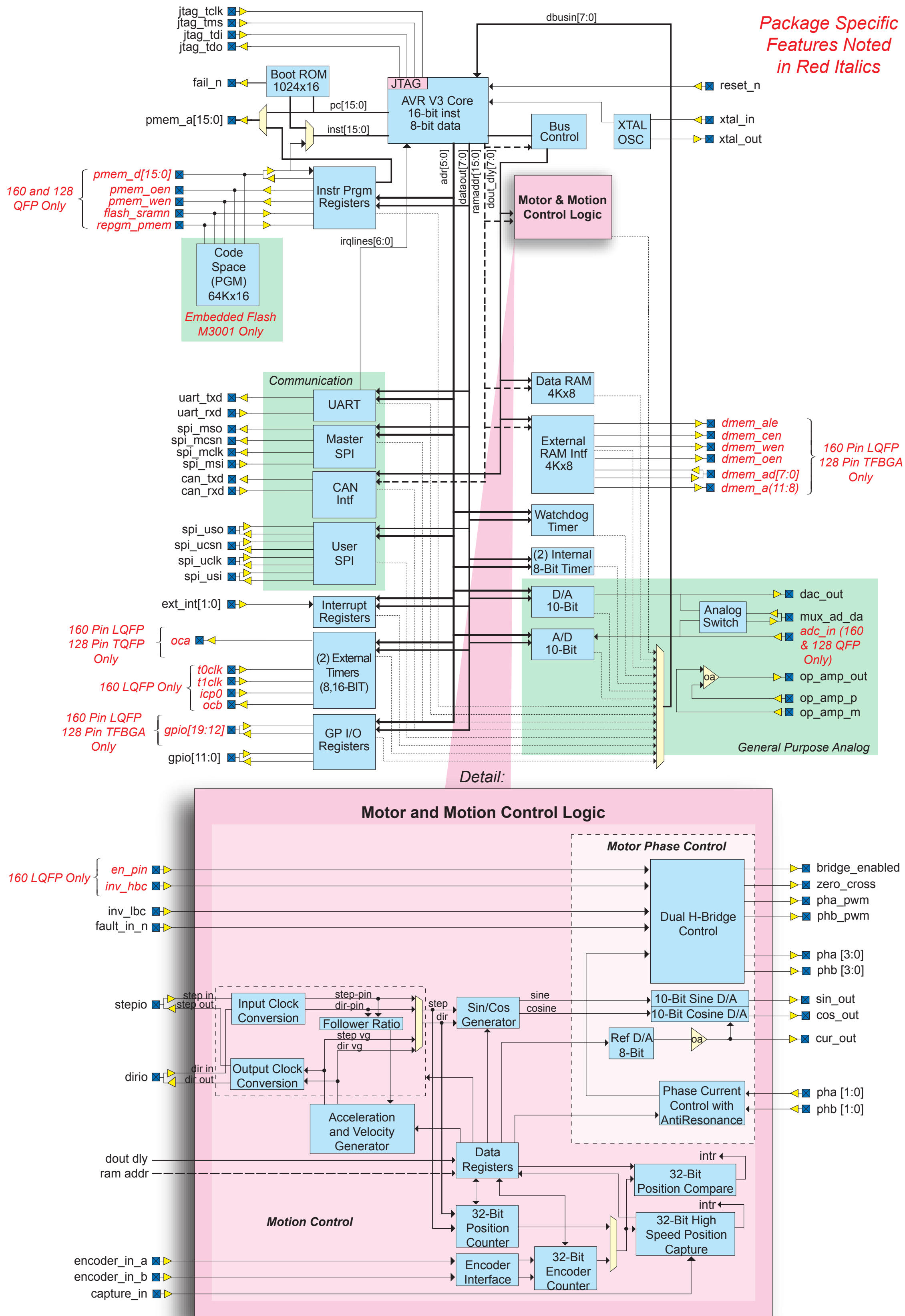


Figure 1.1: M3000 Motor & Motion Controller ASIC Block Diagram

This Page Intentionally Left Blank

SECTION 2
PHYSICAL CHARACTERISTICS

Absolute Maximum Ratings

Absolute Maximum Ratings	
Parameter	**Value**
Operating Junction Temperature	-55°C to 125°C
Storage Temperature	-65°C to 150°C
Operating Ambient	
128 Pin TFBGA (M3001)	+85°C
128 Pin QFP	+85°C
160 Pin QFP	+85°C
Digital Input Voltage	+5.5 VDC
Analog Input Voltage	3.6 VDC
Operating Voltage (V_{dd}, AV_{dd})	+3.6 VDC

Table 2.1: Absolute Maximum Ratings

Analog Characteristics

Analog Characteristics					
Device	**Parameter**	**Minimum**	**Typical**	**Maximum**	**Unit**
Motor Drive DAC	Resolution		10		bit
	Integral Linearity			±2.5	LSb
	Differential Linearity			±1.0	LSb
	Zero Offset			±2.0	LSb
	Reference Voltage	0	V_{dd} - 0.1		V
Motor Drive Reference DAC	Resolution		8		bit
	Integral Linearity			±2.5	LSb
	Differential Linearity			±1.0	LSb
	Zero Offset			±2.0	LSb
	Reference Voltage		V_{dd}		V
General Purpose DAC	Resolution		10		bit
	Integral Linearity			±2.5	LSb
	Differential Linearity			±1.0	LSb
	Zero Offset			±2.0	LSb
	Reference Voltage		V_{dd}		V
General Purpose ADC	Resolution		10		bit
	Integral Linearity			±2.5	LSb
	Differential Linearity			±1.0	LSb
	Zero Offset			±2.0	LSb
	Reference Voltage		V_{dd}		V

Table 2.2: Analog Characteristics

DC Characteristics

Buffer	Used For	Symbol	Parameter	Test Condition	Min.	Typ.	Max	Unit
M3000 DC Electrical Characteristics								
Power (PVSS,PVDD)	Power Supply	Vdd	Supply Voltage		3.0	3.3	3.6	VDC
Supply Current (Max)	M3000	Idd	Supply Current (Max)				119	mA
	M3001	Idd	Supply Current (Max)				131	mA
Input, CMOS, Schmidt, 5V Tolerant (PICSV)	All Inputs and I/O except Program Memory Data I/O	Iih	High-Level Input Current	Vin=Vdd Vdd=Vdd (Max)			10	µA
		Iil	Low-Level Input Current	Vin=Vss Vdd=Vdd (Max)	-10			µA
		Vhys	Schmidt Hysteresis			0.6		V
		Vih	High-Level Input Voltage		2.0			V
		Vil	Low-Level Input Voltage				1.2	V
Input, CMOS, 5V Tolerant (PICV)	Program Memory Data I/O	Iih	High-Level Input Current	Vin=Vdd Vdd=Vdd (Max)			10	µA
		Iil	Low-Level Input Current	Vin=Vss Vdd=Vdd (Max)	-10			µA
		Vih	High-Level Input Voltage		2.0			V
		Vil	Low-Level Input Voltage				1.2	V
Output, CMOS, 2 mA, 5V Tolerant (PO11V)	All Outputs and I/O except CANTX Output	Voh	High-Level Output Voltage	Ioh=1.7 mA Vdd=Vdd (Min)	0.7 Vdd			V
		Vol	Low-Level Output Voltage	Ioh=8mA			0.5	V
		Ios	Output Short Circuit Current	Vout=Vdd Vdd=Vdd (Max)		15		mA
				Vout=Vss Vdd=Vdd (Max)		-11		mA
Output, CMOS, 8 mA, 5V Tolerant (PO44V)	CANTX Output	Voh	High-Level Output Voltage	Ioh=8 mA Vdd=Vdd (Min)				V
		Vol	Low-Level Output Voltage	Ioh=1.4 mA			0.5	V
		Ios	Output Short Circuit Current	Vout=Vdd Vdd=Vdd (Max)		15		mA
				Vout=Vss Vdd=Vdd (Max)		-11		mA
Output	All Tri-stateable Outputs	Ioz	High Impedance State Output Current	Vin=Vdd or Vss Vdd=Vdd (Max) No Pull-Up	-10		10	µA
Oscillator (PX4L)	Oscillator Output	Ios	Output Short Circuit Current	Vout=Vdd Vdd=Vdd (Max)		45		mA
				Vout=Vss Vdd=Vdd (Max)		-42		mA
	Oscillator	Gm	Transconductance				20	mA/V
		Vb	Self Bias Voltage				1.6	V
Pull-Up (PRU1, PRUP1)	Fixed and Programmable Pull-Up and Pull-Down Resistors	Rpu	Pull-Up				20k	Ω
Pull-Down (PRD1, PRDP1)		Rpd	Pull-Down				20k	Ω
Input, Analog (PFIAMUX)	Opamp, ADC_in, MUX_AD_DA Inputs	Iih	High-Level Input Current	Vin= Vdd=			10	µA
		Iil	Low-Level Input Current	Vin= Vdd=	10			µA
		Vih	High-Level Input Voltage			Vdd		V
		Vil	Low-Level Input Voltage			0		V
Output, Analog (PFOAMUX)	DAC_out, MUX_AD_DA, SIN_OUT, COS_OUT Outputs	Voh	High-Level Output Voltage	Ioh=0		Vdd		V
		Vol	Low-Level Output Voltage	Ioh=0		0		V
		Ios	Output Short Circuit Current	Vout=Vdd Vout=Vss		44		mA
						33		mA
Output, Analog (ascop039a)	Opamp, Cur_Out Ouputs	Voh	High-Level Output Voltage				Vdd-0.1	V
		Vol	Low-Level Output Voltage		0.1			V
		Io	Output Current			3		mA

Table 2.3: M3000 DC Electrical Characteristics

Thermal Characteristics

Thermal Ratings	
Package	**Value**
128 Pin TFBGA (M3001)	39.0 °C/W
128 Pin QFP	29.0 °C/W
160 Pin QFP	27.5 °C/W

Table 2.4: Absolute Maximum Ratings

Soldering Characteristics

Land Pattern - M3001 TFBGA Package

The recommended land pattern for the M3001 TFBGA is as follows:

NSMD (non-solder mask defined) Pad diameter..0.012" (0.305 mm)

Paste Mask Diameter ....0.012" (0.305 mm)

Solder Mask Diameter ....0.016 (0.406 mm)

Finished solder balls should be equally compressed on the BGA side and the PCB side to equalize any stresses that may occur.

Reflow Profile - M3001 TFBGA Package

Recommend following J-STD-020 for solder profiles

JOINT IPC/JEDEC STANDARD FOR MOISTURE/REFLOW SENSITIVITY CLASSIFICATION FOR NONHERMETIC SOLID STATE SURFACE-MOUNT DEVICES

This document identifies the classification level of nonhermetic solid-state surface mount devices (SMDs) that are sensitive to moisture-induced stress. It is used to determine what classification level should be used for initial reliability qualification. Once identified, the SMDs can be properly packaged, stored and handled to avoid subsequent thermal and mechanical damage during the assembly solder reflow attachment and/or repair operation. This revision now covers components to be processed at higher temperatures for lead-free assembly.

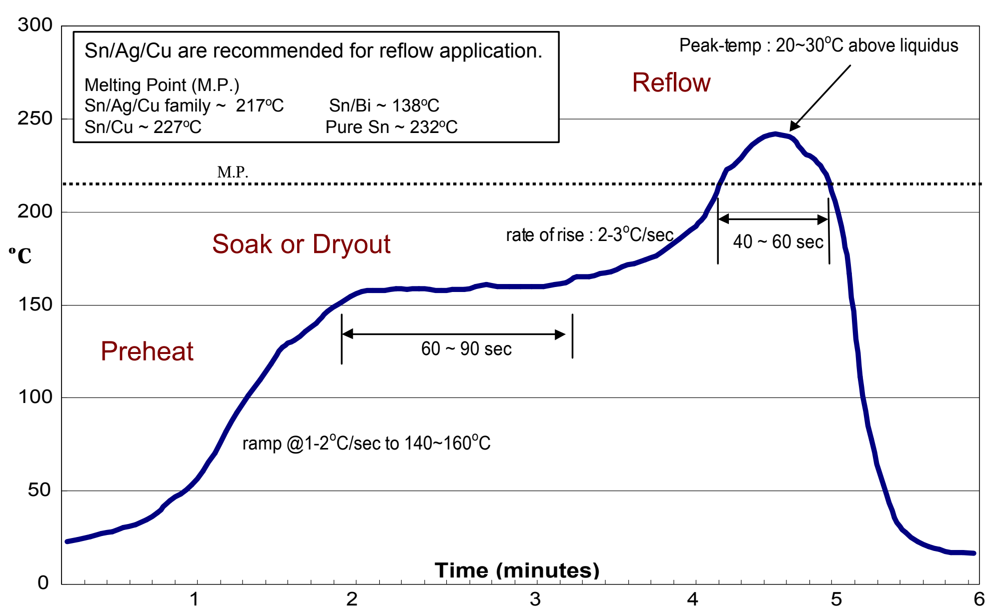


Figure 2.1: Reflow Profile for M3001 TFBGA

This Page Intentionally Left Blank

SECTION 3
MECHANICAL SPECIFICATIONS

M3000F — LQFP-160

Dimensions in Inches (mm)

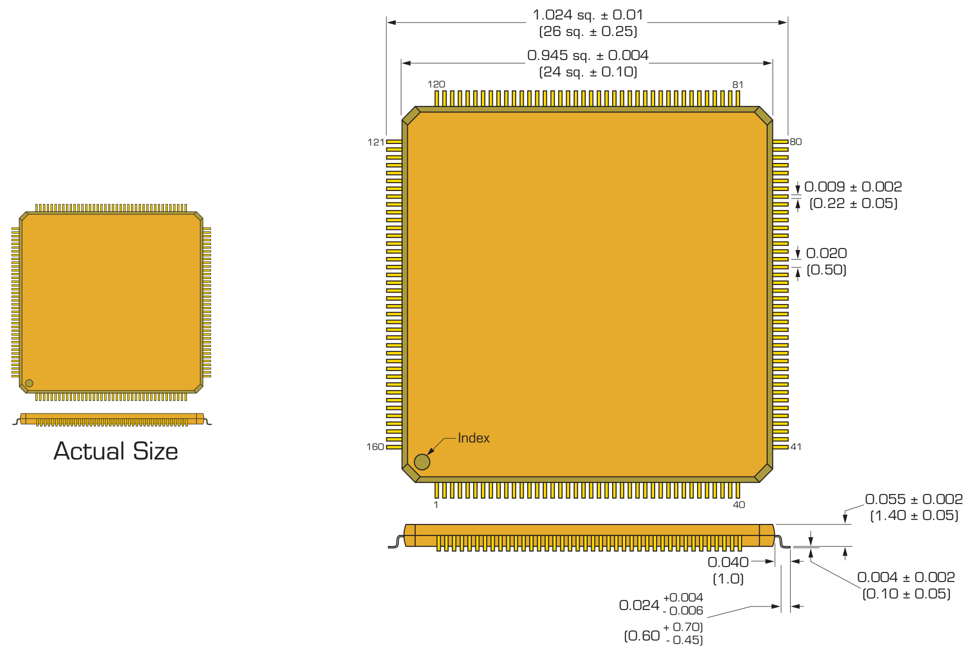


Figure 3.2: M3000F LQFP-160 Dimensions, Quad Package Top View

M3000S — TQFP-128

Dimensions in Inches (mm)

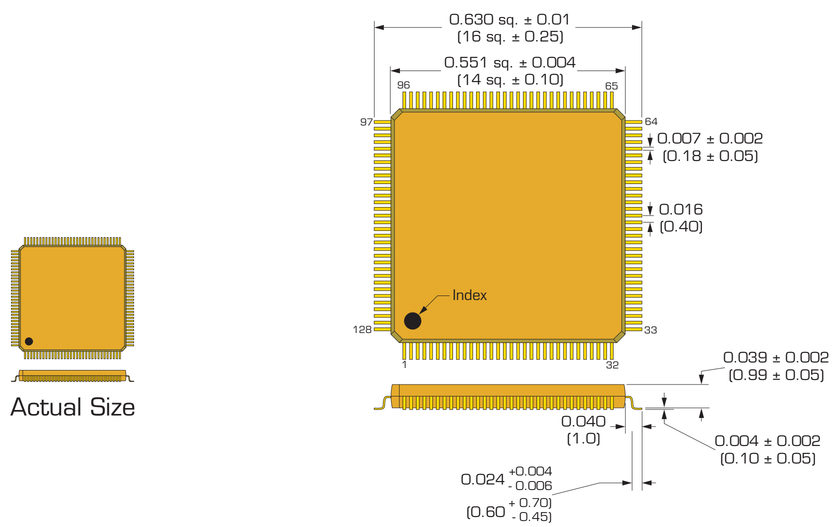


Figure 3.1: M3000S TQFP-128 Dimensions, Quad Package Top View

M3001 — TFBGA-128

Dimensions in Inches (mm)

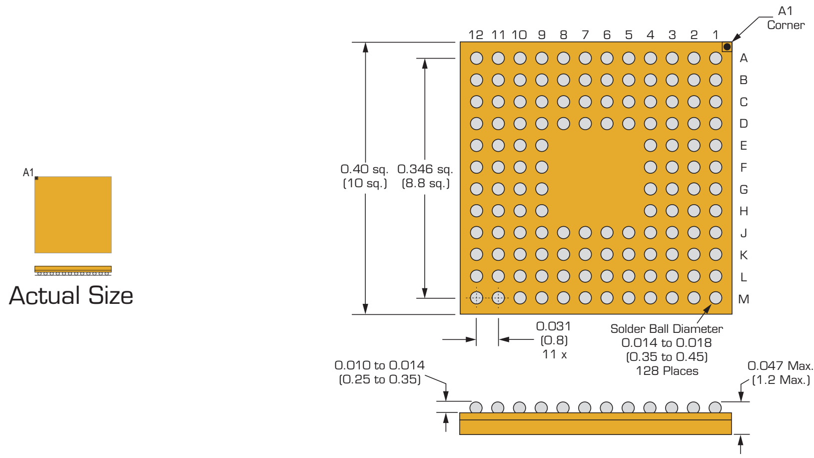


Figure 3.3: M3001 TFBGA-128 Dimensions, BGA Bottom View.

SECTION 4
PIN ASSIGNMENTS

Feature Summary by Package Style

Feature Summary By Package Style				
	FUNCTION	M3000F 160 LQFP	M3000S 128 TQFP	M3001 128 TFBGA
Processor Control				
	JTAG	Y	Y	Y
	IBOOT_N, RePgm_Pmem	Y	Y	Y
	FLASH vs RAM Speed	Y	Y	Y
	ResetN, XTAL, TEST	Y	Y	Y
	Fail_n	Y	Y	N
Memory (Code Space)				
	Pmem..Addr lines, Data lines, we, oe	Y	Y	N
	Memory is Embedded	N	N	Y
	Note: if N, requires external memory			
Communications				
	SPI_USER (configurable as Master or Slave)	Y	Y	Y
	SPI_Master only	Y	Y	Y
	CAN	Y	Y	Y
	UART	Y	Y	Y
Timers				
Timer (Internal: 8bit timer0, 16bit timer1)		Y	Y	Y
Timer (External: 8bit timer0, 16bit timer1)		Y	Y	Y
	T0clk, T1clk, Ipc0 (timer1 only)	Y	N	N
	ocA " "	Y	Y	N
	ocB " "	Y	N	N
Indexer Logic				
	Encoder_In	Y	Y	Y
	STEP_IO, DIR_IO	Y	Y	Y
	CAPTURE_IN, TRIP_Out	Y	Y	Y
	External Interrupts [0..1]	Y	Y	Y
IO Ports / Memory (Extra Data Space)				
	8 GPIO[0..7]	Y	Y	Y
	4 GPIO[8..11] / DataMem_A[8..11]	Y	Y	Y
	8 GPIO[12..19] / DataMem_AD[0..7]	Y	N	Y
	DataMem_[CEN, ALE, WEN, OEN]	Y	N	Y
Analog				
	Op_Amp [_Out, _M, _P]	Y	Y	Y
	adc_in	Y	Y	N
	Mux_ad_da	Y	Y	Y
	dac_out	Y	Y	N
Motor Bridge Driver Stage				
	PhA_[LH, LL, _RH, _RL, _Fwd, _Rev]	Y	Y	Y
	PhB_[LH, LL, _RH, _RL, _Fwd, _Rev]	Y	Y	Y
	BridgeEnabled	Y	Y	Y
	PhA_Pwm, PhB_Pwm	Y	Y	N
	Pwm_Osc, Mask, Sin_Sign	Y	N	N
	Zero_Cross out	Y	Y	Y
	Sin_Out, Cos_Out	Y	Y	Y
	Cur_Out	Y	Y	Y
	Pwm_Current Reference Output	Y	N	N
	dac_SDA, dac_CS_N	Y	N	N
	En_Pin	Y	N	N
	Inv_HBC	Y	N	N
	Fault_IN_N	Y	Y	Y
	Inv_LBC	Y	Y	Y

Table 4.1: Features by Package Style

The following tables illustrate the Motor and Motion Control Pin Assignments for the M3000 in its three different packages:

- M3000F LQFP-160 (Not a stock item. Contact factory for availability.)
- M3000S TQFP-128
- M3001 TFBGA-128

The pins are grouped by function. Table cells marked "X" indicate a feature not available on that particular package.

Processor Control

Group	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#	Pull-Up (PU) Pull-Down (PD) Program (PGM)
JTAG	JTAG Emulator Clock Input	JTAG_TCLK	141	113	B7	PD
JTAG	JTAG Emulator Mode Select Input	JTAG_TMS	146	117	B6	PU
JTAG	JTAG Emulator Data Input	JTAG_TDI	147	118	A6	PU
JTAG	JTAG Emulator Data Output	JTAG_TDO	142	114	A7	
JTAG	JTAG Reset Input	JTAG_TRST_N	2	2	B1	PU
XTAL	Crystal Input	XTAL_IN	99	79	G12	
XTAL	Crystal Output	XTAL_OUT	100	80	G11	
MISC	Internal Boot Input	IBOOT_N	21	17	F3	PD
MISC	Reprogram Flash Memory Input	REPGM_PMEM	28	23	H1	PD
MISC	Program Memory Type Input: Flash=1/SRAM=0	FLASH_SRAM_N	27	22	G4	PU
MISC	Master Reset Input	RESET_N	31	25	H3	PU
MISC	Test Control Input	TEST	24	20	G2	PD
MISC	Fail Boot Test Output and Serial DAC Clock Output	FAIL_N/DAC_CLK	23	19	X	

Table 4.2: Processor Control Pin Functions and Assignment

Communications

Group	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#	Pull-Up (PU) Pull-Down (PD) Program (PGM)
USER SPI	SPI User Clock Output I/O (Master or Slave)	SPI_UCLK	103	83	F12	PU
USER SPI	SPI User Chip Select Output I/O (Master or Slave)	SPI_UCSN	102	82	G9	PU
USER SPI	SPI User Serial In I/O (Master or Slave) (MOSI)	SPI_USI	94	75	H12	PU
USER SPI	SPI User Serial Out I/O (Master or Slave) (MISO)	SPI_USO	95	76	H11	PU
MASTER SPI	SPI Master Clock Output	SPI_MCLK	98	78	H9	
MASTER SPI	SPI Master Chip Select Output	SPI_MCSN	90	72	J12	
MASTER SPI	SPI Master Serial Input	SPI_MSI	91	73	J11	PU
MASTER SPI	SPI Master Serial Output	SPI_MSO	93	74	J10	
CAN	CAN Transmit Data Output	CAN_TXD	107	86	F9	
CAN	CAN Receive Data Input	CAN_RXD	112	90	E9	PU
UART	UART Transmit Data Output	UART_TXD	104	84	F11	
UART	UART Receive Data Output	UART_RXD	106	85	F10	PU

Table 4.3: Communications Pin Functions and Assignment

Memory (Code/Program Space) M3000 128 and 160 QFP Only

NOTE

The M3001 128 FPBGA Package has 64kx16 Program memory (Code Space) internal to the device.

This is not available on the M3000 128 and 160 QFP packages.

	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#
PROGRAM MEMORY ADDRESS LINES	Program Memory SRAM/FLASH Address Bus A0 Output	PMEM_A0	86	69
	Program Memory SRAM/FLASH Address Bus A1 Output	PMEM_A1	76	61
	Program Memory SRAM/FLASH Address Bus A2 Output	PMEM_A2	71	57
	Program Memory SRAM/FLASH Address Bus A3 Output	PMEM_A3	85	68
	Program Memory SRAM/FLASH Address Bus A4 Output	PMEM_A4	83	67
	Program Memory SRAM/FLASH Address Bus A5 Output	PMEM_A5	78	62
	Program Memory SRAM/FLASH Address Bus A6 Output	PMEM_A6	72	58
	Program Memory SRAM/FLASH Address Bus A7 Output	PMEM_A7	67	54
	Program Memory SRAM/FLASH Address Bus A8 Output	PMEM_A8	62	50
	Program Memory SRAM/FLASH Address Bus A9 Output	PMEM_A9	58	46
	Program Memory SRAM/FLASH Address Bus A10 Output	PMEM_A10	53	42
	Program Memory SRAM/FLASH Address Bus A11 Output	PMEM_A11	49	39
	Program Memory SRAM/FLASH Address Bus A12 Output	PMEM_A12	46	37
	Program Memory SRAM/FLASH Address Bus A13 Output	PMEM_A13	45	36
	Program Memory SRAM/FLASH Address Bus A14 Output	PMEM_A14	47	38
	Program Memory SRAM/FLASH Address Bus A15 Output	PMEM_A15	43	35
PROGRAM MEMORY DATA LINES	Program Memory SRAM/FLASH Data Bus D0 I/O	PMEM_D0	87	70
	Program Memory SRAM/FLASH Data Bus D1 I/O	PMEM_D1	89	71
	Program Memory SRAM/FLASH Data Bus D2 I/O	PMEM_D2	79	63
	Program Memory SRAM/FLASH Data Bus D3 I/O	PMEM_D3	75	60
	Program Memory SRAM/FLASH Data Bus D4 I/O	PMEM_D4	66	53
	Program Memory SRAM/FLASH Data Bus D5 I/O	PMEM_D5	70	56
	Program Memory SRAM/FLASH Data Bus D6 I/O	PMEM_D6	74	59
	Program Memory SRAM/FLASH Data Bus D7 I/O	PMEM_D7	68	55
	Program Memory SRAM/FLASH Data Bus D8 I/O	PMEM_D8	63	51
	Program Memory SRAM/FLASH Data Bus D9 I/O	PMEM_D9	59	47
	Program Memory SRAM/FLASH Data Bus D10 I/O	PMEM_D10	54	43
	Program Memory SRAM/FLASH Data Bus D11 I/O	PMEM_D11	64	52
	Program Memory SRAM/FLASH Data Bus D12 I/O	PMEM_D12	60	48
	Program Memory SRAM/FLASH Data Bus D13 I/O	PMEM_D13	55	44
	Program Memory SRAM/FLASH Data Bus D14 I/O	PMEM_D14	50	40
	Program Memory SRAM/FLASH Data Bus D15 I/O	PMEM_D15	51	41
CTRL	Program Memory SRAM/FLASH Write Enable Output	PMEM_WEN	42	34
	Program Memory SRAM/FLASH Output Enable Output	PMEM_OEN	82	66

Table 4.4: Memory (Code/Program Space) Pin Functions and Assignment

Memory (Code/Program Space) M3001 128 TFBGA

Memory (Code/Program Space) Pin Functions and Assignment			
	Function	Pin Name	M3001 128 TFBGA Pin#
PROGRAM MEMORY ADDRESS LINES	Address Bus A0 Output. Test Pin, Factory Use Only	TA0	L12
	Address Bus A1 Output. Test Pin, Factory Use Only	TA1	L9
	Address Bus A2 Output. Test Pin, Factory Use Only	TA2	L8
	Address Bus A3 Output. Test Pin, Factory Use Only	TA3	M12
	Address Bus A4 Output. Test Pin, Factory Use Only	TA4	M11
	Address Bus A5 Output. Test Pin, Factory Use Only	TA5	M9
	Address Bus A6 Output. Test Pin, Factory Use Only	TA6	M8
	Address Bus A7 Output. Test Pin, Factory Use Only	TA7	M7
	Address Bus A8 Output. Test Pin, Factory Use Only	TA8	M6
	Address Bus A9 Output. Test Pin, Factory Use Only	TA9	M5
	Address Bus A10 Output. Test Pin, Factory Use Only	TA10	M4
PROGRAM MEMORY DATA LINES	Data Bus D0 I/O. Test Pin, Factory Use Only	TD0	K12
	Data Bus D1 I/O. Test Pin, Factory Use Only	TD1	K11
	Data Bus D2 I/O. Test Pin, Factory Use Only	TD2	K10
	Data Bus D3 I/O. Test Pin, Factory Use Only	TD3	K9
	Data Bus D4 I/O. Test Pin, Factory Use Only	TD4	L7
	Data Bus D5 I/O. Test Pin, Factory Use Only	TD5	K8
	Data Bus D6 I/O. Test Pin, Factory Use Only	TD6	J9
	Data Bus D7 I/O. Test Pin, Factory Use Only	TD7	J8
	Data Bus D8 I/O. Test Pin, Factory Use Only	TD8	J7
	Data Bus D9 I/O. Test Pin, Factory Use Only	TD9	J6
	Data Bus D10 I/O. Test Pin, Factory Use Only	TD10	J5
	Data Bus D11 I/O. Test Pin, Factory Use Only	TD11	K7
	Data Bus D12 I/O. Test Pin, Factory Use Only	TD12	K6
	Data Bus D13 I/O. Test Pin, Factory Use Only	TD13	K5
	Data Bus D14 I/O. Test Pin, Factory Use Only	TD14	K4
	Data Bus D15 I/O. Test Pin, Factory Use Only	TD15	L4
	Program Memory SRAM/FLASH Chip Enable	PMEM_CEN	L11

Table 4.5: Memory (Code/Program Space) Pin Functions and Assignment - M3001

WARNING! Ball L11 - Chip Enable MUST be connected to Ground for the M3001 to function.

External Timer

	External Timer Pin Functions and Assignment					
	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#	Pull-Up (PU) Pull-Down (PD) Program (PGM)
TIMER	Timer T/C0 Clock Input	T0CLK	105	X	X	PU
	Timer T/C 1 Clock Input	T1CLK	109	X	X	PU
	Timer 0 Capture Input	ICP0	113	X	X	PU
	Timer A Compare Output	OCA	26	21	X	
	Timer B Compare Output	OCB	117	X	X	

Table 4.6: External Timer Pin Functions and Assignment

Indexer Logic

	Indexer Logic Pin Functions and Assignment					
	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#	Pull-Up (PU) Pull-Down (PD) Program (PGM)
ENCODER	Encoder Channel A Input	ENC_IN_A	134	107	D8	PU
	Encoder Channel B Input	ENC_IN_B	139	111	D7	PU
	Encoder Index Input	ENC_IN_IDX	143	115	D6	PU
LOGIC I/O	Capture Input	CAPTURE_IN	131	105	B9	PU
	Trip Output	TRIP_OUT	133	106	A9	
	Step Clock I/O (or Step-up, or CH A)	STEP_IO	137	109	B8	PU
	Direction I/O (or Step-down, or CH B)	DIR_IO	138	110	A8	PU
INT	External Interupt Input 0 (Programmable Polarity)	EXT_INT0	39	31	K1	PU
	External Interupt Input 1 (Programmable Polarity)	EXT_INT1	38	30	J4	PU

Table 4.7: Indexer Logic Pin Functions and Assignment

Analog

	Analog Pin Functions and Assignment					
	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#	Pull-Up (PU) Pull-Down (PD) Program (PGM)
OP-AMP 0...7	General Purpose Op-Amp Output	OP_AMP_OUT	5	4	C1	
	General Purpose Op-Amp Minus Input	OP_AMP_M	8	6	C3	
	General Purpose Op-Amp Plus Input	OP_AMP_P	9	7	D1	
MISC	General Purpose ADC Input	ADC_IN	10	8	X	
	ADC Input or DAC Output	MUX_AD_DA	11	9	D3	
	General Purpose DAC Output	DAC_OUT	14	12	X	

Table 4.8: Analog Pin Functions and Assignment

I/O Ports/Memory

	97I/O Ports/Memory (Extra Data Space) Pin Functions and Assignment					
	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#	Pull-Up (PU) Pull-Down (PD) Program (PGM)
8 GPIO 0...7	Gen. Purpose I/O 0	GPIO0	36	29	J3	PU, PGM
	Gen. Purpose I/O 1	GPIO1	35	28	J2	PU, PGM
	Gen. Purpose I/O 2	GPIO2	34	27	J1	PU, PGM
	Gen. Purpose I/O 3	GPIO3	32	26	H4	PU, PGM
	Gen. Purpose I/O 4	GPIO4	119	95	C12	PU, PGM
	Gen. Purpose I/O 5	GPIO5	118	94	D9	PU, PGM
	Gen. Purpose I/O 6	GPIO6	116	93	D10	PU, PGM
	Gen. Purpose I/O 7	GPIO7	115	92	D11	PU, PGM
GPIO/DATA MEM 8...11	Gen. Purpose I/O 8/Data Memory Address 8	GPIO8/DMEM_A8	114	91	D12	PU, PGM
	Gen. Purpose I/O 9/Data Memory Address 9	GPIO9/DMEM_A9	111	89	E10	PU, PGM
	Gen. Purpose I/O 10/Data Memory Address 10	GPIO10/DMEM_A10	110	88	E11	PU, PGM
	Gen. Purpose I/O 11/Data Memory Address 11	GPIO11/DMEM_A11	108	87	E12	PU, PGM
GPIO 12...19/ DATA MEM 0...8	Gen. Purpose I/O 12/Data Memory Address 0	GPIO12/DMEM_A0	44	X	L1	PU, PGM
	Gen. Purpose I/O 13/Data Memory Address 1	GPIO13/DMEM_A1	48	X	L2	PU, PGM
	Gen. Purpose I/O 14/Data Memory Address 2	GPIO14/DMEM_A2	52	X	M1	PU, PGM
	Gen. Purpose I/O 15/Data Memory Address 3	GPIO15/DMEM_A3	56	X	M2	PU, PGM
	Gen. Purpose I/O 16/Data Memory Address 4	GPIO16/DMEM_A4	65	X	L3	PU, PGM
	Gen. Purpose I/O 17/Data Memory Address 5	GPIO17/DMEM_A5	69	X	M3	PU, PGM
	Gen. Purpose I/O 18/Data Memory Address 6	GPIO18/DMEM_A6	73	X	G3	PU, PGM
	Gen. Purpose I/O 19/Data Memory Address 7	GPIO19/DMEM_A7	77	X	G1	PU, PGM
DATA MEM CTRL	Data Memory Chip Enable	DMEM_CEN	37	X	A5	
	Data Memory Address Latch Enable Output	DMEM_ALE	84	X	E2	
	Data Memory Write Enable Output	DMEM_WEN	88	X	D2	
	Data Memory Output Enable Output	DMEM_OEN	92	X	A10	

Table 4.9: I/O Ports/Memory (Extra Data Space) Pin Functions and Assignment

Motor Bridge Drive Stage

Group	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#	Pull-Up (PU) Pull-Down (PD) Program (PGM)
PHASE A CONTROL	Bridge Control Phase A Left High Output	PHA_LH	122	98	B12	
	Bridge Control Phase A Left Low Output	PHA_LL	123	99	B11	
	Bridge Control Phase A Right High Output	PHA_RH	126	101	A11	
	Bridge Control Phase A Right Low Output	PHA_RL	125	100	A12	
	Bridge Control Phase A Reverse Input	PHA_REV	135	108	C8	PU
	Bridge Control Phase A Forward Input	PHA_FWD	130	104	C9	PU
PHASE B CONTROL	Bridge Control Phase B Left High Output	PHB_LH	156	125	A4	
	Bridge Control Phase B Left Low Output	PHB_LL	159	127	A3	
	Bridge Control Phase B Right High Output	PHB_RH	158	126	B3	
	Bridge Control Phase B Right Low Output	PHB_RL	155	124	B4	
	Bridge Control Phase B Reverse Input	PHB_REV	150	120	C5	PU
	Bridge Control Phase B Forward Input	PHB_FWD	154	123	C4	PU
BRIDGE CONTROL	Bridge Enable Output	BRDG_EN	151	121	B5	
	Bridge Enable Input	EN_PIN	136	X	X	PU
	Invert High Bridge Control Input	INV_HBC	128	X	X	PD
	Invert Low Bridge Control Input	INV_LBC	127	102	B10	PD
PWM	Bridge Control Phase A PWM Output	PHA_PWM	129	103	X	
	Bridge Control Phase B PWM Output	PHB_PWM	152	122	X	
	PWM Oscillator Output	PWM_OSC	29	X	X	
	PWM Current Reference Output	PWM_CUR	132	X	X	
MISC	Zero Crossing Output	ZERO_CROSS	148	119	D5	
	Mask Output	MASK	25	X	X	
	$\overline{\text{Fault Input}}$	FAULT_IN_N	30	24	H2	PU
	Sine Sign Output	SIN_SIGN	33	X	X	
DAC	$\overline{\text{Serial DAC Chip Select Output}}$	DAC_CS_N	18	X	X	
	Serial DAC Data Output	DAC_SDA	19	X	X	
	DAC Sine Output	SIN_OUT	17	15	F1	
	DAC Cosine Output	COS_OUT	16	14	E4	
	Current Reference (Reference DAC Output)	CUR_OUT	7	5	C2	

Table 4.10: Motor Bridge Driver Stage Pin Functions and Assignment

Power and Ground

	Power and Ground Pin Functions and Assignment				
	Function	Pin Name	M3000 160 LQFP Pin#	M3000 128 TQFP Pin#	M3001 128 TFBGA Pin#
DIGITAL POWER	Digital Power	VDD	22	18	F4
			41	33	K3
			61	49	L6
			81	65	M10
			101	81	G10
			121	97	C10
			144	116	C6
			145	X	X
ANALOG POWER	Analog Power	AVDD	3	3	B2
			15	13	E3
DIGITAL GROUND	Digital Ground	GND	96	77	C7
			140	112	H10
			20	16	F2
			40	32	K2
			57	45	L5
			80	64	L10
			120	96	C11
			160	128	A2
			97	X	X
ANALOG GROUND	Analog Ground	AGND	1	1	A1
			12	10	D4
			13	11	E1

Table 4.11: Power and Ground Pin Functions and Assignment

SECTION 5
CPU CORE

Introduction

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory can be Reprogrammable Flash memory or Static Ram.

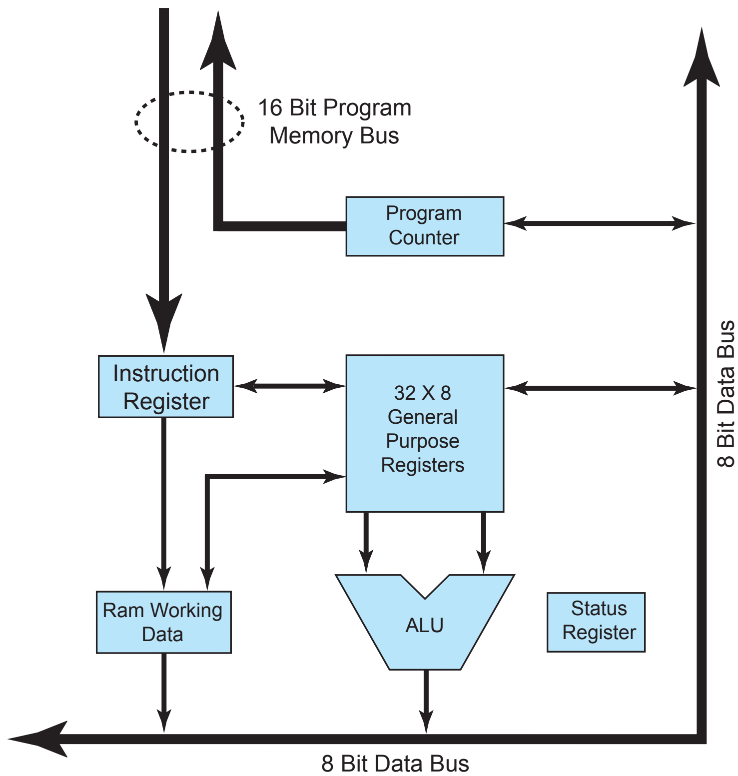


Figure 5.1: AVR MCU Block Diagram

Memory

Program Flash Based memory space is divided in two sections, the Boot program section and the Application Program section. Ram Based Program Memory space can be user configurable and can also be used as data space.

During interrupts and subroutine calls, the return address program counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The Stack Pointer SP is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register file, 0x20 - 0x5F.

ALU – Arithmetic Logic Unit

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the Instruction Set Reference for a detailed description.

Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the program Instruction Set Reference. This will, in many cases, remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash Program memory. These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section.

SREG – Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 5.1: SREG (AVR Status Register)

The fast-access Register file contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register file, the operation is executed, and the result is stored back in the Register file – in one clock cycle.

- **Bit 7 – I: Global Interrupt Enable**

  The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The Ibit can also be set and cleared by the application with the SEI and CLI instructions, as described in Program Instruction Set Reference.

- **Bit 6 – T: Bit Copy Storage**

  The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register file can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register file by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

  The Half Carry Flag H indicates a half carry in some arithmetic operations. Half Carry is useful in BCD arithmetic. See the Program Instruction Set Reference for detailed information.

- **Bit 4 – S: Sign Bit, S = N Å V**

  The S-bit is always an exclusive or between the negative flag N and the Two's Complement Overflow Flag V. See the Program Instruction Set Reference for detailed information.

- **Bit 3 – V: Two's Complement Overflow Flag**

  The Two's Complement Overflow Flag V supports two's complement arithmetic. See the Program Instruction Set Reference for detailed information.

- **Bit 2 – N: Negative Flag**

  The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the Program Instruction Set Reference for detailed information.

- **Bit 1 – Z: Zero Flag**

  The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the Program Instruction Set Reference for detailed information.

- **Bit 0 – C: Carry Flag**

  The Carry Flag C indicates a carry in an arithmetic or logic operation. See the Program Instruction Set Reference for detailed information. The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register file:

  » One 8-bit output operand and one 8-bit result input

  » Two 8-bit output operands and one 8-bit result input

  » Two 8-bit output operands and one 16-bit result input

  » One 16-bit output operand and one 16-bit result input

General Purpose Register File

Table 5.2 illustrates the structure of the 32 General Purpose Working Registers in the CPU.

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

	7 0	Addr.
General Purpose Working Registers	R0	0x00
	R1	0x01
	R2	0x02
	R13	0x0D
	R14	0x0E
	R15	0x0F
	R16	0x10
	R17	0x11
	R26	0x1A X-Register Low Byte
	R27	0x1B X-Register High Byte
	R28	0x1C Y-Register Low Byte
	R29	0x1D Y-Register High Byte
	R30	0x1E Z-Register Low Byte
	R31	0x1F Z-Register High Byte

Table 5.2: General Purpose Working Registers

As shown in Table 5.2, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer Registers can be set to index any register in the file.

The X-Register, Y-Register and Z-Register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the Data Space. The three indirect address registers X, Y, and Z are defined as follows.

In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the Instruction Set Reference for details).

	15	XH			XL	0
X - Register	7		0	7		0
	R27 (0x1B)			R26 (0x1A)		
	15	YH			YL	0
Y - Register	7			7		0
	R29 (0x1D)			R28 (0x1C)		
	15	ZH			ZL	0
Z - Register	7	0		7	0	
	R31 (0x1F)			R30 (0x1E)		

Table 5.3: X-Register, Y-Register, Z-Register

Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the stack. Note that the stack is implemented as growing from higher memory locations to lower memory locations. This implies that a stack PUSH command decreases the Stack Pointer.

The Stack Pointer points to the data SRAM stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above 0x60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI

Stack Pointer Register

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
0x3D (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Table 5.4: Stack Pointer Register

Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clkCPU, directly generated from the selected clock source for the chip. No internal clock division is used.

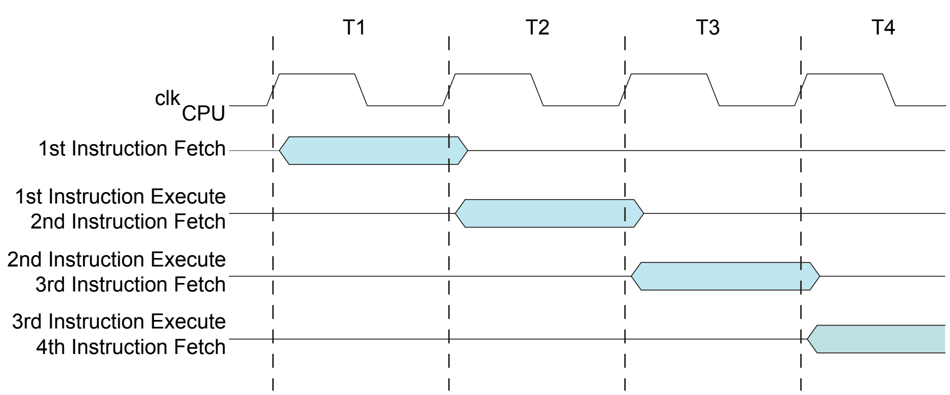


Figure 5.2: Instruction Execution Timing

Figure 5.3 illustrates the internal timing concept for the Register File. In a single clock cycle, an ALU operation using two Register Operands is executed and the result stored back to the destination Register.

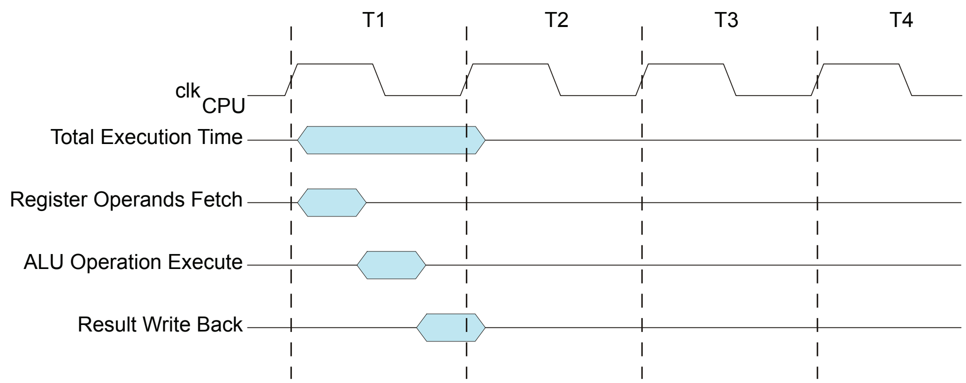


Figure 5.3: Internal Timing Concept for a Register File

Memory Maps

The AVR core is a Harvard architecture RISC processor with separate address and data lines for instruction and data memory. Instruction memory is 64kx16 and data memory is 64kx8.

Instruction Memory

Instruction Memory is separated into two pages with Boot ROM code located on the first page and normal program code located on the second. Once the boot code has completed, it switches pages to reference external program memory and performs a jump to zero

Page	Address	Instruction
Page 1: Boot ROM Code (2048 Instructions) (Internal)	0x0000 - - 0xlast instr-2	Boot Code Instructions
	0xlast instr-1	Switch Page
	0xlast instr	Jmp 0x0000
Page 2 - Program Memory (64k Instructions)	0x0000 - - - 0xFFFF	Program Instructions

Table 5.5: Instruction Memory

Additional mechanisms for programming instruction memory are located in the Boot ROM and are accessed by calling a specific function that switches pages back to the Boot ROM and executes the programming operation. Return to normal program execution is resumed with another page switch and subsequent function return.

Data Memory

Data Memory is separated into three segments: internal registers, I/O register space, and Data SRAM space. Internal and I/O registers are mapped to the lower 96 (0x0000-0x005F) locations of Data SRAM space. The following table shows the location of the regions and their corresponding addresses.

Data Memory Register Files

Register File	Data Address Space
R0	0x0000
R1	0x0001
R2	0x0002
R29	0x001D
R30	0x001E
R31	0x001F

I/O Registers	
0x00	0x0020
0x01	0x0021
0x02	0x0022
....	....
0x3D	0x005D
0x3E	0x005E
0x3F	0x005F

Data SRAM Space	
CAN Interface Registers	0x0100 | 0x01FF
Motor and Motion Control Registers	0x0200 | 0x02FF
Unused	0x0300 | 0x03FD
Reserved	0x03FE
Reserved	0x03FF
Unused	0x0400 | 0x07FF
Internal Timer Registers	0x0800 | 0x080B
Unused	0x080C | 0x0FFF
Internal Data RAM	0x1000 | 0x2FFF
External RAM 256 or 4K (When Selected)	0x2000 | 0x2FFF
Unused	0x3000 | 0xFFFF

Table 5.6: Data Memory Register Files

Data Memory Access Times

The different areas in the Data Memory require different numbers of processor clock cycles in order to access them. These access times are shown in the following table.

Data Memory Access Times	
Data Memory Space	Processor Clocks Per Access
Register File	1
I/O Registers	1
CAN Interface Registers	2
Motor and Motion Control Registers	2
Internal Data RAM	2
External Data RAM	4

Table 5.7: Data Memory Access Times

I/O Register Map

I/O Register Map			
I/O (SRAM) Address	Name	Function	Read/Write
0x00 (0x20)	IPDL	Instruction Program Data Flow	R/W
0x01(0x21)	IPDH	Instruction Program Data High	R/W
0x02 (0x22)	IPAL	Instruction Program Address Low	R/W
0x03 (0x23)	IPAH	Instruction Program Address High	R/W
0x04 (0x24)	IPCR	Instruction Program Control Register	R/W
0x05 (0x25)	ADRSLTLO	A/D Result Low Register	R
0x06 (0x26)	ADRSLTHI	A/D Result Hi Register	R
0x07 (0x27)	ADCTRL	A/D Control Register	R/W
0x08 (0x28)	USPCR	User SPI Control Register	R/W
0x09 (0x29)	USPSR	User SPI Status Register	R/W
0x0A (0x2A)	USPDR	User SPI I/O Data Register	R/W
0x0B (0x2B)	AMUXCTL	Analog MUX Control Register	R/W
0x0C (0x2C)	MSPCR	Master SPI Control Register	R/W
0x0D (0x2D)	MSPSR	Master SPI Status Register	R/W
0x0E (0x2E)	MSPDR	Master SPI I/O Data Register	R/W
0x0F (0x2F)	WDTCR	Watchdog Timer	R/W
0x10 (0x30)	UDR	UART I/O Data Register	R/W
0x11 (0x31)	USR	UART Status Register	R
0x12 (0x32)	UCRA	UART Control Register A	R/W
0x13 (0x33)	UCRB	UART Control Register B	R/W
0x14 (0x34)	UBRRLO	UART Baud Rate Register Low	R/W
0x15 (0x35)	UBRRHI	UART Baud Rate Register High	R/W
0x16 (0x36)	GIFR	General Interrupt Flag Register	R/W
0x17 (0x37)	GIMSK	General Interrupt Mask Register	R/W
0x18 (0x38)	DACVALLO	D/A Conversion Value Low Byte	R/W
0x19 (0x39)	DACVALHI	D/A Conversion Value High Byte	R/W
0x1A (0x3A)	BGPPIN	B GP I/O Input Pins	R
0x1B (0X3B)	BGPDDR	B GP I/O Data Direction Register	R/W
0x1C (0x3C)	BGPPORT	B GP I/O Data Register	R/W
0x1D (0x3D)	AGPPIN	A GP I/O Input Pins	R
0x1E (0x3E)	AGPDDR	A GP I/O Data Direction Register	R/W
0x1F (0x3F)	AGPPORT	A GP I/O Data Register	R/W
0x20 (0x40)	EXTCCR1A	External Timer/Counter 1 Control Register A	R/W
0x21 (0x41)	EXTCCR1B	External Timer/Counter 1 Control Register B	R/W
0x22 (0x42)	EXTCNT1L	External Timer/Counter 1 Low-Byte	R/W
0x23 (0x43)	EXTCNT1H	External Timer/Counter 1 High-Byte	R/W
0x24 (0x44)	EXOCR1AL	External Timer/Counter Output Compare Register A Low-Byte	R/W
0x25 (0x45)	EXOCR1AH	External Timer/Counter Output Compare Register A High-Byte	R/W
0x26 (0x46)	EXOCR1BL	External Timer/Counter Output Compare Register B Low-Byte	R/W
0x27 (0x47)	EXOCR1BH	External Timer/Counter Output Compare Register B High-Byte	R/W
0x28 (0x48)	EXICR1L	External Timer/Counter Input Capture Register Low-Byte	R/W
0x29 (0x49)	EXICR1H	External Timer/Counter Input Capture Register High-Byte	R/W
0x2A (0x4A)	EXTIFR	External Timer/Counter Interrupt Flag Register	R/W
0x2B (0x4B)	EXIMSK	External Timer/Counter Interrupt Mask Register	R/W
0x2C (0x4C)	EXTCNT0	External Timer/Counter 0 (8-bit)	R/W
0x2D (0x4D)	EXTCCR0	External Timer/Counter 0 Control Register	R/W
0x2E (0x4E)	Not Used		
0x2F (0x4F)	Not Used		
0x30 (0x50)	CGPPIN	C GP I/O Input Pins	R
0x31 (0x51)	CGPDDR	C GP I/O Data Direction Register	R/W
0x32 (0x52)	CGPPORT	C GP I/O Data Register	R/W
0x33 (0x53)	MCSR	Micro Controller Status Register	
0x34 (0x54)	Not Used		
0x35 (0x55)	Not Used		
0x36 (0x56)	Not Used		
0x37 (0x57)	IRQADREXT	Interrupt Address Extension Register (Reserved by AVR V3 Core)	
0x38 (0x58)	RAMPD	D Pointer Extension Register (Reserved by AVR V3 Core)	
0x39 (0x59)	RAMPX	X Pointer Extension Register (Reserved by AVR V3 Core)	
0x3A (0x5A)	RAMPY	Y Pointer Extension Register (Reserved by AVR V3 Core)	
0x3B (0x5B)	RAMPZ	Z Pointer Extension Register (Reserved by AVR V3 Core)	
0x3C (0x5C)	EIND	Indir call/jmp Extension Register (Reserved by AVR V3 Core)	
0x3D (0x5D)	SPL	Stack Pointer Low	R/W
0x3E (0x5E)	SPH	Stack Pointer High	R/W
0x3F (0x5F)	SREG	Status Register	R/W

Table 5.8: I/O Register Map

Internal Timer Register Map

Internal Timer Register Map			
Data Memory Address	Name	Function	Read/Write
0x800	INTCCR1A	Internal Timer/Counter 1 Control Register A	R/W
0x801	INTCCR1B	Internal Timer/Counter 1 Control Register B	R/W
0x802	INTCNT1L	Internal Timer/Counter 1 Low-Byte	R/W
0x803	INTCNT1H	Internal Timer/Counter 1 High-Byte	R/W
0x804	INOCR1AL	Internal Timer/Counter 1 Output Compare Register A Low-Byte	R/W
0x805	INOCR1AH	Internal Timer/Counter 1 Output Compare Register A High-Byte	R/W
0x806	INOCR1BL	Internal Timer/Counter 1 Output Compare Register B Low-Byte	R/W
0x807	INOCR1BH	Internal Timer/Counter 1 Output Compare Register B High-Byte	R/W
0x808	INTCNT0	Internal Timer/Counter 0 (8-bit)	R/W
0x809	INTCCR0	Internal Timer/Counter 0 Control Register	R/W
0x80A	INTIFR	Internal Timer/Counter Interrupt Flag Register	R/W
0x80B	INTIMSK	Internal Timer/Counter Interrupt Mask Register	R/W

Table 5.9: Internal Timer Register Map

Peripheral Connections

Connections of the External Timer block for input capture and external timer clocking are implemented similar to the Atmel mega163 part. The T0CLK, and T1CLK pins go through a two 20 MHz clock cycle demetastabilization and are then passed through an edge detection circuit before being forwarded on to the External Timer block. The ICPO pin also passes through a two 20 MHz clock cycle demetastabilization. This implementation requires that a pulse for input capture and the period of a timer0 or timer1 external clock must be at least two 20MHz clock cycles in duration.

External RAM Interface

The External RAM Interface provides a facility for reading and writing to external RAM. Reads and writes are normal AVR data memory reads and writes with two wait states added. Address is driven out first with a half-cycle ALE signal to allow an external latch to capture the address. Data is subsequently driven out on the next clock on a write or is expected at the end of the next clock cycle on a read. A timing diagram describing this function is in the figure below. (Not available on all packages).

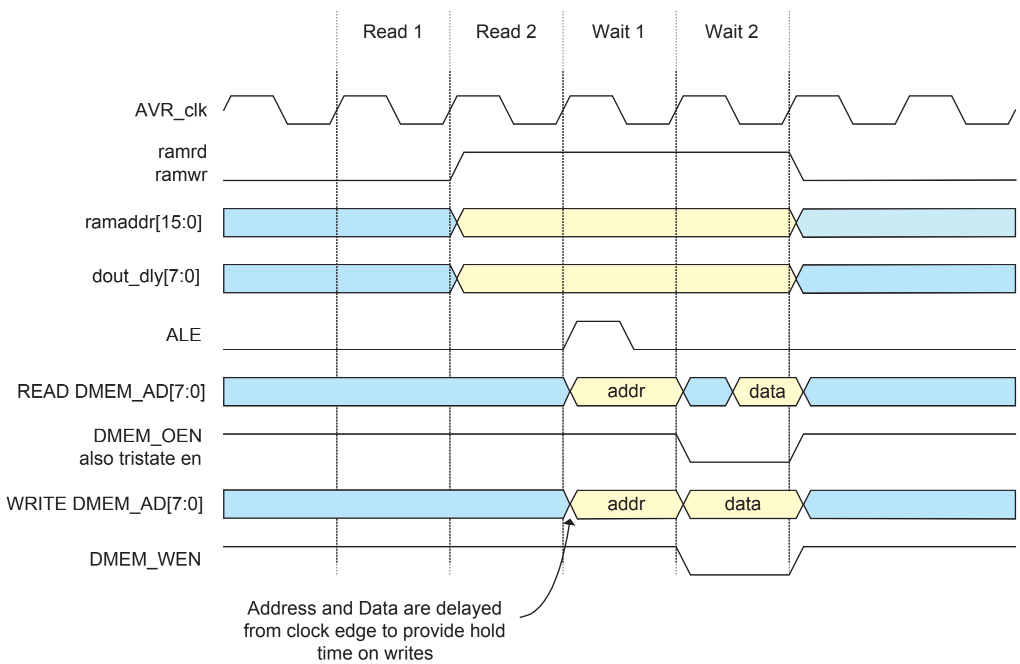


Figure 5.4: External Ram Interface

Page Intentionally Left Blank

SECTION 6
GENERAL PURPOSE I/O

Introduction

NOTE

NOTE: Port C I/O is not available on all packages. Dual function pins may be used for external ram.

The General Purpose I/O consists of 9 registers that drive the 12 + 8* GPI/O pins. The registers consist of three sets of the following registers: a data-direction register, a data (output) register, and a pin (input) register. Each set of three registers provides programmability for the corresponding 4 or 2x8 GPI/O pins. Each pin can be individually programmed to be an input, an input pulled up, or an output. A pin is programmed to be an input by driving the corresponding bit in the data-direction register (AGPDDR, BGPDDR or CGPDDR) to zero. An enabled pull-up resistor can be added by writing a logic one to the corresponding bit in the data register (AGPPORT, BGPPORT or CGPPORT) while the data direction register bit remains zero. A pin becomes an output when the corresponding bit in the data direction register is set to logic one.

General Purpose I/O Registers

AGPPORT (General Purpose I/O Data Register A)

Bit	7	6	5	4	3	2	1	0	
0x1F(0x3F)	GPIO[7:0]								AGPPORT
Read/Write	R/W								
Initial Value	1111_1111								

Table 6.1: General Purpose I/O Data Register A

- **Bits 7..0 – GPDOA[7:0]: General Purpose I/O Data Out A [7:0]**

This represents the value driven onto the GPIO[7:0] pins when the corresponding bit in AGPDDR[7:0] is logic 1. When any of the bits in AGPDDR[7:0] are logic zero, a corresponding logic 1 in GPDOA[7:0] will cause the corresponding GPIO[7:0] pin to be pulled up.

AGPDDR (General Purpose I/O Data Direction Register A)

Bit	7	6	5	4	3	2	1	0	
0x1E(0x3E)	GPIO[7:0]								AGPDDR
Read/Write	R/W								
Initial Value	0000_0000								

Table 6.2: General Purpose I/O Data Direction Register A

- **Bits 7..0 – GPDDRA[7:0]: General Purpose I/O Data Direction A [7:0]**

Logic 1 in this value enables a corresponding bit in AGPPORT[7:0] to be driven onto the GPIO[7:0] pins. When bits in GPDDRA[7:0] are logic zeros, a corresponding logic one in GPDO[7:0] will cause the corresponding GPIO[7:0] pin to be pulled up.

AGPPIN (General Purpose I/O Input Pins A)

Bit	7	6	5	4	3	2	1	0	
0x1D(0x3D)	GPIO[7:0]								AGPPIN
Read/Write	R								
Initial Value	1	1	1	1	1	1	1	1	

Table 6.3: General Purpose I/O Input Pins Register A

- **Bits 7..0 – GPINA[7:0]: General Purpose I/O Input Pins A [7:0]**

This value is the demetastabilized value from the GPIO[7:0] pins.

BGPPORT (General Purpose I/O Data Register B)

Bit	7	6	5	4	3	2	1	0	
0x1C(0x3C)	UCSN	–	SELMEM	SEL4K	GPIO[11:8]				BGPPORT
Read/Write	R/W	R/W	R/W	R/W	R/W				
Initial Value	0	0	0	0	1111				

Table 6.4: General Purpose I/O Data Register B

- **Bit 7 – UCSN: SPI User Chip Select**

  This bit controls the SPI_UCSN pin when the User SPI is in master mode. When this bit is zero, and the USER_SPI is in master mode, the SPI_UCSN pin will be driven with a logic zero.

  (See Section 20 - Serial Peripheral Interface for more information.)

- **Bit 6 – Reserved**
- **Bit 5 – SELMEM: Select External Memory Function for Dual Purpose I/O**

  This bit selects the external memory function for the dual purpose GPIOx /DMEM_Ax pins. When set, some or all pins are used for memory (DMEM_Ax). When cleared, all pins are used for GPIO (GPIOx). Accessing external memory is not possible in all packages.

- **Bit 4 – SEL4K: Select Address Range When External Memory Is Used**

  This bit selects the address range when external memory is used. This bit also affects how many dual purpose GPIO / DMEM_Ax pins are used for memory and GPIO. When set, 4096 bytes are addressable and all dual pins are used for memory. When cleared, 256 bytes are addressable, 8 dual purpose pins are used for memory and 4 dual purpose pins are used for GPIO.

- **Bits 3..0 – GPDOB[3:0]: General Purpose I/O Data Out B [11:8]**

  Represents the value driven onto the GPIO[11:8] pins when the corresponding bit in BGPDDR[3:0] is logic 1. When any of the bits in BGPDDR[3:0] are logic zeros, a corresponding logic 1in GPDOB[3:0] will cause the corresponding GPIO[11:8] pin to be pulled up.

<table>
<tr><th colspan="8">External Memory Function Usage</th></tr>
<tr><th colspan="2">Bits</th><th colspan="3">External Memory</th><th colspan="3">General Purpose I/O</th></tr>
<tr><th>SELMEM</th><th>SEL4K</th><th>Addressable Bytes</th><th>Dual Purpose Pins Used</th><th>Dedicated Pins</th><th>GPIO Available</th><th>Dual Purpose Pins Used</th><th>Dedicated Pins</th></tr>
<tr><td>0</td><td>X</td><td>0</td><td>None</td><td>N/A</td><td>20</td><td>GPIO [19...8] Port C, Port B</td><td rowspan="3">GPIO [7...0] Port A</td></tr>
<tr><td>1</td><td>0</td><td>256</td><td>DMEM_A[7...0]</td><td rowspan="2">DMEM_WEN, CEN, OEN, ALE</td><td>12</td><td>GPIO [11...8] Port B</td></tr>
<tr><td>1</td><td>1</td><td>4096</td><td>DMEM_A[11...8] DMEM_AD[7...0</td><td>8</td><td>None</td></tr>
</table>

Table 6.5: External Memory Function Usage

BGPDDR (General Purpose I/O Data Direction Register B)

Bit	7	6	5	4	3	2	1	0	
0x1B(0x3B)	–	–	–	–	GPIO[11:8]				BGPDDR
Read/Write	R	R	R	R	R/W				
Initial Value	0	0	0	0	0000				

Table 6.6: General Purpose I/O Data Direction Register B

- **Bit 7..4 Reserved**

  These bits are reserved and always read as zero.

- **Bits 3..0 – GPDDRB[3:0]: General Purpose I/O Data Direction B [7:0]**

  Logic 1 in this value enables a corresponding bit in BGPPORT[3:0] to be driven onto the GPIO[11:8] pins. When bits in GPDDRB[3:0] are logic zeros, a corresponding logic one in GPDO[3:0] will cause the corresponding GPIO[11:8] pin to be pulled up.

BGPPIN (General Purpose I/O Input Pins B)

Bit	7	6	5	4	3	2	1	0	
0x1A(0x3A)	–	–	–	–	GPI0[11:8]				BGPPIN
Read/Write	R	R	R	R	R/W				
Initial Value	0	0	0	0	0000				

Table 6.7: General Purpose I/O Input Pins Register B

- **Bit 7..4 Reserved**

These bits are reserved and always read as zero.

- **Bits 7..0 – GPINB[3:0]: General Purpose I/O Input Pins B [3:0]**

This value represents the demetastabilized value read from the GPIO[11:8] pins.

CGPPORT (General Purpose I/O Data Register C)

Bit	7	6	5	4	3	2	1	0	
0x32(0x52)	GPIO[19:12]								CGPPORT
Read/Write	R/W								
Initial Value	1111_1111								

Table 6.8: General Purpose I/O Data Register C

- **Bits 7..0 – GPDOC[7:0]: General Purpose I/O Data Out C [7:0]**

This represents the value driven onto the GPIO[19:12] pins when the corresponding bit in CGPDDR[7:0] is logic 1. When any of the bits in CGPDDR[7:0] are logic zero, a corresponding logic 1 in GPDOC[7:0] will cause the corresponding GPIO[19:12] pin to be pulled up.

CGPDDR (General Purpose I/O Data Direction Register C)

Bit	7	6	5	4	3	2	1	0	
0x31(0x51)	GPIO[19:12]								CGPDDR
Read/Write	R/W								
Initial Value	0000_0000								

Table 6.9: General Purpose I/O Data Direction Register C

- **Bits 7..0 – GPDDRC[7:0]: General Purpose I/O Data Direction C [7:0]**

Logic 1 in this value enables a corresponding bit in CGPPORT[7:0] to be driven onto the GPIO[19:12] pins. When bits in GPDDRC[7:0] are logic zeros, a corresponding logic one in GPDO[7:0] will cause the corresponding GPIO[19:12] pin to be pulled up.

CGPPIN (General Purpose I/O Input Pins C)

Bit	7	6	5	4	3	2	1	0	
0x30(0x50)	GPIO[19:12]								CGPPIN
Read/Write	R								
Initial Value	1	1	1	1	1	1	1	1	

Table 6.10: General Purpose I/O Input Pins Register C

- **Bits 7..0 – GPINC[7:0]: General Purpose I/O Input Pins C [7:0]**

This value is the demetastabilized value from the GPIO[19:12] pins.

I/O Ports

All M3000 ports have true Read-Modify-Write functionality when used as general digital I/O ports. This means that the direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions.

The same applies when enabling/disabling of pull-up resistors (if configured as input). Each output buffer has symmetrical drive characteristics. All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance.

Three I/O memory address locations are allocated for each port, one each for the Data Register – xGPPORT, Data Direction Register – xGPDDR, and the Port Input Pins – xGPPIN. The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write.

Ports as General Digital I/O

General Digital I/O

NOTE: WPx, WDx, RLx, RPx, and RDx are common to all pins within the same port.

The ports are bidirectional I/O ports with optional internal pull-ups. Figure 6.1 shows a functional description of one I/O-port pin, here generically called GPIOx.

Each port pin consists of three register bits: GPDDRxn, GPDOxn and GPINxn. The GPDDRxn bits are accessed at the xGPDDR I/O address, the GPDOxn bits at the xGPPORT I/O address, and the GPINxn bits at the xGPPIN I/O address.

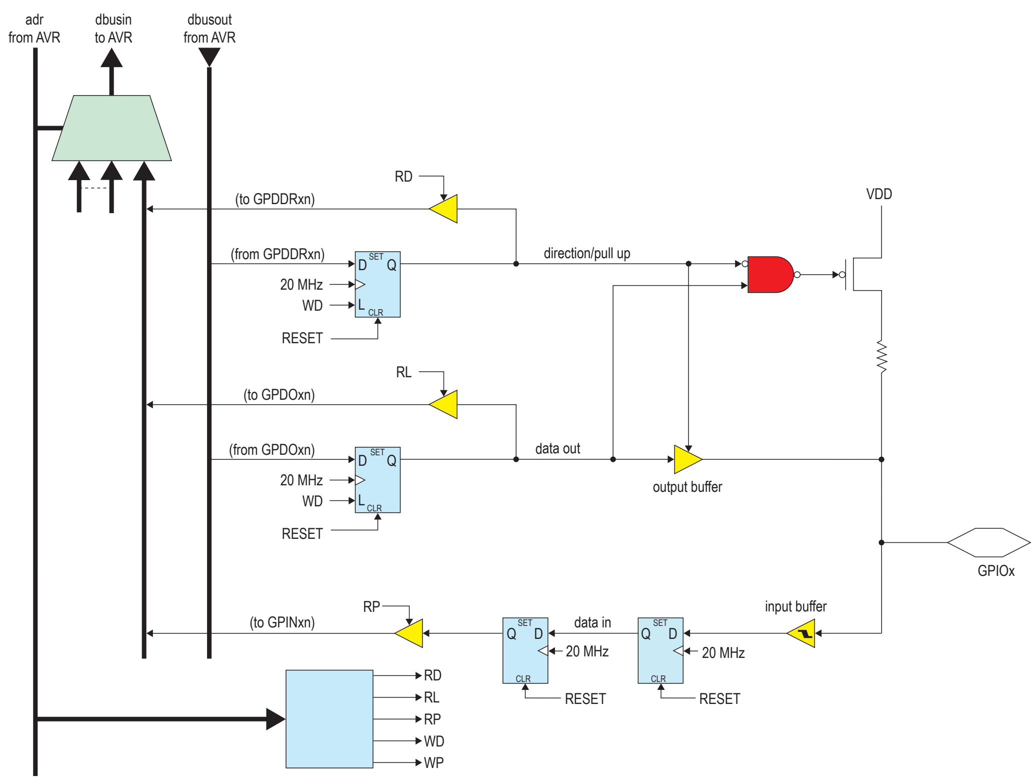


Figure 6.1: General Digital I/O Block Diagram

The GPDDRxn bit in the xGPDDR Register selects the direction of this pin. If GPDDRxn is written logic one, GPIOx is configured as an output pin. If GPDDRxn is written logic zero, GPIOx is configured as an input pin.

If GPDOxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, GPDOxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when a reset condition becomes active, even if no clocks are running.

If GPDOxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If GPDOxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

Port Pin Configurations

When switching between tri-state ({GPDDRxn, GPDOxn} = 0b00) and output high ({GPDDRxn, GPDOxn} = 0b11), an intermediate state with either pull-up enabled ({GPDDRxn, GPDOxn} = 0b01) or output low ({GPDDRxn, GPDOxn} = 0b10) must occur. Normally the pull-up enabled state is fully acceptable, as a high-impedance environment will not notice the difference between a strong high driver and a pull-up.

Switching between input with pull-up and output low generates the same problem. The user must use either the tri-state ({GPDDRxn, GPDOxn} = 0b00) or the output high state ({GPDDRxn, GPDOxn} = 0b11) as an intermediate step.

Table 6.11 summarizes the control signals for the pin value.

I/O Port Pin Configurations				
GPDDRxn	GPDOxn	I/O	Pull-up	Comment
0	0	Input	No	Tri-state (Hi-Z)
0	1	Input	Yes	GPIOx will source current if ext. pulled low
1	0	Output	No	Output Low (Sink)
1	1	Output	No	Output High (Source)

Table 6.11: I/O Port/Pin Configurations

Independent of the setting of Data Direction bit GPDDRxn, the port pin can be read through the GPINxn Register bit. The GPINxn Registers constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay. The figure below shows a timing diagram of the synchronization when reading an externally applied pin value. The maximum and minimum propagation delays are denoted tpd,max and tpd,min respectively.

Synchronization when Reading an Externally Applied Pin Value

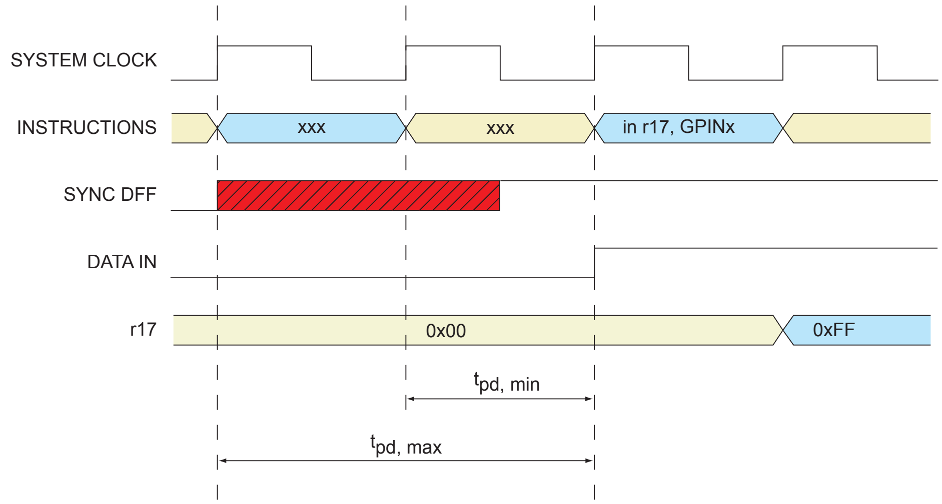


Figure 6.2: Synchronization of an Externally Applied Pin Value

When reading back a software assigned pin value, a nop instruction must be inserted as indicated in the following figure. The out instruction sets the "SYNC LATCH" signal at the positive edge of the clock. In this case, the delay t_{pd} through the synchronizer is one system clock period.

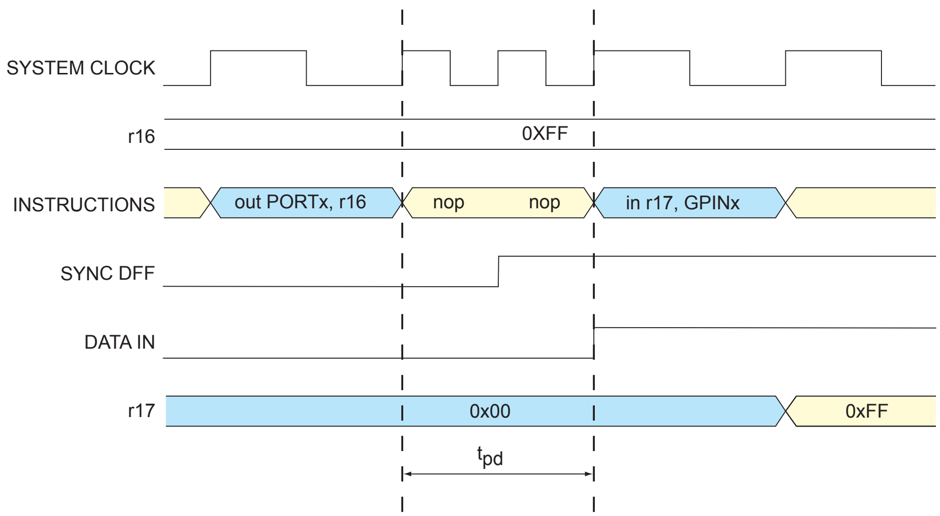


Figure 6.3: Setting the SYNC LATCH

I/O Port Code Examples

The following code examples shows how to set port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7. The resulting pin values are read back again, but as previously discussed, a nop instruction is included to be able to read back the value recently assigned to some of the pins.

C Code Example

```
unsigned char          i;
***
    /* Define pull-ups and set outputs high */
    /* Define directions for port pins */
            PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
            DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
    /* Insert nop for synchronization */
            _NOP ( );
            _NOP ( );
    /* Read port pins */
            i = PINB;
***
```

Page Intentionally Left Blank

SECTION 7
MOTOR AND MOTION CONTROL INTERFACE

Introduction

The AVR core communicates with the Motor and Motion Control Logic through register reads, register writes, and interrupts. Register accesses are memory mapped to Data SRAM address space and take two processor cycles to complete.

Function Blocks

The M3000 Motor and Motion Control Logic is comprised of a comprehensive group of Function Blocks. They are:

- Input/Output Clock Conversion
- Sine/Cosine Generator
- 10-Bit Sine D/A
- 10-Bit Cosine D/A
- Reference D/A
- Acceleration and Velocity Generator
- Data Registers
- 32-Bit Position Counter
- Encoder Interface
- 32 Bit Encoder Counter
- Dual H-Bridge Control
- Phase Current Control with AntiResonance
- 32-Bit Position Compare
- 32-Bit High Speed Position Capture

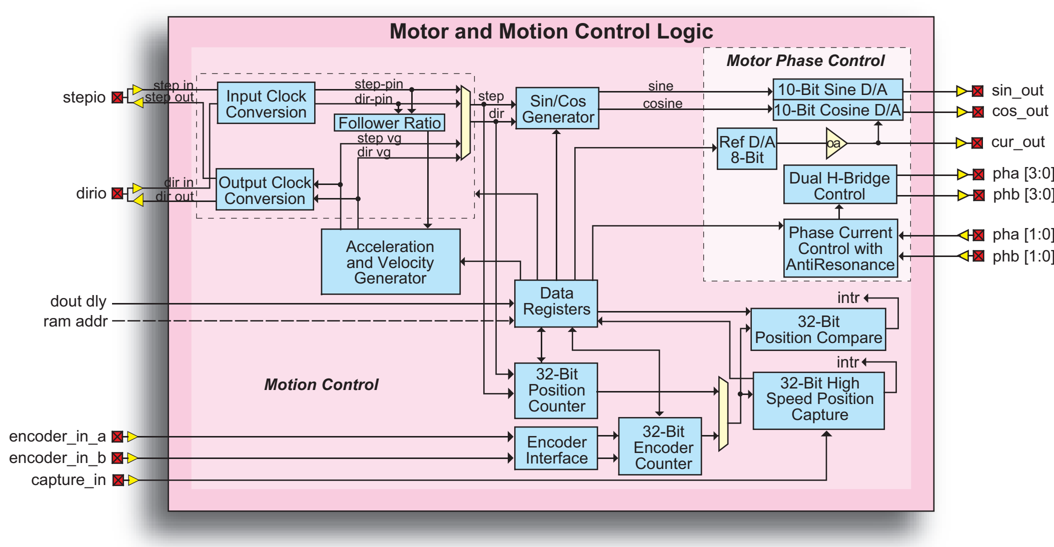


Figure 7.1: Motor and Motion Control Logic Interface

Description of the Motor and Motion Control Function Blocks

Data Registers

NOTE: In the following Text and the Figures: Registers are ALL CAPS and UNDERLINED. Bits are bold_faced. I/O Pins are italicized.

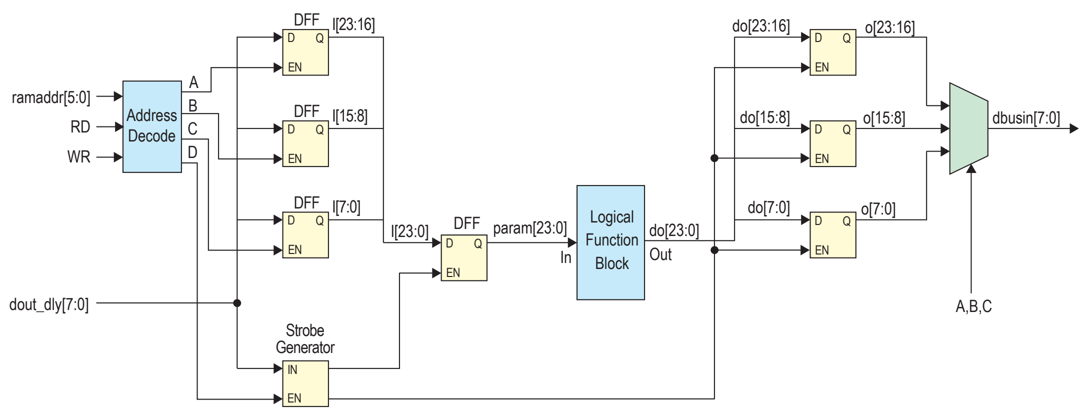
The Motor and Motion Control Logic is treated as a memory mapped device in the system, also many of the parameters required within the Motor and Motion Control Logic are multibyte. This block provides the address decoding and the means to read and write multi-byte parameters.

Some multibyte parameters automatically create a strobe when the MSB is written to transfer the full parameter into the logic, CURRDLY for example. Others are transferred by writing a strobe bit into a register, for example writing the self clearing bit trp_targ_s in register IDXSTRB transfers the parameter IDXTRT to the indexer.

To read multibyte registers, the strobe method is used. The full value is captured when the strobe is created. The individual bytes that make up the parameter can be read at any time. For example, writing the self clearing bit curvel in register VELSTB would capture the full three (3) byte value of the parameter VELCVEL.

There are five (5) types of registers;

- Write, static: primarily used to write a value, a read returns last value written
- Write, dynamic: used to set a value, a read returns value modified by events
- Write, self clear: used to generate strobes, cleared automatically, reads are undefined
- Read, dynamic: read a value modified externally, writes have no effect
- Read, write to clear: read flags, write a “1” to clear



M3000 Multibyte Register Read/Write Logic

Figure 7.2: Typical M3000 Multibyte Register Read/Write Logic Diagram

Input/Output Clock Conversion

This block performs conversion for incoming and outgoing step clocks along with computing ratio parameters when the ASIC is used in the ‘following’ mode. The output of this block goes to the Sin/Cosine generator and to the step position counter of the Indexer.

When the ASIC is used as a drive, clearing the bit **vg_pinn** in register SCIO makes bidirectional pins *step_io* and *dir_io* inputs. The bits **sel_clk[1..0]** set the conversion type to; step and direction, quadrature, or step up / down.

When the input clock type is Step_Up/Dn, clocks on Pin *Step_IO* (Step_Up) will set the internal signal dir. Clocks on pin *DIR_IO* (Step Dn) will clear dir.

When the clock type is Quadrature, *Step_IO* (CHA) leading *DIR_IO* (CHB) will set the internal signal dir. *DIR_IO* (CHB) leading *Step_IO* (CHA) will clear dir.

The step_io and dir_io inputs may be filtered using bits **IOF[3..0]** in register <u>IOF</u>. The minimum detectable pulse width ranges from 50 nS to 12.9 µS. The polarity of step_io and dir_io may be inverted using bits **inv_stp** and **inv_dir** in register SCIO.

When the ratio factor in register SCRF equates to a ratio of one (1), the filtered, converted clock and direction information from the input pins becomes the output of the block.

When the ASIC is used as an indexer or oscillator, setting the bit **vg_pinn** in register SCIO makes bidirectional pins *step_io* and *dir_io* outputs. The bits **sel_clk[1..0]** select what the output from *step_io* and *dir_io* will be; step and direction, quadrature or step up / down. The value in register SCSW determines the width of the step outputs (except quadrature). The bit en_sdo in register SCIO enables the pin output buffers when set. The outputs are high impedance when cleared.

When the output clock type is Step Up/Dn, steps will output on Pin *STEP_IO* (Step Up) when the internal signal dir is set. Steps will output on Pin *DIR_IO* (Step Dn) when dir is cleared.

When the output clock type is quadrature, *Step_IO* (CHA) will lead *DIR_IO* (CHB) when the internal signal dir is set. *DIR_IO* (CHB) will lead *STEP_IO* (CHA) when dir is cleared.

A square wave may be selected for step output by setting bit **sq_sel** in register SCIO. Note the last step of an indexed move will be the width specified in register SCSW.

When used as an indexer or oscillator the output of the velocity generator is the output of the block.

When used in ratio following mode, the ASIC can respond to incoming step clocks with a ratio from 0.001 to 2. Example; When the ratio = 0.001 and the incoming step rate is 10,000 step/sec, the ASIC will generate steps at 10 steps/sec. When the ratio is 2.0, the ASIC will generate steps at 20,000 steps/sec. The ratio factor is set in SCRF. When the ratio factor is not equal to one (1), a velocity parameter is computed based on the ratio factor and the measured period of each incoming step clock. The computed velocity parameter is used by the velocity generator to create step clocks which are routed to the sine generator. Direction information, either directly from the *dir_io* pin or as determined by the clock conversion logic, is also sent to the velocity generator.

Step and Direction Timing

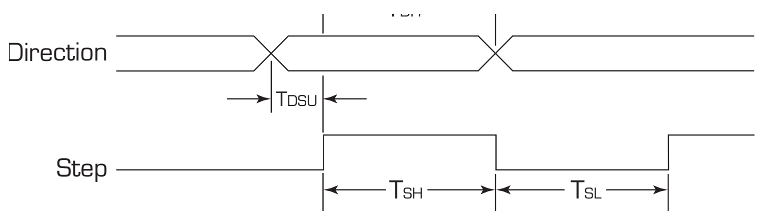


Figure 7.3: Step and Direction Timing Diagram

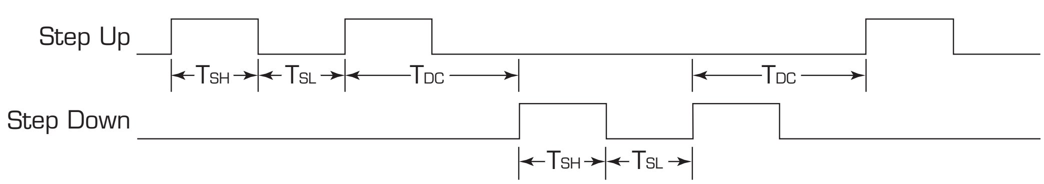


Figure 7.4: Step Up / Step Down Timing Diagram

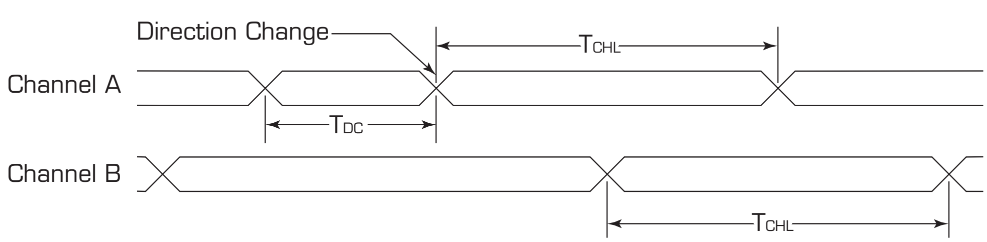


Figure 7.5: Quadrature Timing Diagram

Setup and Direction Timing					
Symbol	Parameter	Type and Value			
		Step and Direction	Step Up / Step Down	Quadrature	Units
T_{DSU}	T Direction Set Up	0	N/A	N/A	nS Min
T_{DH}	T Direction Hold	50	N/A	N/A	nS Min
T_{SH}	T Step High	50	100	N/A	nS Min
T_{SL}	T Step Low	50	100	N/A	nS Min
T_{DC}	T Direction Change	N/A	200	200	nS Min
T_{CHL}	T Channel High/Low	N/A	N/A	400	nS Min
F_{SMAX}	F Step Maximum	5	5	N/A	MHz Max
F_{CHMAX}	F Channel Maximum	N/A	N/A	1.25	MHz Max
F_{ER}	F Edge Rate	N/A	N/A	4	MHz Max

Table 7.1: Step & Direction, Step Up/Step Down, Quadrature Timing Table

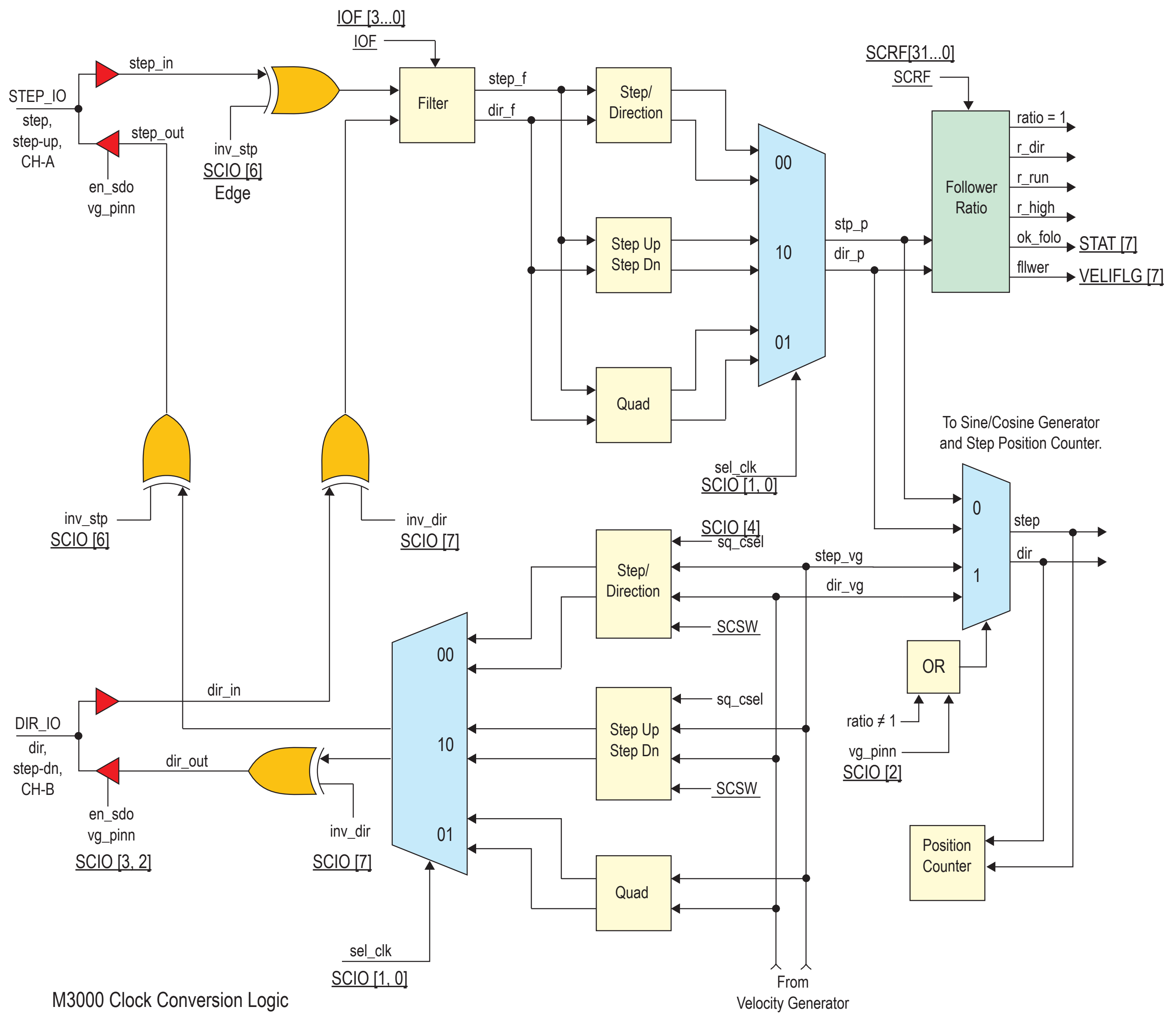


Figure 7.6: M3000 Clock Conversion Logic Diagram

The velocity generator uses the computed velocity parameter instead of the VELHI parameter for the terminal (high) velocity. The VELLOW, VELDEC and VELACC parameters are used normally. The direction information as previously described, is used instead of bit **dirmtn** in register VELVGCTL. Writing a 00 to the register, VELVGCTL creates a register valid signal allowing the velocity generator to operate.

Before following can begin, at least 2 step clocks must be input so a period measurement can be made and the ratio factor must be entered. When those conditions are met the bit **Ok_folo** in register STAT will assert. When the ratio in SCRF is not equal to one (1), the **Ok_folo** bit will automatically “run” the velocity generator. If the step period exceeds the maximum measurable or if the computed velocity parameter is greater than maximum, the velocity generator will be stopped and the bit **fllwer** in register VELIFLG will assert if enabled by bit **fllwer** in register VELIMSK.

Following Parameters			
Parameter	Max.	Min.	Notes
Input Step Frequency	5 MHz	—	Ratio less than 1. Clock conversion limit.
	—	1.193 Hz	Ratio Greater than 0.5. Period counter limit.
Computed Velocity Generator Frequency	5 MHz	0.597 Hz	Above maximum follow error, below minimum, no VG output.

Table 7.2: Following Parameters

Sine/Cosine Generator

The sine / cosine generator creates sine and cosine values that go to the D/A converters. The generator values change in response to the step and direction outputs of the clock conversion block. Sine and cosine current is proportional to a set peak current which allows positioning the step motor between natural motor positions. This is microstepping.

When a step occurs, an address generator increments an amount determined by the bits **MSEL[4:0]** in register MSELR which is the micro step resolution. There are twenty (20) micro step resolutions including values for degrees (180), metric (127), and arc minutes (108). The output of the address generator is used as a pointer into a lookup table. The sine and cosine values are computed based on the lookup value sequentially, then updated simultaneously. The sine and cosine are repeatedly computed regardless of step activity. The update speed is selected using bit **spd_intf** in register MSELR.

MSEL Values can be changed at any time. The new resolution will take effect with the step after the change. There is an exception when switching to Whole Step. The change to Whole Step will not take effect until the phases are at the 0.707 (45°) position in the Sine/Cosine Table. MSEL can not be changed to Whole Step from resolutions $^1/_5$, $^1/_{25}$, $^1/_{125}$ or $^1/_{127}$ because there is no 0.707 (45°) position.

Normally, the sine phase is initialized by reset to 0 (0 deg.) and the cosine phase to 1 (90 deg.). When the bit **init_707** in register MSELR is set each phase will initialize to the 0.707 (45 deg.) position. If desired to initialize to 0.707, the register MSELR should be written before register PWMCTL.

When initializing to 0.707 and selecting slow update, the phases will not change to 0.707 ± 1 step until the first step.

Two (2) interfaces are available for external serial D/A converters. Use bits **en_ser_dac** and **spd_intf** in register MSELR to enable and select a serial D/A interface. The reference for the external D/As is output in a PWM format on Pin *PWM_CUR*. The least significant 7 bits of Register CURIRUN or CURIRED are used. The basic PWM period is 12.8 µs. The serial outputs for the external D/A converters are not available in all packages.

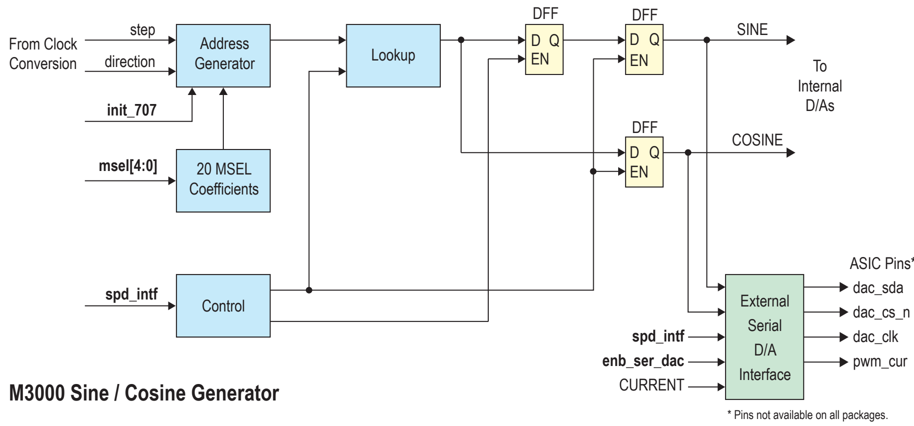


Figure 7.7: M3000 Sine/Cosine Generator Diagram

Internal 10-Bit Sine/Cosine D/As

These D/As convert the sine / cosine values to voltages on pins sin_out and cos_out which are used as set points for the motor phase currents. The sine output is used with step motor phase A. The cosine is used with step motor phase B.

Normally the values for conversion are supplied by the Sine / Cosine generator in response to incoming step clocks. The values for conversion for other purposes can be input to registers SINDACL, SINDACH, COSDACL and COSDACH after setting bits EXSCG and GNEN in register DACCTRL.

Reference D/A

This D/A sets the reference for the Sine / Cosine D/As. This output value defines the peak sine / cosine current. This D/A normally converts one of two values, the run current in register CURIRUN or reduction current in register CURIRED.

The run current value is selected when the step clock is active. A count down timer is loaded with a reduction delay time value in milliseconds in register CURRDLY when each step clock occurs. If the timer counts to zero before a step clock reloads the timer, the reduction current value is selected. When using reduction current, the next step clock will re-select the run current value and re-load the counter.

If the reduction delay time in register CURRDLY is set to zero, reduction current will never be selected. If reduction current is set to 0 (zero) in register CURIRED, the phase bridges are disabled during reduction.

The bit curired in register STAT is set when current reduction is active.

After a reset, the run current value is selected with reduction delay time at maximum. The timer is loaded with the reduction delay time in register CURRDLY when the msb is written and with each step clock. The value for conversion for other purposes can be input to register GAINDAC after setting bits **EXSCG** and **GNEN** in register DACCTRL.

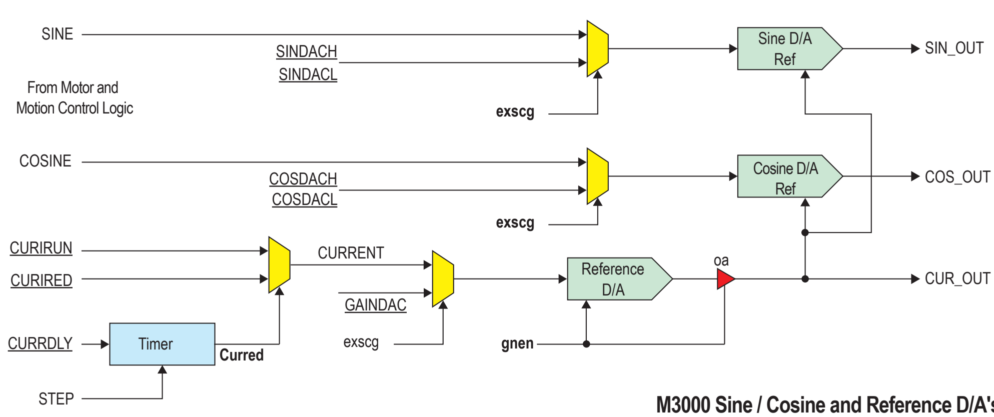


Figure 7.8: M3000 Sine/Cosine and Reference D/A's Diagram

Acceleration and Velocity Generator

The velocity generator creates step clock and direction outputs when the ASIC is used as an oscillator or indexer. It is also used when the asic is used as a drive in a following mode. The outputs are directed to the Clock in / out conversion block. After the parameters are set up, the operation of the velocity generator requires only the **runmtn, dirmtn** and **abortmtn** bits in register VELVGCTL be manipulated as required.

The velocity generator is selected for use as an oscillator or indexer by setting bit **vg_pinn** in register SCIO.

When the ASIC is used as an oscillator or indexer the step clock and direction outputs from the velocity generator are available on step_io and dir_io pins. The type of clock output is specified by the bits **sel_clk[1..0]** in register SCIO. In the case of output types step / direction and step up / down, the width of the pulses is set in register SCSW. The bit **en_sdo** in register SCIO enables the pin output buffers.

The velocity generator exhibits the following behavior under the condition specified:

NOTE:For proper operation, do not change direction and then change the high velocity before the direction change is completed.

Velocity Generator Behavior	
Condition	Response
Run From Stopped	Accelerate from low velocity to high velocity, run at high velocity.
Stop (runmtn de-asserted) from high velocity	Decelerate to low velocity, Stop
Stop (runmtn de-asserted) before high velocity achieved	Decelerate from current velocity to low velocity, Stop.
Runmtn reasserted before deceleration complete	Not recommended. Decelerate from high velocity to low velocity, Run at high velocity.
Direction change while running	Change direction output, Accelerate from low velocity to high velocity Run at high velocity.
Abort	Immediate stop. Note, the run input must be toggled to run again.
Low velocity = high velocity	Slew at velocity with no decel or accel.
Low velocity > high velocity	Slew at high velocity with no accel or decel.
High velocity change while moving	Accel or decel as required from current velocity to reach new velocity.

Table 7.3: Velocity Generator Behavior

NOTE: For proper operation, do not set the deceleration parameter to zero (0) as a means of preventing velocity change.

The following registers contain the parameters required for velocity generator operation;

- VELLOW contains the initial (low) velocity.
- VELHI contains the terminal (high) velocity.
- VELDEC contains the deceleration rate.
- VELACC contains the acceleration rate.

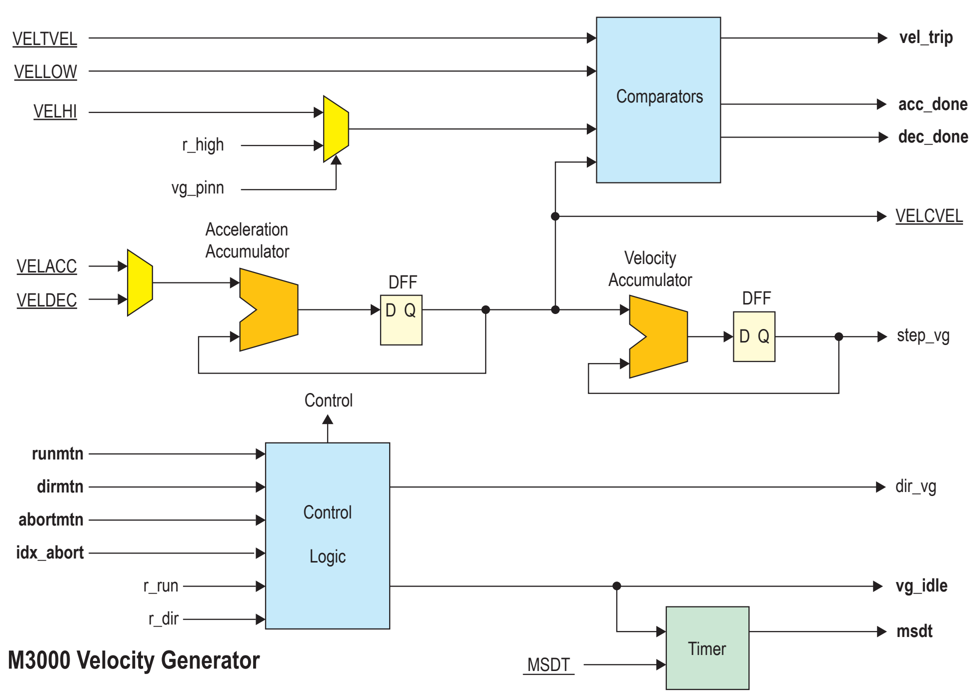


Figure 7.9: M3000 Acceleration and Velocity Generator Diagram

Note: The velocity generator will be aborted immediately if the end position target is past the current position when written.

The above multibyte values are written to the velocity generator when the appropriate self clearing bits in register VELSTB are written.

The register VELVGCTL contains the bits **abortmtn, runmtn** and **dirmtn** that control the operation of the velocity generator as described above. Notes concerning the reading of the **runmt**n and **dirmtn** bits. When ratio following, bits **runmtn** and **dirmtn** reflect the 'following' condition when read, when coordinating with indexer, bit **runmtn** clears when deceleration begins. Writing to register VELVGCTL creates a register valid signal allowing the velocity generator to operate.

A velocity trip comparator is available. When the current velocity matches the value set in register VELTVEL, an interrupt bit **vel_trip** in register VELIFLG will be set if enabled by setting bit **vel_trip** in register VELMSK. The register VELTVEL is written when the self clearing bit **trip_vel** is written in register VELSTB. Note the velocity trip value is not direction dependent.

The current velocity may be captured for read at any time by writing the self clearing bit **curvel** in register VELSTB then reading the VELCVEL registers.

When acceleration or deceleration is complete an interrupt will occur and interrupt bits **acc_done** or **dec_done** will be set in register VELIFLG if the bits with the corresponding name in register VELIMSK are set.

The bit **vg_idle** in register STAT will be set when the velocity generator is idle.

Indexer

The Indexer or Motion Controller is comprised of several blocks. They are:

- 32-bit Position Counter
- 32-bit Encoder Counter
- The Encoder Interface
- 32-bit Position Capture
- 32-bit Position Comparators

The indexer monitors step motor position using 2 counters to count the step clocks produced by the velocity generator and / or the counts produced by an external quadrature encoder if used. Three (3) comparators are available to generate interrupts at position count values of interest. The bit **cmp_sel** in register IDXCTRL chooses the position counter (step or encoder) the comparators monitor.

The trip position comparator is typically used to create a high speed output signal, trip_out which is asserted when the selected count matches the **trp_targ_s** written to registers IDXTRT. The multi-byte value is written to the indexer when the self clearing bit **trp_targ_s** in register IDXSTRB is written. The width of the *trip_out* signal is set in register SCSW. A trip event will also generate an interrupt flag trip, that can be read from register IDXIFLG if enabled by bit **trip** in register IDXIMSK.

The position target comparator is typically used to indicate the position to begin deceleration for a single move (or last in a series of moves) or to indicate the position to change velocity for a blended move. The position target is written to registers IDXPOT. The multi-byte value is written to the indexer when the self clearing bit **pos_targ_s** in register IDXSTRB is written. A position target match will generate an interrupt flag, **pos_targ_s**, that can be read from register IDXIFLG if enabled by bit **pos_targ_s** in register IDXIMSK.

NOTE: For proper operation, do not assert idx_abort during deceleration and then de-assert it during the period between decel complete and end target reached.

The operation of the velocity generator can be automated, if the bit **idx_abort** is set in register IDXCTRL, deceleration will begin automatically when the position target is reached. Note, deceleration will begin immediately if the position target is past the current position. Definition of "past" is direction dependent: When direction (DIRMTN) is 1, position > target, when direction is 0, position < target.

The end position target comparator is typically used to indicate that a move or a series of moves is complete. The end position target is written to registers IDXENT. The multi-byte value is written to the indexer when the self clearing bit **end_targ_s** in register IDXSTRB is written. An end position target match will generate an interrupt flag **end_targ_s** that can be read from register IDXIFLG if enabled by bit **end_targ_s** in register IDXIMSK.

A high speed capture register is available that is typically used to capture the selected count, step or encoder, on the rising edge of the input *capture_in*. The input *capture_in* will generate an interrupt flag, **capture**, that can be read from register IDXIFLG if enabled by bit **capture** in register IDXIMSK. The multibyte value is read from the indexer when the self clearing bit **capture_s** in register IDXSTRB is written. The captured count value can be read from registers IDXCPTP.

The *capture_in* input may be filtered using bits **IOF[7..4]** in register IOF. The minimum detectable pulse width ranges from 50 nS to 12.9 µS. The polarity of *capture_in* may be inverted using bit **inv_capt** in register IDXCTRL.

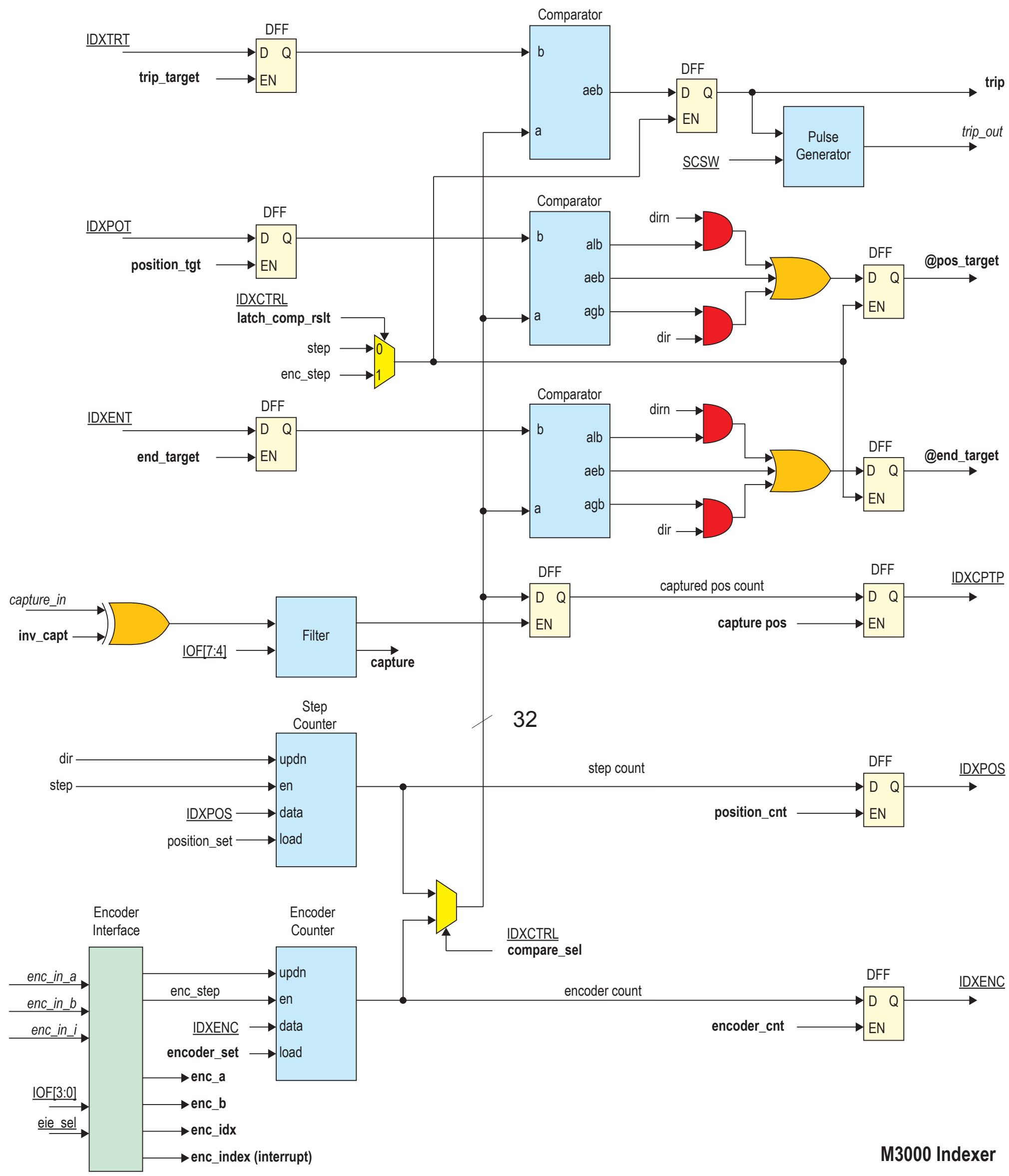


Figure 7.10: Indexer Logic Diagram

The step counter can be preset and read. To preset, write to registers IDXPOS. The multibyte value is written to the indexer when the self clearing bit **pos_set** in register IDXSTRB is written. The multibyte value is read from the indexer when the self clearing bit, **pos_ct** in register IDXSTRB is written. The position count value can be read from registers IDXPOS. The count will increase when the dir signal from the clock conversion logic is high.

An encoder interface converts the input from an external quadrature encoder on pins *ENC_IN_A* and *ENC_IN_B* to step and direction for use by the encoder counter. The encoder inputs may be filtered using bits **IOF[3..0]** in register IOF. The minimum detectable pulse width ranges from 50 nS to 12.9 µS.

The encoder counter can be preset and read. To preset, write to registers IDXENC. The multibyte value is written to the indexer when the self clearing bit **enc_set** in register IDXSTRB is written. The multi-byte value is read from the indexer when the self clearing bit **enc_ct** in register IDXSTRB is written. The encoder count value can be read from registers IDXENC. The count will increase when input *enc_in_a* leads input *enc_in_b*.

An edge of the encoder index mark on pin ENC_IN_IDX may be used to generate an interrupt bit **enc_index** in register IDXIFLG if enabled using bit enc_index in register IDXIFLG. The encoder index mark edge may be selected using bit eie_sel in register IDXCTRL.

The filtered encoder inputs on pins *ENC_IN_A*, *ENC_IN_B* and *ENC_IN_IDX* can be read using the corresponding bits in register STAT.

Dual H-Bridge Control

IMPORTANT!
It is preferable to use hardware pins for bridge control inversion. Software inversion can not be used with bridges requiring both high and low inversion because the bridge would be in a shoot through condition until the software could set the bits.

The PWM control block creates a signal set to operate the sine and cosine phase bridge drivers. The drivers may be monolithic or discrete FETs. The control puts the bridge into one of three states, forward; reverse or recirculate (defined as recirculating the phase current within the bridge).

The signal set will map to many common monolithic bridge drivers.

In the case of a discrete FET bridge implementation, a turn-on delay can be set to prevent shoot through. This delays the assertion of the bridge signal only, not the de-assertion.

The bit **recir** in control parameter PWMCTL determines where the motor current will recirculate in the bridge. The bits **todly[2..0]** in PWMCTL set the turn-on delay. The bit **enable** in PWMCTL allows bridge operation. Note the bridge is held disabled (all outputs de-asserted) until the *PWM_OSC* is running regardless of the **enable** bit.

H-Bridge Control Signal Set								
EN_PIN (Enable*)	Recirc	Bridge State	PHx_LH	PHx_LL	PHx_RH	PHx_RL	PHx_PWM	BRDG_EN
1	0	Recirc Low	0	1	0	1	0	1
1	X	ON, Sign 0	0	1	1	0	1	1
1	X	ON, Sign 1	1	0	0	1	1	1
1	1	Recirc HIGH	1	0	1	0	0	1
0	X	X	0	0	0	0	0	0
X=Doesn't Care 1=Asserted 0=Not Asserted *Not Available in all packages								

Table 7.4: H-Bridge Control Signal Set

Asserting pin *fault_in_n* low will cause a latched fault condition and will disable all the bridge controls. An interrupt bit **fault** in register IDXIFLG will be set if the corresponding bit in register IDXIMSK is set. The bit **fault** will be set in register STAT. The fault condition can be cleared by a power on reset, asserting the reset_n pin low or by setting the self clearing bit **clr_fl**t in register SPWMCTL. An active fault can not be cleared.

The bridge may be disabled by asserting the external pin *en_pin* low. In order to enable the bridge, the *en_pin* must be high and the **enable** bit must be set. The pin is internally pulled up and therefore may be left unconnected (bridge enabled). The pin is not available in all packages.

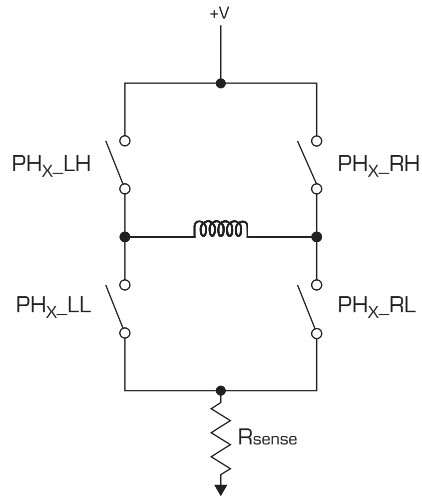


Figure 7.11: M3000 H-Bridge Diagram

The assertion level of the bridge control signals may be inverted via software bits or hardware pin inputs.

The bit **inv_hbc** in register SPWMCTL will invert the high side bridge controls for both phases. The bit **inv_lbc** in register SPWMCTL will invert the low side bridge controls for both phases. These bits should be set before enabling the bridge. When these bits are set the bridge controls are asserted low.

The pin *inv_hbc* will invert the high side bridge controls for both phases. The pin *inv_lbc* will invert the low side bridge controls for both phases. When these pins are high the bridge controls are asserted low. These pins are internally pulled low and may be left unconnected (no inversion) if not required. The pin *inv_hbc* is not available in all packages.

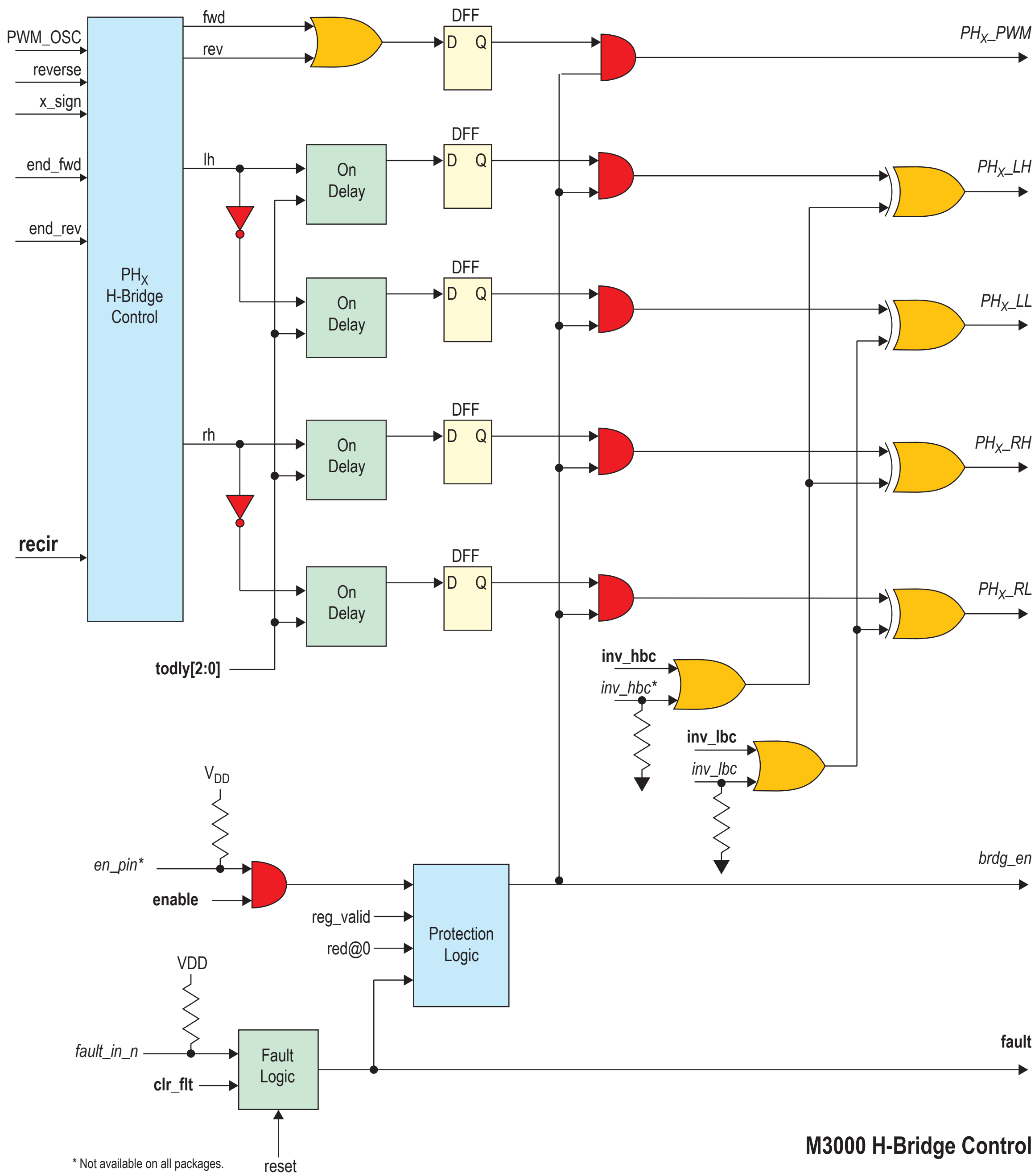


Figure 7.12: M3000 H-Bridge Control Logic Diagram

Phase Current Control with Anti-Resonance

The variable PWM oscillator creates the clock signals required to operate the phase bridges. The internal signal *PWM_OSC* (available as an output pin *PWM_OSC* on some packages) controls the basic timing of bridge operation. The rising edge of *PWM_OSC* signal controls the sine phase, the falling edge of *PWM_OSC* signal controls the cosine phase

The initial and maximum frequency of the PWM oscillator can be changed using parameter PWMSFRQ. The initial frequency is typically set to 20 KHz to avoid the audible frequency range. The maximum frequency is typically set to 60 KHz. The oscillator will adjust the frequency as required to maintain a minimum number of oscillator cycles during the positive front slope of the sine phase. Having a minimum number of edges enables more accurate construction of the phase current for reducing mid-range resonance.

When there are fewer than twenty (20) *PWM_OSC* edges during the front slope period and the maximum frequency as specified in PWMSFRQ has not been reached the *PWM_OSC* frequency is increased by a fixed amount when the period ends. If the number of edges is greater than or equal to twenty (20) and the frequency is above the initial frequency as specified in PWMSFRQ then the *PWM_OSC* frequency will be decreased by a fixed amount when the period ends. When the period of the front slope is stable or changing slowly, the *PWM_OSC* frequency will repetitively step between two frequencies. The stepping between frequencies reduces the excitation of system harmonics, thereby reducing mid-range resonance.

The bridge is turned on in the reverse direction (defined as removing phase current) a fixed amount of time before a *PWM_OSC* edge. There is a minimum forward on time after a *PWM_OSC* edge when the bridge is turned on in the forward direction (defined as increasing phases current toward the set point). The reverse and forward time are set by parameter PWMMSK (MASK is an output pin representing the reverse and forward time that is available on some packages). The typical times are 1.2 µS for each to maintain charge balance.

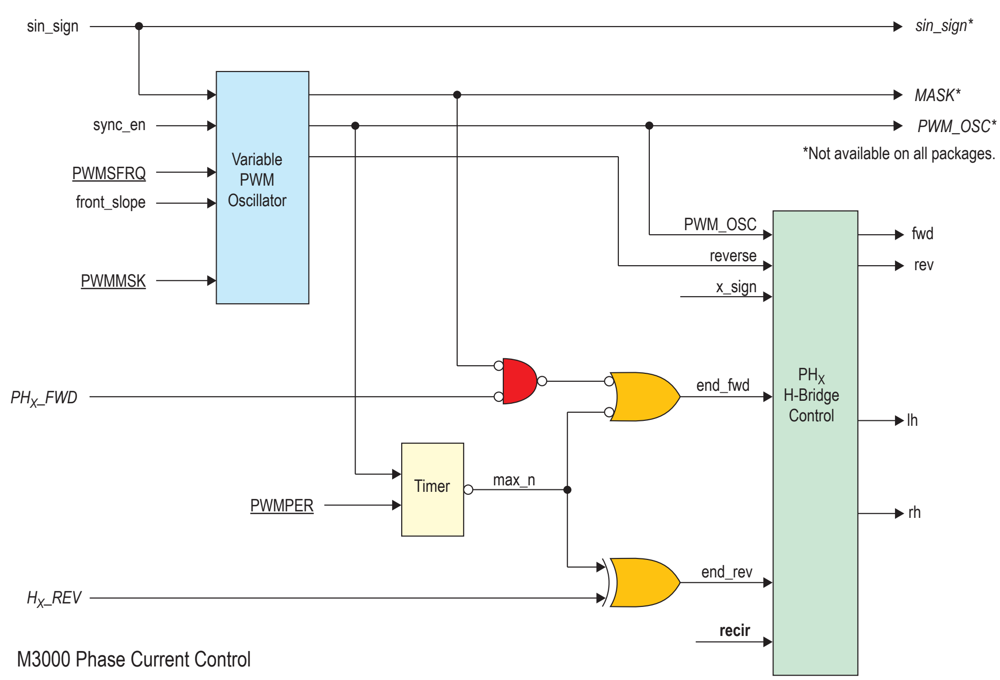


Figure 7.13: PWM Phase Current Control Diagram

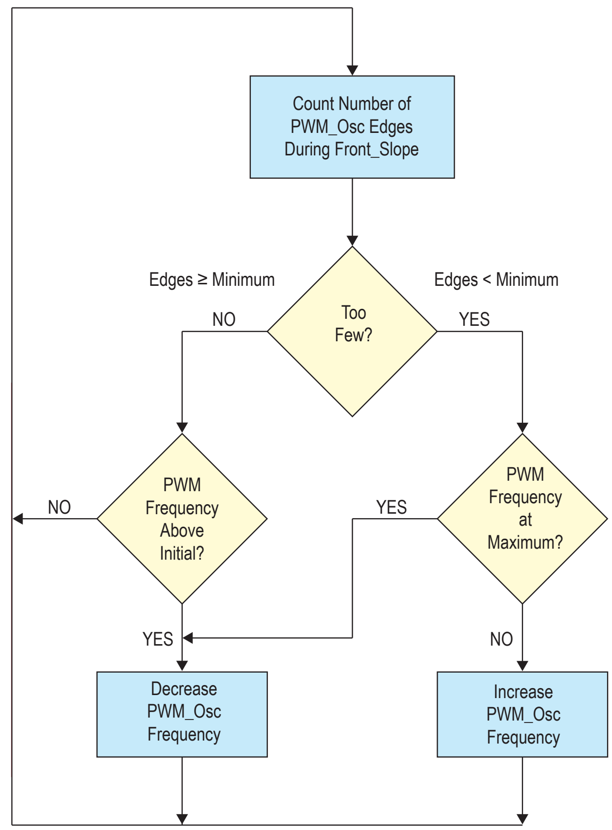


Figure 7.14: PWM Oscillator Frequency Adjustment Block Diagram

The maximum bridge on time per PWM_OSC period can be specified by parameter PWMPER as a percentage of the PWM_OSC period. This forces the bridge into the recirculate state during each PWM_OSC cycle. The typical value is 100% allowing the bridge to stay on as long as required to achieve set current.

The bridge is turned on in the reverse direction for the reverse time (described above) to measure the phase current. If the current is below the set point when the appropriate PWM_OSC edge (described above) changes, the bridge is turned on in the forward direction for the minimum forward time (described above). The bridge will go into the recirculate state if the minimum forward time expires and the set current is reached (pin *PHx_FWD* asserts low) or if the maximum bridge on time (described above) is reached. If the set current or maximum bridge on time are not reached before the next cycle, the forward state will continue, skipping the reverse measure state until *PHx_FWD* asserts. If the current is above the set point (pin *PHx_REV* is asserted low) when the PWM_OSC changes, the bridge stays in the reverse direction until pin *PHx_REV* de-asserts (goes high indicating phase current has fallen to the set point) or the maximum bridge on time is reached. If the set current or maximum bridge on time are not reached before the next cycle, the reverse state will continue until pin *PHx_REV* de-asserts or the sign of the phase changes.

When the Phase Current is at low levels, the MASK signal prevents the premature end to the forward period caused by switching transients. The removal (reverse) and replacement (forward) during the MASK period keeps the Phase Current balanced during the commutation transient period. The Phase Current losses that naturally occur during the recirculate period are accurately replaced when Phase Current flows in the forward direction after MASK ends.

While in motion (step clocks occurring) the *PWM_OSC* signal can be synchronized to the positive rising edge of the sin phase at zero cross by setting the **sync_en** bit in control parameter PWMCTL. When bit is set, the frequency generator is reset each zero cross corresponding to the internal signal *sin_sign* (available as an output pin *sig_sign* on some packages) asserting thereby synchronizing the oscillator to the phase current. Typically the oscillator is synchronized to prevent the frequency of the step input from “beating” against the PWM_OSC frequency, reducing the potential of mid-range resonance. Writing this parameter indicates the registers in this group are valid and is normally done last.

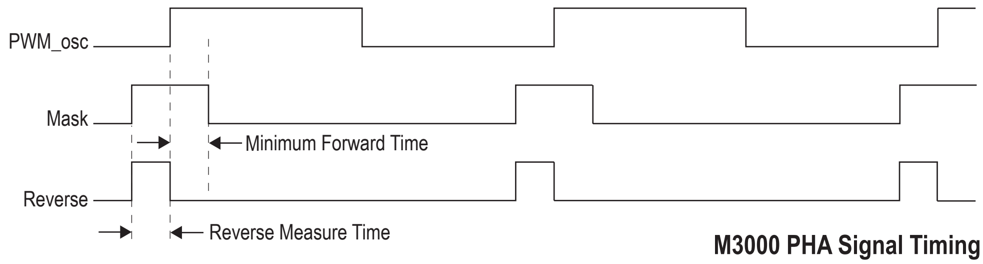


Figure 7.15: PHA Signal Timing Diagram

Motor and Motion Control Register Summary

Register Types

- ***Wr, Static***
  Primarily used to write a value. A read returns the last value written.
- ***Wr, Dynamic***
  Used to set a value. A read returns a value that is modified by events.
- ***Wr, Self Clear***
  Used to generate strobes. Reads are undefined.
- ***Rd, Dynamic***
  Reads a value modified externally. Writes have no effect.
- ***Rd, Write to Clear***
  Read flags, Write to clear.

PWM and Current Control Registers

						Register Bits								
Grp	RAM Address	Name	Function	Bytes	Register Type Strobe	7	6	5	4	3	2	1	0	Detail Page
PWM	0x0200	PWMMSK	PWM Mask	1	Wr. Static	Reverse Measure Time 0-FH 600 ns to 3.4 μs				Minimum Forward Time 0-FH 600 ns to 3.4 μs				7-18
PWM	0x0201	PWMPER	PWM %	1	Wr. Static	Maximum PWM Duty Cycle Percent 1-64H % of PWM Period + 100 ns								7-18
PWM	0x0202	PWMSFREQ	PWM Freq.	1	Wr. Static	Maximum PWM Frequency 0-FH 40 kHz to 70kHz, 2 kHz Resolution				Initial PWM Frequency 0-FH 10 kHz to 25 kHz, 1 kHz Resolution				7-18
PWM	0x0203	PWMCTL	PWN Control	1	Wr. Static	Quiet	—	Sync_En	Recirc 0=Low 1=High	DLY (Turn On) 0-7H 0 to 350 ns, 50 ns Resolution			Enable (Wr, Dyn.)	7-19
CURRENT	0x0204	CURIRUN	Run Current	1	Wr. Static	Run Current 0 - FFH								7-19
CURRENT	0x0205	CURIRED	Reduction Current	1	Wr. Static	Reduction Current, Starts after Reduction Delay 0-FFH								7-20
CURRENT	0x0206-0x0207	CURRDLY	Current Reduction Delay 1 & 2	2	~~Wr. Static~~ Auto Wr When MSB Written	Reduction Delay (From Last Step) 0, 2 - FFFFH 2 ms to 65,535 ms (+0/-1) 1 ms Resolution, 0=Never Reduce								7-20

Table 7.5: Motor and Motion Control Register Summary: PWM and Current Control

Velocity Registers

Grp	RAM Address	Name	Function	Bytes	Register Type Strobe	Register Bits 7	6	5	4	3	2	1	0	Detail Page
VELOCITY	0x0208-0x020A	VELLOW	Velocity Low 1,2 & 3	3	Wr, Static / Manual Wr	Initial (low) Velocity 0-800000H 0 to 5 x 10^6 Steps/Sec, 0.596 Step/Sec Resolution								7-20
	0x020B-0x020D	VELHI	Velocity High 1,2 & 3	3	Wr, Static / Manual Wr	Terminal (high) Velocity 0-800000H 0 to 5 x 10^6 Steps/Sec, 0.596 Step/Sec Resolution								7-21
	0x020E-0x0210	VELDEC	Peak Decel 1,2 & 3	3	Wr, Static / Manual Wr	Deceleration 1-FFFFFFH 90.9 to 1.55 x 10^9 Steps/Sec^2								7-21
	0x0211-0x0213	VELLACC	Peak Accel 1,2 & 3	3	Wr, Static / Manual Wr	Acceleration 1-FFFFFFH 90.9 to 1.55 x 10^9 Steps/Sec^2								7-21
	0x0214-0x0216	VELCVEL	Current Vel 1,2 & 3	3	Rd, Dynamic / Manual Rd	Current Velocity 0-800000H								7-22
	0x0217-0x0219	VELTVEL	Trip Vel 1,2 & 3	3	Wr, Static / Manual Wr	Trip Velocity 0-800000H 0 to 5 x 10^6 Steps/Sec, 0.596 Step/Sec Resolution								7-22
	0x021A	VELVGCTL*	Velocity Generator Control	1	Wr, Dynamic	Abort	—	Run	Dir	Dir_cv (rd, Dyn.)	—	—	—	7-23
	0x021B	VELSTB	Velocity Strobe	1	Wr, Self Clear	Write Strobe Accel	Write Strobe Decel	Write Strobe High	Write Strobe Low	Wrte Strobe Trip Vel	—	—	Read Strobe Current Vel	7-23
	0x021C	VELIFIG	Velocity Interupt Flag	1	Rd, Wr to Clear	Follow Error	—	—	—	—	Dec_ done	Acc_ done	Vel_Trip	7-24
	0x021D	VELIMSK	Velocity Interrupt Mask	1	Wr, Static	Follow Error	—	—	—	—	Dec_ done	Acc_ done	Vel_Trip	7-24

*Note: Write last in group, (reg valid)

Table 7.6: Motor and Motion Control Register Summary: Velocity

Index Registers

Grp	RAM Address	Name	Function	Bytes	Register Type Strobe	Register Bits 7	6	5	4	3	2	1	0	Detail Page
INDEX	0x021E-0x0221	IDXTRT	Index Trip Target 1-4	4	Wr, Static / Manual Wr	Trip Position Target Signed, 0 -7FFFFFFFFH ±2.1 x 10^9 Counts								7-25
	0x0222-0x0225	IDXENT	Index End Target 1-4	4	Wr, Static / Manual Wr	End Position Target Signed, 0 -7FFFFFFFFH ±2.1 x 10^9 Counts								7-25
	0x0226-0x0227	IDXMSDT	Index Motor Settling Delay Time 1-2	2	Wr, Static / Auto Wr When MSB Written	Motor Settling Delay Time 0 - FFFFH 0ms to 65,535 ms, 1 ms Resolution								7-26
	0x0228-0x022B	IDXPOT	Index Position Target 1-4	4	Wr, Static / Manual Wr	Position Target (Slew Position) Signed, 0 -7FFFFFFFFH ±2.1 x 10^9 Counts								7-26
	0x022C-0x022F	IDXPOS	Index Position Count 1-4	4	Wr, Dynamic / Manual Wr/Rd	Position Count Signed, 0 -7FFFFFFFFH ±2.1 x 10^9 Counts								7-27
	0x0230-0x0233	IDXENC	Index Encoder Count 1-4	4	Wr, Dynamic / Manual Wr/Rd	Encoder Count Signed, 0 -7FFFFFFFFH ±2.1 x 10^9 Counts								7.27
	0x0234	IDXCTRL	Index Control	1	Wr, Static	Index_ Abt_En	—	—	—	Latch_ Comp_ Rslt 0=Step 1=Enc.	Inv_ Capt	Eie_Sel 0=Rising 1=Falling	Compare_ Sel 0=Step 1=Enc	7-28
	0x0235	IDXSTRB	Index Strobe	1	Wr, Self Clear	Read_ Strobe Encoder_ Cnt	Read_ Strobe Position_ Cnt	Read_ Strobe Capture_ Pos	Write_ Strobe Encoder_ Set	Write_ Strobe Position_ Set	Write_ Strobe Position_ Tgt	Write_ Strobe End_Tgt	Write_ Strobe Trip_Tgt	7-29
	0x0236-0x0239	IDXCPTP	Index Capture Position	4	Rd, Dynamic / Manual Rd	Capture Position Count Signed, 0 -7FFFFFFFFH ±2.1 x 10^9 Counts								7-30
	0x023A	IDXIFLG	Index Interrupt Flag	1	Rd, Wr to Clear	—	Enc_ Index	@ End_tgt	@ Pos_Tgt	Caputre	Trip	Msdt	Fault	7-30
	0x023B	IDXIMSK	Index Interrupt Mask	1	Wr, Static	—	Enc_ Index	@ End_tgt	@ Pos_Tgt	Caputre	Trip	Msdt	Fault	7-31

Table 7.7: Motor and Motion Control Register Summary: Index

Step Clock and Misc. Registers

Grp	RAM Address	Name	Function	Bytes	Register Type / Strobe	7	6	5	4	3	2	1	0	Detail Page
STEP CLOCK	0x023C	SCIO	Step Clock I/O	1	Wr, Static	Inv_Dir	Inv_Step	—	Sq_Csel	En_Sdo	Vg_Pinn 0=Drive 1=Vel_Gen	Sel_Clk1	Sel_Clk0	7-32
	0x023D	SCSW	Step Clock Width	1	Wr, Static	Step Width (Output) 0-FFH 100 ns to 12.85 µs, 50 ns Resolution								7.33
	0x023E-0x0241	SCRF	Step Clock Ratio Factor 1-4	4	Wr, Static / Auto Wr When MSB Written	Ratio Factor (Drive) Constant (02000000H) x Ratio Range 2 ≥ Ratio ≥ 0.001 8312H - 4000000H								7-33
MISC	0x0242	IOF	Input/ Output Filter	1	Wr, Static	Filter Group 1 (Capture In) 0 - 9H 10 MHz - 38.8 kHz				Filter Group 0 (Step/Dir In, Encoder In) 0 - 9H 10 MHz - 38.8 kHz				7-34
	0x0243	MSEL*	Microstep Step Select	1	Wr, Static	Init_707	Enb_Ser_Dec	Spd_Intf	Msel4	Msel3	Msel2	Msel1	Msel0	7-35
	0x0244	STAT	Status	1	Rd, Dynamic	OK_Folo	Enc_A	Enc_b	VG_Idle	Enc_Idx	Curred	Atzero	Fault	7-36
	0x0245	SPWMCTL**	Special PWM Control	1	Wr	—	—	—	Cir_Flt (Wr, Self Clr)	—	—	Inv_Hbc (Wr, Static)	Inv_Lbc (Wr, Static)	7-36

*Note: Write last in group, (reg valid)
**Note: Write inv_xbc before pwm_ctl

Table 7.8: Motor and Motion Control Register Summary: Step Clock and Miscellaneous

Motor and Motion Control Registers

PWM Registers

PWMMSK (PWM Mask)

Bit	7	6	5	4	3	2	1	0	
0x0200	REVTM[3:0]				FORTM[3:0]				PWMMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.9: PWM Control Register

- **Bits 7...4 - REVTM - Reverse Measure Time**

  This value sets the bridge reverse measure time before the normal forward period.

- **Bits3...0 - FORTM - Minimum Forward on Time**

  This value sets the minimum bridge forward on time. The normal setting is 1.2 µS for each to balance the current (0x66).

Reverse Measure/Minimum Forward On Time							
Hex	Time	Hex	Time	Hex	Time	Hex	Time
0x0	600 ns	0x4	1.0 µs	0x8	1.6 µs	0xC	2.5 µs
0x1	700 ns	0x5	1.1 µs	0x9	1.8 µs	0xD	2.8 µs
0x2	800 ns	0x6	1.2 µs	0xA	2.0 µs	0xE	3.1 µs
0x3	900 ns	0x7	1.4 µs	0xB	2.2 µs	0xF	3.4 µs

Table 7.10: Reverse Measure/Minimum Forward On Time

PWMPER (PWM Percent)

Bit	7	6	5	4	3	2	1	0	
0x0201	Maximum PWM Duty Cycle, Percent								PWMPER
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.11: PWM Percent Register

- **Bits 7..0 – Maximum PWM Duty Cycle, Percent**

  This value sets the maximum duty cycle as a percentage of the bridge PWM period. The range is 1 to 100%. Values above 100% have no meaning. The maximum time is % of bridge PWM period + 100 nS.

PWMSFRQ (PWM Frequency)

Bit	7	6	5	4	3	2	1	0	
0x0202	MAXPWM[3:0]				INPWM[3:0]				PWMSFRQ
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.12: PWM Frequency Register

- **Bits 7..4 – MAXPWM - Maximum Bridge PWM frequency**

  This value sets the maximum bridge PWM frequency. The range is 40 kHz to 70 kHz with 2 kHz resolution. A value of 0 equals 40 kHz with each LSB adding 2 kHz. The normal setting is 60 kHz (0xA).

- **Bits 3..0 – INPWM - Initial Bridge PWM frequency**

  This value sets initial bridge PWM frequency. The range is 10 kHz to 25 kHz with 1 kHz resolution. A value of 0 equals 10 kHz with each LSB adding 1 kHz. The normal setting is 20 kHz (0xA).

PWMCTL (PWM Control)

Bit	7	6	5	4	3	2	1	0	
0x0203	QUIET	–	SYNC_EN	RECIR	TODLY[2:0]			ENABLE	PWMCTL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.12: PWM Control Register

- **Bit 7 – QUIET**

This bit changes PWM operation. When quiet is set, the bridge logic does not enter the reverse measure period, therefore there are fewer transitions. The bridge is disabled during zero cross. This mode is used at rest or when moving very slowly. When quiet is cleared, normal bridge operation is selected.

- **Bit 6 – Reserved**
- **Bit 5 – SYNC_EN**

This bit controls the synchronization of the bridge PWM with the zero cross. When the sync_en bit is set, the bridge PWM will be synchronized with the positive front slope of the sin phase at each zero cross.

- **Bit 4 – RECIR**

This bit controls where the motor current will recirculate within the bridge during the recirculate period. When recirc is set, the motor current will recirculate in the high portion of the bridge. When recir is cleared, the motor current will recirculate in the low portion of the bridge.

- **Bits 3..1 – TODLY - Turn on Delay**

This value sets the bridge control turn on delay to prevent shoot through if a discrete FET bridge is in use. The range is 0 to 350 nS with 50 nS resolution. Each LSB is 50 nS. The normal setting for a bridge driver is 0 nS (0x0).

- **Bit 0 – ENABLE**

This enables the sin & cosine bridge controls.

A read of this bit returns the logical AND of the software bit and the hardware pin EN_PIN (not available in all packages).

This byte must be written last in this group. When written, a register valid signal is generated allowing the bridge PWM logic to start.

CURRENT Registers

CURIRUN (Run Current)

Bit	7	6	5	4	3	2	1	0	
0x0204	RUN CURRENT								CURIRUN
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.13: Run Current Register

- **Bits 7..0 – Run Current**

This value sets the peak run current for each phase as a percentage of full scale. (0xFF) is maximum, (0x0) is minimum.

CURIRED (Current Reduction)

Bit	7	6	5	4	3	2	1	0	
0x0205	REDUCTION CURRENT								CURIRED
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.14: Current Reduction Register

- **Bits 7..0 – Reduction Current**

This value sets the peak reduction current for each phase as a percentage of full scale. (0xFF) is maximum, (0x0) is minimum. The motor current is set to this value an amount of time (specified in CURRDLY) after the last step clock.

CURRDLY (Current Reduction Delay 1 & 2)

NOTE: It may be desired to have the current reduce a specified time after the motor setting delay time (MSDT), In this case the value in CURRDLY should be MSDT plus specified time.

Bit	7	6	5	4	3	2	1	0	
0x0206	CURRENT REDUCTION DELAY (LOW BYTE) [7..0]								CURRDLY1
0x0207	CURRENT REDUCTION DELAY (HIGH BYTE) [15..8]								CURRDLY2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.15: Current Reduction Delay Register 1 & 2

- **Bits 7..0 – CURRENT REDUCTION DELAY 1 (Low Byte)**
- **Bits 15..8 – CURRENT REDUCTION DELAY 2 (High Byte)**

This value sets the delay in mS between the last step clock occurrence and the application of reduction current to the motor. The range is 2 mS to 65535 mS (+0 / -1), 1 mS resolution, 0 = never reduce. Typical setting: 500 mS (0x01F4).

The Low Byte is to be written first.

VELOCITY Registers

VELLOW (Initial Low Velocity 1, 2 & 3)

Bit	7	6	5	4	3	2	1	0	
0x0208	INITIAL VELOCITY 1 [7..0]								VELLOW1
0x0209	INITIAL VELOCITY 2 [15..8]								VELLOW2
0x020A	INITIAL VELOCITY 3 [23..16]								VELLOW3
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.16: Initial Low Velocity Register 1, 2 & 3

- **Bits 7..0 – INITIAL VELOCITY 1**
- **Bits 15..8 – INITIAL VELOCITY 2**
- **Bits 23..16 – INITIAL VELOCITY 3**

NOTE: $10M/2^{24} \cong 0.596$

This value sets the initial (low) velocity for a move. The range is 0 to 5 X 10^6 steps/sec with 0.596 step/sec resolution. Maximum register value is (0x800000). The 24-bit word is loaded when the low bit in register VELSTB is written.

VELHI (Terminal High Velocity 1, 2 & 3)

Bit	7	6	5	4	3	2	1	0	
0x020B	TERMINAL VELOCITY 1 [7..0]								VELHI1
0x020C	TERMINAL VELOCITY 2 [15..8]								VELHI2
0x020D	TERMINAL VELOCITY 3 [23..16]								VELHI3
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.17: Terminal High Velocity Register 1, 2 & 3

- **Bits 7..0 – TERMINAL VELOCITY 1**
- **Bits 15..8 – TERMINAL VELOCITY 2**
- **Bits 23..16 – TERMINAL VELOCITY 3**

This value sets the terminal (high) velocity for a move. The range is 0 to 5 X 10^6 steps/sec with 0.596 step/sec resolution. Maximum register value is (0x800000). The 24 bit word is loaded when the high bit in register VELSTB is written.

VELDEC (Velocity Deceleration 1, 2 & 3)

Bit	7	6	5	4	3	2	1	0	
0x020E	DECEL 1 [7..0]								VELDEC1
0x020F	DECEL 2 [15..8]								VELDEC2
0x0210	DECEL 3 [23..16]								VELDEC3
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.18: Velocity Deceleration Register 1, 2 & 3

NOTE

NOTE: $10M/2^{16}$ x 0.596 ≅ 90.9

- **Bits 7..0 – DECEL 1**
- **Bits 15..8 – DECEL 2**
- **Bits 23..16 – DECEL 3**

This value sets the deceleration rate. The range is 90.9 to 1.5 X 10^9 steps/sec^2 with 90.9 step/sec^2 resolution. The 24-bit word is loaded when the decel bit in register VELSTB is written.

VELACC (Velocity Acceleration 1, 2 & 3)

NOTE

NOTE: VELDEC and VELACC take effect after it is strobed and then when the velocity generator is commanded (including VELLOW and VELHI).

Bit	7	6	5	4	3	2	1	0	
0x0211	ACCEL 1 [7..0]								VELACC1
0x0212	ACCEL 2 [15..8]								VELACC2
0x0213	ACCEL 3 [23..16]								VELACC3
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.19: Velocity Acceleration Register 1, 2 & 3

- **Bits 7..0 – ACCEL 1**
- **Bits 15..8 – ACCEL 2**
- **Bits 23..16 – ACCEL 3**

This value sets the acceleration rate. The range is 90.9 to 1.5 X 10^9 steps/sec^2 with 90.9 step/sec^2 resolution . The 24-bit word is loaded when the accel bit in register VELSTB is written.

VELCVEL (Current Velocity 1, 2 & 3)

Bit	7	6	5	4	3	2	1	0	
0x0214	CURRENT VELOCITY 1 [7..0]								VELCVEL1
0x0215	CURRENT VELOCITY 2 [15..8]								VELCVEL2
0x0216	CURRENT VELOCITY 3 [23..16]								VELCVEL3
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.20: Current Velocity Register 1, 2 & 3

- **Bits 7..0 – CURRENT VELOCITY 1**
- **Bits 15..8 – CURRENT VELOCITY 2**
- **Bits 23..16 – CURRENT VELOCITY 3**

This value represents the current velocity of the velocity generator. The range is 0 to 0x800000. This 24 bit word is latched for read when the self clearing bit, current vel, in register VELSTB is written.

VELTVEL (Trip Velocity 1, 2 & 3)

Bit	7	6	5	4	3	2	1	0	
0x0217	TRIP VELOCITY 1 [7..0]								VELTVEL1
0x0218	TRIP VELOCITY 2 [15..8]								VELTVEL2
0x0219	TRIP VELOCITY 3 [23..16]								VELTVEL3
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.21: Trip Velocity Register 1, 2 & 3

- **Bits 7..0 – TRIP VELOCITY 1**
- **Bits 15..8 – TRIP VELOCITY 2**
- **Bits 23..16 – TRIP VELOCITY 3**

This value represents a trip velocity during a move profile. The value is compared to the current velocity of the velocity generator. An interrupt (optional) is generated on a match. The range is 0 to 0x800000. This 24-bit word is loaded when the self clearing bit, target vel, in register VELSTB is written.

VELVGCTL (Velocity Generator Control)

Bit	7	6	5	4	3	2	1	0	
0x021A	ABORTMTN	–	RUNMTN	DIRMTN	DIR_CV	–	–	–	VELVGCTL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	1	0	0	0	0	

Table 7.22: Velocity Generator Control Register

- **Bit 7 – ABORTMTN**

  This causes the velocity generator to stop with out deceleration.

- **Bit 6 – Reserved**
- **Bit 5 – RUNMTN**

  Used to start the velocity generator. When set, the velocity generator will start creating step clocks at the frequency specified by the value in register VELLOW. When runmtn is de-asserted, the velocity generator will begin deceleration specified by the value in register VELDEC from the current velocity. If the abortmtn bit is asserted when runmtn is set, runmtn must be de-asserted before motion can be started. A read returns preset runmtn. This can be used during ratio following. When it is ok to follow, this bit is set.

- **Bit 4 – DIRMTN**

  This bit sets the direction when stepping through the sine and cosine tables. A read returns present direction. This can be used during following.

- **Bit 3 – DIR_CV**

  This bit indicates the direction at the time the current velocity was captured.

- **Bits 2.0 – Reserved**

VELSTB (Velocity Strobe)

	Write Strobe							Read Strobe	
Bit	7	6	5	4	3	2	1	0	
0x021B	ACC_STB	DEC_STB	VHI_STB	VLOW_STB	TRPVEL_STB	–	–	CURVEL	VELSTB
Read/Write	W	W	W	W	W	W	W	W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.23: Velocity Strobe Register

- **Bit 7 – ACC_STB**

  This is a self clearing bit used as a write strobe to set the value specified in register VELACC into the velocity generator.

- **Bit 6 – DEC_STB**

  This is a self clearing bit used as a write strobe to set the value specified in register VELDEC into the velocity generator.

- **Bit 5 – VHI_STB**

  This is a self clearing bit used as a write strobe to set the value specified in register VELHI into the velocity generator.

- **Bit 4 – VLOW_STB**

  This is a self clearing bit used as a write strobe to set the value specified in register VELLOW into the velocity generator.

- **Bit 3 – TRIPVEL_STB**

  This is a self clearing bit used as a write strobe to set the value specified in register VELTVEL into the velocity generator.

- **Bits 2..1 – Reserved**
- **Bit 0 – CURVEL**

  This is a self clearing bit used as a read strobe to capture the current velocity from the velocity generator for subsequent reads from the VELCVEL registers.

VELIFLG (Velocity Interrupt Flag)

Bit	7	6	5	4	3	2	1	0	
0x021C	FLLWER	–	–	–	–	DEC_DONE	ACC_DONE	VEL_TRIP	VELIFLG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.24: Velocity Interrupt Flag Register

NOTE

NOTE: Flags are only active when the corresponding mask is set. Interrupts must be enabled to see the flag.

- **Bit 7 – FLLWER**

When this bit is set there is an error in ratio following, either the measured period is too long or the computed velocity is too high.

- **Bits 6..3 – Reserved**
- **Bit 2 – DEC_DONE**

When this bit is set, the velocity generator has finished the deceleration section of the move profile. This will occur when finished decelerating to VELLOW, or to VELHI (VELHI changed to a lower value), or to r_high (when ratio following and r_high changed to a lower value).

NOTE

NOTE: Write a 1 to the bit to clear the flag.

- **Bit 1 – ACC_DONE**

When this bit is set, the velocity generator has finished the acceleration section of the move profile. This will occur when finished accelerating to VELHI (initial move or VELHI changed to a higher value), or to r_high (when ratio following and r_high changed to a higher value).

- **Bit 0 – VEL_TRIP**

When this bit is set, it indicates the current velocity in the velocity generator matches(ed) the trip velocity specified in register VELTVEL.

Write a 1 to clear a flag. If an interrupt occurs simultaneously while attempting to clear, the flag will not clear and another interrupt will be generated.

VELIMSK (Velocity Interrupt Mask)

Bit	7	6	5	4	3	2	1	0	
0x021D	FLLWER	–	–	–	–	DEC_DONE	ACC_DONE	VEL_TRIP	VELIMSK
Read/Write	R/W	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.25: Velocity Interrupt Mask Register

- **Bit 7 – FLLWER**
- **Bits 6..3 – Reserved**
- **Bit 2 – DEC_DONE**
- **Bit 1 – ACC_DONE**
- **Bit 0 – VEL_TRIP**

When these bits are set, an incoming interrupt of the same name will set the corresponding flag and generate a processor interrupt.

INDEX Registers

IDXTRT (Index Trip Target 1, 2, 3 & 4)

Bit	7	6	5	4	3	2	1	0	
0x021E	POSITION TRIP 1 [7..0]								IDXTRT1
0x021F	POSITION TRIP 2 [15..8]								IDXTRT2
0x0220	POSITION TRIP 3 [23..16]								IDXTRT3
0x0221	POSITION TRIP 4 [31..24]								IDXTRT4
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.26: Index Trip Target Registers 1, 2, 3 & 4

- **Bits 7..0 – POSITION TRIP 1**
- **Bits 15..8 – POSITION TRIP 2**
- **Bits 23..16 – POSITION TRIP 3**
- **Bits 31..24 – POSITION TRIP 4**

This signed value represents a trip position during a move. The value is compared to the position count (either step or encoder). An output signal, trip_out and an interrupt (optional) are generated on a match.

The range is +/- 2.1 x 10^9 counts. The 32-bit word is loaded when the self clearing bit, trip_tgt in register IDXSTRB is written.

IDXENT (Index End Target 1, 2, 3 & 4)

Bit	7	6	5	4	3	2	1	0	
0x0222	POSITION END TARGET 1 [7..0]								IDXENT1
0x0223	POSITION END TARGET 2 [15..8]								IDXENT2
0x0224	POSITION END TARGET 3 [23..16]								IDXENT3
0x0225	POSITION END TARGET 4 [31..24]								IDXENT4
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.27: Index End Target Registers 1, 2, 3 & 4

- **Bits 7..0 – POSITION END TARGET 1**
- **Bits 15..8 – POSITION END TARGET 2**
- **Bits 23..16 – POSITION END TARGET 3**
- **Bits 31..24 – POSITION END TARGET 4**

This signed value represents the end position of a move. The value is compared to the position count (either step or encoder). An interrupt (optional) is generated on a match and the velocity generator will be aborted (without any deceleration) if bit idx_abort is set in register IDXCTRL.

The range is +/- 2.1 x 10^9 counts. The 32-bit word is loaded when the self clearing bit, end tgt, in register IDXSTRB is written.

IDXMSDT (Index Motor Settling Delay Time 1& 2)

NOTE: MSDT and HCDT (Hold Current Delay Time) are additive to make CURRDLY

Bit	7	6	5	4	3	2	1	0	
0x0226	MSDT [7..0]								IDXMSDT1
0x0227	MSDT [15..8]								IDXMSDT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.28: Index Motor Settling Delay Time Registers 1 & 2

- **Bits 7..0 – MSDT**
- **Bits 15..8 – MSDT**

This value sets the motor settling delay time in mS. The purpose is to allow the motor to mechanically settle before the next move. When the timer expires, an interrupt (optional) is generated. The timer starts when the end target is reached. The range is 0 to 65535 mS with 1 mS resolution.

The 16-bit word is automatically transferred when the high byte is written.

IDXPOT (Index Position Target 1, 2, 3 & 4)

Bit	7	6	5	4	3	2	1	0	
0x0228	START DECEL POSITION TARGET 1 [7..0]								IDXPOT1
0x0229	START DECEL POSITION TARGET 2 [15..8]								IDXPOT2
0x022A	START DECEL POSITION TARGET 3 [23..16]								IDXPOT3
0x022B	START DECEL POSITION TARGET 4 [31..24]								IDXPOT4
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.29: Index Position Target Registers 1, 2, 3 & 4

- **Bits 7..0 – START DECEL POSITION TARGET 1**
- **Bits 15..8 – START DECEL POSITION TARGET 2**
- **Bits 23..16 – START DECEL POSITION TARGET 3**
- **Bits 31..24 – START DECEL POSITION TARGET 4**

This signed value represents a position in a move. The value is compared to the position count (either step or encoder). An interrupt (optional) is generated on a match and the velocity generator will be stopped (start deceleration using value in VELDEC register) if bit idx_abort is set in register IDXCTRL.

The range is +/- 2.1 x 10^9 counts. The 32-bit word is loaded when the self clearing bit, pos_targ_s in register IDXSTRB is written.

IDXPOS (Index Position Count 1, 2, 3 & 4)

Bit	7	6	5	4	3	2	1	0	
0x022C	POSITION 1 [7..0]								IDXPOS1
0x022D	POSITION 2 [15..8]								IDXPOS2
0x022E	POSITION 3 [23..16]								IDXPOS3
0x022F	POSITION 4 [31..24]								IDXPOS4
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.30: Index Position Count Registers 1, 2, 3 & 4

NOTE

NOTE: IDXPOS contains 2 Buffers, 1 for Read and 1 for Write

- **Bits 7..0 – POSITION 1**
- **Bits 15..8 – POSITION 2**
- **Bits 23..16 – POSITION 3**
- **Bits 31..24 – POSITION 4**

This signed value represents the position in step counts from the step counter. The range is +/- 2.1 x 10^9 counts. The counter may be preloaded with a count or cleared by writing 0. The 32-bit word is loaded when the self clearing bit, pos_set in register IDXSTRB is written.

The 32 bit word is loaded when the self clearing bit, Pos_set, in Register IDXSTRB is written. The counter value may be captured for read when the self clearing bit, Pos_ct, in register IDXSTRB is written.

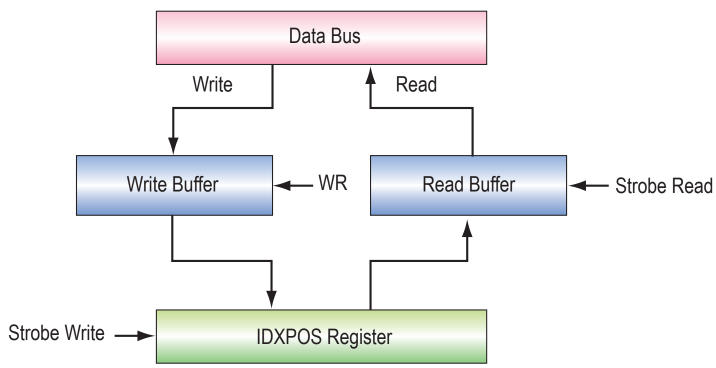


Figure 7:.16: Index Position Count Read/Write Buffers

IDXENC (Index Encoder Count 1, 2, 3 & 4)

Bit	7	6	5	4	3	2	1	0	
0x0230	ENCODER POSITION 1 [7..0]								IDXENC1
0x0231	ENCODER POSITION 2 [15..8]								IDXENC2
0x0232	ENCODER POSITION 3 [23..16]								IDXENC3
0x0233	ENCODER POSITION 4 [31..24]								IDXENC4
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.31: Index Encoder Count Registers 1, 2, 3 & 4

- **Bits 7..0 – ENCODER POSITION 1**
- **Bits 15..8 – ENCODER POSITION 2**
- **Bits 23..16 – ENCODER POSITION 3**
- **Bits 31..24 – ENCODER POSITION 4**

This signed value represents the position in counts from a quadrature encoder. The counter may be preloaded with a count or cleared by writing 0. The 32-bit word is loaded when the self clearing bit, enc_set, in register IDXSTRB is written. The counter value may be captured for read when the self clearing bit, enc_ct, in register IDXSTRB is written. The range is +/- 2.1 x 10^9 counts.

IDXCTRL (Index Control)

Bit	7	6	5	4	3	2	1	0	
0x0234	IDX_ABORT	–	–	–	LATCH_COMP_RSLT	INV_CAPT	EIE_SEL	CMP_SEL	IDXCTRL
Read/Write	R/W	R/W			R/W	R/W	R/W	R/W	
Initial Value	0	0			0	0	0	0	

Table 7.32: Index Control Register

- **Bit 7 – IDX_ABORT**

  This bit, when set, will affect velocity generator operation. When set, deceleration will begin when the count (step or encoder) matches the value specified in the register IDXPOT and the velocity generator will abort when the count (step or encoder) matches the value specified in register IDXENT.

- **Bits 6..4 – Reserved**
- **Bits 3 – LATCH_COMP_RSLT**

  This bit selects what latches the position comparators result. When cleared, the result is latched after a step occurs. When set, the result is latched after an encoder edge occurs. NOTE: The latching of the comparators result is independent of what is being compared (Step vs. target or encoder vs. target).

- **Bit 2 – INV_CAPT**

  This bit controls the input level of the capt_ in input that causes the count (step or encoder) to be captured. When set, the input should go low to capture the count. Capture on falling edge. When cleared, the input should go high to capture the count. Capture on rising edge.

- **Bit 1 – EIE_SEL**

  This bit selects which edge of the encoder index mark will generate an interrupt (optional). When set, the falling edge is used. When cleared, the rising edge is used.

- **Bit 0 – CMP_SEL**

  This bit selects which count source (step or encoder) will be compared with target, end and trip position.

  When set, the encoder count will be compared with target, end and trip position.When cleared, the step count will be compared with target, end and trip position.

IDXSTRB (Index Strobe)

Bit	7	6	5	4	3	2	1	0	
	Read Strobes			Write Strobes					
0x0235	ENC_CT	POS_CT	CAPTURE_S	ENC_SET	POS_SET	POS_TARG_S	END_TARG_S	TRP_TARG_S	IDXSTRB
Read/Write	W	W	W	W	W	W	W	W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.33: Index Strobe Register

- **Bit 7 – ENC_CT - Encoder Count**

  This is a self clearing bit used as a read strobe to capture the count from the encoder counter for subsequent reads from the IDXENC registers.

- **Bit 6 – POS_CT - Position Count**

  This is a self clearing bit used as a read strobe to capture the count from the position counter for subsequent reads from the IDXPOS registers.

- **Bit 5 – CAPTURE_S - Capture Position**

  This is a self clearing bit used as a read strobe to capture the count from the capture register for subsequent reads from the IDXCPTP registers.

- **Bit 4 – ENC_SET - Encoder Set**

  This is a self clearing bit used as a write strobe to preset the value specified in the register IDXENC into the encoder counter.

- **Bit 3 – POS_SET - Position Set**

  This is a self clearing bit used as a write strobe to preset the value specified in the register IDXPOS into the position counter.

- **Bit 2 – POS_TARG_S - Position Target**

  This is a self clearing bit used as a write strobe to set the value specified in the register IDXPOT. This value is compared to a count source (step or encoder).

- **Bit 1 – END_TARG_S - End Target**

  This is a self clearing bit used as a write strobe to set the value specified in the register IDXENT. This value is compared to a count source (step or encoder).

- **Bit 0 – TRIP_TARG_S - Trip Target**

  This is a self clearing bit used as a write strobe to set the value specified in the register IDXTRT. This value is compared to a count source (step or encoder).

IDXCPTP (Index Capture Position 1, 2, 3 & 4)

Bit	7	6	5	4	3	2	1	0	
0x0236	CAPTURED POSITION 1 [7..0]								IDXCPTP1
0x0237	CAPTURED POSITION 2 [15..8]								IDXCPTP2
0x0238	CAPTURED POSITION 3 [23..16]								IDXCPTP3
0x0239	CAPTURED POSITION 4 [31..24]								IDXCPTP4
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.34: Index Capture Position Registers 1, 2, 3 & 4

- **Bits 7..0 – CAPTURED POSITION 1**
- **Bits 15..8 – CAPTURED POSITION 2**
- **Bits 23..16 – CAPTURED POSITION 3**
- **Bits 31..24 – CAPTURED POSITION 4**

This signed value represents the count (step or encoder) that was captured when the filtered, edge (selectable with bit inv_capt in register IDXCTRL) of the external input, capt_in was asserted. The range is +/- 2.1 x 10^9 counts.

The 32-bit word is latched for read when the self clearing bit, capture_s in register IDXSTRB is written.

IDXIFLG (Index Interrupt Flag)

Bit	7	6	5	4	3	2	1	0	
0x023A	–	ENC_INDX	END_TARG	POS_TARG	CAPTURE	POS_TRIP	MSDT_DN	FAULT_INT	IDXIFLG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.35: Index Interrupt Flag Register

NOTE

NOTE: Flags are only active when the cooresponding mask is set. Interrupts must be enabled to see the flag.

NOTE

NOTE: Write a 1 to the bit to clear the flag.

- **Bit 7 – Reserved**
- **Bit 6 – ENC_INDX**

When this bit is set, the index mark of an external encoder has been asserted.

- **Bit 5 – END_TARG**

When this bit is set, the value set in registers, IDXENT matches or past the count (step or encoder).

- **Bit 4 – POS_TARG**

When this bit is set, the value set in registers, IDXPOT matches or past the count (step or encoder).

- **Bit 3 – CAPTURE**

When this bit is set, it indicates a new count (step or encoder) has been latched by the assertion of the external signal capt_in.

- **Bit 2 – POS_TRIP**

When this bit is set, the value set in registers, IDXTRT matches the count (step or encoder).

- **Bit 1 – MSDT_DN**

When this bit is set, the period set by the value in register IDXMSDT has expired. The timer is started when the velocity generator enters the idle state. Typically used to allow a motor to settle after an index is complete or velocity has gone to zero for a direction change.

- **Bit 0 – FAULT_INT**

When this bit is set, an external fault has been detected. The bridge controls are disabled. The ASIC must be reset or set bit CLR_FLT in Register SPWMCTL in order to operate the bridge again.

If an interrupt occurs simultaneously while attempting to clear, the flag will not clear and another interrupt will be generated.

IDXIMSK (Index Interrupt Mask Register)

Bit	7	6	5	4	3	2	1	0	
0x023B	–	ENC_INDX	END_TARG	POS_TARG	CAPTURE	POS_TRIP	MSDT_DN	FAULT_INT	IDXIMSK
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.36: Index Interrupt Mask Enable Register

- **Bit 7 – Reserved**
- **Bit 6 – ENC_INDX**
- **Bit 5 – END_TARG (IDXENT)**
- **Bit 4 – POS_TARG (IDXPOT)**
- **Bit 3 – CAPTURE**
- **Bit 2 – POS_TRIP**
- **Bit 1 – MSDT_DN**
- **Bit 0 – FAULT**

When these bits are set, an incoming interrupt of the same name will set the corresponding flag and generate a processor interrupt.

STEP CLOCK Registers

SCIO (Step Clock I/O)

Bit	7	6	5	4	3	2	1	0	
0x023C	INV_DIR	INV_STP	–	SQ_CSEL	EN_SDO	VG_PINN	SEL_CLK1	SEL_CLK0	SCIO
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.37: Step Clock I/O Register

- **Bit 7 – INV_DIR**

This bit inverts the dir_io input / output and changes the direction of motor rotation. Exception: The direction of the motor will not change if VG_PINN=1.

- **Bit 6 – INV_STP**

This bit inverts the step_io input / output and makes the falling edge the active edge.

NOTE

NOTE: When Square Wave is selected, the last step of an index move will be the width specified in register SCSW.

- **Bit 5 – Reserved**
- **Bit 4 – SQ_CSEL**

This bit selects square wave step clock output. When set and the step_io pin is an output, square wave step clocks will be output. When cleared, register SCSW sets the step clock output width.

- **Bit 3 - EN_SDO**

This bit enables step_io and dir_io when Vg_pinn sets the pins as outputs. When set, enabled. When cleared, tristate.

- **Bit 2 – VG_PINN**

This bit selects the source of step clock and direction directed to the internal sine / cosine generator and defines the external pins step_io and dir_io an input or output

When set, the ASIC is used as an indexer or oscillator, the velocity generator sources step and direction to the Sine/Cosine Generator. The external pins are outputs.

When cleared and the following ratio is equal to one, the ASIC is used as a drive, the external pins step_io and dir_io are inputs, source, step and direction to the Sine/Cosine Generator.

When cleared and following ratio is not equal to one, the ASIC is used as a ratioed following drive, the velocity generator sources step and direction to the Sine/Cosine Generator in response to external pins step_io and dir_io which are inputs.

NOTE

NOTE: For quadrature; channel A leading B sets direction. For step up / down; step up sets direction when Inv_Dir is cleared.

- **Bits 1..0 – SEL_CLK 1..0**

These bits select clock type conversion. When the pins are outputs, the conversion type selects what clock type to convert to. When the pins are inputs, the conversion type selects what clock type to convert from.

Clock Type Conversions				
SEL_CLK Bits		Conversion Type	Step_IO	Dir_IO
Bit 1	Bit 0			
0	0	Step and Direction	Step	Direction
0	1	Quadrature	Channel A	Channel B
1	0	Step Up/Down	Step Up	Step Down

Table 7.38: Clock Type Conversions

SCSW (Step Clock Step Width)

NOTE

NOTE: This will also set the width of the last step of an index move if the bit sq_csel is set in register SCIO. SCSW values of 0, 1, 2 set the pulse width to 200 ns.

Bit	7	6	5	4	3	2	1	0	
0x023D	Step Width								SCSW
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.39: Step Clock Step Width Register

- **Bits 7..0 – Step Width**

This value sets the step clock width when pin step_io is an output. This also sets the output pulse width on the trip out pin.The width range is 100 nS to 12.85 µS with 50 nS resolution (0x0 to 0xFF).

SCRF (Step Clock Ratio Factor 1, 2, 3 & 4)

Bit	7	6	5	4	3	2	1	0	
0x023E	STEP CLOCK RATIO FACTOR 1 [7..0]								SCRF1
0x023F	STEP CLOCK RATIO FACTOR 2 [15..8]								SCRF2
0x0240	STEP CLOCK RATIO FACTOR 3 [23..16]								SCRF3
0x0241	STEP CLOCK RATIO FACTOR 4 [31..24]								SCRF4
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.40: Step Clock Ratio Factor Registers 1, 2, 3 & 4

- **Bits 7..0 – STEP CLOCK RATIO FACTOR 1**
- **Bits 15..8 – STEP CLOCK RATIO FACTOR 2**
- **Bits 23..16 – STEP CLOCK RATIO FACTOR 3**
- **Bits 31..24 – STEP CLOCK RATIO FACTOR 4**

This value is a factor used for setting the ratio to be used when following an external clock in drive mode.

Ratio factor = 0x02000000 x desired following ratio.

Acceptable following ratio range: 2 >= following ratio >= 0.001 (0x04000000 to 0x8312).

For a ratio of 1, ratio factor = 0x02000000.

The 32-bit word is automatically transferred when the high byte is written.

Miscellaneous Registers

IOF (Input/Output Filter)

Bit	7	6	5	4	3	2	1	0	
0x0242	FG1[3:0]				FG2[3:0]				IOF
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.41: Input/Output Filter Register

- **Bits 7..4 – FG1 - Filter group 1**

  This value sets group 1 filtering; capture input.

- **Bits 3..0 – FG2 - Filter group 2**

  This value sets group 2 filtering; step and direction inputs and encoder inputs. The filter range is 10 MHz to 38.8 KHz (0x0 to 0x9).

Filter Settings		
Filter Group Value	Minimum Detectable Pulse Width	Cutoff Frequency
0	50 ns	10 MHz
1	150 ns	3.3 MHz
2	200 ns	2.5 MHz
3	300 ns	1.67 MHz
4	500 ns	1 MHz
5	900 ns	555.5 kHz
6	1.7 μs	294.1 kHz
7	3.3 μs	151.5 kHz
8	6.5 μs	76.9 kHz
9	12.9 μs	38.8 kHz

Table 7.42: Input/Output Filter Range

MSELR (Microstep/Step Select Register)

Bit	7	6	5	4	3	2	1	0	
0x0243	INI_707	EN_SER_DAC	SPD_INTF	MSEL[4:0]					MSELR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.43: Microstep/Step Select Register

- **Bit 7 – INI_707**

This bit selects how the phases initialize. When set, the phases are initialized to the 0.707 (45 deg) position. If used, this should be set before pwm_ctrl. When cleared, the sine phase initializes to 0 (0 deg), the cosine phase to 1 (90 deg).

- **Bit 6 – EN_SER_DAC**

This bit selects external serial D/A interface. When set, the bit spd_intf selects the interface. The output pins may not be available in all packages. When cleared the internal DACs are used and the bit spd_intf selects the update speed.

- **Bit 5 – SPD_INTF**

This bit selects the update speed when the internal DACs are used or the interface when the external DACs are used.

NOTE

NOTE: External Serial DAC signals are not available on all packages.

DAC Select			
EN_SER_DAC	SPD_INTF	DAC	Update Speed
0	0	Internal DAC	200 kHz
0	1	Internal DAC	5 MHz
1	0	External DAC - LT1661	290 kHz
1	1	External DAC - TLV5617A	290 kHz

Table 7.44: DAC Table

- **Bits 4..0 – MSEL 4..0**

These bits select micro step resolution according to the following table.

NOTE

NOTE: All MSEL Bits 4...0 not shown in Table 7.45 will set the Microstep Resolution to 256 microstep/step.

Microstep Resolution Select															
MSEL Bits					Hex	Resolution (Microstep/Step)		MSEL Bits					Hex	Resolution (Microstep/Step)
4	3	2	1	0				4	3	2	1	0		
0	0	0	0	0	0x000	2 (Half Stepping)		0	1	0	1	0	0x00A	25*
0	0	0	0	1	0x001	4		0	1	0	1	1	0x00B	50
0	0	0	1	0	0x002	8		0	1	1	0	0	0x000C	125*
0	0	0	1	1	0x003	16		0	1	1	0	1	0x00D	250
0	0	1	0	0	0x004	32		1	0	0	0	0	0x010	1 (Full stepping)
0	0	1	0	1	0x005	64		1	0	0	0	1	0x011	180
0	0	1	1	0	0x006	128		1	0	0	1	0	0x012	100
0	0	1	1	1	0x007	256		1	0	0	1	1	0x013	127*
0	1	0	0	0	0x008	5*		1	0	1	0	0	0x014	200
0	1	0	0	1	0x009	10		1	0	1	0	1	0x015	108

* Can not switch to full stepping (1/1 - 0x010) From these Resolutions.

Table 7.45: Microstep/Step Select Table

STAT (Status)

Bit	7	6	5	4	3	2	1	0	
0x0244	OK_FOLO	ENC_A	ENC_B	VG_IDLE	ENC_IDX	CURRED	ATZERO	FAULT	STAT
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	1	0	0	0	0	

Table 7.46: M3000 Status Register

- **Bit 7 – OK_FOLO**

  When set, indicates when in ratioed following mode that it is okay to follow the incoming step clock. If ratio is not equal to 1, the velocity generator will start automatically when this bit sets.

- **Bit 6 – ENC_A**

  When set, indicates encoder channel A is asserted high.

- **Bit 5 – ENC_B**

  When set, indicates encoder channel B is asserted high.

- **Bit 4 – VG_IDLE**

  When set, indicates the velocity generator is idle (not moving).

- **Bit 3 – ENC_IDX**

  When set, indicates the encoder index mark is asserted high.

- **Bit 2 – CURRED**

  When set, indicates when the phase current has been reduced to the value specified in register CURIRED.

- **Bit 1 – ATZERO**

  When set, indicates when the sine or cosine phase is at 0 current.

- **Bit 0 – FAULT**

  When set, indicates cleared by SPWMCTL bit 4 or asserting reset_n. An external fault has been detected and latched. The bridge controls are disabled when a fault occurs.

SPWMCTL (Safety, PWM Control)

IMPORTANT note regarding inversion of bridge controls! It is preferable to use hardware pins (not available on all packages). Software inversion cannot be used with bridges requiring both high and low inversion because all FETs would be on “shoot through” until the software could set bits. A solution could be to externally use hardware inverters for the high controls and use software to invert the low controls. The low side of the bridge would be on (recirculate – a benign condition) until the software bit was set.

Bit	7	6	5	4	3	2	1	0	
0x0245	–	–	–	CLR_FLT	–	–	INV_HBC	INV_LBC	SPWMCTL
Read/Write	R	R	R	W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.47: Safety, PWM Control Register

- **Bits 7..5 Reserved**
- **Bit 4 – CLR_FLT**

  A self clearing bit used to clear a fault condition. Writing a 1 to this bit will restart the bridge controls. For safety, this bit will not override an incoming fault.

- **Bits 3..2 – Reserved**
- **Bit 1 – INV_HBC**

  This bit inverts the high side bridge controls. When set, the high side bridge controls are asserted low. This bit should be set before enabling bridge (bit en in register PWMCTL).

- **Bit 0 – INV_LBC**

  This bit inverts the low side bridge controls. When set, the low side bridge controls are asserted low. This bit should be set before enabling bridge (bit en in register PWMCTL).

Sine and Cosine DACs

Description

The Sine and Cosine DACs can receive data from the Sine/Cosine Generator or from the AVR.

In the automatic (default) mode, the Gain, Sine and Cosine data will be received from the Motor and Motion Control Logic. Only the enable signal (GNEN) from the DACCTRL Register must be set. (See the first figure below.)

The GAIN, SINE and COSINE may also be controlled by the AVR. This is accomplished by setting the GNEN and EXSCG bits in the DACCTRL Register. (See the second figure below.)

GAINDAC; SINDACL; SINDACH; COSDACL; COSDACH

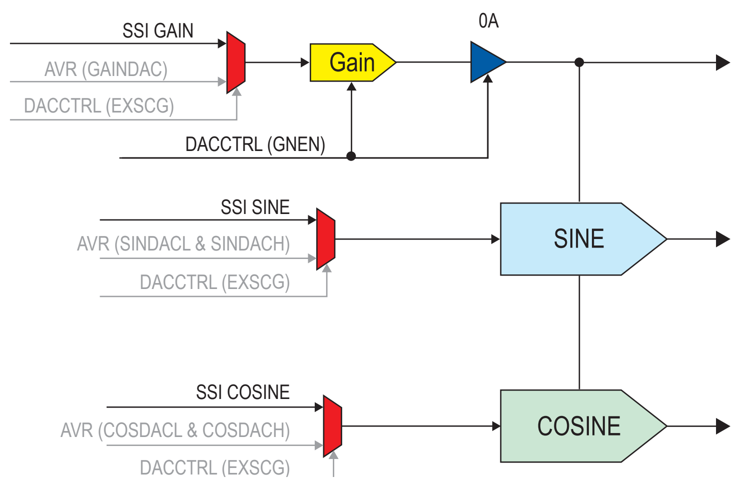


Figure 7.17: Default (Start-up) Sine and Cosine DACs Utilizing the External SSI Data Bits

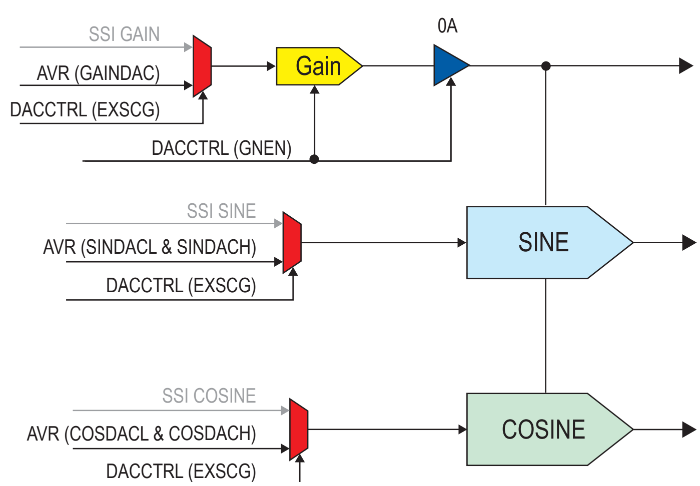


Figure 7.18: Sine and Cosine DACs Controlled by the Internal AVR Data Bits

DAC Registers

DACCTRL (D/A Converter Control)

Bit	7	6	5	4	3	2	1	0	
0x0285	–	–	–	–	EXSCG	–	GNEN	–	DACCTRL
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7.48: D/A Converter Control Register

- **Bits 7..4 – Reserved**
- **Bit 3 – EXSCG: Motor and Motion Control Gain**

When this bit is set, the DACs will use the AVR SIN, COS and GAIN values (from the SINDACL, SINDACH, COSDACL, COSDACH and GAINDAC registers). When cleared, the DAC's will use the SIN, COS values from the Sine/Cosine generator and the Gain value from the motor and motion control.

- **Bit 2 Reserved**
- **Bit 1 – GNEN: Gain Enable**

When this bit is set, the gain DAC is turned on. When this bit is cleared, the gain DAC is disabled.

- **Bit 0 – Reserved**

SINDACL (Sine D/A Converter Low Bits)

Bit	7	6	5	4	3	2	1	0	
0x0280	–	–	–	–	–	–	SINDAC[1:0]		SINDACL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
Initial Value	0	0	0	0	0	0	0		

Table 7.49: Sine D/A Converter Low Bits Register

- **Bits 7..2 – Reserved**

These bits are reserved and always read as zero.

- **Bits 1..0 – SINDAC[1:0]:**

These bits represent bits 1..0 of the sine DAC value.

SINDACH (Sine D/A Converter High Bits)

NOTE: SINDACH should be written first with SINDACL written second.

Bit	7	6	5	4	3	2	1	0	
0x0281	SINDAC[9:2]								SINDACH
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
Initial Value	0	0	0	0	0	0	0		

Table 7.50: Sine D/A Converter High Bits Register

- **Bits 7..0 – SINDAC[9:2]:**

These bits represent bits 9..2 of the sine DAC value.

COSDACL (Cosine D A Converter Low Bits)

Bit	7	6	5	4	3	2	1	0	
0x0282	–	–	–	–	–	–	COSDAC[1:0]		COSDACL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
Initial Value	0	0	0	0	0	0	0		

Table 7.51: Cosine D/A Converter Low Bits Register

- **Bits 7..2 – Reserved**

  These bits are reserved and always read as zero.

- **Bits 1..0 – COSDAC[1:0]:**

  These bits represent bits 1..0 of the cosine DAC value.

COSDACH (Cosine D/A Converter High Bits)

NOTE

NOTE: COSDACH should be written first with COSDACL written second.

Bit	7	6	5	4	3	2	1	0	
0x0283	COSDAC[9:2]								COSDACH
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
Initial Value	0	0	0	0	0	0	0		

Table 7.52: Cosine D/A Converter High Bits Register

- **Bits 7..0 – COSDAC[9:2]:**

  These bits represent bits 9..2 of the cosine DAC value.

GAINDAC (D/A Converter Gain)

Bit	7	6	5	4	3	2	1	0	
0x0284	GAINDAC[7:0]								GAINDAC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W		
Initial Value	0	0	0	0	0	0	0		

Table 7.53: D/A Converter Gain Register

- **Bits 7..0 – GAINDAC[7:0]:**

  These bits represent bits 7..0 of the gain DAC value.

Page Intentionally Left Blank

SECTION 8
INTERRUPTS

Interrupt Registers and Functions

Several of the peripherals use interrupts to inform the AVR of completion of an operation and the System Semiconductor Motion Controller ASIC has two pins for external interrupt generation. The AVR has 7 interrupt inputs with associated vector addresses for servicing each interrupt line.

Interrupts are grouped according to function in capture registers accessible by the AVR. The bits in these registers will then be 'or'ed together to create a signal which will drive the AVR interrupt inputs. Thus when an interrupt is detected by the AVR, the software routine handling the interrupt must read the interrupt capture register corresponding to the interrupt line to determine which peripheral function generated the interrupt.

The following table describes the interrupt capture registers and the corresponding priority of the generated interrupt. Priority 0 is highest and the priority corresponds to the AVR interrupt vector - 1. NOTE: Priority 0 and Priority 1 are not controlled by the Global Interrupt Enable bit "I" in the Status Register (SREG).

Interrupt Capture Registers

Interrupt Capture Register Summary					
Data Space	Register Name	Priority	Description	Clear	Mask Register
0x033	MCSR[3:1]	0	External Reset, Watchdog Timer, Reset Non-Existent Memory Reset.	Manual	WDTCR
0x016	GIFR[7:5]	1	Captures External Interrupt Inputs and SSI Motor and Motion Logic Interrupt	Manual	GIMSK
0x80A	INTIFR	2	Captures Internal Timer Interrupts	Manual	INTIMSK
0x104	CMCR[6]	3	Captures CAN Interrupts	Manual	CMCR
0x016	GIFR[4:2]	4	Captures UART Interrupts	Auto/Manual	UCRA
0x016	GIFR[1:0]	5	Captures Master/Slave SPI Interrupts	Manual	MSPCR, USPCR
0x02A	EXTIFR	6	Captures External Timer Inputs	Manual	EXTIMSK
0x007	ADCSR[4]	7	Captures A/D Conversion Interrupts	Manual	ADCSR

Table 8.1: Interrupt Capture Registers

These interrupts and the separate reset vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt vectors. The complete list of vectors is shown in the Interrupt Vector Table on the following page.

The list also determines the priority levels of the different interrupts. The lower the address the higher is

External Interrupts

External interrupts are implemented with programmable polarity meaning that each interrupt line is individually selectable for a rising or falling edge detection to generate an interrupt. Bits [1:0] in the GIMSK register control this functionality (see register description) and default to falling edge detect. To cause an interrupt, one of the EXT_INT0 or EXT_INT1 pins must be held low for two 20MHz clock cycles to enable proper demetastabilization of the input pins. Note that the interrupts are edge sensitive and that holding either EXT_INT0 or EXT_INT1 pin low will not cause multiple interrupts.

Interrupt Clearing

Interrupts that require manual clearing are cleared by writing logic '1' to the corresponding bit in the interrupt flag register. Automatic clearing of the UART RXIF and UDREIF flags in the GIFR occur when the UDR is read (clears RXIF) or the UDR is written (clears UDREIF).

Typical Code for Initializing Interrupts

The most typical and general program setup for the Reset and Interrupt Vector Addresses is as follows.

Code Space

Address	Code	Labels	Comments
0x0000	jmp	RESET	; Reset Handler
0x0002	jmp	IDLIF	; Motor & Motion Ctrl. Interrupt
0x0004	jmp	EXT_INTF1	; External Interrupt Flag 1
0x0006	jmp	EXT_INTF0	; External Interrupt Flag 0
0x0008	jmp	CAN_INT	; CAN Interrupt Flags
0x000A	jmp	UART_INT	; UART Interrupt Flags
0x000C	jmp	SPI_INT	; SPI Master/Slave Interrupt Flags
0x000E	jmp	ADC_INT	; A/D Conversion Interrupt Flags

Interrupt Handling

The M3000 has three Interrupt Mask control registers:

1. GIMSK – General Interrupt Mask Register
2. EXTIMSK – External Timer/Counter Interrupt Mask Register
3. INTIMSK - Internal Timer/Counter Interrupt Mask Register.

NOTE

NOTE: The Status Register is not automatically stored when entering an interrupt routine, or when returning from an interrupt routine. This is handled by software. When using the CLI (Clear Global Interrupt) instruction to disable interrupts, the interrupts will be immediately disabled except RESET and GIFR[7:5]. Only the interrupts RESET and GIFR[7:5] can be executed after the CLI instruction, even if they occur simultaneously with the CLI instruction. No other interrupts will be executed if CLI has been set.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the interrupt flag. For these interrupts, the Program Counter is vectored to the actual interrupt vector in order to execute the interrupt handling routine, and hardware clears the corresponding interrupt flag. Interrupt flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the interrupt flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the global interrupt enable bit is cleared, the corresponding interrupt flag(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present.

These interrupts do not necessarily have interrupt flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered. When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles, the program vector address for the actual interrupt handling routine is executed. During this 4-clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in Sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these 4-clock cycles, the Program Counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set.

Interrupt Registers

GIMSK (General Interrupt Mask)

	Motor	Externals							
Bit	7	6	5	4	3	2	1	0	
0x17(0x37)	IDL	INT1	INTØ	–	–	–	POL1	POLØ	GIMSK
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 8.2: General Interrupt Mask Register

- **Bit 7 – IDL: Motor and Motion Control Logic Interrupt Enable**

  When this bit is set the Motor and Motion Control Logic interrupt is activated.

- **Bit 6 – INT1: External Interrupt Request 1 Enable**

  When this bit is set the External pin 1 interrupt is activated.

- **Bit 5 – INT0: External Interrupt Request 0 Enable**

  When this bit is set the External pin 0 interrupt is activated.

- **Bits 5..2 Reserved Bits**

  These bits are reserved and always read as zero.

- **Bit 1 – POL1: External Interrupt Polarity**

  When this bit is set a rising edge on the External Interrupt Pin 1 causes an interrupt. When this bit is cleared, a falling edge on the External Interrupt Pin 1 causes an interrupt.

- **Bit 0 – POL0: External Interrupt Polarity**

  When this bit is set a rising edge on the External Interrupt Pin 0 causes an interrupt. When this bit is cleared, a falling edge on the External Interrupt Pin 0 causes an interrupt.

NOTE

NOTE: Do to an internal interaction when POL0 is LOW, to POL1 circuitry, certain external interrupt configurations are not acceptable.

POL1	POL0	Comment
1	1	Acceptable Configuration
0	1	" "
1	0	Not Acceptable Configuration
0	0	" "

GIFR (General Interrupt Flag)

NOTE: GIFR[7:5] will not be controlled by the Global Interrupt Enable bit "I" in the Status Register (SREG).

	Motor	Externals		UART			SPI		
Bit	7	6	5	4	3	2	1	0	
0x16(0x36)	IDLIF	INTF1	INTFØ	RXIF	TXCIF	UDREIF	MSPIF	USPIF	GIFR
Read/Write	R	R/W	R/W	R	R/W	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 8.3: General Interrupt Flag Register

- **Bit 7 – IDLIF: Motor and Motion Control Logic Interrupt Flag.**

  When this bit is set the Motor and Motion Control Logic interrupt is activated.

- **Bit 6 – INTF1: External Interrupt Request Flag 1**

  This bit is set when a selected edge occurs on the INT1 pin. The bit is cleared by writing a logical one to it.

- **Bit 5 – INTF0: External Interrupt Request Flag 0**

  This bit is set when a selected edge occurs on the INT0 pin. The bit is cleared by writing a logical one to it.

- **Bit 4 – RXIF: RX Complete Interrupt Flag**

  When this bit is set, a Receive Complete Interrupt request has occurred. This bit is cleared according to internal UART functions.

- **Bit 3 – TXCIF: TX Complete Interrupt Flag**

  When this bit is set, a Transmit Complete Interrupt request has occurred. This bit is cleared by writing logic one to the flag.

- **Bit 2 – UDREIF: UART Data Register Empty Interrupt Flag**

  When set, a UART Data Register Empty Interrupt request has occurred. This bit is cleared according to internal UART functions.

- **Bit 1 – MSPIF: Master SPI Interrupt Flag**

  When this bit is set, an interrupt request from the Master SPI has occurred. This bit is cleared by writing logic one to the flag.

- **Bit 0 – USPIF: User SPI Interrupt Flag**

  When this bit is set, an interrupt request from the User SPI has occurred. This bit is cleared by writing logic one to the flag.

EXTIMSK (External Timer/ Counter Interrupt Mask)

Bit	7	6	5	4	3	2	1	0	
0x2B(0x4B)	TOIE1	OCIE1A	OCIE1B	–	TICIE1	–	TOIEØ	–	EXTIMSK
Read/Write	R/W	R/W	R/W	R	R/W	R	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 8.4: External Timer/Counter Interrupt Mask Register

- **Bit 7 – TOIE1: Timer/Counter 1 Overflow Interrupt Enable**

  When the TOIE1 bit is set, an Overflow interrupt is enabled for the External Timer.

- **Bit 6 – OCIE1A: Output Compare Interrupt Enable 1A**

  When the OCE1A bit is set, an Output Compare A interrupt is enabled for the External Timer.

- **Bit 5 – OCIE1B: Output Compare Interrupt Enable 1B**

  When OCIE1B bit is set, an Output Compare B interrupt is enabled for the External Timer.

- **Bit 4 Reserved**

  This bit is reserved and always read as zero.

- **Bit 3 – TICIE1: Timer/Counter 1 Input Capture Interrupt Enable**

  When TICIE1 bit is set, an Input Capture interrupt is enabled for the External Timer.

- **Bit 2 Reserved**

  This bit is reserved and always read as zero.

- **Bit 1 – TOIE0: Timer/Counter 0 Overflow Interrupt Enable**

  When the TOIE0 bit is set, an Overflow interrupt is enabled for the External Timer.

- **Bit 0 Reserved**

  This bit is reserved and always read as zero.

EXTIFR (External Timer/ Counter Interrupt Flag)

Bit	7	6	5	4	3	2	1	0	
0x2A(0x4A)	TOV1	OCF1A	OCF1B	–	ICF1	–	TOVØ	–	EXTIFR
Read/Write	R/W	R/W	R/W	R	R/W	R	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 8.5: External Timer/Counter Interrupt Flag Register

- **Bit 7 – TOV1: Timer/Counter 1 Overflow Flag**

  TOV1 is set when an overflow occurs in Timer/Counter1. TOV1 is cleared by writing logic one to the flag.

- **Bit 6 – OCF1A: Output Compare Flag 1A**

  When this bit is set, a compare match has occurred between the Timer/Counter1 and the data in OCR1A - Output Compare register 1A. OCF1A is cleared by writing logic one to the flag.

- **Bit 5 – OCF1B: Output Compare Flag 1B**

  When this bit is set, a compare match has occurred between the Timer/Counter1 and the data in OCR1B - Output Compare register 1B. OCF1B is cleared by writing logic one to the flag.

- **Bit 4 Reserved**
- **Bit 3 – ICIF1: Input Capture Flag 1**

  When set, an input capture event has occurred, indicating that the Timer/Counter1 value has been transferred to the input capture register – ICR1. ICF1 is cleared by writing logic one to the flag.

- **Bit 1 TOV0**

  The TOV0 is set when an overflow occurs in Timer/Counter0. TOV0 is cleared by writing logic one to the flag.

- **Bit 0 Reserved**

INTIMSK (Internal Timer/ Counter Interrupt Mask)

Bit	7	6	5	4	3	2	1	0	
0x80B	TOIE1	OCIE1A	OCIE1B	–	TICIE1	–	TOIEØ	–	INTIMSK
Read/Write	R/W	R/W	R/W	R	R/W	R	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 8.6: Internal Timer/Counter Interrupt Mask Register

- **Bit 7 – TOIE1: Timer/Counter 1 Overflow Interrupt Enable**

  When TOIE1 is set, an Overflow interrupt is enabled for the Internal Timer.
- **Bit 6 – OCIE1A: Output Compare Interrupt Enable 1A**

  When OCE1A is set, an Output Compare A interrupt is enabled for the Internal Timer.
- **Bit 5 – OCIE1B: Output Compare Interrupt Enable 1B**

  When OCIE1B is set, an Output Compare B interrupt is enabled for the Internal Timer.
- **Bit 4 Reserved**

  This bit is reserved and always read as zero.
- **Bit 3 – TICIE1: Timer/Counter 1 Input Capture Interrupt Enable**

  This bit should be cleared (logic zero) at all times since input capture is not available in the Internal Timer.
- **Bit 2 Reserved**

  This bit is reserved and always read as zero.
- **Bit 1 – TOIE0: Timer/Counter 0 Overflow Interrupt Enable**

  When TOIE0 is set, an Overflow interrupt is enabled for the Internal Timer.
- **Bit 0 Reserved**

  This bit is reserved and always read as zero.

INTIFR (Internal Timer Counter Interrupt Flag)

Bit	7	6	5	4	3	2	1	0	
0x80A	TOV1	OCF1A	OCF1B	–	–	–	TOVØ	–	INTIFR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 8.7: Internal Timer/Counter Interrupt Flag Register

- **Bit 7 – TOV1: Timer/Counter 1 Overflow Flag**

  The TOV1 is set when an overflow occurs in Timer/Counter1. TOV1 is cleared by writing logic one to the flag.
- **Bit 6 – OCF1A: Output Compare Flag 1A**

  When this bit is set, a compare match has occurred between the Timer/Counter1 and the data in OCR1A- Output Compare register 1A. OCF1A is cleared by writing logic one to the flag.
- **Bit 5 – OCF1B: Output Compare Flag 1B**

  When this bit is set, a compare match has occurred between the Timer/Counter1 and the data in OCR1B Output Compare register 1B. OCF1B is cleared by writing logic one to the flag.
- **Bits 4..2 Reserved**

  These bits are reserved and always read as zero.
- **Bit 1 TOV0**

  The TOV0 is set when an overflow occurs in Timer/Counter0. TOV0 is cleared by writing logic one to the flag.
- **Bit 0 Reserved**

  This bit is reserved and always read as zero.

SECTION 9
INSTRUCTION MEMORY PROGRAMMING

Introduction

This section describes the software architecture for the Boot Monitor contained in the Motion Controller ASIC (MCA).

Description

The boot code is the first software to execute after any reset condition. It will provide the following functions:

- Run internal self tests.
- Load application if required by External Execution Memory.
- Determine location of Warm Boot Loader or Application (either Serial FLASH or PARALLEL FLASH).
- Copy the application SERIAL FLASH to SRAM.
- Validate code.
- Write an application to PARALLEL FLASH memory upon command.
- Write an application to SERIAL FLASH upon command.
- Provide Interface to Warm Boot Loader or Application to perform Write and Erase Program Memory Operations.

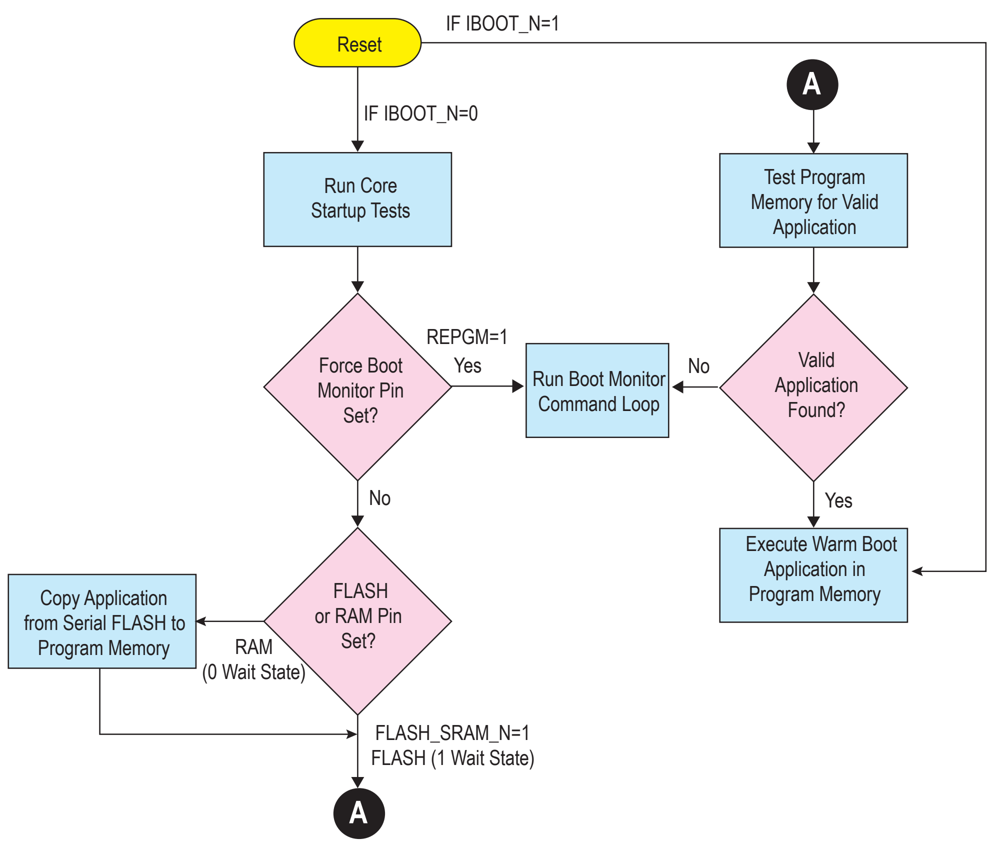


Figure 9.1: Boot Code Flow Chart

Definition of Software Code Types

Boot Loader

The Boot Loader handles initialization, programming utilities and testing. It is hard coded in the M3000.

Warm Boot Loader (Optional)

A software structure could use a warm boot loader. The warm boot loader resides in the bottom boot sector of program memory. The warm boot loader takes responsibility for programming the user application using the programming utilities in the boot loader.

Application Code

The Application Code resides at the start of program memory (bottom) or at any location above the boot sector when a warm boot loader is used.

System Memory Configurations

Normal

The Normal System Memory Configuration (Parallel Flash, Figure 9.2):

- Operates on a 20 MHz AVR processor with 1 wait state for program memory access.
- Stores the Warm Boot Loader and/or Application Code in 64K x 16 parallel FLASH.
- Executes code from the Parallel FLASH.
- Can utilize an optional external Serial EEPROM as storage for user programs, variables and other data.

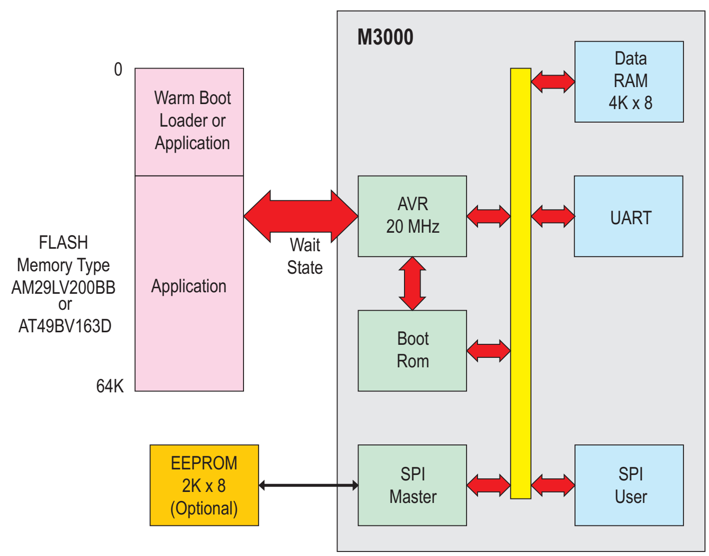


Figure 9.2: Normal System Memory Configuration

High Speed (With Expanded Ram Capabilities)

The High Speed System Memory Configuration (Figure 9.3):

- Operates on a 20 MHz AVR processor with no wait states for program memory access.
- Stores the Warm Boot Loader and/or Application Code in 1Mbit Serial FLASH.
- Executes code from External RAM.

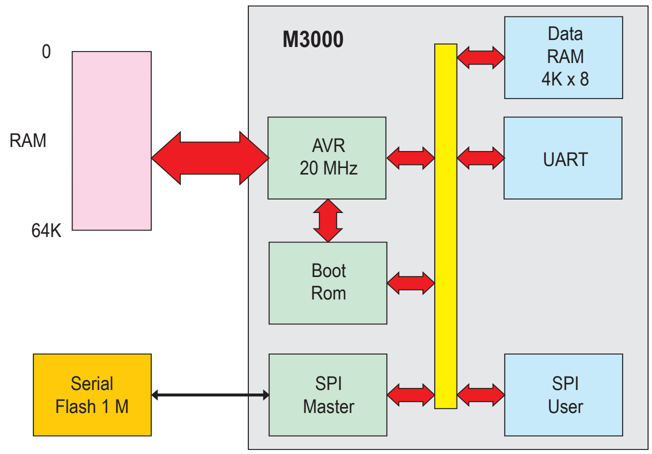


Figure 9.3: High Speed System Memory Configuration

Data Memory Register Files

When using register indirect addressing modes with automatic pre-decrement and postincrement, the address registers X, Y and Z are decremented or incremented. The 32 general purpose working registers, 64 I/O Registers, and the 4096 bytes of internal data SRAM in the System Semiconductor M3000 are all accessible through these addressing modes.

Register File	Data Address Space
R0	0x0000
R1	0x0001
R2	0x0002
R29	0x001D
R30	0x001E
R31	0x001F

I/O Registers	
0x00	0x0020
0x01	0x0021
0x02	0x0022
....	....
0x3D	0x005D
0x3E	0x005E
0x3F	0x005F

Data SRAM Space	
CAN Interface Registers	0x0100 | 0x01FF
Motor and Motion Control Registers	0x0200 | 0x02FF
Unused	0x0300 | 0x03FD
Reserved	0x03FE
Reserved	0x03FF
Unused	0x0400 | 0x07FF
Internal Timer Registers	0x0800 | 0x080B
Unused	0x080C | 0x0FFF
Internal Data RAM	0x1000 | 0x2FFF
External RAM 256 or 4K (When Selected)	0x2000 | 0x2FFF
Unused	0x3000 | 0xFFFF

Table 9.1: Data Memory Map

Data Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk CPU cycles as described in the figure below.

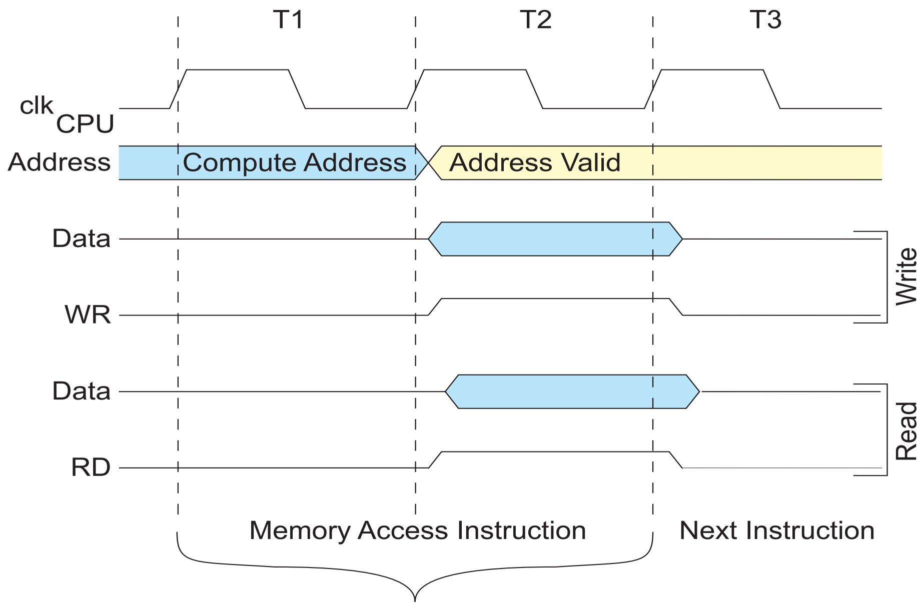


Figure 9.4: Data Memory Access Times

I/O Memory

The I/O locations are accessed by the IN and OUT instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.

Refer to the Program Instruction Set /Reference for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

The I/O and peripherals control registers are explained in later sections.

Programming and Operating Scenarios

Boot (New Production or Reprogram)

Cold Boot Events		
Memory Configuration	Software Code Type	Basic Actions
Normal	Boot Loader	1. Perform Startup Tests. 2. Detect *reprog_mem* pin. 3. Communicating through UART or User SPI an external utility uses the Cold Boot Loader commands to program the boot sector of the Parallel FLASH with the Warm Boot Loader or application.
High Speed	Boot Loader	1. Perform Startup Tests. 2. Detect *reprog_mem* pin. 3. Communicating through UART or User SPI an external utility uses the Cold Boot Loader commands to program the boot sector of the Serial FLASH with the Warm Boot Loader or application.

Table 9.2: Cold Boot Table of Events

Operation (Warm Boot)

Operation Warm Boot Events		
Memory Configuration	Software Code Type	Basic Actions
Normal	Boot Loader	1. Perform Startup Tests. 2. Detect *flash_sramin* pin. 3. Test for valid App Code in Flash and send control to the Warm Boot Loader or application.
	Warm Boot Loader or Application	1. Housekeeping. 2. Operate
High Speed	Boot Loader	1. Perform Startup Tests. 2. Detect *flash_sramin* pin. 3. Transfer code from Serial FLASH to Parallel RAM. 4. Test for valid App Code in Flash and send control to the Warm Boot Loader or application.
	Warm Boot Loader or Application	1. Housekeeping. 2. Operate

Table 9.3: Operation Warm Boot Table of Events

Architecture

Power Up Sequence

The boot monitor will perform a self-check sequence to ensure proper operation of the AVR core and the ability to write and read internal registers properly.

Internal Tests

The Boot Monitor will run a series of Internal Tests on the M3000 to verify that all blocks and circuits are operational.

Forced Boot Monitor Execution

The boot monitor will check the external pin REPGM_PGM_MEM to see if forced boot monitor operation is selected. This is to ensure that even though a valid application exists, operation will continue in boot mode.

Application Code Locale Determination

The boot monitor will then check the PARALLEL FLASH_SRAMN pin to determine if external PARALLEL FLASH or SRAM is connected. If SRAM is present, the boot monitor will attempt to load code to the SRAM from the SERIAL FLASH connected via the SPI Port.

Validation of Application Code

The boot monitor will then determine if a valid application exists by performing a checksum of the code and comparing the checksum located in the Image Footer. If a valid application is found, the code will then be executed; otherwise, execution of the boot monitor command mode will commence.

An Image Footer is used by the Cold Boot code to determine the length and expected checksum. The contents of the Image Footer are shown in Table 9.4 below.

Image Footer Contents		
Field	Length (Bytes)	Description
Signature	4	Digital Signature = 0xC53AA35C
Image Length	2	# of Words in the Image
Image Checksum	2	Computed Checksum (MSW)
	2	Computed Checksum (LSW)

Table 9.4: Image Footer Contents

The Image Footer is located at different locations depending on the size of the boot code. It can be located at any of the following addresses: 0x1FFB, 0x3FFB, 0x5FFB, 0x7FFB, 0x9FFB, 0xBFFB, 0xDFFB or 0xFFFB. The boot code determines the location by checking for the Digital Signature at the start of the header. The Boot code searches the above memory locations starting from 0x1FFB and working up. The First location with the correct digital signature will be used.

Program Memory Functions

The Boot Code is designed so that different memory devices can be used for the Program memory space. Typically these will either be an SRAM [the Application code is stored in a SERIAL FLASH] or a PARALLEL FLASH. Since programming algorithms for the PARALLEL FLASH parts are different for each vendor and memory type, they cannot be stored in the Boot ROM. Instead, a scheme that uses basic programming instructions is provided in the Cold Boot Loader.

There are three data structures used to perform the programming algorithms on the Program Memory. These are shown in the figure on the following page. The Program Memory Function Table contains an encoding of the programming algorithm to be used. The Function Data Buffer contains the data that is used by the algorithm. The Function Response Buffer contains any output data generated by the algorithm.

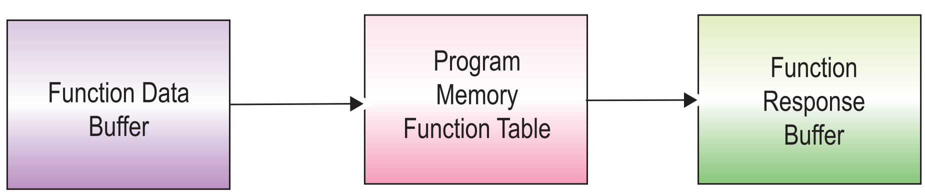


Figure 9.5: Program Memory Function Data Structure

Program Memory Function Table

The Boot code makes one very basic assumption about the Program Memory Device. That all device required functions can be broken down to a set of operations each consisting of either a Memory Read, a Memory Write or a Time Delay. These operations will be stored in RAM in the Program Memory Function Table. Each entry in the table will correspond to one Program Memory Operation. The Set of these operations is combined in the Program Memory Function Table and is capable of performing one High-Level function specific to the memory device. The Typical Functions used on the Program Memory would be Parallel FLASH Erase, Parallel FLASH Write, or SRAM Write, but could also consist of Read Vendor ID etc.

The OPERATION Field is an 8-bit value and contains the FLASH operation to perform. The full list of supported Program Memory Operations is shown in the table below. Each Program Memory Operation can use up to 2 arguments stored in the entry. These arguments are 16-bit unsigned values shown as OP_ARG1 and OP_ARG2.

Program Memory Options		
Mnemonic	Opcode	Description
START_OF_CMD	0x00	Required Table Header.
READ	0x01	Read PGM Memory.
RD_ARG	0x02	Read PGM Memory with Address passed as an argument when command is executed.
WRITE	0x03	Writes PGM Memory.
WR_ARG_ADDR	0x04	Writes PGM Memory with Address passed as an argument when command is executed.
WR_ARG_DATA	0x05	Writes PGM Memory with Data passed as an argument when command is executed.
WR_ARG_ARG	0x06	Writes PGM Memory with Address and Data passed as an argument when command is executed.
PAUSE	0x07	Pauses Execution for a set time.
VERIFY	0x08	Verifies that the Address and Data passed as arguments is written properly.
VERIFY_WRITE	0x09	Verifies that the previous Write Operation reads back the same data that was written.
END_OF_CMD	0x0A	Required Table Footer.

Table 9.5: Program Memory Operations

Program Memory Function

The Program Memory Function allows programming of these types of memory:

- Parallel FLASH.
- Serial FLASH, to be later transferred to SRAM.
- Directly to SRAM.

To write or read memory, via the Cold Boot Loader, there are eleven (11) commands available. (See Table 9.5)

Each entry in the Program Memory Function Table consists of three fields and is stored in memory in the following format:

OPCODE: Byte

ARG1: Word

ARG2: Word

To write to Parallel FLASH, the following sequence of commands must be sent to the Cold Boot Loader.

Cold Boot Loader Commands			
Opcode	ARG1	ARG2	Comments
START_OF_CMD	FUNC_WRITE_FLASH	0x0004	Function name is a number 0x0000 to 0xFFFF.
WRITE	0x5555	0x00AA	Sequence to Unlock.
WRITE	0x2AAA	0x0055	Flash to Allow Writes.
WRITE	0x5555	0x00A0	
WR_ARG_ARG	0x0000	0x0000	Looks for Address and Data in FuncData Array.
VERIFY_WRITE	0x0002	0x0000	ARG1 is the number of milliseconds to wait for a response.
END_OF_CMD	0x0000	0x0000	Command is finished.

Table 9.6: Cold Boot Loader Commands

To execute the above command, send the following command to the Cold Boot Loader for each Address and Data.

```
EXE_PGM_MEM_FUNC, FUNC_WRITE, ADDR, DATA
```

To send the above commands to the Cold Boot Loader, either via the UART or the SPI, they must be sent as follows:

- Formatted with a Message Header (<10><SOH>) prefix.
- Followed by a Cold Boot Loader command <20>.
- The Function.
- A message trailer (<10><EOT>) suffix.
- The entire message will be followed with a one-byte 2's complement checksum.

NOTE: 1. If a byte should be <10>, it should be prefixed with another <10>.
2. Words are sent LSB first, followed by the MSB.
3. <XX> represents one byte in hexadecimal notation.

<10><SOH><20><00><00><10><10><04><00><03><55><55><AA><00><03><AA>

<2A><55><00><03><55><55><A0><00><06><00><00><00><00><09><02><00>

<00><00><00><00><00><00><00><10><EOT><CKSUM>

CKSUM = ((SUM(<10>...<EOT>) * -1) AND <FF>) = <B6>

The Program Memory Function also allows applications to have the ability to Read and Write to FLASH via similar access parameters. To Write or Read FLASH, a table must be created in RAM. This table will be similar to the Function in the table on the previous page.

To create a typed array, and fill it with the required data, perform the following. Please see the SSI CD for sample programs on:

- Functions for Writing to and Reading FLASH in "C".
- Headers to define Global Definitions in "C".
- Headers to define M3000 Registers and Bits in "C".

C Examples:

```
// Create a type definition
typedef  struct
{
    uchar ucCMD;
    ushort usARG1;
    ushort usARG2;
} BT_FUNC;

// create an array to hold the commands
BT_FUNC btfWr_Flash[7];
// create an array to hold the function data
ushort usFuncData[3];
// create an array to hold the function response
uchar ucFuncRsp[10];
// create a variable to hold the number of function bytes
uchar ucFuncBytes;

// fill the array
btfWr_Flash[0].ucCMD = START_OF_CMD;
btfWr_Flash[0].usARG1 = FUNC_WRITE;
btfWr_Flash[0].usARG2 = 0x0004;
          •
          •
          •
btfWr_Flash[0].ucCMD = END_OF_CMD;
btfWr_Flash[0].usARG1 = 0x0000;
btfWr_Flash[0].usARG2 = 0x0000;

/*
** ucExecutePgmMemFunc
*  FILENAME:
*  PARAMETERS: Function Table pointer, Data Table
*  pointer, Response Buffer pointer, numbytes pointer
*  DESCRIPTION: Sets up and executes Command Function
*  RETURNS: pointer to results
**/
uchar ucExePgmMemFunc(uchar *pTbl, uchar *pFuncData, uchar
*pRspBuf, / uchar *pNumBytes)
                      {
//Setup global memory.
*FUNC_TBL_PTR_L = ((ushort)pTbl)&0xFF;
*FUNC_TBL_PTR_H = ((ushort)pTbl)>>8;

*FUNC_DATA_PTR_L = ((ushort)pFuncData)&0xFF;
*FUNC_DATA_PTR_H = ((ushort)pFuncData)>>8;

*FUNC_RSP_PTR_L = ((ushort)pRspBuf)&0xFF;
*FUNC_RSP_PTR_H = ((ushort)pRspBuf)>>8;

*FUNC_NUM_BYTES = (*pNumBytes)&0xFF;

// Save All Registers (change to suit your compiler)
asm("push r0  \n\t push r1  \n\t push r2  \n\t push r3");
asm("push r4  \n\t push r5  \n\t push r6  \n\t push r7;");
asm("push r8  \n\t push r9  \n\t push r10 \n\t push r11;");
asm("push r12 \n\t push r13 \n\t push r14 \n\t push r15;");
asm("push r16 \n\t push r17 \n\t push r18 \n\t push r19;");
asm("push r20 \n\t push r21 \n\t push r22 \n\t push r23;");
asm("push r24 \n\t push r25 \n\t push r26 \n\t push r27;");
asm("push r28 \n\t push r29 \n\t push r30 \n\t push r31;");
```

```
// Load Registers for function call
asm("lds r26, 0x800 \n\t lds r27, 0x801");
asm("lds r28, 0x804 \n\t lds r29, 0x805");
asm("lds r30, 0x802 \n\t lds r31, 0x803");
asm("lds r2,  0x806");

//Switch To Cold Boot Code Context
asm("ldi r16, 0 \n out 0x04, r16");
asm("call 0x0020");
asm("sts 0x807,r17");
asm("sts 0x806,r2");

// Restore All Registers (change to suit your compiler)
asm("pop r31 \n\t pop r30 \n\t pop r29 \n\t pop r28");
asm("pop r27 \n\t pop r26 \n\t pop r25 \n\t pop r24");
asm("pop r23 \n\t pop r22 \n\t pop r21 \n\t pop r20");
asm("pop r19 \n\t pop r18 \n\t pop r17 \n\t pop r16");
asm("pop r15 \n\t pop r14 \n\t pop r13 \n\t pop r12");
asm("pop r11 \n\t pop r10 \n\t pop r9  \n\t pop r8");
asm("pop r7  \n\t pop r6  \n\t pop r5  \n\t pop r4");
asm("pop r3  \n\t pop r2  \n\t pop r1  \n\t pop r0");


*pNumBytes = *FUNC_NUM_BYTES;

return (*FUNC_STATUS);
}

// to use  the above function in code, then create a function similar to
this one:
/*
** usWrite_Flash
*  FILENAME: *
*  PARAMETERS: Address, Data
*  DESCRIPTION: Writes Data into Flash at Address
*  RETURNS: Fault Code
**/
uchar ucWrite_Flash(ushort usAddress, ushort usData)
{
uchar ucFault;

// Use the Function Table to Program the Flash.
usFuncData[0] = FUNC_WRITE_FLASH;
usFuncData[1] = usAddress;
usFuncData[2] = usData;
ucFuncBytes = 4;

    ucFault = ucExePgmMemFunc((uchar*)btfWr_Flash,(uchar*)usFuncData,ucFuncRs
p,&ucFuncBytes)
    return(ucFault);
}
//end of c code example
```

See the System Semiconductor CD for a complete listing of examples.

Program Memory Function Data Buffer

The Program Memory Function Data Buffer contains the data that will be used by the algorithm in the Function Table. The format of this buffer is shown in Table 9.7. The first two bytes in this buffer contain the Function Name that is to be executed. This MUST match the function name that is stored in the Function Table. The contents of the rest of the buffer are specific to the Function that is being performed. Data is taken from the buffer, as needed, by the each program memory operation.

FUNC_NAME_L
FUNC_NAME_H
Data 0
Data 1
...
Data N

Table 9.7: Program Memory Function Data Buffer Format

Program Memory Function Response Buffer

The Program Memory Response Buffer contains the data that is generated by execution of the Program Memory Function Table. The format of this buffer is shown in Table 9.8. All read operations, either READ or RD_ARG, will generate 2 bytes of data. This data is stored to the response buffer in the order that was read.

Data 0
Data 1
...
Data N

Table 9.8: Program Memory Response Buffer Format

Program Memory Command Example

An example of an entire Function Sequence is shown below. This example shows how to write address 0x1234 to 0x5678.

- First Setup the Function Table as described in the above example.
- Then setup the Function Data as:
- U16 funcData[3] = { FUNC_WRITE_FLASH, 0x1234, 0x5678}

This specifies that the function to Write FLASH will be performed at the address of 0x1234 with data of 0x5678.

Finally Execute The Command. This can be done either through the UART or the Boot Code API code designed for this purpose. In this case there are no Program Memory Reads so there will be no data stored to the response buffer. The function will however return a status of ERR_OK or ERR_PGM_MEM_TIMEOUT.

Program Memory Access from SSI Code and User Code

The Boot Code is designed so that the PROGRAM MEMORY Command Functions will be accessible to the other code images. This will be done through a single API call to the Function CALL_PROGRAM MEMORY_CMD. The Address of this API will be fixed as the memory location after the Final Interrupt Vector in the Interrupt Table.

Boot Monitor Command Mode

The Boot Monitor will support commands that allow the downloading and verification of application code. The host will communicate with the boot monitor via either

RS-232 serial port or the User SPI Port. A message based low-level protocol is used to ensure data integrity.

The commands supported by the boot monitor are shown in Table 9.9.

Commands

Cold Boot Loader Commands		
Mnemonic	Opcode	Description
SET_PGM_MEM_FUNC	0x20	Sets up Program Memory Function Table.
EXE_PGM_MEM_FUNC	0x21	Executes Program Memory Function Table.
SET_SER_PGM	0x22	Program Serial FLASH Device.
VER	0x23	Verify that a Valid Program Application is loaded into Parallel FLASH/SRAM.
START	0x24	Executes code loaded into Program Memory.
RESET	0x25	Resets the Boot Monitor.
XFER_SER_PGM	0x26	Copies Serial Memory into Program Memory.
ERASE_SER_PGM	0x27	Erase Serial FLASH.
RD_SER_PGM	0x28	Read Data from Serial FLASH.

Table 9.9: Cold Boot Loader Commands

When the UART is used for communication it will use a standard baud rate of 38400 bps. When the SPI port is used for communication the baud rate is determined by the master device and data is clocked into the SPI on the FIRST rising edge of the ACTIVE-HIGH clock.

Boot Monitor Command Protocol

The low level protocol consists of a message header, data payload, message trailer and checksum. The header and trailer are detected by a two-character sequence beginning with the Data Link Escape (DLE) character (0x10). The header is indicated by a start of Header (SOH) character (0x01). An End of Text (EOT) character (0x04) indicates the trailer. The checksum (CSM) is the two's complement of the sum of all characters, excluding the checksum, in the message. Any data value in the payload that has a value of 0x10 is preceded with a DLE character.

[DLE][SOH][DT0]…[DTN-1][DTN][DLE][EOT][CSM]

On transmissions to the Boot Monitor the first byte in the Data Payload will always be the command opcode followed by any arguments specific to that command. On successful command responses the data payload will start with an Acknowledged (ACK) character (0x06) optionally followed by any command specific data.

[DLE][SOH][ACK][Data0]…[DataN-1][DataN][DLE][EOT][CSM]

On unsuccessful command responses the data payload will start with a Not Acknowledged (NAK) character (0x15) followed by an 8-bit error code.

[DLE][SOH][NAK][Error Code][EOT][CSM]

Boot Monitor Error Codes

Table 9.10 shows the different error codes that can be returned by Boot Monitor commands.

Boot Monitor Error Codes	
Error	Value
ERR_OK	0X00
ERR_MEM_MISCOMPARE	0XF0
ERR_REG_TEST_1_FAILED	0XF1
ERR_REG_TES_2_FAILED	0XF2
ERR_IRAM_ADDR_FOLD_FAILED	0XF3
ERR_IRAM_AADR_ZERO_FAILED	0XF4
ERR_STACK_TEST_FAILED	0XF5
ERR_UART_TEST_FAILED	0XF6
ERR_SPI_TEST_FAILED	0XF7
ERR_TIMER_TEST_FAILED	0XF8
ERR_IMG_BAD_HEADER	0XF9
ERR_IMG_BAD_CKSUM	0XFA
ERR_UNASSIGNED_ISR	0XFB
ERR_MSG_BAD_CKSUM	0XE0
ERR_MSG_INV_OPCODE	0XE1
ERR_PGM_BAD_TABLE_OP	0XD0
ERR_PGM_MEM_WRONG_FUNC_NAME	0XD1
ERR_PGM_MEM_WRONG_NUM_ARGS	0XD2
ERR_PGM_MEM_TIMEOUT	0XD3

Table 9.10: Boot Monitor Error Codes

Instruction Memory Programming Registers

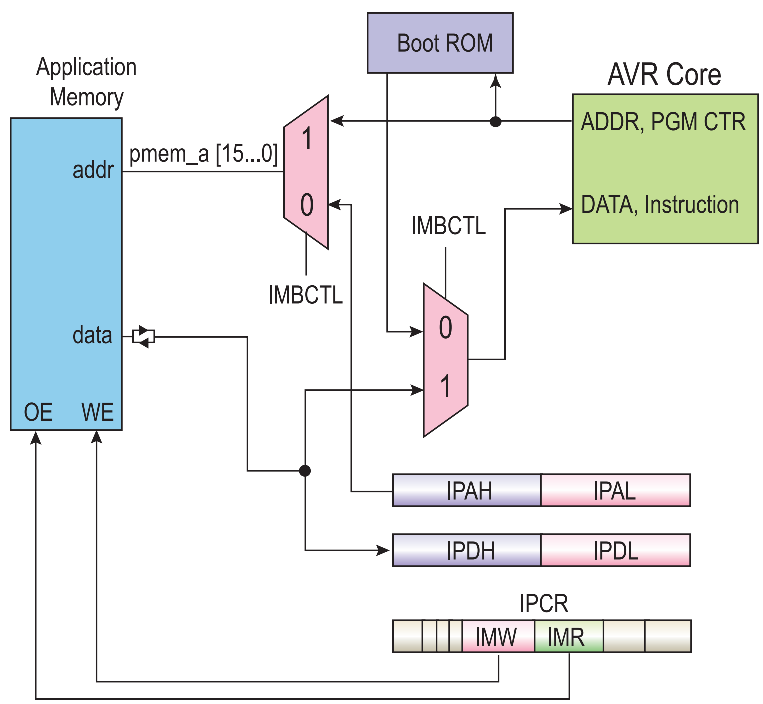


Figure 9.6: Instruction Memory Programming Registers

IPCR (Instruction Program Control)

Bit	7	6	5	4	3	2	1	0	
0x04(0x24)	IMBCTL	–	–	–	IMW	IMR	RPM	F_SN	IPCR
Read/Write	R/W	R	R	R	R/W	R/W	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 9.11: Instruction Program Control Register

- **Bit 7 – IMBCTL: Instruction Memory Bus Control**

  When this bit is set, instruction addresses come from the pc[15:0] bus and are routed to the external instruction memory. Also, instructions come from external instruction memory and are placed on the instr[15:0] bus. When this bit is zero, instruction addresses from the processor are routed to the internal BOOT ROM. Instructions from the BOOT ROM are placed on the instr[15:0] bus. Also, external memory addresses originate from [IPAH, IPAL] and external memory data (on a write) are driven from [IPDH, IPDL] registers.

- **Bits 6..4 Reserved bits**

  These bits are reserved and always read as zero.

- **Bit 3 – IMW: Instruction Memory Write**

  Writing logic one to this bit causes the processor to generate a write pulse to the external memory. The bit will subsequently be cleared by hardware.

- **Bit 2 – IMR: Instruction Memory Read**

  Writing logic one to this bit causes the processor to generate an output enable pulse to the external memory and subsequently capture the value returned in the IPDH and IPDL registers. The bit will subsequently be cleared by hardware.

- **Bit 1 – RPM: Reprogram Program Memory**

  This bit is a registered version of the REPGM_PGM_MEM input pin provided to allow software to read the status of the pin.

- **Bit 0 – F_SN: Flash or SRAM**

  This bit is a registered version of the FLASH_SRAMN input pin provided to allow software to read the status of the pin.

IPAH (Instruction Program Address High)

Bit	7	6	5	4	3	2	1	0	
0x03(0x23)	Instruction Program Address High								IPAH
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 9.12: Instruction Program Address High Register

- **Bits 7..0 Instruction Program Address High**

  These bits drive the upper 8 bits of the external instruction memory address when IPCR bit 7 is cleared.

IPAL (Instruction Program Address Low)

Bit	7	6	5	4	3	2	1	0	
0x02(0x22)	Instruction Program Address Low								IPAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 9.13: Instruction Program Address Low Register

- **Bits 7..0 Instruction Program Address Low**

These bits drive the lower 8 bits of the external instruction memory address when IPCR bit 7 is set.

IPDH (Instruction Program Data High)

Bit	7	6	5	4	3	2	1	0	
0x01(0x21)	Instruction Program Data High								IPDH
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 9.14: Instruction Program Data High Register

- **Bits 7..0 Instruction Program Data High**

These bits drive the upper 8 bits of the external instruction memory data bus when IPCR bit 7 is set and a write occurs (IPCR bit 3 set) to external instruction memory. On a read of external instruction memory (IPCR bit 2 set), this register captures the upper 8 data bits from external instruction memory.

IPDL (Instruction Program Data Low)

Bit	7	6	5	4	3	2	1	0	
0x00(0x20)	Instruction Program Data Low								IPDL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 9.15: Instruction Program Data Low Register

- **Bits 7..0 Instruction Program Data Low**

Program Memory Function Command Descriptions

START_OF_CMD

The First operation in the Function Table MUST be the START_OF_CMD operation. It is used to validate the input arguments whenever a Program Memory command is executed. OP_ARG1 specifies the Program Memory FUNCTION_NAME. OP_ARG2 contains the number of bytes data bytes required to execute the function. Data can be passed to the Function upon execution in a Data Buffer, this is used to optimize performance. For example when programming Parallel FLASH via the UART, the Address and Data to be programmed is NOT stored in the Function Table. It is stored in the Function Table Data Buffer. This way the Function Table does not have to be resent every time one word needs to be programmed.

Start of Command Operation	
Start of Command	Description
Required Function Table Header. Does Not Access the Program.	
OP_ARG1	Contains Function Name.
OP_ARG2	Contains the number of additional data bytes to perform function.

Table 9.16: Start of Command Operation

READ

The READ Operation will read one word from the Program Memory. OP_ARG1 specifies the Program Memory Address to read. OP_ARG2 is not used. The Data read will be saved to the Function Response Buffer.

Read Operation	
Read	Description
Reads one word of from the Program Memory, Data Read will be stored in the Function Response Buffer.	
OP_ARG1	Contains Address to Read.
OP_ARG2	Not Used.

Table 9.17: Read Operation

RD_ARG

The RD_ARG Operation will read one word where the address is read from the Function Data Buffer. OP_ARG1 and OP_ARG2 are not used. The Data read will be saved to the Function Response Buffer.

Rd_Arg Operation	
Read	Description
Reads one word of from the Program Memory, Address will be read from the Function Data Buffer. Data Read will be stored in the Function Response Buffer.	
OP_ARG1	Contains Address to Read.
OP_ARG2	Not Used.

Table 9.18: Read Argument Operation

WRITE

The WRITE Operation will write one word to Program Memory where OP_ARG1 specifies the Address to write to and OP_ARG2 specifies the DATA to write.

Write Operation	
Read	Description
Reads one word of the Program to Memory.	
OP_ARG1	Contains Address to Write.
OP_ARG2	Contains Data to Write.

Table 9.19: Write Operation

WR_ARG_ADDR

The WR_ARG_ADDR Operation will write one word to Program Memory where the Address read from the Function Data Buffer and the Data is in OP_ARG2.

Write Argument Address Operation	
Read	Description
Reads one word from the Program to Memory, where the Address to write is read from the Function Data Buffer.	
OP_ARG1	Contains Address to Write.
OP_ARG2	Contains Data to Write.

Table 9.20: Write Argument Address Operation

WR_ARG_DATA

The WR_ARG_DATA Operation will write one word to Program Memory where the Address is specified in OP_ARG1 and the data is read from the Function Data Buffer.

Write Argument Data Operation	
Read	Description
Reads one word from the Program to Memory, where the Data to write is read from the Function Data Buffer.	
OP_ARG1	Contains Address to Write.
OP_ARG2	Not Used.

Table 9.21: Write Argument Data Operation

WR_ARG_ARG

The WR_ARG_ARG Operation will write one word to PROGRAM MEMORY where both the Address and the Data are read from the Function Data Buffer. (Address precedes Data in the Function Data Buffer.)

Write Argument Data Operation	
Read	Description
Reads one word from the Program to Memory, where the Address and Data to write are read from the Function Data Buffer.	
OP_ARG1	Contains Address to Write.
OP_ARG2	Not Used.

Table 9.22: Write Argument-Argument Operation

PAUSE

The PAUSE Operation does not actually access program memory. It simply pauses for a number of milliseconds as specified in OP_ARG1. This is useful for some Parallel FLASH functions such as BLOCK Protecting that require a time delay before the command is finished.

Pause Operation	
Read	Description
Pause Function Execution for a set amount of time.	
OP_ARG1	Contains Time in milliseconds.
OP_ARG2	Not Used.

Table 9.23: Pause Operation

VERIFY

The VERIFY Operation will continually read Program Memory at the Address read from the Function Data Buffer until the Data matches the specified pattern or until the operation times out. OP_ARG1 specifies a timeout in milliseconds to give the Program Memory to complete the operation. If the memory does not match within the timeout, the status in the response buffer will be ERR_PGM_MEM_TIMEOUT. However the boot code will continue processing the Program Memory Function until it is completed. This allows for error recovery when a Parallel FLASH may need to be reset before it is accessed for a read again.

Verify Operation	
Read	Description
Verifies that the contents of program memory. Address and data to verify are read from the Function Data Buffer	
OP_ARG1	Contains Time in milliseconds.
OP_ARG2	Not Used.

Table 9.24: Verify Operation

VERIFY_WRITE

The VERIFY_WRITE Operation is a special operation used to Verify that a write operation has completed. It works exactly the same as the VERIFY Operation but for the Address and Data to verify, it uses that of the previous Operation. This enhancement prevents the Address/Data from being specified twice in the Function Data Buffer.

Verify Write Operation	
Read	Description
Verifies one Word from Program Memory.	
OP_ARG1	Contains Time in milliseconds.
OP_ARG2	Not Used.

Table 9.25: Verify Write Operation

END_OF_CMD

The END_OF_CMD Operation is used as the Command Table footer and MUST be the last command in the command table.

Verify Operation	
END_OF_CMD	Description
Required Function Table Footer. Does Not Access Program	
OP_ARG1	Not Used.
OP_ARG2	Not Used.

Table 9.26: End of Command Operation

Boot Monitor Command Descriptions

The specific format of each of the supported Commands for Command Data and Response Data is described below.

SET_PGM_MEM_FUNC

This command is used to download the Program Memory Function. This command must be performed prior to the Execute Program Memory Function Command.

Command Data: [SET_PGM_MEM_FUNC] [TBL DATA0][TBL DATA1]…[TBL DATAN]
Response Data:

[ACK]	if success
[NAK] [ERROR CODE]	if failure

EXE_PGM_MEM_FUNC

This command is used to execute the Program Memory Function that has already been set up.

Command Data: [EXE_PGM_MEM_FUNC] [CMD_NAME LSB] [CMD_NAME MSB] [D0][D1]…[Dn]
Response Data:

[ACK] [STATUS] [D0] [D1]…[Dn]	if command sent properly
[NAK] [ERROR CODE]	if failure sending command

PRG_SER_PGM

This command programs 64 words (128 bytes) of the SERIAL FLASH. The SERIAL FLASH MUST be present for this command to work properly.

Command Data: [PRG_SER_PGM] [ADDR LSB] [ADDR MSB] [D_0 LSB] [D_0 MSB] [D_1 LSB] [D_1 MSB] … [D_{63} LSB] [D_{63} MSB]
Response Data:

[ACK]	if success
[NAK] [ERROR CODE]	if failure

START

Command Data:
[START OPCODE]
Response Data:
None. The Boot Monitor will immediately transfer control to Program Memory.

RESET

Command Data:
[RESET OPCODE]
Response Data:
None. The Boot Monitor will immediately reset.

RD_SER_PGM

Command Data:
[RD_SER_PGM OPCODE] [ADDR LSB] [ADDR MSB]
Response Data:
[ACK] [D_0 LSB] [D_0 MSB] [D_1 LSB] [D_1 MSB] … [D_{63} LSB] [D_{63} MSB] if success
[NAK] [ERROR CODE] if failure.

Boot Monitor Bypass

The operation of the Boot Monitor may be bypassed using the IBOOT_N input pin. When this pin is low or unconnected (internal pull-down), the Boot Monitor performs internal tests etc., after a reset as described previously.

When this pin is high, program memory execution from FLASH/RAM commences after reset, bypassing Boot Monitor actions. This allows flexibility using the JTAG emulator.

This Page Intentionally Left Blank

SECTION 10
CAN Controller

Introduction

This module performs the functions necessary to implement the CAN transfer layer as defined in the CIA (CAN In Automation) specification 2.0, Part B Active and ISO/11898 CAN Specification 2.0B, September 1991.

Figure 10.1 illustrates the basic elements of the CAN controller. The shaded area is the function covered in this section of the specification.

The CAN Controller consists of the following elements:

- CAN Interface Macrocell
- CAN Control and Status Registers
- Transmit buffers 0 - 2, with Remote Transmission Request (RTR)
- Foreground and Background Receive Buffers
- Seven Receiver Acceptance Filters (Code and Mask Combinations)

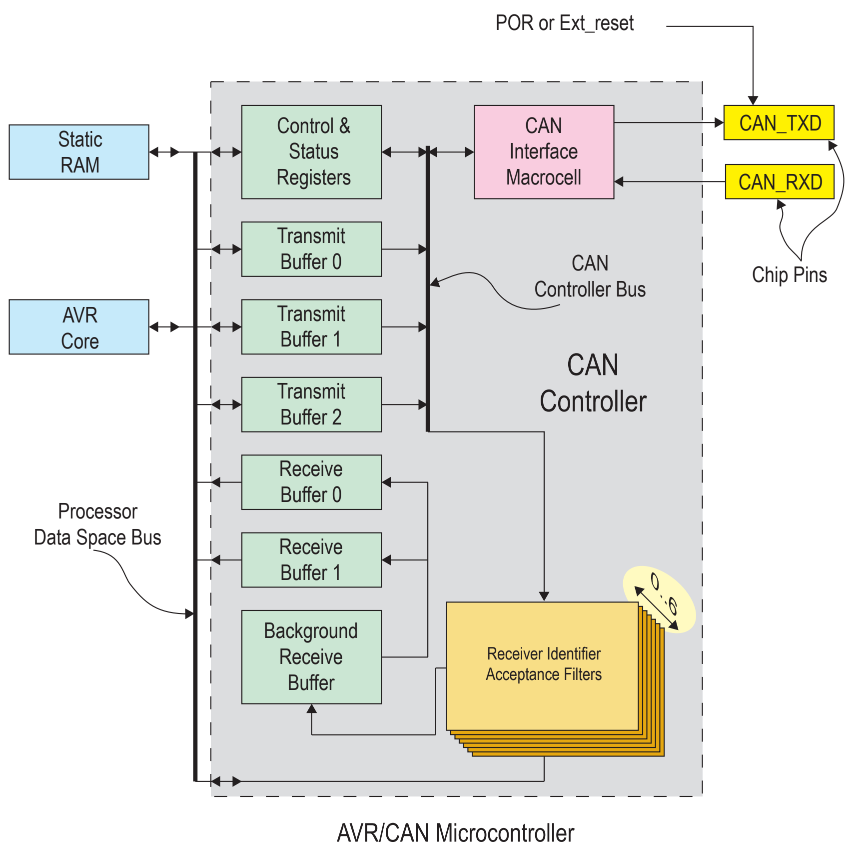


Figure 10.1: CAN Controller and Interface Block Diagram

Interfaces

All CAN Registers are mapped into the AVR processors Data Space and are accessible via ordinary memory access instructions. See the “CAN Controller Register Data Space Memory Map” later in this section for details.

The Transmit/Receive serial data lines are connected to their respective pins on the chip. The Transmit pin is an enhanced drive (7mA) CMOS compatible output pad with tristate during power on reset (POR) or external reset (ext_rst). The Receive pin is a conventional CMOS input pad.

Receiving Messages

Incoming messages appearing on the CAN_RXD input to the CAN Interface Microcell will be de-serialized. The Background Receive Buffer is utilized as a CAN message is being received. If the message is of type RTR, and the IDE&ID matches one of the 3 Tx Buffers set for automatic RTR return request (CCFLG.RRENn=1), then the message is no longer processed as a receive message. Instead the matched Tx Buffer is scheduled for transmittal as a reply to the RTR request.

If the message (IDE, ID, RTR) passes at least one of the 7-enabled Rx Identifier Acceptance Filters then the entire incoming message (IDE, ID, RTR, DLC and DATA) is held in the background buffer. After a complete error free message is received, the message is moved to an available Receive Buffer. Then the Receiver Full flag (CCFLG.RXFn) for the appropriate receiver is set. The interrupt FIFO’s will store the status in the CAN Controller Interrupt Status Register (CCISR) and any error flags (in CEFR) that are associated with the message. An interrupt will be generated if enabled.

For clarity, Table 10.1 helps indicate the logic for processing RTR messages. It separates the actions of the receiver modules versus that of the transmitter module.

CAN Remote Transmit Request (RTR) Operation

Is RTR on Bus?	Has ID & IDE or RTR Hit to any RX Filter?		Does ID & IDE Match and TXn Buffer?	IS RRENn set in TX Buffer?	RX Action	TX Action
	ID & IDE	RTR				
X	—	—	X	N		None
X	—	—	N	Y		None
N	—	—	Y	Y		None
Y	—	—	Y	Y		Send RTR Response Message
X	N	X	—	—	Message Ignored	
X	Y	N	—	—	Message Ignored	
X	Y	Y	—	—	Message Ignored	
Exception to the Above Table						
Y	Y	Y	Y	Y	Message Ignored	Send RTR Response Message

Key:

X - Doesn't care, can be high or low, match or no match

Y - Function bit is set, N - Function bit is not set

Table 10.1: CAN Remote Transmit Request (RTR) Operation

If both Receive Buffers are available, the message will be written into Receive Buffer 0. If neither Receive Buffer is available the newly received message will remain in the Background Receive Buffer until one of the Receive Buffers becomes available.

If a message is received which passes any of the Identifier Acceptance Filters while the Background Receive Buffer is full, then the CAN Controller will send out an Overload Frame. The Overload Frame is an indication to the CAN bus that the message was not processed by the receiver. If another message passes the filters while the Background Receive Buffer is full, then a second Overload Frame is sent.

This process of rejecting the message and sending the Overload Frame is done for two consecutive times. On the 3rd receipt (match) of a CAN message, the Overload Frame is not sent. Instead the CAN message is loaded into the Background Buffer, overwriting the previous message. Then a receive overrun will be indicated in the CAN Error Flag Register (CEFR.OVR=1).

The AVR processor may only read the contents of Receive Buffers 0 and 1. The Background Receive Buffer is not accessible to the AVR processor. After the AVR processor reads the complete message from a Receive Buffer, the processor must clear the appropriate CCFLG.RXFn flag. Writing to the CCISR register clears the Receive Message Interrupt and cycles the interrupt FIFOs.

Note that the initial processing of the received message will be the same whether the RTR bit is set (recessive on the media: i.e. the received message is a remote frame request) or cleared (dominant on the media: i.e. the received message is an ordinary message). If a received message fails all enabled Rx Identifier Acceptance Filters and fails to match any Tx Buffer set for automatic RTR Return Enabled (RREN=1), then further processing of the message is abandoned and the CAN Cell will wait for the next message on the bus.

Messages with errors detected are still received and placed into a receive buffer. The error bits, in the message status, are set accordingly.

Transmitting Messages

When the Can Controller is transmitting, the controller listens to its own messages but does NOT overwrite the Background Receive Buffer, generate an interrupt, or acknowledge its own messages.

The CAN Controller has three independent Transmit Buffers, two of which can be turned on or off for improved power management. The structure of these buffers is similar to the Receive Buffer and is shown in “Transmit Buffer Structure Details”. The Transmit Buffer contains the Transmit Buffer Priority Register (TBPR) at location 0xnnnD. To transmit a message the AVR processor identifies an empty Transmit Buffer by a set Transmit Buffer Empty flag (TXEMn=1) in the CAN Communications Flag Register (CCFLG). The AVR processor writes the message to be transmitted (identifier, control bits, data, byte count and priority) into the appropriate Transmit Buffer and then enables the buffer for transmission by clearing the appropriate Transmit Buffer Empty Flag (TXEMn=0).

The CAN Controller schedules all enabled messages for transmission. Successful transmission of a message is indicated when the controller sets the appropriate Transmit Buffer Empty Flag (TXEMn=1) and a transmit interrupt is generated (if the transmit interrupt is enabled). If a transmit buffer is not enabled its TXEN bit will remain in the “not empty” state (TXEMn=0). When a transmit buffer is disabled, the corresponding TXEMn bit shall be changed to “not empty” and no message transmission shall occur with the change.

When more than one Transmit Buffer is scheduled for transmission, the CAN Controller uses the Transmit Buffer Priority Register (TBPR) to determine which message will be sent first. The lowest numerical value TBPR has the highest priority and will be sent first. If more than one pending transmission message has the same priority, the lowest value numerical buffer that is enabled (TXBENn=1) and not empty (TXEM=0) will be sent first (i.e. Transmit Buffer 0 is sent first, then Transmit Buffer 1, then Transmit Buffer 2).

To abort (cancel) the transmission of a previously scheduled message the AVR processor must request that the message transmission be aborted. To request transmit message abort the processor must set the corresponding Abort Request Flag (ABTRQ) in the CAN Transmit Abort Request Register (CTARR). The CAN Controller will grant the request, if possible, by setting the corresponding Abort Request Acknowledge Flag (ABTAK), and the TXEMn Flag to release the Transmit buffer and generate a transmit interrupt.

Messages that are already under transmission cannot be aborted (canceled). In order to verify that an abort request was honored (or not), the processor must wait for a transmit interrupt to occur (either as a result of a successful abort request or the end of message transmission of a failed abort request), and then read the ABTAKn flag in the CAN Controller Interrupt Status Register (CCISR). If the abort request occurred before message transmission was started, the ABTAKn flag will be set. If the abort request failed, the ABTAKn flag will be cleared.

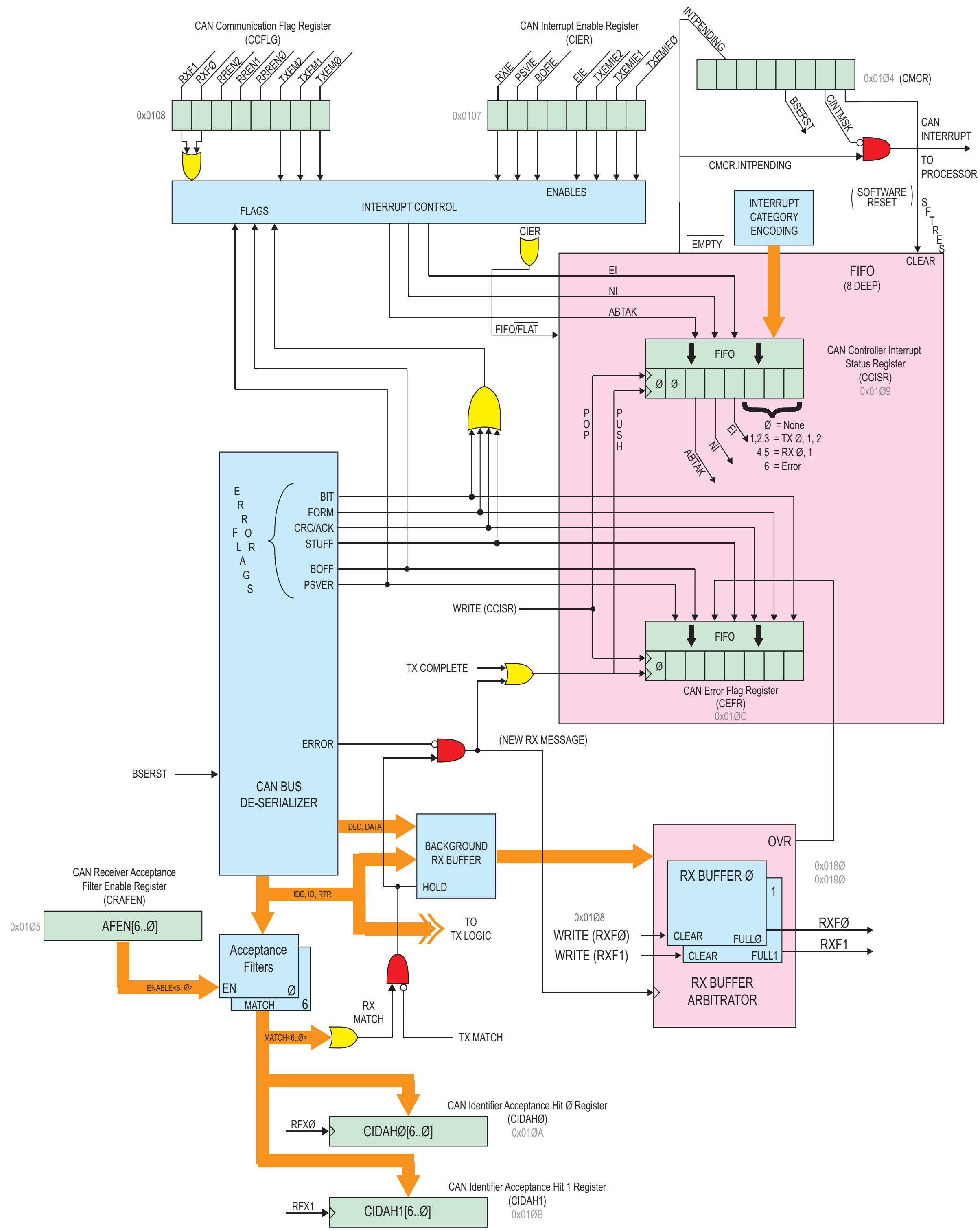


Figure 10.2: CAN Receiver Block Diagram

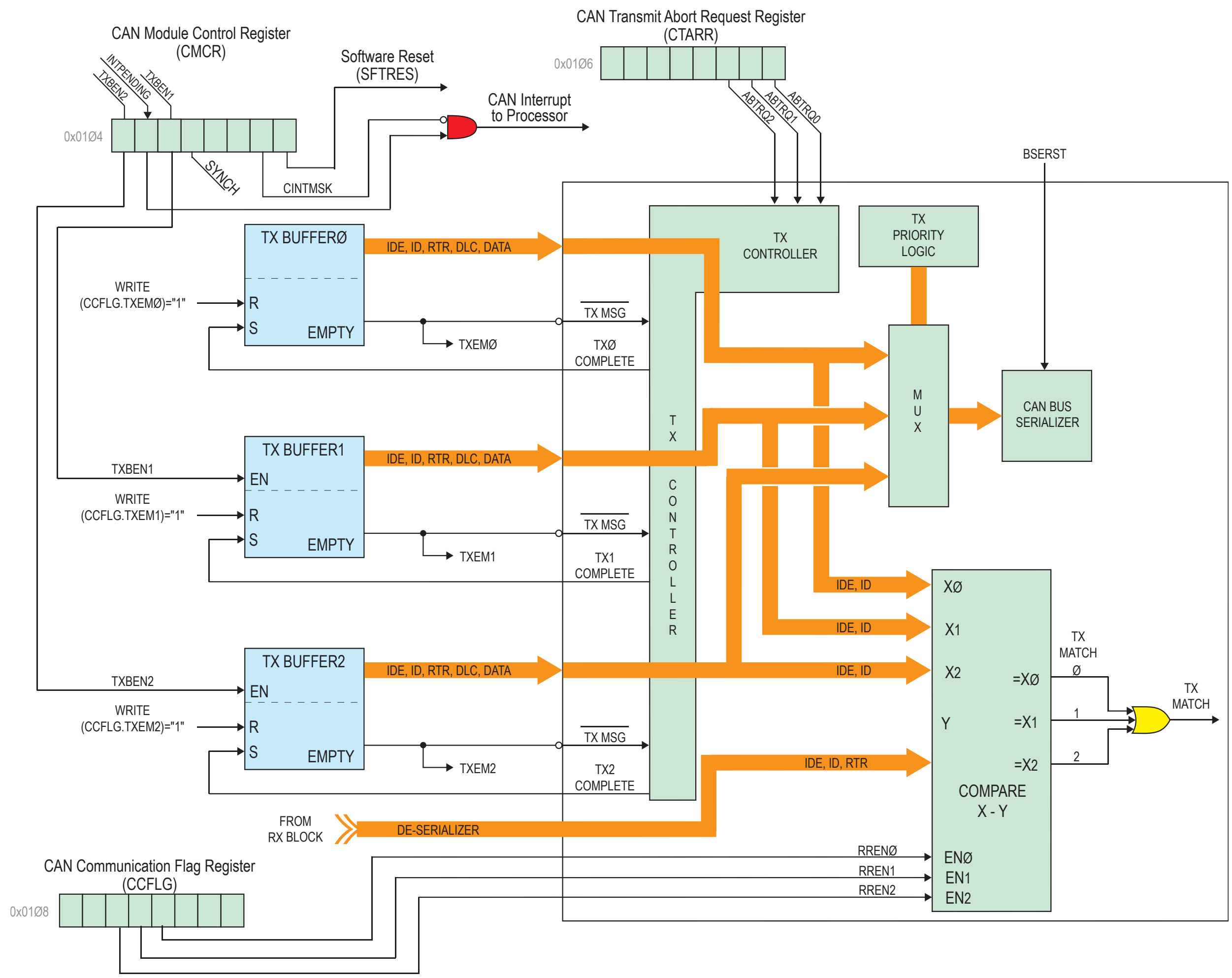


Figure 10.3: CAN Transmitter Block Diagram

CAN Controller Register Map

CAN Controller Register Map			
Group	Data Space	Name	Description
BAUD Rate	0x0100	CDIVCAN	CAN Enable and Clock Prescale
	0x0101 - 0x0103	CBTR[0..2]	CAN Bus Timing Registers
CAN Control and Enable Registers	0x0104	CMCR	Can Module Control Register
	0x0105	CRAFEN	CAN Reciever Acceptance Filter Enable
	0x0106	CTARR	CAN Transmit Abort Request Register
	0x0107	CIER	CAN Interrupt Enable Register
CAN Interrupt, Flag and Status Registers	0x0108	CCFLG	CAN Communications Flag Register
	0x0109	CCISR	CAN Communications Interrupt Status Register
	0x010A	CIDAH0	CAN Indentifier Acceptance Hit 0 Register
	0x010B	CIDAH1	CAN Indentifier Acceptance Hit 1 Register
CAN Error Specific Registers	0x010C	CEFR	CAN Error Flag Register
	0x010D	CRXERR	CAN Recieve Error Counter
	0x010E	CTXERR	CAN Transmit Error Counter
Version	0x010F	CVER	CAN Hardware Version
RX0 Codes and Masks	0x0110 - 0x0113	CIDAC0R[0..3]	CAN Indentifier Acceptance Code 0 Register
	0x0114 - 0x0117	CIDM0R[0..3]	CAN Indentifier Mask 0 Register
RX1 Codes and Masks	0x0118 - 0x011B	CIDAC1R[0..3]	CAN Indentifier Acceptance Code 1 Register
	0x011C - 0x011F	CIDM1R[0..3]	CAN Indentifier Mask 1 Register
RX2 Codes and Masks	0x0120 - 0x0123	CIDAC2R[0..3]	CAN Indentifier Acceptance Code 2 Register
	0x0124 - 0x0127	CIDM2R[0..3]	CAN Indentifier Mask 2 Register
RX3 Codes and Masks	0x0128 - 0x012B	CIDAC3R[0..3]	CAN Indentifier Acceptance Code 3 Register
	0x012C - 0x012F	CIDM3R[0..3]	CAN Indentifier Mask 3 Register
RX4 Codes and Masks	0x0130 - 0x0133	CIDAC4R[0..3]	CAN Indentifier Acceptance Code 4 Register
	0x0134 - 0x0137	CIDM4R[0..3]	CAN Indentifier Mask 4 Register
RX5 Codes and Masks	0x0138 - 0x013B	CIDAC5R[0..3]	CAN Indentifier Acceptance Code 5 Register
	0x013C - 0x013F	CIDM5R[0..3]	CAN Indentifier Mask 5 Register
RX6 Codes and Masks	0x0140 - 0x0143	CIDAC6R[0..3]	CAN Indentifier Acceptance Code 6 Register
	0x0144 - 0x0147	CIDM6R[0..3]	CAN Indentifier Mask 6 Register
Unused	0x0148 - 0x014F	Unused	Unused
TX Buffers	0x0150 - 0x015F	CTXB0	CAN Transmit Buffer 0
	0x0160 - 0x016F	CTXB1	CAN Transmit Buffer 1
	0x0170 - 0x017F	CTXB2	CAN Transmit Buffer 2
RX Buffers	0x0180 - 0x018F	CRXB0	CAN Recieve Buffer 0
	0x0190 - 0x019F	CRXB1	CAN Recieve Buffer 1

Table 10.2: CAN Controller Register Data Space Memory Map

Programmer's Model of Control and Interrupt Registers

NOTE: A 0-1 transition on TXEMn triggers a Normal Message Transmit Interrupt and the state of TXEMn is observable in CCFLG, but the interrupt service routine should read the CCISR for an indication of the source of the interrupt.

CAN Controller Register Summary										
Data Space Addr.	Register		Register Bits							
			7	6	5	4	3	2	1	0
0x0100	CDIVCAN	Read	CANEN	0	0	0	0	CDIV[2..0]		
		Write								
0x0101	CBTR0	Read	SJW[1..0]		BRP[5..0]					
		Write								
0x0102	CBTR1	Read	SAMP	PSEG2[2..0]			0	PSEG1[2..0]		
		Write								
0x0103	CBTR2	Read	0	0	0	0	0	PRPT[2..0]		
		Write								
0x0104	CMCR	Read	TXBEN2	INT_ PENDING	TXBEN1	SYNCH	BESERST	MONITOR	CINTMSK	SFTRES
		Write								
0x0105	CRAFFEN	Read	0	AFEN6	AFEN5	AFEN4	AFEN3	AFEN2	AFEN1	AFEN0
		Write								
0x0106	CTARR	Read	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
		Write								
0x0107	CIER	Read	RXIE	PSVIE	BOFIE	0	EIE	TXEMIE2	TXEMIE1	TXEMIE0
		Write								
0x0108	CCFLG	Read	RXF1	RXF0	RRE2	RRE1	RRE0	TXEM2	TXEM1	TXEM0
		Write								
0X0109	CCISR	Read	0	0	ABTAK	NI	EI	CIID[2..0]		
		Write								
0X010A	CIDAH0	Read	0	HIT6	HIT5	HIT4	HIT3	HIT2	HIT1	HIT0
		Write								
0X010B	CIDAH1	Read	0	HIT6	HIT5	HIT4	HIT3	HIT2	HIT1	HIT0
		Write								
0X010C	CEFR RX	Read	0	PSVER	BOFF	OVR	STUFF	CRC	FORM	0
		Write								
	CEFR TX	Read	0	PSVER	BOFF	OVR	STUFF	ACK	FORM	BIT
		Write								
0X010D	CRXERR	Read	RXERR[7..0]							
		Write								
0X010E	CTXERR	Read	TXERR[7..0]							
		Write								

Table 10.3: CAN Control Registers Summary

NOTE: The Abort Request Acknowledge indication is a combination of ABTAK and CIID2-0. Also, the interrupt mask for the Abort Request Acknowledge is tied to the TXEMIEn mask for each Transmit Message Buffer.

CAN Controller Interrupt Sources						
Function	Source Signal	Source Register	Mask	Mask Register	CMCR	
					Signal	Mask
Normal Message Transmit Interrupts	TXEM0	CCISR 0x0109	TXEMIE0	CIER 0X0107	Interrupt Pending	CINTMSK
	TXEM1		TXEMIE1			
	TXEM2		TXEMIE2			
Normal Message Receive Interrupts	RXF0		RXIE			
	RXF1					
Transmission Abort Acknowledge	ABTAK		TXEMIEn			
Transmit Error Interrupts	PSVER	CEFR 0X010C	PSVIE			
	BOFF		BOFIE			
	STUFF		EIE			
	ACK		EIE			
	FORM		EIE			
	BIT		EIE			
Recieve Interrupts	PSVER		PSVIE			
	BOFF		BOFIE			
	OVR		EIE			
	STUFF		EIE			
	CRC		EIE			
	FORM		EIE			

Table 10.4 CAN Controller Interrupt Sources

CAN Baud Rate Control

NOTE: 1/Tb = CAN Baud Rate

The M3000 uses the external clock as the source clock to the CAN module. The clock can be scaled by means of CDIV<2..0> and BRP<5..0> to obtain the desired Time Quanta (Tq). According to CAN Specification, there must be from 8 to 25 Tq's to comprise 1 (one) CAN bit time (Tb).

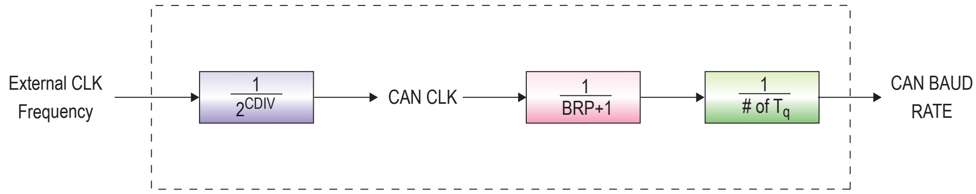


Figure 10.4: CAN System Clock to Baud Rate

Figure 10.5 below gives an overview of the CAN segment settings. It is the users responsibility to configure the bit settings to be in compliance with the CAN standard.

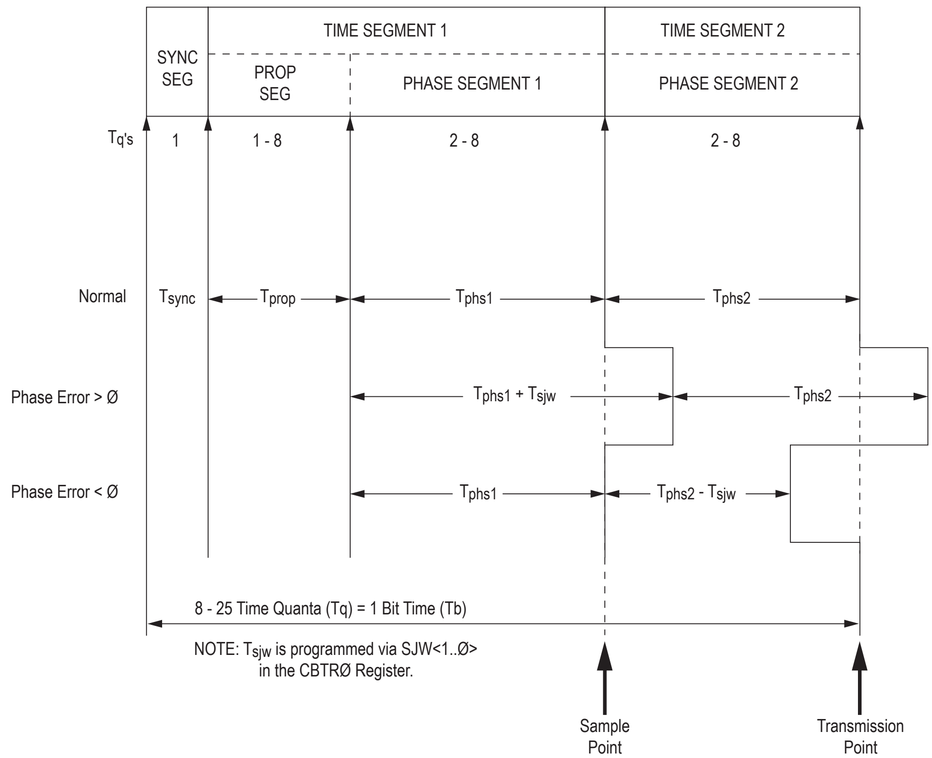


Figure 10.5: CAN Segment Bit Timing

The formulas that apply are:

Eq 1: $Tq = (2^{CDIV} \times (BRP + 1) / \text{External Clock Frequency})$

Eq 2: $Tb = (\text{Synch Seg} + \text{Prop Seg} + \text{Phase Seg 1} + \text{Phase Seg 2}) \times Tq$

Eq 3: $Tb = (1 + (PRPT + 1) + (PSEG1X + 1) + (PSEG2X + 1)) \times Tq$

The following table demonstrates the recommended settings for desired Baud Rates, assuming a 20 MHz External Processor Clock.

Recommended BAUD Rate Settings													
					CDIVCAN	CBTR0		CBTR1				CBTR2	
Xtal (MHz)	CAN BAUD (MHz)	Quantas Avail.	Total Divider	New Tq(s) Avail.	CDIV [2..0]	SJW [7..6]	BRP [5..0]	SAMP [7]	PSEG2 [6..4]	[3]	PSEG1 [2..0]	[7..3]	PRPT [2..1]
20	1	20	1	20	0		0		5	0	4	0	7
20	0.8	25	1	25	0		0		7	0	7	0	7
20	0.5	40	2	20	0		1		5	0	4	0	7
20	0.25	80	5	16	0		4		3	0	2	0	7
20	0.125	160	10	16	0		9		3	0	2	0	7
20	0.1	200	10	20	0		9		5	0	4	0	7
20	0.05	400	25	16	0		24		3	0	2	0	7
20	0.02	1000	50	20	0		49		5	0	4	0	7
20	0.01	2000	100	20	0		49		5	0	4	0	7

Table 10.5: Recommended Baud Rate Settings

CAN Controller Registers

CDIVCAN (CAN Clock Prescaler)

Bit	7	6	5	4	3	2	1	0	
0x0100	CANEN	–	–	–	–	CDIV			CDIVCAN
Read/Write	R/W	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	1	

Table 10.6: CAN Clock Prescaler

- **Bit 7 – CAN EN**

  This bit is used to enable the CAN Controller and make the Transmit and Receive Pins active. Zero is disabled. While disable the CAN Section draws no power. No other CAN registers functional unless CANEN is enabled.

- **Bits 6..3 – Res: Reserved bits**

  (Read Only) These bits are reserved bits and always read as zero.

- **Bits 2..0 – CDIV Clock Division Factor**

  The Clock Prescaler Select bits 2,1 and 0 define the divide by prescale factor for the CAN Clock. (CAN Clock = 1 / 2^{CDIV} * External Clock)

CBTR0 (CAN Bus Timing Register 0)

Bit	7	6	5	4	3	2	1	0	
0x0101	SJW		BRP						CBTR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.7: CAN Bus Timing Register 0

- **Bits 7..6 – SJW - Synchronization Jump Width**

  $T_{sjw} = (SJW+1) * T_q$

  The Synchronization Jump Width (SJW) defines the maximum number of time quanta (Tq) clock cycles by which a bit may be shortened or lengthened to achieve re-synchronization on data transitions on the bus.

- **Bits 5..0 – BRP - Baud Rate Prescaler**

  These bits determine the time quanta (Tq) clock which is used to build up the individual bit timing, per Eq1.

CBTR1 (CAN Bus Timing Register 1)

Bit	7	6	5	4	3	2	1	0	
0x0102	SAMP	PSEG2			-	PSEG1			CBTR1
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.8: CAN Bus Timing Register 1

- **Bit 7 – SAMP - Sampling**
  - 0 – One sample per bit
  - 1 – Three samples per bit (PHASE_SEG1 must be at least 2 time quanta)

  This bit determines the number of serial bus samples to be taken per bit time. If set, three samples are to be taken, the regular one and two preceding samples, using a majority rule. For higher bit rates, SAMP should be cleared, which means that only one sample will be taken per bit.
- **Bits 6..4 – PSEG2 Phase Segment 2 Time**

  T_{phs2} = Phase_Segment2_Time = (PSEG2 + 1) * T_q
- **Bit 3 - Reserved**
- **Bits 2..0 – PSEG1<2..0> - Phase Segment 1 Time**

  T_{phs1} = Phase_Segment1_Time = (PSEG1 + 1) * T_q

CBTR2 (CAN Bus Timing Register 2)

Bit	7	6	5	4	3	2	1	0	
0x0103	-	-	-	-	-	PRPT			CBTR2
Read/Write	R	R	R	R	R	R/W			
Initial Value	0	0	0	0	0	0			

Table 10.9: CAN Bus Timing Register 2

- **Bits 7..3 - Reserved**
- **Bits 2..0 – PRPT<2..0> - Propogation Segment Time**

  Encoding

  T_{prop} = Propogation_Segment_Time = (PRPT + 1) * T_q

CMCR (CAN Module Control Register)

Bit	7	6	5	4	3	2	1	0	
0x0104	TXBEN2	INTPENDING	TXBEN1	SYNCH	BSERST	MONITOR	CINTMSK	SFTRES	CMCR
Read/Write	R/W	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	1	1	

Table 10.10: CAN Module Control Register

- **Bit 7 – TXBEN2 - Transmit Buffer 2 Enable**
  - » 0 – Transmit Buffer 2 Disabled
  - » 1 – Transmit Buffer 2 Enabled
- **Bit 6 – INTPENDING - Interrupt pending from the CAN core to the processor**
  - » 0 – No Interrupt Pending
  - » 1 – Interrupt Pending
- **Bit 5 – TXBEN1 - Transmit Buffer 1 Enable**
  - » 0 – Transmit Buffer 1 Disabled
  - » 1 – Transmit Buffer 1 Enabled
- **Bit 4 – SYNCH - Synchronized Status**
  - » 0 – CAN Controller Not Synchronized
  - » 1 – CAN Controller is Synchronized with the CAN bus and can participate in communications.
- **Bit 3 – BSERST - Bit Stream Engine Reset**

Software bit to reset the CAN interface macrocell. To fully reset, 128 clock cycles are required. When releasing BSERST, one must release SFTRES at the same time.

- **Bit 2 – MONITOR - Monitor Mode**
- **Bit 1 – CINTMSK - CAN Interrupt Mask**
  - » 0 – Enable CAN Interrupts to processor
  - » 1 – Disable CAN Interrupts to processor

Must be cleared to 0 to unmask CAN interrupts.

If 0 => 8 deep que status que (CCISR)

1 => 1 deep que (flat status que)

- **Bit 0 – SFTRES - Soft Reset.**
  - » 0 – Normal operation
  - » 1 – In Soft Reset State / Initiate Software Reset

This bit is set by the CPU or a Reset (RESET_N = 0). When set, the CAN controller immediately enters the Soft Reset state. Any message transmission or reception is immediately terminated and bus synchronization is lost. Interrupts are disabled. RX and TX Error Counters are cleared.

The following registers enter the reset state.

- CCFLG (CAN Communications Flag Register)
- CCISR (CAN Controller Interrupt Status Register)
- CTARR (CAN Transmit Abort Request Register)
- CEFR (CAN Error Flag Register)

When the Soft Reset bit is cleared (0), the CAN controller tries to synchronize to the CAN bus. If the CAN controller is not in the “Bus Off” State (BOFF=0 in CEFR) the controller will be synchronized after the next occurrence of 11 recessive bits on the bus. If the CAN Controller is in the “Bus Off” state (BOFF=1 in CEFR) the controller continues to wait for 128 occurrences of 11 recessive bits. When clearing SFTRES and writing to other bits in CMCR it is recommended that they be done in separate instructions.

CRAFEN (CAN Receiver Acceptance Filter Enable)

Bit	7	6	5	4	3	2	1	0	
0x0105	-	AFEN6	AFEN5	AFEN4	AFEN3	AFEN2	AFEN1	AFEN0	CRAFEN
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.11: CAN Receiver Acceptance Filter Enable Register

- **Bit 7 – Reserved**
- **Bit <6..0> – CAN Acceptance Filter Enable <6..0>**
  - » 0 – Acceptance Filter NOT Enabled
  - » 1 – Acceptance Filter Enabled

Disabling a receiver will disable the "hit" for that receiver, and disable the interrupt path for that receiver filter. No data will be transferred to receiver buffers 0 or 1.

Please note that the CAN Controller will continue the filtering process of the CAN messages from the enabled Receiver Acceptance Filters. When a bit is cleared, only then is it allowed that the software make changes to the associated Code and Mask Register.

Enabling a Receiver Acceptance Fiter will begin the filtering process at the beginning of the next CAN frame signified by the recognition of the SOF (Start Of Frame) bit.

CTARR (CAN Transmit Abort Request Register)

Bit	7	6	5	4	3	2	1	0	
0x0106	-	-	-	-	-	ABTRQ2	ABTRQ1	ABTRQ0	CTARR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.12: CAN Transmit Abort Request Register

CTARR is held in reset while the Soft Reset bit is asserted (SFTRES=1 in CMCR).

- **Bits 7..3 – Reserved**
- **Bits 2..0 – ABTRQ2..0 - Abort Request**
  - » [0 at Reset]
  - » 0 – No abort request - 1 – Abort request Pending

Setting ABTRQn requests the CAN controller to abort scheduled message transmission on the associated Transmit Buffer. A successful Abort Request will generate an interrupt and set the ABTAK and appropriate CIIDn bits in the CCISR if the corresponding TXEMIEn (TX EMpty Interrupt Enable) bit in the CIER is set.

If the CAN message has not yet been transmitted, the ABTRQn bit will be cleared. If the CAN message is in progress of being transmitted, the ABTRQn will remain set until the completion of the transmit.

CIER (CAN Interrupt Enable Register)

Bit	7	6	5	4	3	2	1	0	
0x0107	RXIE	PSVIE	BOFIE	-	EIE	TXEMIE2	TXEMIE1	TXEMIE0	CIER
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.13: CAN Interrupt Enable Register

CIER is held in reset while the Soft Reset bit in CAN Module Control Register is asserted (SFTRES=1 in CMCR) and during a hard reset.

- Bit 7 – RXIE - Receive Buffer full Interrupt Enable
  - » (Read/Write) [0 at Reset]
  - » 0 – An Interrupt will not be generated from a receive buffer full event
  - » 1 – A Receiver Buffer full event will result in an interrupt (RXFn = 1 in CCFLG)
- **Bit 6 – PSVIE - Passive Error Interrupt Enable**
  - » (Read/Write) [0 at Reset]
  - » 0 – An interrupt will not be generated for Passive Error event
  - » 1 – A Passive Error Event will result in an error interrupt (PSVER = 1 in CEFR)
- **Bit 5 – BOFIE - Bus Off Interrupt Enable**
  - » (Read/Write) [0 at Reset]
  - » 0 – Bus Off interrupt disabled
  - » 1 – A Bus Off event will result in an error interrupt (BOFF =1 in CEFR)
- **Bit 4 – Reserved**
- **Bit 3 – EIE - Error Interrupt Enable**
  - » (Read/Write) [0 at Reset]
  - » 0 – An Error Interrupt will not be generated for an Error event that is not a Passive Error
  - » 1 – An Error event that is not a Passive Error event will result in an error interrupt (If any of OVR,BIT,FORM,CRC,STUFF = 1 in CEFR)
- **Bit 2 – TXEMIE2 - Transmit Buffer 2 Empty Interrupt Enable**
- **Bit 1 – TXEMIE1 - Transmit Buffer 1 Empty Interrupt Enable**
- **Bit 0 – TXEMIE0 - Transmit Buffer 0 Empty Interrupt Enable**
  - » (Read/Write) [0 at Reset]
  - » 0 – An interrupt will not be generated for a Transmit Buffer Empty event
  - » 1 – A Transmit Buffer Empty event will result in an interrupt (TXEMn = 1 in CCFLG)

CCFLG (CAN Communication Flag Register)

Bit	7	6	5	4	3	2	1	0	
0x0108	RXF1	RXF0	RREN2	RREN1	RREN0	TXEM2	TXEM1	TXEM0	CCFLG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	1	

Table 10.14: CAN Communication Flag Register

CCFLG is held in reset while the Soft Reset bit in CAN Module Control Register is asserted (SFTRES=1 in CMCR) and during a hard reset.

- **Bit 7 – RXF1 - Receive Buffer 1 Full Flag**
- **Bit 6 – RXF0 - Receive Buffer 0 Full Flag**
  - » (Read/write 1 to clear)
  - » 0 – The indicated Receive Buffer is empty and available for new message
  - » 1 – The indicated Receive Buffer contains a received message and is not available for new received messages.

The appropriate bit is set by the CAN Controller when a message has been stored in the corresponding Rx Buffer. The processor can read the bit or write a one to clear the bit. Writing a zero to RXFn has no effect. Reading the associated Receive Message Buffer when RXFn is cleared will produce indeterminate data.

- **Bit 5 – RREN2 - Remote Request Enable for Tx Buffer 2**
- **Bit 4 – RREN1 - Remote Request Enable for Tx Buffer 1**
- **Bit 3 – RREN0 - Remote Request Enable for Tx Buffer 0**
  - » [0 at Reset]
  - » 0 – Remote Requests are disabled for the indicated transmit buffer
  - » 1 – Remote Requests are enabled for the indicated transmit buffer

Setting RRENn enables Remote Request Messages for the indicated Transmit Buffer. The RRENn and corresponding TXEMn bits cannot both have their respective functions enabled simultaneously. If Remote Request is enabled attempts to enable the same transmit buffer for scheduled transmission will be ignored. Conversely, if a buffer is enabled for scheduled transmission, attempts to enable remote request will be ignored.

- **Bit 2 – TXEM2 - Transmitter Buffer 2 Empty Flag**
- **Bit 1 – TXEM1 - Transmitter Buffer 1 Empty Flag**
- **Bit 0 – TXEM0 - Transmitter Buffer 0 Empty Flag**
  - » (Write 1 to Clear) [0/0/1 at Reset]
  - » 0 – The associated transmit buffer is full and scheduled for transmittal
  - » 1 – The associated transmit buffer is empty

This flag indicates the state of the associated Transmit Buffer. The CPU must clear this flag (TXEMn=0) after a complete message has been setup in the associated Transmit Buffer. The TXEMn flag is cleared by writing a 1 to the respective bit in CCFLG. Writing a 0 to TXEMn has no effect.

The CAN Controller sets this bit (TXEMn=1) when any one of three events occurs: a successfully transmitted message, a message transmission is successfully aborted due to a pending Abort Request (See “CAN Transmit Abort Request Register (CTARR)”, or an error occurs in the transmission of a message (See “CAN Error Flag Register (CEFR)”. Registers in the associated Transmit Message Buffer should not be written when this flag is cleared (TXEMn=0).

Transmit buffer 0 is available on reset and therefore TXEM0 is empty on reset(TXEM0=1). Transmit buffer 1 and 2 must be enabled after reset. Because of this, the corresponding TXEMn bit indicates buffer is full on reset (TXEM2=TXEM1=0).

Upon enabling of the TXBEN1 or TXBEN2, the corresponding TXEMn bit shall change to the buffer is empty state (TXEMn=1) without generating a transmit interrupt.

CCISR (CAN Controller Interrupt Status Register)

Bit	7	6	5	4	3	2	1	0	
0x0109	-	-	ABTAK	NI	EI	CIID<2..Ø>			CCISR
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.15: CAN Controller Interrupt Status Register

The CCISR contains a summary of the reason for the CAN Controller interrupt. This register should be the first read in the CAN Controller interrupt service routine. The value encoded in CIID<2..0> indicates which message buffer caused the interrupt and bits 3–5 are set as appropriate for that message buffer. If there are interrupts pending for more than one message buffer, the interrupts are presented in the order of occurrence.

- **Bits 7..6 – Reserved**
- **Bit 5 – ABTAK Abort Acknowledge**
  - » 0 – Message transmission not aborted
  - » 1 – Message transmission aborted

  This bit acknowledges that a previously scheduled message transmission has been aborted due to a request from the CPU (see CAN Transmit Abort Request Register (CTARR)). The value of CIID<2..0> indicates Transmit Message Buffer for which the transmission was aborted. The ABTAK, NI, and EI bits are mutually exclusive. A successful transmit abort (ABTAK set) automatically sets the appropriate TXEMn bit in the CCFLG Register (Transmit buffer not available for transmission). Writing to the CCISR causes the next lower pending interrupt status to be loaded. If there are no pending interrupts the CCISR will be cleared.
- **BIT 4 – NI - Normal Interrupt**
  - » 0 – No normal communications interrupt occurred.
  - » 1 – An interrupt occurred as a result of the normal transmission or reception of a CAN message

  The NI bit is set, if enabled in the CIER, as a result of normal communications activity (e.g. no errors or abort requests). The ABTAK, NI and EI bits are mutually exclusive. Writing to the CCISR causes the next lower pending interrupt status to be loaded. If there are no pending interrupts, the CCISR will be cleared.
- **Bit 3 – EI - Error Interrupt**
  - » 0 – No communications error interrupt occurred
  - » 1 – An interrupt occurred as a result of an error during the transmission or reception of a CAN message

  The EI bit is set, if enabled in the CIER, as a result of an error. The specific error will be indicated in the CAN Error Flag Register (CEFR). The ABTAK, NI and EI bits are mutually exclusive. When CCISR indicates a message error, the CAN Error Flag Register (CEFR) can be read to determine the type of error. Writing to the CCISR Register causes the next lower pending interrupt status to be loaded in the CCISR. If there are no pending interrupts the CCISR will be cleared.
- **Bits <2..0> – CIID<2..0> CAN Interrupt Category Encoding**

  The numerical value of CIID<2..0> indicates the Message buffer associated with the current interrupt as shown in Table 10:16.

CAN Interrupt Category Encoding			
CIID2	CIID1	CIID0	Message Buffer
0	0	0	No Pending Communications Interrupts
0	0	1	Transmit Message Buffer 0
0	1	0	Transmit Message Buffer 1
0	1	1	Transmit Message Buffer 2
1	0	0	Receive Message Buffer 0
1	0	1	Receive Message Buffer 1
1	1	0	Bit Stream Engine Error
1	1	1	Reserved

Table 10.16: CAN Interrupt Category Encoding

CIDAH0 (CAN Identifier Acceptance Hit6..0 Register for Rx Buffer 0)

Bit	7	6	5	4	3	2	1	0	
0x010A	-	HIT6	HIT5	HIT4	HIT3	HIT2	HIT1	HIT0	CIDAH0
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.17: CAN Identifier Acceptance Hit0 Register

Software Coding

CIDAH[0/1]

CIDAH0 indicates which Identifier Acceptance Filter (or Filters) were triggered (Hit), and the corresponding received CAN message is in Receive Buffer 0. All bits are held in reset (all bits = 0) while the Soft Reset bit in CAN Module Control Register is asserted (SFTRES=1 in CMCR) and during a hard reset.

- **Bit 7 – Reserved**
- **Bits 6..0 – ID0HIT6..0 - Identifier Acceptance Filter Hit 6..0 Indicator**
  - » 0 – Message in Receive Buffer 0 did NOT pass Identifier Accept Filter 6..0
  - » 1 – Message in Receive Buffer 0 passed Identifier Accept Filter 6..0

CIDAH1 (CAN Identifier Acceptance Hit6..0 Register for Rx Buffer 1)

Bit	7	6	5	4	3	2	1	0	
0x010B	-	HIT6	HIT5	HIT4	HIT3	HIT2	HIT1	HIT0	CIDAH1
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.18: CAN Identifier Acceptance Hit1 Register

CIDAH1 indicates which Identifier Acceptance Filter (or Filters) were triggered (Hit), and the corresponding received CAN message is in Receive Buffer 1. All bits are held in reset while the Soft Reset bit in CAN Module Control Register is asserted (SFTRES=1 in CMCR) and during a hard reset.

- **Bit 7 – Reserved**
- **Bits 6..0 – ID1HIT 6..0 - Identifier Acceptance Filter Hit 6..0 Indicator**
  - » 0 – Message in Receive Buffer 1 did NOT pass Identifier Accept Filter 6..0
  - » 1 – Message in Receive Buffer 1 passed Identifier Accept Filter 6..0

CEFR (CAN Error Flag Register)

Bit	7	6	5	4	3	2	1	0	
0x010C	-	PSVER	BOFF	OVR	STUFF	CRC	FORM	-	CEFR (Receive)
0x010C	-	PSVER	BOFF	OVR	STUFF	ACK	FORM	BIT	CEFR (Transmit)
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.19: CAN Error Flag Register (Receive or Transmit)

All bits are read only. If the EI bit (Error Interrupt) was set in the CCISR (CAN Controller Interrupt Status Register), the CEFR should be the next register read during a CAN communications interrupt service routine. The CEFR will show either the Receive or Transmit Error status depending on which message buffer type was indicated by CIID<2..0> in the CCISR. If there are no further pending CAN communications interrupts, writing CCISR will clear the bits in the CEFR as well as the CCISR. If there is a pending CAN communications interrupt, writing to CCISR will result in both the CCISR and the CEFR registers being loaded with the appropriate information pending interrupt for the next value of CIID<2..0>. For any CAN communications interrupts caused by NI (Normal Interrupt) or ABTAK (Abort Acknowledge) the CEFR will remain cleared.

CEFR is held in reset while the Soft Reset bit in CAN Module Control Register is asserted (SFTRES=1 in CMCR).

- **Bit 7 – Reserved**
- **Bit 6 – PSVER - Passive Error Interrupt Flag**
  - » {Receive or Transmit} [0 at Reset]
  - » 0 – No passive error interrupt has occurred
  - » 1 – A passive error interrupt has occurred

The flag is set, if PSVIE (Enable) is set in the CIER (CAN Interrupt Enable Register), and the CAN Controller enters the error passive status due to either the Receive or Transmit Error Counters (CRXERR or CTXERR) exceeding a count of 127 and the Controller is not in the Bus Off state. Since this interrupt occurs as a result of either the Receive or Transmit Error Counters exceeding 127, the error cannot be attributed to a particular message transmission or reception. Both CRXERR or CTXERR must be read to identify the cause of the interrupt.

$PSVER = [(127 < CRXERR)\ \ OR\ \ (127 < CTXERR \leq 255)]\ AND\ NOT(BOFF)$

- **Bit 5 – BOFF - Bus Off Interrupt Flag**
  - » {Receive or Transmit} [0 at Reset]
  - » 0 – No bus off
  - » 1 – A Bus Off interrupt has occurred.

The BOFF flag is set, if BOFIE (Enabled) is set in the CIER. When the CAN Controller enters the Bus Off status due to the Transmit Error Counter (CTXERR) exceeding a count of 255, the CAN Controller cannot be reset until it has monitored 128 occurrences of 11 consecutive 'recessive' bits on the bus. Since this interrupt occurs as a result from the Transmit Error Counter exceeding 255, the error cannot be attributed to a particular message transmission or reception. Both CRXERR or CTXERR must be read to identify the cause.

- **Bit 4 – OVR- Overrun Interrupt Flag (Receive ONLY)**
- **Bit 4 – OVR - Overload Interrupt Flag (Transmit ONLY)**
  - » [0 at Reset]
  - » 0 – No Receiver Overrun or no Transmit Overload Interrupt has occurred
  - » 1 – A Receive Overrun or a Transmit Overload Interrupt has occurred

The OVR flag is set, if EIE is set in the CIER and the CAN Controller detects a Receive overrun condition. A receive overrun condition occurs when both of the Foreground Receive Buffers are full (RXF1=RXF0=1), the Background Receive Buffer is full and a CAN message is received which passes at least one of the Identifier Acceptance Filters.

The OVR flag is set, if the EIE bit is set in the CIER and the CAN Controller receive an overload frame in response to a transmission.

- **Bit 3 – STUFF- Stuff Error Interrupt Flag**
  - » {Receive or Transmit} [0 at Reset]
  - » 0 – No Stuff Error interrupt has occurred
  - » 1 – A Stuff Error has occurred

The STUFF flag is set, if EIE is set in the CIER and the CAN Controller detects a bit stuffing error in a transmitted or received message. A STUFF error is six consecutive bits of the same polarity in a message field that should have been coded by the bit stuffing method required by the CAN protocol.

- **Bit 2 – CRC- CRC Error Interrupt Flag (Receive ONLY)**
- **Bit 2 – ACK - An ACK error was detected (Transmit ONLY)**
  - » [0 at Reset]
  - » 0 – No CRC error interrupt; 0 – No ACK error interrupt
  - » 1 – A CRC interrupt has occurred; 1 – An ACK error interrupt occurred

The ACK flag is set, if EIE is set in the CIER and the CAN Controller detects an ACK error in a transmitted message. An ACK error occurs when the CAN Controller did not detect a dominant bit in the ACK field of the transmitted message. This can happen when there are no other stations on the network or they are not operating properly.

- **Bit 1 – FORM (Receive ONLY)**
- **Bit 1 – FORM (Transmit ONLY)**
  - » 0 – No Form error interrupt; 0 – No Form Error
  - » 1 – A Form error interrupt occurred;
  - » 1 – A Form error was detected in the transmitted message. One of the fixed form fields of the message contains one or more illegal bits

The FORM flag is set, if EIE is set in the CIER and the CAN Controller detects a Form error in a transmitted or received message. A Form error occurs when one of the fixed form fields of the message contains one or more illegal bits. A dominant bit in the last bit of an End Of Frame of a received message is NOT treated as a Form Error.

- **Bit 0 – Reserved (Receive ONLY)**
- **Bit 0 – BIT Error (Transmit ONLY)**
  - » 0 – No Bit Error
  - » 1 – A Bit error was detected in the transmitted message

Error Counters and Fault Confinement

With respect to Fault Confinement, a unit may be in one of the three following states:

- Error Active
- Error Passive
- Bus Off

An Error Active unit takes part in bus communication and can send an Active Error frame when the CAN Macro detects an error.

An Error Passive unit cannot send an active error frame. It takes part in bus communication, but when an error is detected a passive error frame is sent. In addition, after a transmission an error passive unit will wait before initiating further transmission.

A Bus Off unit is not allowed to have any influence on the bus. For fault confinement, two error counters, Transmit Error Counter (TXERR) and Receive Error Counter (RXERR), are implemented.

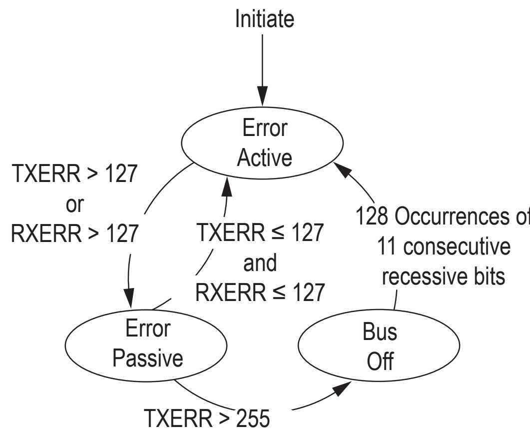


Figure 10.6: Line Error Mode

CRXERR (CAN Receive Error Counter)

Bit	7	6	5	4	3	2	1	0	
0x010D	RXERR<7..0>								CRXERR
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.20: CAN Receive Error Counter

- **Bits 7..0 – Receive Error Counter**

  The CAN Receive Error Counter (CRXERR) shows the number of receive errors detected by the CAN Controller. The value of this register is incremented or decremented as described in the Fault Containment section of the CAN Specification. BSERST in register CMCR will clear this register.

CTXERR (CAN Transmit Error Counter)

Bit	7	6	5	4	3	2	1	0	
0x010E	TXERR<7..0>								CTXERR
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 10.21: CAN Transmit Error Counter

- **Bits 7..0 – Transmit Error Counter**

  The CAN Transmit Error Counter (CTXERR) shows the number of Transmit errors detected by the CAN Controller. The value of this register is incremented or decremented as described in the Fault Containment section of the CAN Specification. BSERST in register CMCR will clear this register.

CVER (CAN Module Firmware Version)

Bit	7	6	5	4	3	2	1	0	
0x010F	CVER								CVER
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	1	0	1	1	0	

Table 10.22: CAN Module Firmware Version

- **Bits 7..0 – M3000's CAN Module Firmware Version**

CAN RX Acceptance Filtering

There are 7 (0..6) acceptance filters associated with the receiver which are used to screen incoming messages. Each acceptance filter (screen) is comprised of four (4) Code Registers and four (4) corresponding Mask Registers.

The combination of the Code and Mask Registers filter (screen) the Identifier and IDE portion of the CAN message frame. A match of the identifier and IDE to the Code and Mask screen is called a “hit”. The Data portions of the CAN Message frame will be accepted into one of the two (2) receiver buffers provided there are no CAN errors.

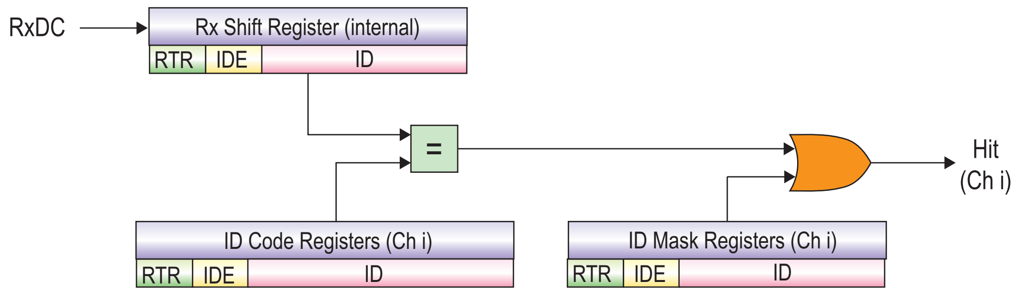


Figure 10.7: Acceptance Filter Block Diagram

Example:

To accept only ID = 318h
ID CODE = 011 0001 1000 b
ID MSK* = 000 0000 0000 b

*ID MSK Note:

1 = Don’t Care
0 = Must Match

CIDACnR<0..3> CIDMnR<0..3> (CAN Identifier Acceptance Code and Mask Registers)

Software Coding

CID[uIAC / uIAM]

ID Code Registers	CIDACnR3				CIDACnR2	CIDACnR1	CIDACnR0
	Bit-7	Bit-6	Bit-5	Bits<4..0>7	Bits<7..0>7	Bits<7..0>7	Bits<7..0>7
CAN Identifier Bit Positions, IDE and RTR		↓	↓	↓ ↓ ↓	↓ ↓ ↓	↓ ↓ ↓	↓ ↓ ↓
		RTR	IDE	ID<28..24>	ID<23..16>	ID<15..08>	ID<07..00>
		↑	↑	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑	↑ ↑ ↑
ID Mask Registers	Bit-7	Bit-6	Bit-5	Bits<4..0>7	Bits<7..0>7	Bits<7..0>7	Bits<7..0>7
	CIDMnR3				CIDMnR2	CIDMnR1	CIDMnR0

Table 10.23: CAN Identifier Acceptance Code and Mask Registers

If the corresponding bit of the Mask Register is 1 then the match is ignored. If the corresponding bit of the Mask Register is 0 a match is considered for that bit.

The Boolean expression for the bit-wise comparison is given below. A hit is indicated if “accept_this_bit” is true for all bits for the length of the identifier (ID), IDE and RTR.

accept_this_bit = (ID_bit = code_bit) OR (mask_bit = 1)

There are seven (7) Code/Mask Combinations. (See the table that follows.) Under the heading “Data Space Address”, in the first column titled “Base” the Rx Buffer numbers are <0..6>. The Base Data Address (0xnnnn) is to the right.

Example: The third Rx Buffer is #2 and would be written as CIDAC<2>Rn

The second column under Data Space Address is the “Offset”. The Offset Numbers <0..4> are for one of four Registers under the Base Buffer Address. If you were addressing Rx Buffer #2, Register #3, (CIDAC2R2) the Register Address would be 0x0122.

The Offset numbers for the Mask Registers are R<4..7>. If you were addressing Mask Register CIDM2R2 the Data Address would be 0x0126.

Data Space Address Base	Offset	Code Registers		Bit-7	Bit-6	Bit-5	Bits-4..0
Rx Buffer 0 - Ox0110	+0	CIDACnR0	Read: Write:	AC<7..0>			
Rx Buffer 1 - Ox0118	+1	CIDACnR1	Read: Write:	AC<15..8>			
Rx Buffer 2 - Ox0120	+2	CIDACnR2	Read: Write:	AC<23..16>			
Rx Buffer 3 - Ox0128	+3	CIDACnR3	Read: Write:	0	RTRAC	IDEAC	AC<28..24>
Rx Buffer 4 - Ox0130		Mark Registers		Bit-7	Bit-7	Bit-7	Bits-4..0
Rx Buffer 5 - Ox0138	+4	CIDMnR0	Read: Write:	AM<7..0>			
Rx Buffer 6 - Ox0140	+5	CIDMnR1	Read: Write:	AM<15..8>			
	+6	CIDMnR2	Read: Write:	AM<23..16>			
	+7	CIDMnR3	Read: Write:	1	RTRAM	IDEAM	AM<28..24>

Table 10.24: CAN Identifier Acceptance Code and Mask Register Map

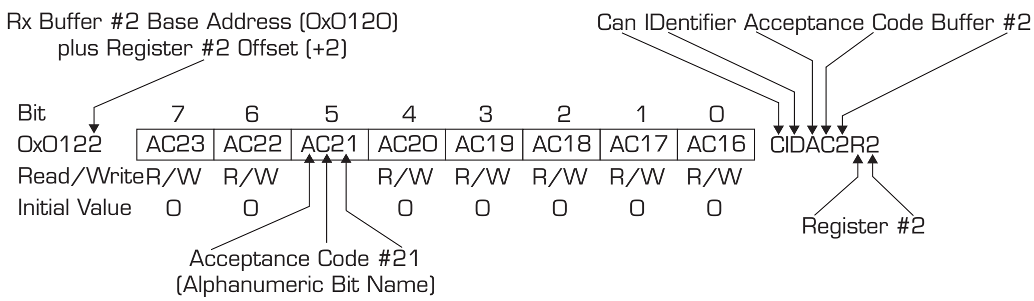


Figure 10.8: Example of a CAN Identifier Acceptance Code & Mask Register

All Bits are (Read/Write) [Code Bits are 0 at Reset - Mask Bits are 1 at Reset] To write to ID Code Registers and ID Mask Registers, the corresponding Acceptance Filter (CRAFEN) must be disabled.

- **AC<28..0> - Acceptance Code**
  - » 0 – Standard Format (11-bit ID) use AC<28..18>
  - » 1 – Extended Format (29-bit ID) use AC<28..0>
- **IDEAC - Identifier Acceptance Code Format**
  - » 0 – Standard format (11 Bit) identifier
  - » 1 – Extended format (29 Bit) identifier
- **RTRAC - Remote Return Request Acceptance Code Format**
  - » 0 – Standard Message
  - » 1 – RTR Message
- **AM<28..0> - Acceptance Mask**
  - » 0 – Match corresponding Acceptance Code Register Bit
  - » 1 – Ignore corresponding Acceptance Code Register Bit
- **IDEAM - Identifier Acceptance Mask**
  - » 0 – Match Corresponding IDE bit
  - » 1 – Ignore Corresponding IDE bit
- **RTRAM - Remote Return Request Acceptance Mask**
  - » 0 – Match corresponding RTR Bit
  - » 1 – Ignore corresponding RTR Bit

IDEAC indicates whether the Identifier Acceptance Filter should look at Standard or Extended Identifiers. If the Standard Identifier is selected only the 11 most significant bits of the filter are compared and subject to the Mask Register.

Message Handling Overview

COB = Communication Object

dlc = data length code

RTR = Remote Request Return

IDE = (0 = 11 Bit ID, 1 = 29 Bit ID)

The following table illustrates the general organization of the Transmit and Receive Message Buffer. Both buffers have the same general structure. All buffers reside on memory boundaries which have beginning Data Addresses at 0x150.

CAN Controller Register Map			
Data Space Address		Register Name	Comment
Base	Offset		
TX Buffer 0 — 0x0150 TX Buffer 1 — 0x0160 TX Buffer 2 — 0x0170 RX Buffer 0 — 0x0180 RX Buffer 1— 0x0190	+0	Identifier Register 0	11 or 29 Bit COB-ID
	+1	Identifier Register 1	
	+2	Identifier Register 2	
	+3	Identifier Register 3	
	+4	Data Segment Register 0	Up to 8 Bytes of CAN Data
	+5	Data Segment Register 1	
	+6	Data Segment Register 2	
	+7	Data Segment Register 3	
	+8	Data Segment Register 4	
	+9	Data Segment Register 5	
	+A	Data Segment Register 6	
	+B	Data Segment Register 7	
	+C	Message Control and Data Length Register	RTR, IDE. dlc
	+D	Transmit Buffer Priority	TX Priority
	+E	Unused	
	+F	Unused	

* The priority register is only used in the Transmit Buffer. It is not used in the Receive Buffer.

Table 10.25: Organization of a Transmit/Receive Message Buffer

Programmers Model of Message Storage

Transmit Buffer Structure Description

The Transmit Buffer Structures in the tables below illustrate the details of the structure of the Transmit Buffers. All Transmit Buffers are write-only. Reads of the Transmit Buffers will result in all zeros. Locations 0xnnnE and 0xnnnF are not implemented. Writes to these two bytes will have no effect. Those bits shaded gray are also not implemented and writing to those bits will have no effect. The IDE bit (bit 4 in the DLR) indicates whether the message to be transmitted has a standard or extended identifier. If IDE=0 the transmitted message uses a standard identifier and bits ID17 - ID0 will be ignored. The 11-bit standard identifier is in location ID<28..18>.

When more than one Transmit Buffer is scheduled for transmission, the CAN Controller uses the Transmit Buffer Priority Register (TBPR) to determine which message will be sent first. The lowest numerical value TBPR has the highest priority and will be sent first. If more than one pending transmission message has the same priority, the lowest value numerical buffer enabled (TXE=0) will be sent first (i.e. Transmit Buffer 0 is sent first, then Transmit Buffer 1, then Transmit Buffer 2).

Transmit Buffer Structure (Extended Identifier)

Software Coding

CTXB [ulIDR ucDSR[ ] ucDLR ucTBPR]

Transmit Buffer Structure - Extended Identifier											
Group	Data Offsets	Register		Register Bits							
				7	6	5	4	3	2	1	0
COB-ID	0	IDR0	Write:	← ID[7..0] →							
	1	IDR1	Write:	← ID[15..8] →							
	2	IDR2	Write:	← ID[23..16] →							
	3	IDR3	Write:	0			← ID[28..24] →				
Up to 8 Bytes of CAN Data	4..8	DSR[0..7]	Write:	← DB[0..63] →							
RTR, IDE, dlc	C	DLR	Write:	0		RTR	IDE(=1)	← DLC[3..0] →			
Transmit Buffer Priority Register	D	DBPR	Write:	0						PRI[1..0]	
Unused	E..F		Write:								

Table 10.26: Transmit Buffer Structure - Extended Identifier

Transmit Buffer Structure (Standard Identifier)

Group	Data Offsets	Register		7	6	5	4	3	2	1	0
Transmit Buffer Structure - Standard Identifier											
				Register Bits							
COB-ID	0	IDR0	Write:								
	1	IDR1	Write:								
	2	IDR2	Write:	← ID[23..16] →							
	3	IDR3	Write:	0			← ID[28..24] →				
Up to 8 Bytes of CAN Data	4..8	DSR[0..7]	Write:	← DB[0..63] →							
RTR, IDE, dlc	C	DLR	Write:	0		RTR	IDE(=0)	← DLC[3..0] →			
Transmit Buffer Priority Register	D	DBPR	Write:	0						PRI[1..0]	
Unused	E..F		Write:								

Table 10.27: Transmit Buffer Structure - Standard Identifier

Receive Buffer Structure Description

The Receive Buffer Structures in the tables below illustrate the details of the structure of the Receive Buffers. All Receive Buffers are read-only. Writes to the Receive Buffers will have no effect. The IDE bit (bit 4 in the DLR) indicates whether the received message has a standard or extended Identifier. If IDE=0 the received message had a standard identifier and bits ID17 - ID0 will read zero. The standard identifier is in location ID<28..18>.

Receive Buffer Structure (Extended Identifier)

Software Coding

CRXB [ulIDR ucDSR[ ] ucDLR ucTBPR]

Group	Data Offsets	Register		7	6	5	4	3	2	1	0
Recieve Buffer Structure - Extended Identifier											
				Register Bits							
COB-ID	0	IDR0	Read:	← ID[7..0] →							
	1	IDR1	Read:	← ID[15..8] →							
	2	IDR2	Read:	← ID[23..16] →							
	3	IDR3	Read:	← 0 →			← ID[28..24] →				
Up to 8 Bytes of CAN Data	4..8	DSR[0..7]	Read:	← DB[0..63] →							
RTR, IDE, dlc	C	DLR	Read:	← 0 →		RTR	IDE(=1)	← DLC[3..0] →			
Transmit Buffer Priority Register	D	DBPR	Read:	← 0 →						PRI[1..0]	
Unused	E..F		Read:	← 0 →							

Table 10.28: Receive Buffer Structure - Extended Identifier

Receive Buffer Structure (Standard Identifier)

Group	Data Offsets	Register		7	6	5	4	3	2	1	0
Receive Buffer Structure - Standard Identifier											
				Register Bits							
COB-ID	0	IDR0	Read:	← 0 →							
	1	IDR1	Read:	← 0 →							
	2	IDR2	Read:	← ID[23..18] →						← 0 →	
	3	IDR3	Read:	← 0 →			← ID[28..24] →				
Up to 8 Bytes of CAN Data	4..8	DSR[0..7]	Read:	← DB[0..63] →							
RTR, IDE, dlc	C	DLR	Read:	← 0 →		RTR	IDE(=0)	← DLC[3..0] →			
Transmit Buffer Priority Register	D	DBPR	Read:	← 0 →							
Unused	E..F		Read:	← 0 →							

Table 10.29: Receive Buffer Structure - Standard Identifier

SECTION 11
GENERAL PURPOSE A/D INTERFACE

Introduction

The M3000 features an Analog to Digital converter consisting of a 10-bit Cascaded Potentiometric Digital to Analog Converter connected to a Sample and Hold Comparator, and a Logic Block.

This DAC is based on a string of 64 polysilicon resistors connected between reference input Vrefp and the ground. Tap-off points are provided at intervals along the string.

Features

- 10-bit Resolution
- ±2.5 LSB Integral Non-linearity
- ±1 Differential Non-Linearity
- 27.5 µs Conversion Time
- 36 kSPS at Maximum Resolution
- 0 - VCC ADC Input Voltage Range
- Interrupt on ADC Conversion Complete

Specifications

Analog to Digital Converter Specifications					
Parameter	Condition	Min.	Typ.	Max	Unit
Resolution			10		Bit
Conversion Time	11 clk (1 for sample, 10 for conversion)	27.5			µs
Startup Time			2	4	µs

Table 11.1: Analog to Digital Converter Specifications

Circuit Operation

When initiating a conversion by setting the ADSC bit in ADCSR, the conversion starts at the following rising edge of the ADC clock cycle. A normal conversion takes 11 ADC clock cycles.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set and ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

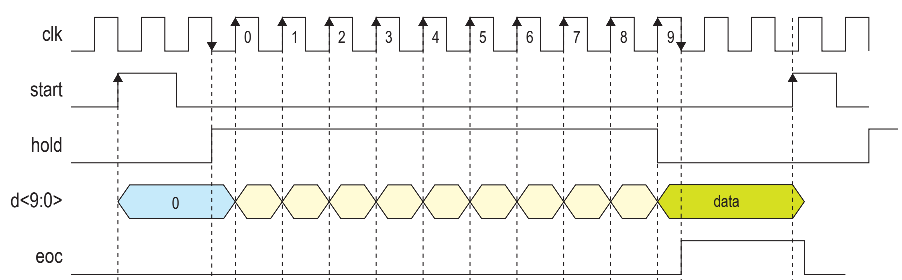


Figure 11.1: Analog to Digital Converter Timing Chart

General Purpose A/D Register Interface

A register interface wrapper for the ADC019 module allows the AVR core to initiate and subsequently read converted values. The interface consists of three registers: a control register and two data registers to capture the 10-bit converted value. The processor initiates a conversion by setting bit 6 of the ADCSR to logic one. Several cycles later, the ADC will generate an interrupt, indicating that its conversion has completed. Once conversion is completed, the value will be stored in the result registers until another conversion has completed.

ADC Registers

ADCSR (ADC Control and Status)

Bit	7	6	5	4	3	2	1	0	
0x07(0x27)	ADCON	ADSC	–	ADIF	ADIE	–	–	ADBSY	ADCSR
Read/Write	R	R/W	R	R/W	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 11.2: ADC Control and Status Register

- **Bit 7 – ADCON: ADC On**

  Setting this bit turns the ADC power on.
- **Bit 6 – ADSC: ADC Start Conversion**

  Setting bit starts conversion. Cleared when conversion complete.
- **Bit 5 Reserved**

  This bit is reserved and always read as zero.
- **Bit 4 – ADIF: ADC Interrupt Flag**

  This bit is set to when a conversion is complete. It is 'anded' with the ADIE bit to cause a processor interrupt. This bit is cleared when logic one is written to the flag.
- **Bit 3 – ADIE: ADC Interrupt Enable**

  When set, an ADC interrupt is enabled to the processor core.

Bits 2..1 Reserved

These bits are reserved and always read as zero.

- **Bit 0 – ADBSY: ADC Conversion Busy**

  When this bit is set, the ADC is performing a conversion .

ADRSLTHI (ADC Result High)

Bit	7	6	5	4	3	2	1	0	
0x06(0x26)	–	–	–	–	–	–	ADC[9:8]		ADRSLTHI
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 11.3: ADC Result High Register

- **Bits 7..2 Reserved**

  These bits are reserved and always read as zero
- **Bits 1..0 – ADC[9:8]: ADC Value [9:8]**

  This value contains bits 9 and 8 of the converted ADC value.

ADRSLTLO (ADC Result Low)

Bit	7	6	5	4	3	2	1	0	
0x05(0x25)	ADC[7:0]								ADRSLTLO
Read/Write	R								
Initial Value	0000_0000								

Table 11.4: ADC Result Low Register

- **Bits 7..0 – ADC[7:0]: ADC Value [7:0]**

  This value contains bits 7 through 0 of the converted ADC value.

SECTION 12
GENERAL PURPOSE D/A INTERFACE

Introduction

The General Purpose D/A interface is a 10-bit Cascaded Potentiometric Digital to Analog Converter. This cell is based on a string of 64 Polysilicon Resistors connected between reference inputs and ground. Tap-off points are provided at intervals along the string.

The six most significant bits of the digital number to be converted select two of these tap-off points to be connected to intermediate signals A and B. The voltage difference is applied across a string of transmission gates. The four least significant bits of the number to be converted select a tap-off point from this string to be connected to the Analog Ouput signal.

The D/A Converter has no sample-and-hold facility. When given a digital number, it outputs the analog equivalent after the internal delay (flash). The analog output voltage persists as long as, and only as long as, the digital number is available at the input. To minimize transients when changing the digital input code, this changing should be synchronous.

General Purpose D/A Register Interface

The interface consists of two registers (DACVALHI, DACVALLO) that are written by the processor to provide a digital value to be converted.

DAC Registers

DACVALHI (DAC Conversion Value High Byte)

Bit	7	6	5	4	3	2	1	0	
0x19(0x39)	DACON	–	–	–	–	–	DAC[9:8)		DACVALHI
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 12.1: DAC Conversion Value High Byte Register

- **Bit 7 – DACON: DAC On**

  When this bit is set, the DAC is powered. When it is cleared, the DAC is on Standby.

- **Bits 7..2 Reserved**

  These bits are reserved and always read as zero.

- **Bits 1..0 – DACVAL[9:8]: DAC Conversion Value [9:8]**

  This value contains bits 9 and 8 of the Digital value to be converted.

DACVALLO (DAC Conversion Value Low Byte)

Bit	7	6	5	4	3	2	1	0	
0x18(0x38)	DAC[7:0]								DACVALLO
Read/Write	R/W								
Initial Value	0000_0000								

Table 12.2: DAC Conversion Value Low Byte Register

NOTE

Note: The data is not buffered. The bytes are transferred as they are written, therefore there will be an intermediate value when the word changes across the byte boundary.

- **Bits 7..0 – DACVAL[7:0]: DAC Conversion Value [7:0]**

  This value contains bits 7 through 0 of the Digital value to be converted.

Page Intentionally Left Blank

SECTION 13
ANALOG SWITCH

Introduction

An analog switch is provided to switch a single pin on the M3000 between being an analog input for the General Purpose A/D converter and being an output from the General Purpose D/A converter under the control of an I/O register. The following diagram depicts this function and the actual implementation of the Analog Switch.

Analog Switch Control

The switch is controlled by the AMUXCTL register. When bit 7 of the register is set to logic one, the AD_IN_OUT pin is driven by the General Purpose D/A and ISET_AD_IN is connected to the General Purpose A/D. When bit 7 is cleared (logic zero) AD_IN_OUT drives the General Purpose A/D

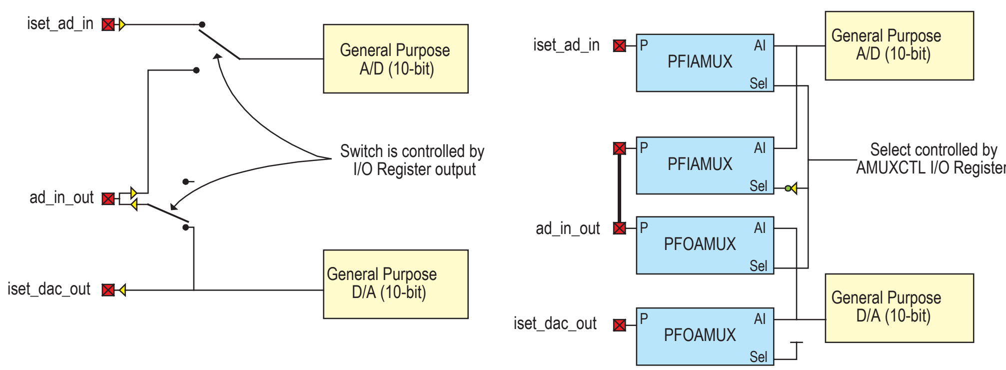


Figure 13.1: Analog Switch Logic

Control Register

AMUXCTL (Analog Mux Control)

NOTE

Note: When using the MUX_AD_DA as an input there will be a contention until AMUX is cleared. Protect against short circuit.

Bit	7	6	5	4	3	2	1	0	
0x0B(0x2B)	AMUX	–	–	–	–	–	–	–	AMUXCTL
Read/Write	R/W	R	R	R	R	R	R	R	
Initial Value	1	0	0	0	0	0	0	0	

Table 13.1: Analog MUX Control Register

- **Bit 7 – AMUX: Analog Mux Select**

  When this bit is set (logic one), the MUX_ADC_DAC pin is an output driven by the General Purpose DAC (DAC10). When cleared (logic zero), the MUX_ADC_DAC pin is an input that drives the General Purpose A/D converter (ADC10)

- **Bit 6..0 Reserved**

  These bits are reserved and always read as zero

Page Intentionally Left Blank

SECTION 14
CRYSTAL OSCILLATOR

Introduction

The M3000 Motor and Motion Control ASIC operates on a frequency of 20 MHz. The oscillator is a customer supplied item.

A typical Crystal Oscillator diagram is shown in Figure 14.1.

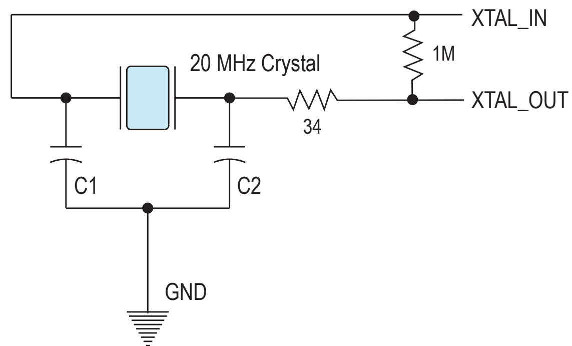


Figure 14.1: Crystal, Capacitor and Resistor Connections

The optimal value of the capacitors and resistors depends on the crystal in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Use capacitor and resistor values recommended by the Crystal manufacturer.

A 20 MHz Ceramic Resonator may also be used. Ceramic Resonators are manufactured with the capacitors built in. A bias resistor is typically required.

A typical Ceramic Resonator diagram is shown in Figure 14.2.

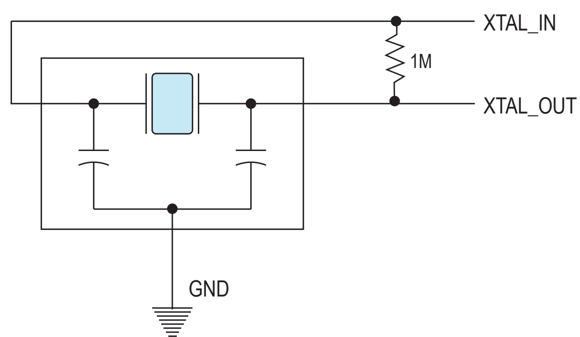


Figure 14.2: Typical Ceramic Resonator Installation

Page Intentionally Left Blank

SECTION 15
RESET

Introduction

For proper operation, the M3000 must be reset. There are several sources/conditions that will reset all or a portion of the device. Initially, a power-on Reset must be asserted on the input pin RESET_N. Reset generation comes from an external reset pin (RESET_N), a nonexistent memory timeout or a watchdog timer timeout.

Reset Sources

External

At power up, an active low reset must be asserted on input pin reset_n.

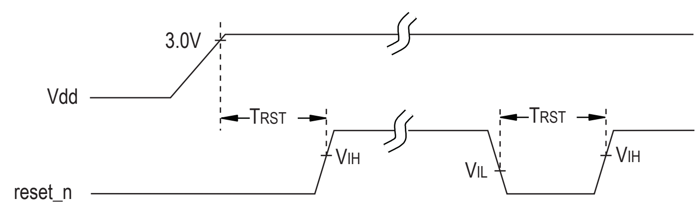


Figure 15.1: Reset Timing Diagram

Reset Timing Chart				
Symbol	Parameter	Min.	Max	Unit
T_{RST}	Reset Period	5	—	ns
V_{IH}	Input High	2.0	—	V
V_{IL}	Input Low	—	1.2	V

Table 15.1: Reset Timing Chart

Watchdog

The internal watchdog timer causes a reset when the timer period expires and the watchdog is enabled.

NEMR

An attempted access to Nonexistent Program Memory will cause a reset.

JTAG

A JTAG commanded reset through the AVR core.

Operation

The reset structure splits the system into sections such that the source of the reset determines what is reset. The sections are: The AVR core, the logic (motor / motion, CAN and peripherals) and the JTAG controller. (See the Reset Structure Block Diagram on the following page.)

External activation will reset the entire ASIC. The JTAG tap controller is placed in the Test-Logic-Reset state, the AVR is set to the reset vector and the logic is set to the initial values. External resets are asynchronously asserted and released for the JTAG controller and synchronously released for the AVR core and logic. In the event the system clock fails to start, the ASIC will be held in reset. During the external reset period the CAN_TXD pin is tri-stated.

A watchdog timeout or a nonexistent memory access will cause a one clock cycle pulse to begin a reset. The synchronous release registers will extend the period by 2 clock cycles, resetting the AVR core and the logic. The JTAG controller is not reset and the CAN_TXD pin is not tri-stated.

A JTAG command reset through the AVR core will reset the logic only. The AVR and JTAG controller will not be reset. The CAN_TXD pin is not tri-stated.

As previously noted, the JTAG tap controller is reset to the Test-Logic-Reset by an external reset. The tap controller (only) may also be reset by asserting the JTAG_TRST pin low or by asserting the JTAG_TMS input high and applying five rising edges at pin JTAG_TCLK.

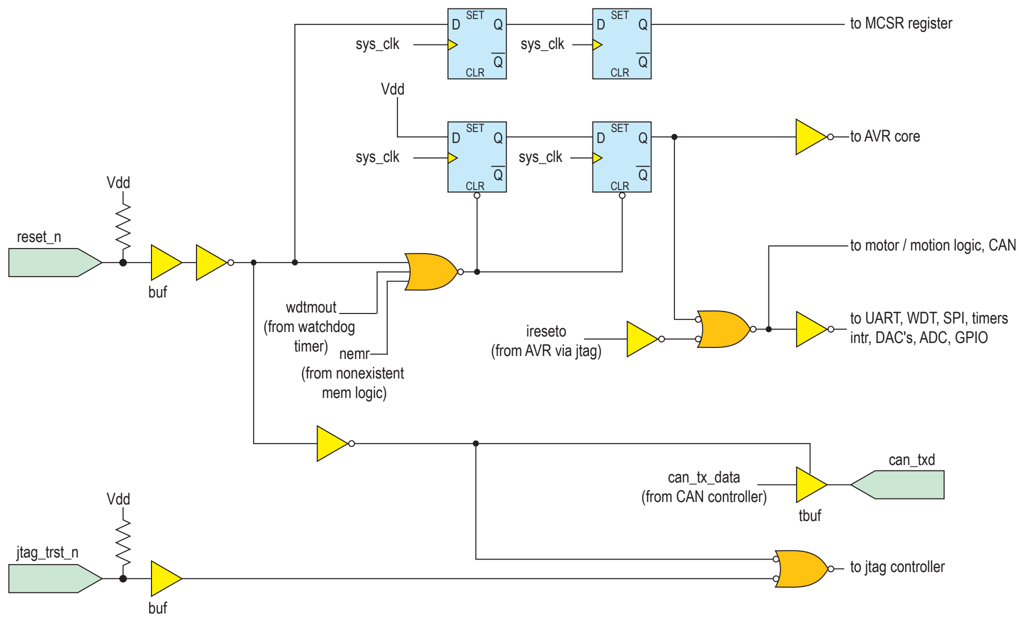


Figure 15.2: Reset Structure Block Diagram

MCSR Register

This register can be used to determine the source of the reset. A reset caused by the external pin, a watchdog timeout or a nonexistent memory access will set a corresponding bit in the register.

The function of this register is covered in detail in Section 16 of this document.

SECTION 16
MICRO CONTROLLER STATUS REGISTER

Introduction

The Micro Controller Status Register monitors several fault and reset conditions on the M3000. They are:

- Boot ROM Test Fail
- Watchdog Reset
- External Reset
- Nonexistent Memory Access Reset

Micro Controller Status Register

MCSR (Micro Controller Status)

Bit	7	6	5	4	3	2	1	0	
0x33(0x53)	FAILN	–	–	–	NEMR	WDTR	EXTR	–	MCSR
Read/Write	R/W	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 16.1: Micro Controller Status Register

- **Bit 7 – FAILN: BootROM Test FAIL (low asserted)**

  When this bit is cleared, the BootROM tests have failed. The BootROM will set this bit if it's internal tests pass. This bit drives the FAIL_N pin

- **Bits 6..4 Reserved bits**

  These bits are reserved and always read as zero

- **Bit 3 – NEMR: Non-Existent Memory Access Reset**

  This bit is set if the memory access is out of the 64k range, causing a device reset

- **Bit 2 – WDTR: Watchdog Timer Reset**

  This bit is set when a watchdog timer time-out caused a device reset

- **Bit 1 – EXTR: External Reset**

  This bit is set when an External Reset (RESET_N pin is being driven to a logic 0). It is cleared on a watchdog timer generated reset or non-existent memory reset.

- **Bit 0 – Reserved bit**

  This bit is reserved and always read as zero

The following table describes the type of reset occurrence and the resulting value of the EXTR, WDTR and NEMR bits

Reset Cause Table	
Cause	EXTR, WDTR, NEMR
External Reset	001
Watchdog Timer Timout	010
Non-Existent Memory Access Reset	100

Table 16.2: Reset Cause Table

Page Intentionally Left Blank

SECTION 17
EXTERNAL AND INTERNAL TIMER/COUNTERS

Introduction

NOTE

NOTE: Not all External Timer/ Counter Signals are available on all packages.

Two general purpose Timer/Counters are supplied with the M3000. Each Timer/Counter consists of one 8-bit (T/C0) and one 16-bit (T/C1) Timer/Counter. The Timer/Counters have individual prescaling selection from the same 10-bit prescaling timer. One Timer/Counter is used as an Internal Timer/Counter and the second is used as an External Timer/Counter.

Features

- 8-bit Timer/Counter
- 16-bit Timer/Counter includes:
  - » Two Output Compare Functions
  - » One Input Capture Function
- 8- to 10-bit Pulse Width Modulator
- Individual 10-bit Prescaler
- Prescaled Clocking or External Clocking Schemes

The following descriptions and explanations apply to both the internal counters and external counters, except where noted.

Any of the following can be selected as clock sources for the two Timer/Counters.

- The 20 MHz master clock
- The four prescaled selections: clk/8; clk/64; clk/256; clk/1024
- An external source (t0clk for T/C0 and t1clk for T/C1)
- Stop

8-Bit Timer/Counter0

The 8-bit Timer/Counter0 can select a clock source from the 20 MHz clock, prescaled 20 MHz clock, or an external pin t0clk*. In addition it can be stopped as described in the specification for the Timer/Counter0 Control Register - EXTCCR0 and INTCCR0. Control signals may also be found in the Timer/Counter0 Control Register - EXTCCR0 and INTCCR0.

The t0clk pin* goes through two 20 Mhz clock demetastabilization cycles and then it is forwarded to the timer block. This implementation requires a period of at least two 20 MHz clock cycles.

The 8-bit Timer/Counter0 features both a high resolution and a high accuracy usage with the lower prescaling opportunities. Similarly, the high prescaling opportunities make the Timer/Counter0 useful for lower speed functions or exact timing functions with infrequent actions.

* Not available on all packages.

Timer/Counter Prescaler

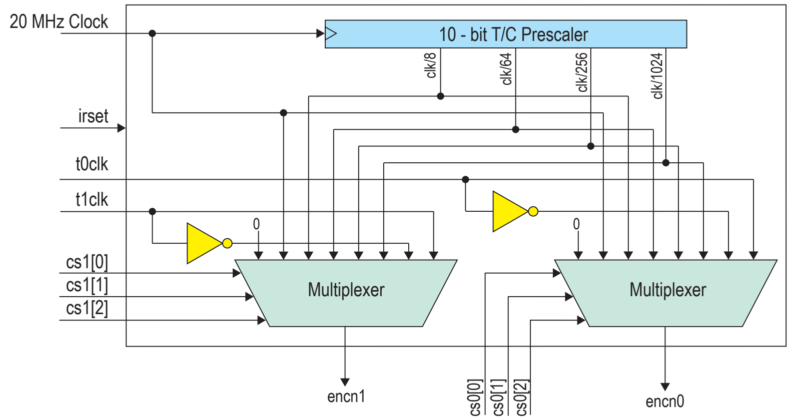


Figure 17.1: Timer/Counter Prescaler

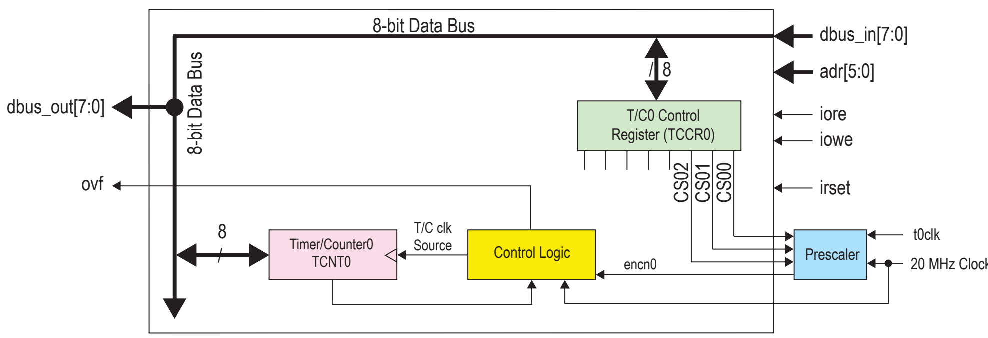


Figure 17.2: 8-Bit Timer/Counter0 Block Diagram

16-Bit Timer/Counter1

NOTE

NOTE: The t1clk pin and the External Counter/Timer OCB Output Pin is not available on all packages.

The 16-bit Timer/Counter1 can select its clock source from the 20 MHz Clock, prescaled 20 MHz Clock, or an external pin t1clk. It can also be stopped as described in the specification for the Timer/Counter1 Control Registers - EXTCCR1A, INTCCR1A, EXTCCR1B, and INTCCR1B. The different status flags (overflow, compare match and capture event) and control signals are found in the Timer/Counter1 Control Registers - EXTCCR1A, EXTCCR1B, INTCCR1A and INTCCR1B.

The t1clk pin* goes through two 20 Mhz clock demetastabilization cycles and then it is forwarded to the timer block. This implementation requires a period of at least two 20 MHz clock cycles.

The 16-bit Timer/Counter1 features both a high resolution and a high accuracy usage with the lower prescaling opportunities. Similarly, the high prescaling opportunities make the Timer/Counter1 useful for lower speed functions or exact timing functions with infrequent actions.

The Timer/Counter1 supports two Output Compare functions using the Output Compare Register 1A and 1B - EXOCR1A, EXOCR1B as the data sources to be compared to the Timer/Counter1 contents. The Output Compare functions include optional clearing of the counter on CompareA match, and actions on the Output Compare pins on both compare matches. The I/O Pins are disabled on the Internal Timer/Counter. However, the interrupts are available.

Timer/Counter1 can also be used as a 8, 9 or 10-bit Pulse Width Modulator. In this mode the counter and the EXOCR1A/EXOCR1B and INOCR1A/INOCR1B registers serve as a dual glitch-free stand-alone PWM with centered pulses. The I/O Pins are disabled on the Internal Timer/Counter, however, the interrupts are available.

The Input Capture function provides a capture of the External Timer/Counter1 contents to the Input Capture Register - ICR1. This is triggered by an external event on the Input Capture Pin - ICP0. The actual capture event settings are defined by the External Timer/Counter1 Control Register - EXTCCR1B. The ICP0 pin logic is shown in Figure 17.3.

NOTE: The ICP0 Pin is not available on all packages.

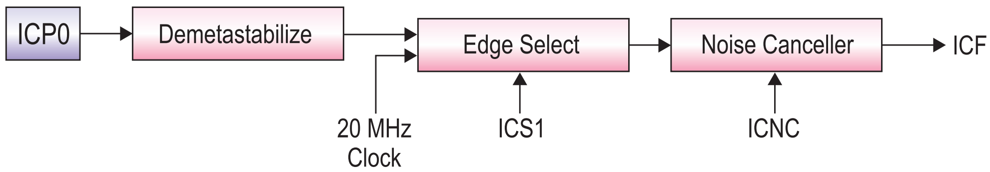


Figure 17.3: ICP Pin Schematic

The ICP0 Pin goes through two 20 MHz clock demetastabilization cycles and is then forwarded to the Timer. This implementation requires that the pulse for Input Capture must be at least two (2) clock cycles in duration.

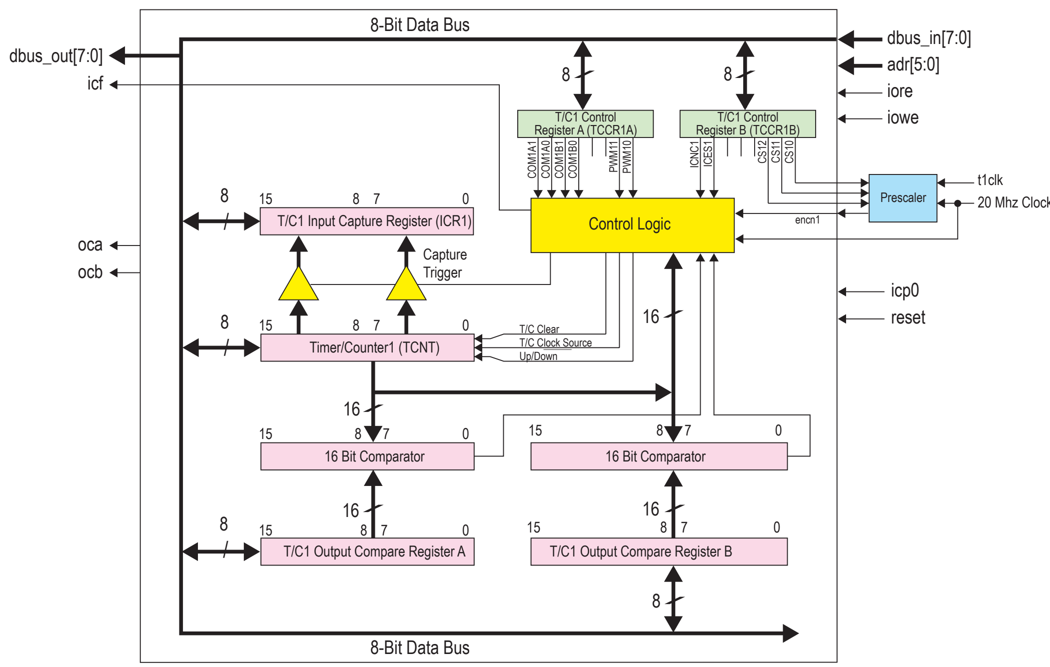


Figure 17.4: 16-Bit Timer/Counter1 Block Diagram

Input Capture Noise Canceler

If the noise canceler function is enabled, the actual trigger condition for the capture event is monitored over 4 samples before the capture is activated. The input pin signal is sampled at CPU Clock frequency.

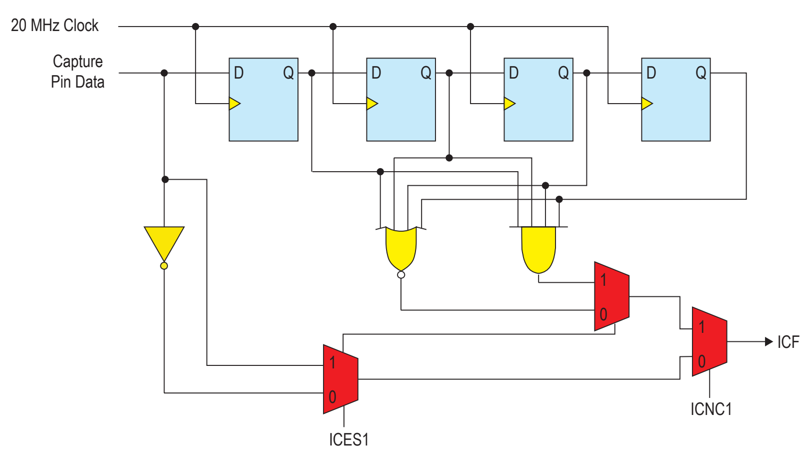


Figure 17.5: Input Capture Noise Cancelation

Timer/Counter1 in PWM Mode

NOTE:The OCB Pin on the External Timer/Counter is not available on all packages.

The OCA and OCB Pins are not available on the Internal Timer/Counter, however the interrupts are available.

When the PWM mode is selected, Timer/Counter1 and the Output Compare Register1A - EXOCR1A or INOCR1A, and the Output Compare Register1B - EXOCR1B or INOCR1B, form a dual 8, 9 or 10-bit free-running, glitch-free and phase correct PWM with outputs on the OCA and OCB* pins. Timer/Counter1 acts as an up/down counter, counting up from 0x0000 to TOP (see table), when it turns and counts down again to zero before the cycle is repeated. When the counter value matches the contents of the 10 least significant bits of EXOCR1A / INOCR1A or EXOCR1B / INOCR1B, the OCA/OCB** pins are set or cleared according to the settings of the COM1A1/COM1A0 or COM1B1/COM1B0 bits in the Timer/Counter1 Control Register EXTCCR1A / INTCCR1A. Refer to Tables 17.1 and 17.2 for details.

PWM Frequency Table		
PWM Resolution	Timer Top Value	Frequency
8-bit	0x00FF	$f_{TC1}/510$
9-bit	0x01FF	$f_{TC1}/1022$
10-bit	0x03FF	$f_{TC1}/2046$

Table 17.1: PWM Frequency Chart

Compare Effects on TCX1		
COM1X1	COM1X0	Effect pm on OCX1
0	0	Not Connected
0	1	Not Connected
1	0	Cleared on compare match, upcounting. Set on compare match, downcounting (non-inverted PWM)
1	1	Cleared on compare match, downcounting. Set on compare match, upcounting (inverted PWM)

Table 17.2: Compare Effects on OCX1

Note that in the PWM mode, the 10 least significant EXOCR1A / EXOCR1B and INOCR1A / INOCR1B bits, when written, are transferred to a temporary location.

They are latched when Timer/Counter1 reaches the value TOP. This prevents the occurrence of odd-length PWM pulses (glitches) in the event of an unsynchronized EXOCR1A / EXOCR1B or INOCR1A / INOCR1B write.

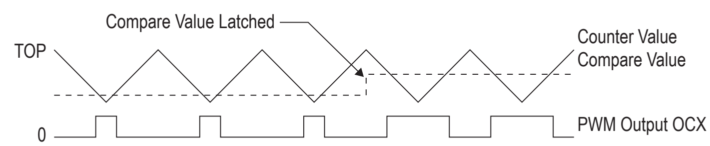


Figure 17.6: Synchronized OCR1X Latch

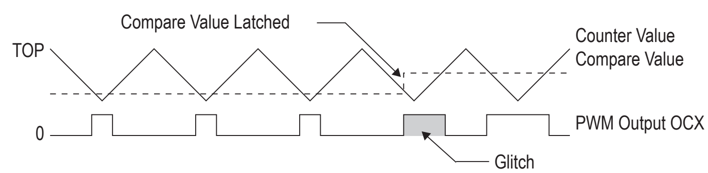


Figure 17.7: Unsynchronized OCR1X Latch

When OCR1 contains 0x0000 or TOP, the output OCA/OCB is held low or high according to the settings of COM1A1/COM1A0 or COM1B1/COM1B0. This is shown in Table 17.3.

PWM OCX Outputs			
COM1X1	COM1X0	OCR1X	Output OCX
1	0	0x0000	L
1	0	TOP	H
1	1	0x0000	H
1	1	Top	L

Table 17.3: PWM OCX Outputs

In PWM mode, the Timer Overflow Flag1 (OVF) is set when the counter changes direction at 0x0000. Timer Overflow Interrupt operates exactly as in normal Timer/Counter mode.

External Timer/Counter Registers

EXTCCR0 (External Timer/Counter0 Control Register)

Bit	7	6	5	4	3	2	1	0	
0x2D(0x4D)	–	–	–	–	–	CS02	CS01	CS00	EXTCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.4: External Timer/Counter0 Control Register

- **Bits 7..3 Reserved**

These bits are reserved and always read as zero

- **Bits 2..0 – CS0[2:0]: Clock Select0 bits 2:0**

The Clock Select0 bits 2:0 define the prescaling source of Timer0 according to Table 17.5:

The Stop condition provides a Timer Enable/Disable function. The divided modes are scaled directly from the 20 MHz clock.

Timer/Counter0 Prescaling (20 MHz Clock)			
CS02	CS01	CS00	Description
0	0	0	Stop! The Timer/Counter0 has stopped
0	0	1	CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	External Pin t0clk, falling edge
1	1	1	External Pin t0clk, rising edge

Table 17.5: Timer/Counter0 Prescaling

EXTCNT (External Timer/Counter0)

Bit	7	6	5	4	3	2	1	0	
0x2C(0x4C)	EXTCNT0[7:0]								EXTCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.6: External Timer/Counter0 Register

- **Bit 7:0 EXTCNT0[7:0]**

Timer/Counter0 is implemented as an up-counter with read and write access. If the Timer/Counter0 register is written and a clock source is present, the Timer/Counter0 continues counting in the clock cycle following the write operation.

EXTCCR1A (External Timer/ Counter1 Control Register A)

Bit	7	6	5	4	3	2	1	0	
0x20(0x40)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	PWM11	PWM10	EXTCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.7: External Timer/Counter1 Control Register A

- **Bits 7..6 – COM1A[1:0]: Compare Output Mode1 A [1:0]**

These bits determine output pin actions following an OCR1A compare match to Timer/Counter1. Output pin actions affect pin EXT_OCA. The configuration is shown in Table 17.8.

COM1 Timer Table		
COM1X1	COM1X0	Description
0	0	Timer Counter1 disconnected from output pin OCX
0	1	Toggle the OCX output line
1	0	Clear the OCX output line (to zero)
1	1	Set the OCX output line (to one)

Note: OCX=OCA or OCB, the external OCB Output pin is not available on all packages.

Table 17.8: COM1 Timer Table

- **Bits 5..4 – COM1B[1:0]: Compare Output Mode1 A [1:0]**

These bits determine output pin actions following an OCR1B compare match to Timer/Counter1. Output pin actions affect pin EXT_OCB*. The configuration is shown in the table below:

In PWM mode, these bits have a different function.

When changing the COM1X1/COM1X0 bits, Output Compare Interrupts 1 must be disabled by clearing their Interrupt Enable bits the corresponding external IRQ register. Otherwise an interrupt can occur when the bits are changed.

- **Bits 3..2 Reserved**

These bits are reserved and always read as zero

- **Bits 1..0 – PWM1[1:0]: Pulse Width Modulator Select bits [1:0]**

These bits select PWM operation of the Timer/Counter1 as specified in Table 17.9.

PWM Mode Select		
PWM11	PWM10	Description
0	0	PWM operation of the Timer/Counter1 is disabled
0	1	Timer/Counter1 is an 8-bit PWM
1	0	Timer/Counter1 is an 9-bit PWM
1	1	Timer/Counter1 is an 10-bit PWM

Table 17.9: PWM Mode Select

EXTCCR1B (External Timer/ Counter1 Control Register B)

Bit	7	6	5	4	3	2	1	0	
0x21(0x41)	ICNC1	ICES1	–	–	CTC1	CS12	CS11	CS10	EXTCCR1B
Read/Write	R/W	R/W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.10: External Timer/Counter1 Control Register B

- **Bit 7 – ICNC1: Input Capture1 Noise Canceler (4 - 20 MHz clocks)**

  When the ICNC1 bit is cleared (logic zero), the input capture trigger noise canceler function is disabled. The input capture is triggered at the first rising/falling edge sampled on the EXT_ICP0 input capture pin. When the ICNC1 bit is set (logic one), four successive samples are measured on the EXT_ICP0 input capture pin and all samples must be high/low according to the input capture trigger specification in the ICES1 bit. The actual sampling frequency is the 20 MHz clock frequency.

- **Bit 6 – ICES1: Input Capture1 Edge Select**

  While the ICES1 bit is cleared (logic zero), the Timer/Counter1 contents are transferred to the Input Capture Register (ICR1) on the falling edge of the input capture pin EXT_ICP0. While the ICES1 bit is set (logic one), the Timer/Counter1 contents are transferred to the Input Capture Register ICR1 on the rising edge of the input capture pin EXT_ICP0.

- **Bits 5..4 Reserved**

  These bits are reserved and always read as zero.

- **Bit 3 – CTC1: Clear Timer/Counter1 on Compare Match**

  When the CTC1 control bit is set (logic one), the Timer/Counter1 is reset to 0x0000 in the clock cycle after a Compare/A match. If the CTC1 control bit is cleared, the Timer/Counter1 continues counting until it is stopped, cleared, wraps around (overflow), or changes direction. In PWM mode, this bit has no effect.

- **Bits 2..0 – CS1[2:0]: Clock Select bits 2:0**

  These bits define the prescaling source of Timer/Counter1 according to Table 17.11. The Stop condition provides a Timer Enable/Disable function. The divided modes are scaled directly from the 20 MHz clock.

Clock1 Prescaling (20 MHz Clock)			
CS02	CS01	CS00	Description
0	0	0	Stop! The Timer/Counter1 has stopped
0	0	1	CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	External Pin t1clk, falling edge
1	1	1	External Pin t1clk, rising edge

Table 17.11: Clock1 Prescaling

EXTCNT1H and EXTCNT1L (External Timer/Counter1 Register High Byte, Low Byte)

Bit	7	6	5	4	3	2	1	0	
0x23(0x43)	TCNT1[15:8]								EXTCNT1H
0x22(0x42)	TCNT1[7:0]								EXTCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.12: External Timer/Counter1 Register (High Byte/Low Byte)

- **Bits 7..0 (EXTCNT1H) – TCNT1[15:8]: Timer/Counter 1 Bits 15:8**

  These bits represent the upper 8 bits of the Timer/Counter 1

- **Bits 7..0 (EXTCNT1L) – TCNT1[7:0]: Timer/Counter Bits 7:0**

  These bits represent the lower 8 bits of the Timer/Counter 1

To ensure that both the high and low bytes are read and written simultaneously when the CPU access these registers, the access is performed using an 8-bit temporary register (TEMP).

TCNT1 Timer/Counter1 Write: When the CPU writes to the high byte EXTCNT1H, the written data is placed in the TEMP register. Next, when the CPU writes the low byte EXTCNT1L, this byte of data is combined with the byte of data in the TEMP register and all 16 bits are written to the TCNT1 Timer/Counter1 register simultaneously. Consequently, the high byte EXTCNT1H must be written first for a full 16-bit register write operation.

TCNT1 Timer/Counter1 Read: When the CPU reads the low byte EXTCNT1L, the data of the low byte EXTCNT1L is sent to the CPU and the data of the high byte EXTCNT1H is placed in the TEMP register. When the CPU reads the data in the high byte EXTCNT1H, the CPU received the data in the TEMP register. Consequently, the low byte TCNT1L must be accessed first for a full 16-bit register read operation.

Timer/Counter1 is implemented as an up or up/down (in PWM mode) counter with read and write access. If Timer/Counter1 is written to and a clock source is selected, the Timer/Counter1 continues counting in the timer clock cycle after it is preset with the written value.

EXOCR1AH and EXOCR1AL (External Timer/Counter1 Output Compare Register A High Byte, Low Byte)

Bit	7	6	5	4	3	2	1	0	
0x25(0x45)	OCR1A[15:8]								EXOCR1AH
0x24(0x44)	OCR1A[7:0]								EXOCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.13: External Timer/Counter1 Output Compare Register A

- **its 7..0 (EXOCR1AH) – OCR1A[15:8] Output Compare Register A Bits 15:8**

  These bits represent the upper 8 bits of the Output Compare Register A

- **Bits 7..0 (EXOCR1AL) – OCR1A[7:0] Output Compare Register A Bits 7:0**

  These bits represent the lower 8 bits of the Output Compare Register A

The External Timer/Counter 1 Output Compare Register A contains data that is continuously compared with External Timer/Counter1. Actions on compare matches are specified in the External Timer/Counter1 Control and Status Register.

Since the Output Compare Register A is a 16-bit value, a temporary register TEMP is used when OCR1A is written to ensure that both bytes are updated simultaneously. When the CPU writes the high byte EXOCR1AH, the data is temporarily stored in the TEMP register. When the CPU writes the low byte EXOCR1AL, the TEMP register is simultaneously written to the EXOCR1AH. Consequently, the high byte EXOCR1AH must be written first for a full 16-bit register write operation.

EXOCR1BH and EXOCR1BL (External Timer/Counter1 Output Compare Register B High Byte, Low Byte)

Bit	7	6	5	4	3	2	1	0	
0x27(0x47)	OCR1B[15:8]								EXOCR1BH
0x26(0x46)	OCR1B[7:0]								EXOCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.14: External Timer/Counter1 Output Compare Register B

- **Bits 7..0 (EXOCR1BH) – OCR1B[15:8] Output Compare Register B Bits 15:8**

These bits represent the upper 8 bits of the Output Compare Register B.

- **Bits 7..0 (EXOCR1BL) – OCR1B[7:0] Output Compare Register B Bits 7:0**

These bits represent the lower 8 bits of the Output Compare Register B.

The External Timer/Counter 1 Output Compare Register B contains data that is continuously compared with External Timer/Counter1. Actions on compare matches are specified in the External Timer/Counter1 Control and Status Register.

Since the Output Compare Register B is a 16-bit value, a temporary register TEMP is used when OCR1B is written to ensure that both bytes are updated simultaneously. When the CPU writes the high byte EXOCR1BH, the data is temporarily stored in the TEMP register. When the CPU writes the low byte EXOCR1BL, the TEMP register is simultaneously written to the EXOCR1BH. Consequently, the high byte EXOCR1BH must be written first for a full 16-bit register write operation.

EXICR1H and EXICR1L (External Timer/Counter1 Input Capture Register High Byte, Low Byte)

Bit	7	6	5	4	3	2	1	0	
0x29(0x49)	ICR1[15:8]								EXICR1H
0x28(0x48)	ICR1[7:0]								EXICR1L
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.15: External Timer/Counter1 Input Capture Registers

- **Bits 7..0 (EXICR1H)– ICR1[15:8]: Input Capture Register Bits 15:8**

These bits represent the upper 8 bits of the Input Capture Register

- **Bits 7..0 (EXICR1L)– ICR1[7:0]: Input Capture Register Bits 7:0**

These bits represent the lower 8 bits of the Input Capture Register

When the rising or falling edge (according to the input capture edge setting ICES1) of the signal on the input capture pin (EXT_ICP0) is detected, the current value of the External Timer/Counter1 is transferred to the Input Capture Register (ICR1[15:0]). At the same time, the input capture flag (ICF) is set (logic one). Since the Input Capture Register is a 16-bit value, a temporary register TEMP is used when ICR1 is read to ensure that both bytes are read simultaneously. When the CPU reads the low byte EXICRL1, ICR1[7:0] is sent to the CPU and the high byte ICR1[15:8] is stored in the TEMP register. When the CPU reads the high byte EXICR1H, the value stored in the TEMP register is sent to the CPU. Consequently, the low byte EXICR1L must be accessed first for a full 16-bit register read operation.

Internal Timer/Counter Registers

INTCCR0 (Internal Timer/ Counter0 Control Register)

Bit	7	6	5	4	3	2	1	0	
0x809	–	–	–	–	–	CS02	CS01	CS00	INTCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.16: Internal Timer/Counter0 Control Register

- **Bits 7..3 Reserved**

These bits are reserved and always read as zero

- **Bits 2..0 – CS0[2:0]: Clock Select0 bits 2:0**

The Clock Select0 bits 2:0 define the prescaling source of Timer0 according to Table 17.17. The Stop condition provides a Timer Enable/Disable function. The divided modes are scaled directly from the 20 MHz clock.

Timer/Counter0 Prescaling (20 MHz Clock)			
CS02	CS01	CS00	Description
0	0	0	Stop! The Timer/Counter0 has stopped
0	0	1	CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	External Pin t01clk, falling edge
1	1	1	External Pin t0clk, rising edge

Table 17.17: Timer/Counter0 Prescaling

INTCNT0 (Internal Timer/Counter0)

Bit	7	6	5	4	3	2	1	0	
0x808	MSB							LSB	INTCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.18: Internal Timer/Counter0 Register

- **Bits 7..0 – TCNT0[7:0]: Timer/Counter0 bits 7:0**

Timer/Counter0 is implemented as an up-counter with read and write access. If the Timer/ Counter0 register is written and a clock source is present, the Timer/Counter0 continues counting in the clock cycle following the write operation.

INTCCR1A (Internal Timer/ Counter1 Control Register A)

Bit	7	6	5	4	3	2	1	0	
0x800	COM1A1	COM1A0	COM1B1	COM1B0	–	–	PWM11	PWM10	INTCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.19: Internal Timer/Counter1 Control Register A

- **Bits 7..6 – COM1A[1:0]: Compare Output Mode1 A [1:0]**

These bits have no function as the OCA pin on the Internal Timer/Counter1 is disabled

COM1 Timer Table		
COM1X1	COM1X0	Description
0	0	Timer Counter1 disconnected from output pin OCX
0	1	Toggle the OCX output line
1	0	Clear the OCX output line (to zero)
1	1	Set the OCX output line (to one)

Note: OCX=OCA or OCB

Table 17.20: Compare1 Output Description

- Bits 5..4 – COM1B[1:0]: Compare Output Mode1 B [1:0]

These bits have no function as the OCA pin on the Internal Timer/Counter1 is disabled

- **In PWM mode, these bits have a different function**

When changing the COM1X1/COM1X0 bits, Output Compare Interrupts 1 must be disabled by clearing their Interrupt Enable bits the corresponding external IRQ register. Otherwise an interrupt can occur when the bits are changed.

- **Bits 3..2 Reserved**

These bits are reserved and always read as zero

- **Bits 1..0 – PWM1[1:0]: Pulse Width Modulator Select bits [1:0]**

These bits select PWM operation of the Timer/Counter1 as specified in Table 17.21

PWM Mode Select		
PWM11	PWM10	Description
0	0	PWM operation of the Timer/Counter1 is disabled
0	1	Timer/Counter1 is an 8-bit PWM
1	0	Timer/Counter1 is an 9-bit PWM
1	1	Timer/Counter1 is an 10-bit PWM

Table 17.21: PWM Mode Select

INTCCR1B (Internal Timer/ Counter1 Control Register B)

Bit	7	6	5	4	3	2	1	0	
0x801	ICNC1	ICES1	–	–	CTC1	CS12	CS11	CS10	INTCCR1B
Read/Write	R/W	R/W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.22: Internal Timer/Counter1 Control Register B

- **Bit 7 – ICNC1: Input Capture1 Noise Canceller (4 - 20 MHz clocks)**

  This bit has no function as input capture is disabled in the Internal Timer/Counter1

- **Bit 6 – ICES1: Input Capture1 Edge Select**

  This bit has no function as input capture is disabled in the Internal Timer/Counter1

- **Bits 5..4 Reserved**

  These bits are reserved and always read as zero

- **Bit 3 – CTC1: Clear Timer/Counter1 on Compare Match**

  When the CTC1 control bit is set (logic one), the Timer/Counter1 is reset to 0x0000 in the clock cycle after a compareA match. If the CTC1 control bit is cleared, the Timer/Counter1 continues counting until it is stopped, cleared, wraps around (overflow), or changes direction. In PWM mode, this bit has no effect.

- **Bits 2..0 – CS1[2:0]: Clock Select bits 2:0**

  These bits define the prescaling source of Timer/Counter1 according to Table 17.23 The Stop condition provides a Timer Enable/Disable function. The divided modes are scaled directly from the 20 MHz clock.

Timer/Counter1 Prescaling (20 MHz Clock)			
CS12	CS11	CS10	Description
0	0	0	Stop! The Timer/Counter1 has stopped
0	0	1	CLK
0	1	0	CLK/8
0	1	1	CLK/64
1	0	0	CLK/256
1	0	1	CLK/1024
1	1	0	External Pin t11clk, falling edge
1	1	1	External Pin t1clk, rising edge

Table 17.23: Timer/Counter1 Prescaling

INTCNT1H and INTCNT1L (Internal Timer/Counter1 Register High Byte, Low Byte)

Bit	7	6	5	4	3	2	1	0	
0x803	TCNT1[15:8]								INTCNT1H
0x802	TCNT1[7:0]								INTCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.24: Internal Timer/Counter1 Register (High Byte - Low Byte)

- **Bits 7..0 (INTCNT1H) – TCNT1[15:8]: Timer/Counter 1 Bits 15:8**

These bits represent the upper 8 bits of the Timer/Counter 1

- **Bits 7..0 (INTCNT1L) – TCNT1[7:0]: Timer/Counter Bits 7:0**

These bits represent the lower 8 bits of the Timer/Counter 1

To ensure that both the high and low bytes are read and written simultaneously when the CPU access these registers, the access is performed using an 8-bit temporary register TEMP.

TCNT1 Timer/Counter1 Write: When the CPU writes to the high byte INTCNT1H, the written data is placed in the TEMP register. Next, when the CPU writes the low byte INTCNT1L, this byte of data is combined with the byte of data in the TEMP register and all 16 bits are written to the TCNT1 Timer/Counter1 register simultaneously. Consequently, the high byte INTCNT1H must be written first for a full 16-bit register write operation.

TCNT1 Timer/Counter1 Read: When the CPU reads the low byte INTCNT1L, the data of the low byte INTCNT1L is sent to the CPU and the data of the high byte INTCNT1H is placed in the TEMP register. When the CPU reads the data in the high byte INTCNT1H, the CPU received the data in the TEMP register. Consequently, the low byte TCNT1L must be accessed first for a full 16-bit register read operation.

Timer/Counter1 is implemented as an up or up/down (in PWM mode) counter with read and write access. If Timer/Counter1 is written to and a clock source is selected, the Timer/Counter1 continues counting in the timer clock cycle after it is preset with the written value.

INOCR1AH and INOCR1AL (Internal Timer/Counter Compare Register A High Byte, Low Byte)

Bit	7	6	5	4	3	2	1	0	
0x805	OCR1A[15:8]								INOCR1AH
0x804	OCR1A[7:0]								INOCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.25: Internal Timer/Counter Compare Register A (High Byte/Low Byte)

- **Bits 7..0 (INOCR1AH) – OCR1A[15:8]:**

Output Compare Register A Bits 15:8. These bits represent the upper 8 bits of the Output Compare Register A.

- **Bits 7..0 (INOCR1AL) – OCR1A[7:0]:**

Output Compare Register A Bits 7:0. These bits represent the lower 8 bits of the Output Compare Register A.

The Internal Timer/Counter1 Output Compare Register A contains data that is continuously compared with Internal Timer/Counter1. Actions on compare matches are specified in the Internal Timer/Counter1 Control and Status Register.

Since the Output Compare Register A is a 16-bit value, a temporary register TEMP is used when OCR1A is written to ensure that both bytes are updated simultaneously. When the CPU writes the high byte INOCR1AH, the data is temporarily stored in the TEMP register. When the CPU writes the low byte INOCR1AL, the TEMP register is simultaneously written to the INOCR1AH. Consequently, the high byte INOCR1AH must be written first for a full 16-bit register write operation.

INOCR1BH and INOCR1BL (Internal Timer/Counter1 Output Compare Register B High Byte, Low Byte

Bit	7	6	5	4	3	2	1	0	
0x807	OCR1B[15:8]								INOCR1BH
0x806	OCR1B[7:0]								INOCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 17.26: Internal Timer/Counter1 Output Compare Register B

- **Bits 7..0 (INOCR1BH) – OCR1B[15:8]:**

  Output Compare Register B Bits 15:8. These bits represent the upper 8 bits of the Output Compare Register B.

- **Bits 7..0 (INOCR1BL) – OCR1B[7:0]:**

  Output Compare Register B Bits 7:0. These bits represent the lower 8 bits of the Output Compare Register B.

The Internal Timer/Counter1 Output Compare Register/B contains data that is continuously compared with Internal Timer/Counter1. Actions on compare matches are specified in the Internal Timer/Counter1 Control and Status Register.

Since the Output Compare Register B is a 16-bit value, a temporary register TEMP is used when OCR1B is written to ensure that both bytes are updated simultaneously. When the CPU writes the high byte INOCR1BH, the data is temporarily stored in the TEMP register. When the CPU writes the low byte INOCR1BL, the TEMP register is simultaneously written to the INOCR1BH. Consequently, the high byte INOCR1BH must be written first for a full 16-bit register write operation.

SECTION 18
WATCHDOG TIMER

Introduction

The Watchdog Timer generates a time-out signal if it has not been reset after a certain number of clock cycles. This can be used to exit from endless loops.

The watchdog timer is a fully synchronous peripheral.

Features

- 28-bit Prescaler
- Programmable Time-out Period
- Fully Synchronous
- Protected Turnoff Sequence

Functional Description

The watchdog timer is clocked by the 20 MHz system clock. By controlling the watchdog timer prescaler, the watchdog time-out interval can be adjusted from 800µs to 13.1 sec. The WDR - Watchdog Reset - instruction resets the watchdog timer. Fifteen different multiples of the clock cycle period can be selected to determine the time-out period. If the time-out period expires without another watchdog reset, the AVR core, motor and motion logic, peripherals, and CAN logic will be reset. See Figure 18.2

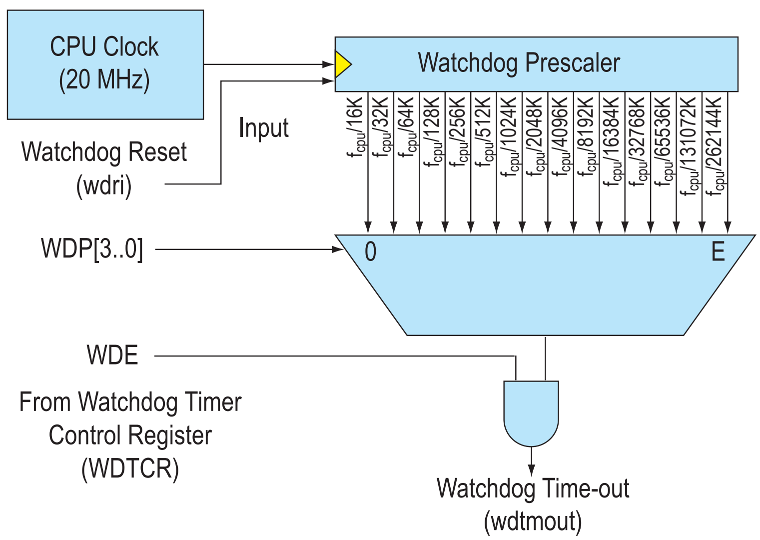


Figure 18.1: Watchdog Timer Prescaler

Watchdog Timer Periods				
WDP[3..0]				Timeout Period
3	2	1	0	
0	0	0	0	800 µs
0	0	0	1	1.6 ms
0	0	1	0	3.2 ms
0	0	1	1	6.4 ms
0	1	0	0	12.8 ms
0	1	0	1	25.6 ms
0	1	1	0	51.2 ms
0	1	1	1	102.4 ms
1	0	0	0	204.8 ms
1	0	0	1	409.6 ms
1	0	1	0	819.2 ms
1	0	1	1	1.6 s
1	1	0	0	3.3 s
1	1	0	1	6.6 s
1	1	1	0	13.1 s
1	1	1	1	Reserved

Table 18.1 Watchdog Timer Periods

Code Examples

The following code example shows one assembly and one C function for turning off the WDT. The example assumes that interrupts are controlled (for example by disabling interrupts globally) so that no interrupts will occur during execution of these functions.

C Code Example

```
void   WDT_off (void)
{
    /* Write logical one to WDTOE and WDE */
    WDTCR = (1<<WDTOE)   |   (1<<WDE)
    /* Turn off WDT */
    WDTCR = 0x00;
}
```

WDR Reset Instructions

***WD Timer Restart Operation*:** This instuction resets the Watchdog Timer. This instruction must be executed within a limited time given by the WD prescaler. See Program Instruction Set, WDR.

WDTCR (Watchdog Timer Control Register)

Bit	7	6	5	4	3	2	1	0	
0x0F(0x2F)	WDIE	WDIRQ	WDTOE	WDE	WDP[3..0]				WDTCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 18.2: Watchdog Timer Register

- **Bit 7 – WDIE: Watchdog Interrupt Enable**

This bit forces the Watchdog to generate an interrupt instead of a Watchdog Time-out Reset.

- **Bit 6 – WDIRQ: Watchdog Interrupt Flag**

This IRQ Flag is set if WDIE is high and a Watchdog Time-out occurs. It is reset by ireset, wdt_irqack, or by a one written in the WDIRQ bit.

- **Bit 5 – WDTOE: Watchdog Turnoff Enable**

This bit must be set (Logic One) when the WDE bit is cleared. Otherwise, the Watchdog will not be disabled. Once set, hardware will clear this bit after four clock cycles. Refer to the description of the WDE bit for a Watchdog disable procedure.

- **Bit 4 – WDE: Watchdog Enable**

When the WDE is set (logic one), the Watchdog Timer is enabled and if the WDE is cleared (logic zero), the Watchdog Timer function is disabled. WDE can only be cleared if the WDTOE bit is set to logic level one. To disable an enabled Watchdog Timer, perform the following:

1. In the same operation, write a logic one to WDTOE and WDE. A logic one must be written to WDE even though it is already set to one before the disable starts.
2. Within the next four clock cycles, write a logic zero to WDE. This disables the Watchdog.

- **Bits 3:0 – WD[3:0]: Watchdog Timer Prescaler**

The WDP[3:0] bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Time-out Periods are shown in the preceding table.

Page Intentionally Left Blank

SECTION 19
UART
Introduction

The Universal Asynchronous serial Receiver and Transmitter (UART) is a highly flexible serial communication device.

Features

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous Operation
- Master or Slave Clocked Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8 or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data Over Run Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

Function of the UART

A simplified block diagram of the UART transmitter is shown below.

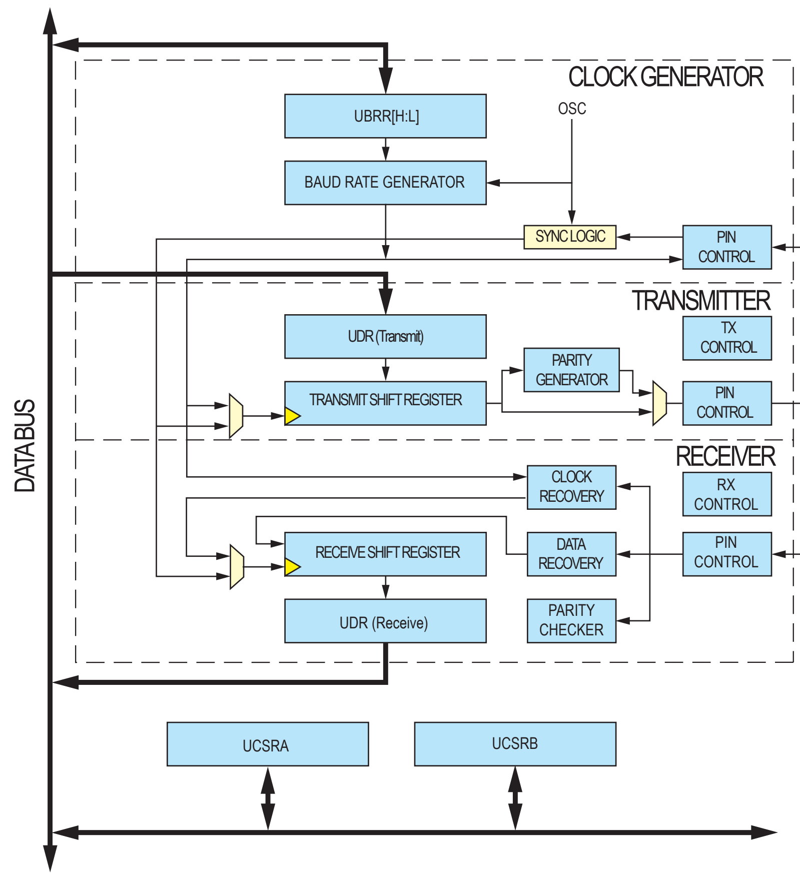


Figure 19.1: UART Tramsmitter Block Diagram

The dashed boxes in the block diagram separate the three main parts of the UART (listed from the top):

- Clock Generator
- Transmitter
- Receiver

Control registers are shared by all units. The clock generation logic consists of synchronization logic for external clock input used by synchronous slave operation and the baud rate generator. The XCK (Transfer Clock) pin is only used by Synchronous Transfer mode. The Transmitter consists of a single write buffer, a serial shift register, parity generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the UART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the receiver includes a parity checker, control logic, a Shift Register and a two level receive buffer (UDR). The receiver supports the same frame formats as the transmitter and can detect frame error, data overrun and parity errors.

Clock Generation

The clock generation logic generates the base clock for the Transmitter and Receiver. The UART supports normal Asynchronous mode of operation.

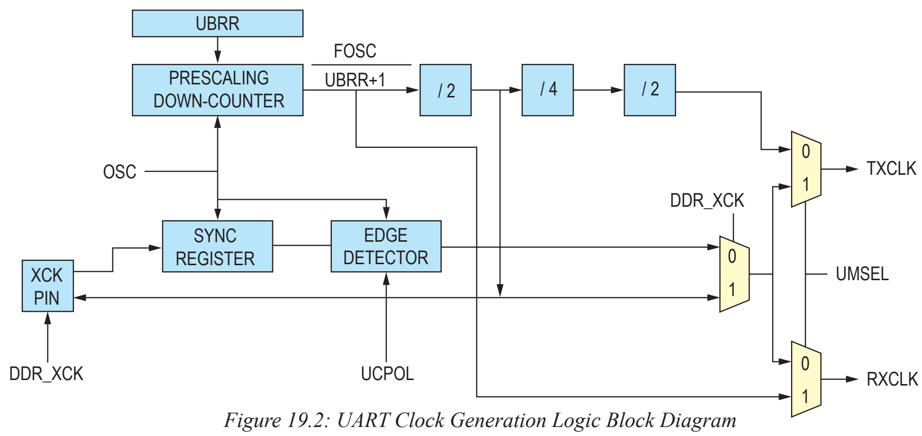


Figure 19.2: UART Clock Generation Logic Block Diagram

- **Signal description:**
  - » ***txclk:*** Transmitter clock (Internal Signal).
  - » ***rxclk:*** Receiver base clock (Internal Signal).
  - » ***xcki:*** Input from XCK pin (Internal Signal). Used for synchronous slave operation.
  - » ***xcko:*** Clock output to XCK pin (Internal Signal). Used for synchronous master operation.
  - » ***fosc:*** XTAL pin frequency (System Clock).

Frame Formats

A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The UART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8 or 9 data bits
- parity; none, even or odd
- 1 or 2 stop bits

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. The figure below illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

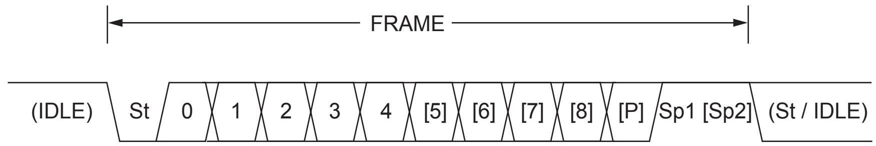


Figure 19.3: Combinations of UART Frame Formats

- St — Start bit, always low.
- (n) — Data bits (0 to 8).
- P — Parity bit. Can be odd or even.
- Sp — Stop bit, always high.
- IDLE — No transfers on the communication line (RxD or TxD). An IDLE line must be high.

The frame format used by the UART is set by the UCSZ2:0, UPM1:0], and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The UART Character Size (UCSZ2:0) bits select the number of data bits in the frame. The UART Parity Mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the UART Stop Bit Select (USBS) bit. The receiver ignores the second stop bit. An FE (Frame Error) will therefore only be detected in the cases where the first stop bit is zero.

Parity Bit Calculation

The parity bit is calculated by doing an exclusive-OR of all the data bits. If odd parity is used, the result of the exclusive-OR is inverted. The relation between the parity bit and data bits is as follows:

$$P_{EVEN} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_1 \oplus d_1 \oplus d_0 \oplus 0$$

$$P_{ODD} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_1 \oplus d_1 \oplus d_0 \oplus 1$$

- PEVEN — Parity bit using even parity.
- PODD — Parity bit using odd parity.
- dn — Data bit n of the character.

If used, the parity bit is located between the last data bit and first stop bit of a serial frame.

UART Initialization

The UART has to be initialized before any communication can take place. The initialization process normally consists of setting the baud rate, setting frame format and enabling the Transmitter or the Receiver depending on the usage. For interrupt driven UART operation, the global interrupt flag should be cleared (and interrupts globally disabled) when doing the initialization.

Before doing a re-initialization with changed baud rate or frame format, be sure that there are no ongoing transmissions during the period the registers are changed. The TXC flag can be used to check that the Transmitter has completed all transfers and the RXC flag can be used to check that there are no unread data in the receive buffer.

Note: The TXC flag must be cleared before each transmission (before UDR is written) if it is used for this purpose.

The following simple UART initialization code examples show one assembly and one C function that are equal in functionality. The examples assume asynchronous operation using polling (no interrupts enabled) and a fixed frame format. The baud rate is given as a function parameter. For the assembly code, the baud rate parameter is assumed to be stored in the r17:r16 registers. When the function writes to the UCSRC Register, the URSEL bit (MSB) must be set due to the sharing of I/O location by UBRRH and UCSRC.

Code Example - UART Initialization

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
void UART_Init( unsigned int baud )
{
    /* Set baud rate */
       UBRRH = (unsigned char)(baud>>8);
       UBRRL = (unsigned char)baud;
    /* Enable receiver and transmitter */
       UCSRB = (1<<RXEN)|(1<<TXEN);
    /* Set frame format: 8data, 2stop bit */
       UCSRC = (1<<URSEL) | (1<<USBS) | (3<<UCSZ0);
}
```

More advanced initialization routines can be made that include frame format as parameters, disable interrupts and so on. However, many applications use a fixed setting of the baud and control registers and for these types of applications the initialization code may be placed directly in the main routine or combined with initialization code for other I/O modules.

UART Data Transmitter

Introduction

The UART Transmitter is enabled by setting the Transmit Enable (TXEN) bit in the UCSRB Register. When the Transmitter is enabled, the normal port operation of the TxD pin is overridden by the UART and given the function as the transmitter's serial output. Before doing any transmissions, the baud rate, mode of operation and frame format must be set up. If synchronous operation is used, the clock on the XCK pin will be overridden and used as transmission clock.

Sending Frames with 5 to 8 Data Bit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the UDR I/O location. The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the baud register, U2X bit or XCK depending on mode of operation.

Code Example - UART Transmit Function

The following code examples show a simple UART transmit function based on polling of the Data Register Empty (UDRE) flag. When using frames with less than eight bits, the most significant bits written to the UDR are ignored. The UART has to be initialized before the function can be used. For the assembly code, the data to be sent is assumed to be stored in Register R16.

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
void UART_Transmit( unsigned char data )
{
    /* Wait for empty transmit buffer */
       while ( !( UCSRA & (1<<UDRE)) );
    /* Put data into buffer, sends the data */
       UDR = data;
}
```

The function simply waits for the transmit buffer to be empty by checking the UDRE flag, before loading it with new data to be transmitted. If the data register empty interrupt is utilized, the interrupt routine writes the data into the buffer.

Code Example - Sending Frames with 9 Data Bits

If 9-bit characters are used (UCSZ = 7), the ninth bit must be written to the TXB8 bit in UCSRB before the low byte of the character is written to UDR. The following code examples show a transmit function that handles 9-bit characters. For the assembly code, the data to be sent is assumed to be stored in Registers R17:R16.

NOTE

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
void UART_Transmit( unsigned int data )
    {
/* Wait for empty transmit buffer */
       while ( !( UCSRA & (1<<UDRE))) )
       ;
    /* Copy 9th bit to TXB8 */
       UCSRB &= ~(1<<TXB8);
       if ( data & 0x0100 )
       UCSRB |= (1<<TXB8);
    /* Put data into buffer, sends the data */
       UDR = data;
}
```

The ninth bit can be used for indicating an address frame when using multi processor communication mode or for other protocol handling as, for example, synchronization.

Transmitter Flags and Interrupts

The UART transmitter has two flags that indicate its state: UART Data Register Empty (UDRE) and Transmit Complete (TXC). Both flags can be used for generating interrupts.

NOTE

NOTE: These transmit functions are written to be general functions. They can be optimized if the contents of the UCSRB are static (i.e., only the TXB8 bit of the UCSRB Register is used after initialization)

The Data Register Empty (UDRE) flag indicates whether the transmit buffer is ready to receive new data. This bit is set when the transmit buffer is empty, and cleared when the transmit buffer contains data to be transmitted that has not yet been moved into the Shift Register. For compatibility with future devices, always write this bit to zero when writing the UCSRA register.

When the Data Register empty Interrupt Enable (UDRIE) bit in UCSRB is written to one, the UART Data Register Empty Interrupt will be executed as long as UDRE is set (provided that global interrupts are enabled). UDRE is cleared by writing UDR. When interrupt-driven data transmission is used, the data register empty Interrupt routine must either write new data to UDR in order to clear UDRE or disable the Data Register empty Interrupt, otherwise a new interrupt will occur once the interrupt routine terminates.

The Transmit Complete (TXC) flag bit is set once when the entire frame in the transmit Shift Register has been shifted out and there is no new data currently present in the transmit buffer. The TXC flag is useful in half-duplex communication interfaces (like the RS485 standard), where a transmitting application must enter receive mode and free the communication bus immediately after completing the transmission.

When the Transmit Complete Interrupt Enable (TXCIE) bit in UCSRB is set, the UART Transmit Complete Interrupt will be executed when the TXC flag becomes set (provided that global interrupts are enabled). Note: Flags are only active when the corresponding mask is set. Interrupts must be enabled to see the flag The interrupt handling routine must clear the TXC flag.

Parity Generator

The parity generator calculates the parity bit for the serial frame data. When parity bit is enabled (UPM1 = 1), the transmitter control logic inserts the parity bit between the last data bit and the first stop bit of the frame that is sent.

Disabling the Transmitter

The disabling of the transmitter (setting the TXEN to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD pin.

UART Data Receiver

Introduction

The UART Receiver is enabled by writing the Receive Enable (RXEN) bit in the

UCSRB Register to one. When the receiver is enabled, the normal pin operation of the RxD pin is overridden by the UART and given the function as the receiver's serial input. Before any serial reception can be done, the baud rate, mode of operation and frame format must be set up. If synchronous operation is used, the clock on the XCK pin will be used as transfer clock.

Receiving Frames with 5 to 8 Data Bits

The receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCK clock, and shifted into the receive Shift Register until the first stop bit of a frame is received. A second stop bit will be ignored by the receiver. When the first stop bit is received, i.e., a complete serial frame is preset in the receive Shift Register, the contents of the Shift Register will be moved into the receive buffer. The receive buffer can then be read by reading the UDR I/O location.

Code Example - UART Receive Function

The following code example shows a simple UART receive function based on polling of the Receive Complete (RXC) flag. When using frames with less than eight bits, the most significant bits of the data read from the UDR will be masked to zero. The UART has to be initialized before the function can be used.

NOTE

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
unsigned char UART_Receive( void )
{
    /* Wait for data to be received */
        while ( !(UCSRA & (1<<RXC)) )
        ;
    /* Get and return received data from buffer */
        return UDR;
}
```

The function simply waits for data to be preset in the receive buffer by checking the RXC flag, before reading the buffer and returning the value.

Receiving Frames with 9 Databits

If 9-bit characters are used (UCSZ=7) the ninth bit must be read from the RXB8 bit in UCSRB before reading the low bits from the UDR. This rule applies to the FE, DOR and PE status flags as well. Read status from UCSRA, then data from UDR. Reading the UDR I/O location will change the state of the receive buffer FIFO and consequently the TXB8, FE, DOR and PE bits, which all are stored in the FIFO, will change.

Code Examples - Receiving Frames with 9 Data Bits

NOTE

NOTE: The example code assumes that the part specific header file is included.

The following code example shows a simple UART receive function that handles both 9-bit characters and the status bits.

C Code Example

```
unsigned int UART_Receive( void )
{
        unsigned char status, resh, resl;
    /* Wait for data to be received */
        while ( !(UCSRA & (1<<RXC)) );
    /* Get status and 9th bit, then data */
    /* from buffer */
        status = UCSRA;
        resh = UCSRB;
        resl = UDR;
    /* If error, return -1 */
        if ( status & (1<<FE)|(1<<DOR)|(1<<PE) )
        return -1;
    /* Filter the 9th bit, then return */
        resh = (resh >> 1) & 0x01;
        return ((resh << 8) | resl);
}
```

The receive function example reads all the I/O registers into the register file before any computation is done. This gives an optimal receive buffer utilization since the buffer location read will be free to accept new data as early as possible.

Receive Compete Flag and Interrupt

The UART Receiver has one flag that indicates the receiver state.

The Receive Complete (RXC) flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled (RXEN = 0), the receive buffer will be flushed and consequently the RXC bit will become zero.

Receiver Error Flags

When the Receive Complete Interrupt Enable (RXCIE) in UCSRB is set, the UART Receive Complete Interrupt will be executed as long as the RXC flag is set (provided that global interrupts are enabled). When interrupt-driven data reception is used, the receive complete routine must read the received data from UDR in order to clear the RXC flag, otherwise a new interrupt will occur once the interrupt routine terminates.

The UART Receiver has three error flags: Frame Error (FE), Data OverRun (DOR) and Parity Error (PE). All can be accessed by reading UCSRA. Common for the error flags is that they are located in the receive buffer together with the frame for which they indicate the error status. Due to the buffering of the error flags, the UCSRA must be read before the receive buffer (UDR), since reading the UDR I/O location changes the buffer read location. Another equality for the error flags is that they cannot be altered by software doing a write to the flag location. However, all flags must be set to zero when the UCSRA is written for upward compatibility of future UART implementations. None of the error flags can generate interrupts.

The Frame Error (FE) flag indicates the state of the first stop bit of the next readable frame stored in the receive buffer. The FE flag is zero when the stop bit was correctly read (as one), and the FE flag will be one when the stop bit was incorrect (zero). This flag can be used for detecting out-of-sync conditions, detecting break conditions and protocol handling. The FE flag is not affected by the setting of the USBS bit in UCSRC since, except for the first, the receiver ignores all stop bits. For compatibility with future devices, always set this bit to zero when writing to UCSRA.

The Data OverRun (DOR) flag indicates data loss due to a receiver buffer full A Data OverRun occurs when the receive buffer is full (two characters), it is a new character waiting in the receive Shift Register, and a new start bit is detected. If the DOR flag is set there was one or more serial frame lost between the frame last read from UDR and the next frame read from UDR. For compatibility with future devices, always write this bit to zero when writing to UCSRA. The DOR flag is cleared when the frame received was successfully moved from the Shift Register to the receive buffer.

The Parity Error (PE) flag indicates that the next frame in the receive buffer had a parity error when received. If parity check is not enabled the PE bit will always be read zero. For compatibility with future devices, always set this bit to zero when writing to UCSRA.

Parity Checker

The Parity Checker is active when the high UART Parity mode (UPM1) bit is set. Type of parity check to be performed (odd or even) is selected by the UPM0 bit. When enabled, the parity checker calculates the parity of the data bits in incoming frames and compares the result with the parity bit from the serial frame. The result of the check is stored in the receive buffer together with the received data and stop bits. The Parity Error (PE) flag can then be read by software to check if the frame had a parity error.

The PE bit is set if the next character that can be read from the receive buffer had a parity error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read.

Disabling the Receiver

In contrast to the Transmitter, disabling of the Receiver will be immediate. Data from ongoing receptions will therefore be lost. When disabled (i.e., the RXEN is set to zero) the Receiver will no longer override the normal function of the RxD port pin. The receiver buffer FIFO will be flushed when the receiver is disabled. Remaining data in the buffer will be lost.

Flushing the Receive Buffer

The receiver buffer FIFO will be flushed when the Receiver is disabled, i.e., the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation due to, for instance, an error condition, read the UDR I/O location until the RXC flag is cleared. The following code example shows how to flush the receive buffer.

Code Example - Flushing the Receive Buffer

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
void UART_Flush( void )
{
    unsigned char dummy;
    while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```

Asynchronous Data Reception

The UART includes a clock recovery and a data recovery unit for handling asynchronous data reception. The clock recovery logic is used for synchronizing the internally generated baud rate clock to the incoming asynchronous serial frames at the RxD pin. The data recovery logic samples and low pass filters each incoming bit, thereby improving the noise immunity of the receiver. The asynchronous reception operational range depends on the accuracy of the internal baud rate clock, the rate of the incoming frames, and the frame size in number of bits.

Asynchronous Clock Recovery

The clock recovery logic synchronizes internal clock to the incoming serial frames. The following figure illustrates the sampling process of the start bit of an incoming frame. The sample rate is 16 times the baud rate for Normal mode, and 8 times the baud rate for Double Speed mode. The horizontal arrows illustrate the synchronization variation due to the sampling process. Samples denoted zero are samples done when the RxD line is idle (i.e., no communication activity).

Start Bit Sampling

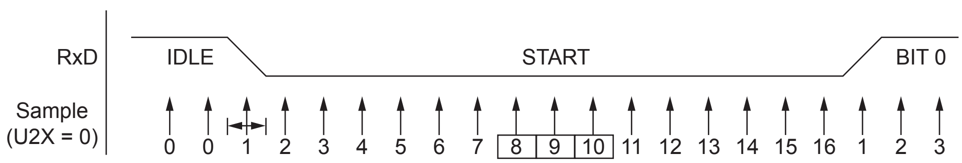


Figure 19.4: UART Start Bit Sampling

When the clock recovery logic detects a high (idle) to low (start) transition on the RxD line, the start bit detection sequence initiated. Let sample 1 denote the first zero-sample as shown in the figure above. The clock recovery logic then uses samples 8, 9 and 10 for Normal mode, and samples 4, 5 and 6 for Double Speed mode (indicated with sample numbers inside boxes on the figure), to decide if a valid start bit is received. If two or more of these three samples have logical high levels (the majority wins), the start bit is rejected as a noise spike and the receiver starts looking for the next high to low-transition. If however, a valid start bit is detected, the clock recovery logic is synchronized and the data recovery can begin. The synchronization process is repeated for each start bit.

Asynchronous Data Recovery

When the receiver clock is synchronized to the start bit, the data recovery can begin. The data recovery unit uses a state machine that has 16 states for each bit in normal mode and 8 states for each bit in Double Speed mode. The figure below shows the sampling of the data bits and the parity bit. Each of the samples is given a number that is equal to the state of the recovery unit.

Sampling of Data and Parity Bit

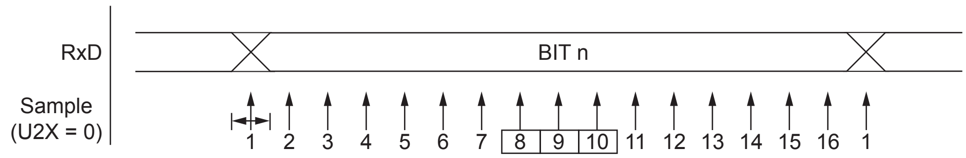


Figure 19.5: UART Data and Parity Bit Sampling

The decision of the logic level of the received bit is taken by doing a majority voting of the logic value to the three samples in the center of the received bit. The center samples are emphasized on the figure by having the sample number inside boxes. The majority voting process is done as follows: If two or all three samples have high levels, the received bit is registered to be a logic 1. If two or all three samples have low levels, the received bit is registered to be a logic 0. This majority voting process acts as a low pass filter for the incoming signal on the RxD pin. The recovery process is then repeated until a complete frame is received. Including the first stop bit. Note that the receiver only uses the first stop bit of a frame.

Stop Bit Sampling and Next Start Bit Sampling

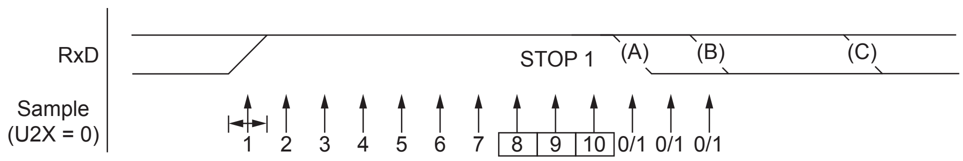


Figure 19.6: UART Stop Bit Sampling

The same majority voting is done to the stop bit as done for the other bits in the frame. If the stop bit is registered to have a logic 0 value, the Frame Error (FE) flag will be set. A new high to low transition indicating the start bit of a new frame can come right after the last of the bits used for majority voting. For Normal Speed mode, the first low level sample can be at point marked (A) in the figure above. For Double Speed mode the first low level must be delayed to (B). (C) marks a stop bit of full length. The early start bit detection influences the operational range of the receiver.

Asynchronous Operational Range

The operational range of the receiver is dependent on the mismatch between the received bit rate and the internally generated baud rate. If the Transmitter is sending frames at too fast or too slow bit rates, or the internally generated baud rate of the receiver does not have a similar base frequency, the receiver will not be able to synchronize the frames to the start bit.

The following equations can be used to calculate the ratio of the incoming data rate and internal receiver baud rate.

$$R_{slow} = \frac{(D+1)S}{S - 1 + D \cdot S + S_F}$$

$$R_{fast} = \frac{(D+2)S}{(D+1)S + S_M}$$

D Sum of character size and parity size (D = 5 to 10 bit).

S Samples per bit. S = 16 for Normal Speed mode.

S_F First sample number used for majority voting. S_F = 8 for Normal Speed mode.

S_M Middle sample number used for majority voting. S_M = 9 for Normal Speed mode.

R_{slow} The ratio of the slowest incoming data rate that can be accepted in relation to the receiver baud rate.

R_{fast} Is the ratio of the fastest incoming data rate that can be accepted in relation to the receiver baud rate.

Maximum Receiver Baud Rate Error

Tables 19.1 and 19.2 below list the maximum receiver baud rate error that can be tolerated. Note that Normal Speed mode has higher tolerance of baud rate variations.

Recommended Max. Receiver BAUD Rate Error (Normal Mode)				
D# (Data + Parity Bit)	Rslow (%)	Rfast (%)	Max. Total Error (%)	Recommended Max. Receiver Error (%)
5	93.20	106.67	+6.67/-6.8	±3.0
6	94.12	105.79	+5.79/-5.88	±2.5
7	94.81	105.11	+5.11/-5.19	±2.0
8	95.36	104.58	+4.58/-4.54	±2.0
9	95.81	104.14	+4.14/-4.19	±1.5
10	96.17	103.78	+3.78/-3.83	±1.5

Table 19.1: Recommended Max Receiver Baud Rate Error (Normal Mode)

The recommendations of the maximum receiver baud rate error were made under the assumption that the receiver and transmitter equally divide the maximum total error.

There are two possible sources for the receivers baud rate error. The receiver's system clock (XTAL) will always have some minor instability over the supply voltage range and the temperature range. When using a crystal to generate the system clock, this is rarely a problem, but for a resonator the system clock may differ more than 2% depending of the resonators tolerance. The second source for the error is more controllable. The baud rate generator can not always do an exact division of the system frequency to get the baud rate wanted. In this case an UBRR value that gives an acceptable low error can be used if possible.

Multiprocessor Communication Mode

Setting the Multiprocessor Communication Mode (MPCM) bit in UCSRA enables a filtering function of incoming frames received by the UART Receiver. Frames that do not contain address information will be ignored and not put into the receive buffer. This effectively reduces the number of incoming frames that has to be handled by the CPU, in a system with multiple MCUs that communicate via the same serial bus. The Transmitter is unaffected by the MPCM setting, but has to be used differently when it is a part of a system utilizing the Multiprocessor Communication mode.

If the receiver is set up to receive frames that contain 5 to 8 data bits, then the first stop bit indicates if the frame contains data or address information. If the receiver is set up for frames with nine data bits, then the ninth bit (RXB8) is used for identifying address and data frames. When the frame type bit (the first stop or the ninth bit) is one, the frame contains an address. When the frame type bit is zero the frame is a data frame.

The Multiprocessor Communication mode enables several slave MCUs to receive data from a master MCU. This is done by first decoding an address frame to find out which MCU has been addressed. If a particular Slave MCU has been addressed, it will receive the following data frames as normal, while the other slave MCUs will ignore the received frames until another address frame is received.

For an MCU to act as a master MCU, it can use a 9-bit character frame format (UCSZ = 7). The ninth bit (TXB8) must be set when an address frame (TXB8 = 1), or cleared when a data frame (TXB = 0), is being transmitted. The slave MCUs must in this case, be set to use a 9-bit character frame format.

Using MPCM

The following procedure should be used to exchange data in Multiprocessor Communication mode:

1. All slave MCUs are in Multiprocessor Communication mode (MPCM in UCSRA is set).
2. The Master MCU sends an address frame and all slaves receive and read this frame. In the Slave MCUs, the RXC flag in UCSRA will be set as normal.
3. Each Slave MCU reads the UDR Register and determines if it has been selected. If so, it clears the MPCM bit in UCSRA, otherwise it waits for the next address byte and keeps the MPCM setting.
4. The addressed MCU will receive all data frames until a new address frame is received. The other slave MCUs, which still have the MPCM bit set, will ignore the data frames.
5. When the last data frame is received by the addressed MCU, the addressed MCU sets the MPCM bit and waits for a new address frame from Master. The process then repeats from 2.

Using any of the 5- to 8-bit character frame formats is possible, but impractical since the receiver must change between using n and n+1 character frame formats. This makes full-duplex operation difficult since the transmitter and receiver use the same character size setting. If 5- to 8-bit character frames are used, the transmitter must be set to use two stop bit (USBS = 1) since the first stop bit is used for indicating the frame type.

Do not use Read-Modify-Write instructions (SBI and CBI) to set or clear the MPCM bit. The MPCM bit shares the same I/O location as the TXC flag and this might accidentally be cleared when using SBI or CBI instructions.

Accessing UBRRH/UCSRC Registers

The UBRRH Register shares the same I/O location as the UCSRC Register. Therefore some special consideration must be taken when accessing this I/O location.

Write Access

When doing a write access of this I/O location the high bit of the value written, the UART Register Select (URSEL) bit, controls which one of the two registers will be written. If URSEL is zero during a write operation, the UBRRH value will be updated. If URSEL is one, the UCSRC setting will be updated.

Code Example - Writing to the UBRRH/UCSRC Registers

The following code example shows how to access the two registers.

NOTE

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
/* Set UBRRH to 2 */
   UBRRH = 0x02;
/* Set the USBS and the UCSZ1 bit to one, and */
/* the remaining bits to zero. */
   UCSRC = (1<<URSEL)|(1<<USBS)|(1<<UCSZ1);
```

As the code example illustrates, write accesses of the two registers are relatively unaffected by the sharing of I/O location.

Read Access

Doing a read access to the UBRRH or UCSRC Register is a more complex operation. However, in most applications, it is rarely necessary to read any of these registers.

The read access is controlled by a timed sequence. Reading the I/O location once returns the UBRRH Register contents. If the register location was read in previous system clock cycle, reading the register in the current clock cycle will return the UCSRC contents. Note that the timed sequence for reading the UCSRC is an automatic operation. Interrupts must therefore be controlled (for example, by disabling interrupts globally) during the read operation.

Code Example - Reading the UBRRH/UCSRC Registers

The following code example shows how to read the UCSRC Register contents.

NOTE

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
unsigned char UART_ReadUCSRC( void )
{
    unsigned char ucsrc;
    /* Read UCSRC */
       ucsrc = UBRRH;
       ucsrc = UCSRC;
       return ucsrc;
}
```

Reading the UBRRH contents is not an automatic operation and therefore it can be read as an ordinary register, as long as the previous instruction did not access the register location.

UART Registers

UBRRHI (UART Baud Rate High)

Bit	7	6	5	4	3	2	1	0	
0x15(0x35)	UOSR [4..0]					UBRR [10..8]			UBRRHI
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	1	1	1	1	0	0	0	

Table 19.2: UART Baud Rate High

- **Bits 7..3 – UOSR: UART Over Sampling Register**

This value defines the number of samples to take for each bit interval. Together with the UBRR, USOR is used to generate an accurate baud rate. Permitted values for UOSR are from 7 to 31. Note that the default value is 15, so 16 samples are taken per bit interval by default.

- **Bits 2..0 – UBRR High bits**

These are the three most significant bits of the 11-bit UART baud rate register (UBRR).

UBRRLO (UART Baud Rate Low)

Bit	7	6	5	4	3	2	1	0	
0x14(0x34)	UBRR [7..0]								UBRRLO
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 19.3: UART Baud Rate Low

- **Bits 7..0 – UBRR[7:0]: UART Baud Rate Register bits 7:0**

These bits are the eight least significant bits of the 11-bit UBRR internal UART register that define the baud rate used by specifying the number of clock cycles between two consecutive samples.

BAUD Rate Table	
UBRR=	BAUD Rate
0x780A	115200
0x7820	38400
0x7840	19200
0x7881	9600
0x7903	4800

Table 19.4: Baud Rate Table

UCRB (UART Control Register B)

Bit	7	6	5	4	3	2	1	0	
0x13(0x33)	–	MPCM	PAR [2..0]			–	CHRL [1..0]		UCRB
Read/Write	R	R/W	R/W	R/W	R/W	R	R/W	R/W	
Initial Value	0	1	0	0	0	0	1	0	

Table 19.5: UART Control Register B

- **Bit 7 Reserved**

This bit is reserved and always read as zero

- **Bit 6 – MPCM: Multi-processor Communication Mode**

This bit is used to enter multiprocessor communication mode. The bit is set when the slave MCU waits for an address byte to be received. When the MCU has been addressed, the MCU switches off the MPCM bit and starts data reception.

- **Bits 5..3 PAR: Parity Mode Selection**

Parity Mode Selection			
PAR Bits			Mode
2	1	0	
0	0	0	Even Parity
0	0	1	Odd Parity
0	1	0	Parity Forced to 0 (space)
0	1	1	Parity Forced to 1 (mark)
1	x	x	No Parity

Table 19.6: Parity Mode Selection

These bits select the parity to be generated when transmitting and checked when receiving. The following modes can be selected: The actual sequence of bits transmitted and received by the UART is: Start bit + 6,7,8 or 9 (depending on CHRL) Data Bits + Parity Bit (only if parity is used) + Stop Bit.

- **Bit 2 Reserved**

This bit is reserved and always read as zero

- **Bit 1..0 CHRL: Character Length**

NOTE

NOTE: The default value after Reset is 10, i.e., 8-bit characters are used.

These bits select the width of the data words to be transmitted and received according to Table 19.7.

Character Length		
CHRL		Mode
1	0	
0	0	6 Bits
0	1	7 Bits
1	0	8 Bits
1	1	9 Bits

Table 19.7: UART Character Length Table

UCRA (UART Control Register A)

Bit	7	6	5	4	3	2	1	0	
0x12(0x32)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	–	RXB8	TXB8	UCRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 19.8: UART Control Register A

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

  When this bit is set (logic one), a setting of the RXC bit in the USR will cause the Receive Complete Interrupt routine to be executed provided that global interrupts are enabled.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

  When this bit is set (logic one), a setting of the TXC bit in the USR will cause the Transmit Complete Interrupt routine to be executed provided that global interrupts are enabled.

- **Bit 5 – UDRIE: UART Data Register Empty Interrupt Enable**

  When this bit is set (logic one), a setting of the UDRE bit in the USR will cause the UART Data Register Empty Interrupt routine to be executed provided that global interrupts are enabled.

- **Bit 4 – RXEN: Receiver Enable**

  This bit enables the UART receiver when set (logic one). When disabled, the RXC, OR and FE status flags cannot be set. If these flags are set, turning of RXEN does not cause them to be cleared.

- **Bit 3 – TXEN: Transmitter Enable**

  This bit enables the UART transmitter when set (logic one). If disabling the transmitter is requested while transmitting a character, the transmitter is not disabled before the character in the shift registers plus any following character in the UDR has been completely transmitted.

- **Bit 2 Reserved**

  This bit is reserved and always read as zero

- **Bit 1 – RXB8: Receive Data Bit 8**

  When CHRL = 11, RXB8 is the ninth data bit of the received character

- **Bit 0 – TXB8: Transmit Data Bit 8**

  When CHRL = 11, TXB8 is the ninth data bit of the character to be transmitted

USR (UART Status Register)

Bit	7	6	5	4	3	2	1	0	
0x11(0x31)	RXC	TXC	UDRE	FE	OR	PE	NE	–	USR
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	1	0	0	0	0	0	

Table 19.8: UART Status Register

- **Bit 7 – RXC: UART Receive Complete**

  Set to logic one when a received character is transferred from the Receiver Shift Register to the UDR. The bit is set regardless of any detected framing errors. When the RXCIE bit in UCRA is set, the UART Receive Complete interrupt will be raised when the RXC is set (logic one). RXC is cleared by reading the UDR. When interrupt-driven data reception is used, the UART Receive Complete Interrupt routine must read UDR in order to clear RXC, otherwise a new interrupt will occur once the interrupt routine terminates.

  Set to logic one when the entire character (including the stop bit) in the Transmit Shift Register has been shifted out and no new data has been written to the UDR. This flag is especially useful in half-duplex communications interfaces where a transmitting application must enter receive mode and free the communications bus immediately after completing the transmission. When the TXCIE bit in UCRA is set, a set of TXC causes the UART Transmit Complete interrupt to be raised. TXC can be cleared by hardware when executing the corresponding interrupt handling vector which writes a logic one to the TXC bit in the GIFR. Alternatively, the TXC bit is cleared (logic zero) by writing a logical one to the bit.

- **Bit 5 – UDRE: UART Data Register Empty**

  Set to logic one when a character written to the UDR is transferred to the Transmit Shift Register. Setting of this bit indicates that the transmitter is ready to receive a new character for transmission. When the UDRIE bit in the UCRA is set, the UART Transmit Complete interrupt is raised when UDRE is set. UDRE is cleared by writing UDR. When interrupt driven data transmission is used, the UART Data Register Empty Interrupt routine must write UDR in order to clear UDRE, otherwise a new interrupt will occur once the interrupt routine terminates. UDRE is set (one) during reset to indicate that the transmitter is ready.

- **Bit 4 – FE: Framing Error**

  This bit is set if a Framing Error condition is detected, i.e., when the stop bit of an incoming character is zero. The FE bit is cleared when the stop bit of received data is one.

- **Bit 3 – OR: Overrun Flag**

  This bit is set if an Overrun condition is detected, i.e., when a character already present in the UDR is not read before the next character has been shifted into the Receiver Shift Register. The OR bit is buffered, which means that it will be updated once the valid data still in UDR is read. The OR bit is cleared (logic zero) when data is received and transferred to the UDR.

- **Bit 2 – PE: Parity Error**

  This bit is set whenever the parity of the received character does not match current parity (PAR bits in the UCRB). The PE bit is updated at each new received character.

- **Bit 1 – NE: Noise Error**

  This bit is set when noise has been detected (three samples not identical) during the last reception (including the parity and the stop bit). The NE bit is updated at each new received character.

- **Bit 0 Reserved**

  This bit is reserved and always read as zero

UDR (UART I/O Data Register)

Bit	7	6	5	4	3	2	1	0	
0x10(0x30)	UDR[7:0]								UDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 19.10: UART I/O Data Register

- **Bits 7..0 – UDR: UART Data register bits 7:0**

  The UDR is actually two physically separate registers sharing the same I/O address. When writing to the register, the UART Transmit Data Register is written. When reading from the UDR, the UART Receive Data Register is read.

Baud Rate Setting

Baud rates for asynchronous operation can be generated by using the UBRR settings in the table below. UBRR values which yield an actual baud rate differing less than 0.5% from the target baud rate are shown as bold in the table. Higher error ratings are acceptable, but the receiver will have less noise resistance when the error ratings are high, especially for large serial frames (see "Asynchronous Operational Range" earlier in this section. The error values are calculated using the following equation:

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1 \right) \bullet 100\%$$

UBRR Settings

fosc=20.0000 MHz	
U2X=0	
UBRR	Error (%)
259	0.16
129	0.16
64	0.16
32	-1.36
10	-1.4

Table 19.11: UBRR Settings for Common Oscillator Frequencies

SECTION 20
SERIAL PERIPHERAL INTERFACE (SPI)

Introduction

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the M3000 and peripheral devices, or between several M3000 devices. The SPI includes the following features:

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- Maximum Bit Frequency of 5 M-bits/second
- LSB First or MSB First Data Transfer
- Four Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection

The M3000 utilizes two (2) SPI blocks. They are:

- The Master SPI
- The User SPI

The Master SPI

The Master SPI is not switchable from Master to Slave. This SPI Block manages the outputs for the Master Serial Out, Master Chip Select Out, Master Clock Out and Master Serial Input.

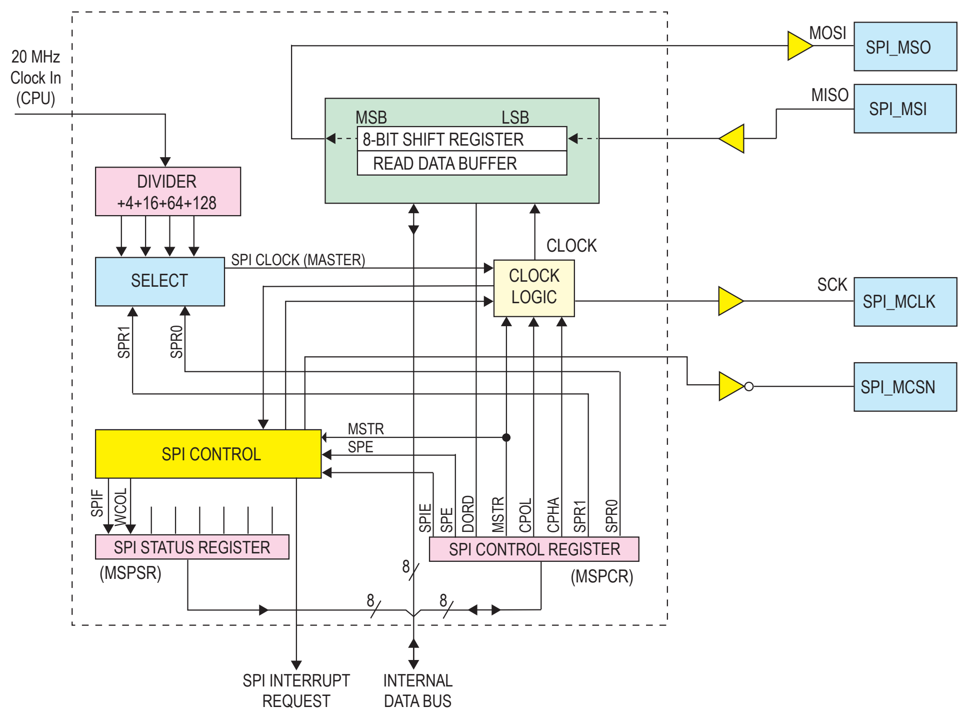


Figure 20.1: Master SPI Block Diagram

The User SPI may be configured as a Master or as a Slave.

When configured as a Master, the User SPI interface has no automatic control of the SPI_UCSN line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, SPI_UCSN line. The last incoming byte will be kept in the buffer register for later use.

When configured as a Slave, the User SPI interface will remain sleeping with MISO tri-stated as long as the SPI_UCSN pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the SPI_UCSN pin is driven low. As one byte has been completely shifted, the end of transmission flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the buffer register for later use.

User SPI

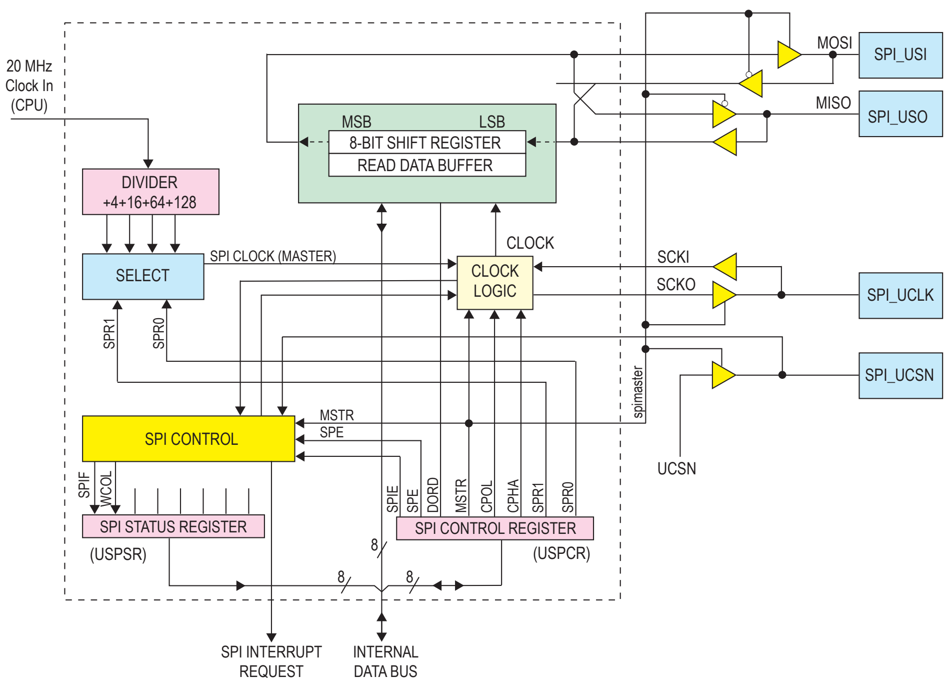


Figure 20.2: User SPI Block Diagram

The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed. When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost.

SPI_UCSN Pin Functionality

When the SPI is enabled, the data direction of the SPI_USI, SPI_UCK, SPI_UCSN and SPI_USO pins are overridden according to the table below.

NOTE: While an input, SPI_UCSN serves as the Slave Select input. When in Master mode, this functionality is disabled and SPI_UCSN is controlled by bit 7 of the BGPPORT register.

SPI Pin Overrides			
MSTR	Mode	Input	Output
0	Slave	SPI_USI, SPI_UCK, SPI_UCSN	SPI_USO
1	Master	SPI_USO	SPI_USI, SPI_UCK, SPI_UCSN

Table 20.1: SPI Pin Overrides

Slave Mode

When the SPI is configured as a Slave, the Slave Select (SPI_UCSN) pin is always input. When SPI_UCSN is held low, the SPI is activated and MISO becomes an output. All other pins are inputs. When SPI_UCSN is driven high, all pins are inputs, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI Logic will be reset once the SPI_UCSN pin is driven high.

The SPI_UCSN pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the SPI_UCSN pin is driven high, the SPI Slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

Code Example - Initializing SPI as a Master

The following code examples show how to initialize the SPI as a master and how to perform a simple transmission. DDR_SPI in the examples must be replaced by the actual Data Direction Register controlling the SPI pins. DD_MOSI, DD_MISO and DD_SCK must be replaced by the actual data direction bits for these pins. For example if MOSI is placed on pin PB5, replace DD_MOSI with DDB5 and DDR_SPI with DDRB.

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
void SPI_MasterInit(void)
    {
    /* Set MOSI and SCK output, all others input */
       DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
    /* Enable SPI, Master, set clock rate fck/16 */
       SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
    }

void SPI_MasterTransmit(char cData)
    {
    /* Start transmission */
       SPDR = cData;
    /* Wait for transmission complete */
       while (!(SPSR & (1<<SPIF)));
    }
```

Code Example - Initializing SPI as a Slave

The following code examples show how to initialize the SPI as a Slave and how to perform a simple reception.

NOTE: The example code assumes that the part specific header file is included.

C Code Example

```
void SPI_SlaveInit(void)
{
    /* Set MISO output, all others input */
       DDR_SPI = (1<<DD_MISO);
    /* Enable SPI */
       SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
    /* Wait for reception complete */
       while(!(SPSR & (1<<SPIF)));
    /* Return data register */
       return SPDR;
}
```

Master SPI Registers

MSPDR (Master SPI Data Register)

Bit	7	6	5	4	3	2	1	0	
0x0E(0x2E)	MSB	–	–	–	–	–	–	LSB	MSPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 20.2: Master SPI Data Register

- **Bits 7..0 – MSPDR[7:0]: Master SPI Data Register bits 7:0**

The Master SPI Data Register is a read/write register used for data transfer between the register file and the SPI Shift register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

MSPSR (Master SPI Status Register)

Bit	7	6	5	4	3	2	1	0	
0x0D(0x2D)	SPIF	WCOL	–	–	–	–	–	–	MSPSR
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 20.3: Master SPI Status Register

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF bit is set (logic one) and an interrupt is generated if SPIE in SPCR is set (logic one) and global interrupts are enabled. SPIF is cleared by hardware when executing the corresponding interrupt handling vector and writing logic one to the corresponding interrupt bit (MSPIF) in the GIFR. Alternatively, the SPIF bit is cleared by first reading the SPI status register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write Collision Flag**

The WCOL bit is set if the SPI data register (SPDR) is written during a data transfer. During data transfer, the result of reading the SPDR register may be incorrect and writing to it will have no effect. The WCOL bit (and the SPIF bit) are cleared (logic zero) by first reading the SPI Status register with WCOL set and then accessing the SPI Data register.

- **Bits 5..0 Reserved**

These bits are reserved and always read as zero.

MSPCR (Master SPI Control Register)

Bit	7	6	5	4	3	2	1	0	
0x0C(0x2C)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR [1..0]		MSPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 20.4: Master SPI Control Register

- **Bit 7 – SPIE: SPI Interrupt Enable**

  This bit causes setting of the SPIF bit in the SPSR register to execute the SPI interrupt provided that global interrupts are enabled.

- **Bit 6 – SPE: SPI Enable**

  When the SPE bit is set (logic one), the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

  When the DORD bit is set (logic one), the LSB of the data word is transmitted first. When the DORD bit is cleared (logic zero), the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: SPI_MCSN Control**

  This bit selects Master SPI mode when set (logic one) and Slave SPI mode when cleared (logic Zero). This bit is inverted and driven out on the SPI_MCSN pin.

- **Bit 3 – CPOL: Clock Polarity**

  When this bit is set (logic one), SCK is high when idle. When CPOL is cleared (logic zero), SCK is low when idle.

- **Bit 2 – CPHA: Clock Phase**

  This bit when set causes data to be transmitted or received on the non-idle (according to CPOL) transition of SCK. When this bit is cleared, data is transmitted and received during the middle of the SCK idle (according to CPOL) period.

- **Bits 1..0 – SPR[1:0]: SPI Clock Rate Select**

  This value controls the SCK rate of the device configured as a master. SPR[1:0] has no effect on a slave configured device. The relationship between SCK and the Oscillator clock frequency (Fclk = 20 MHz) is shown in table 20.5.

SPI Clock Frequency		
SPR[1..0] Bits		SCK Frequency
1	0	
0	0	5 MHz
0	1	1.25 MHz
1	0	312.5 kHz
1	1	156.25 kHz

Table 20.5: SPI Clock Frequency

User SPI Registers

USPDR
(User SPI Data Register)

Bit	7	6	5	4	3	2	1	0	
0x0A(0x2A)	MSB	–	–	–	–	–	–	LSB	USPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 20.6: User SPI Data Register

- **Bits 7..0 – USPDR[7:0]: User SPI Data Register bits 7:0**

The User SPI Data Register is a read/write register used for data transfer between the register file and the SPI Shift register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

USPSR
(User SPI Status Register)

Bit	7	6	5	4	3	2	1	0	
0x09(0x29)	SPIF	WCOL	–	–	–	–	–	–	USPSR
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Table 20.7: User SPI Status Register

- **Bit 7 – SPIF: SPI Interrupt Flag**

When a serial transfer is complete, the SPIF bit is set (logic one) and an interrupt is generated if SPIE in SPCR is set (logic one) and global interrupts are enabled. SPIF is cleared by hardware when executing the corresponding interrupt handling vector and writing logic one to the corresponding interrupt bit (USPIF) in the GIFR. Alternatively, the SPIF bit is cleared by first reading the SPI status register with SPIF set, then accessing the SPI Data Register (SPDR).

- **Bit 6 – WCOL: Write Collision Flag**

The WCOL bit is set if the SPI data register (SPDR) is written during a data transfer. During data transfer, the result of reading the SPDR register may be incorrect and writing to it will have no effect. The WCOL bit and the SPIF bit are cleared (logic zero) by first reading the SPI Status register with WCOL set, and then accessing the SPI Data register.

- **Bits 5..0 Reserved**

These bits are reserved and always read as zero

USPCR
(User SPI Control Register)

Bit	7	6	5	4	3	2	1	0	
0x08(0x28)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	USPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 20.8: User SPI Control Register

- **Bit 7 – SPIE: SPI Interrupt Enable**

This bit causes setting of the SPIF bit in the SPSR register to execute the SPI interrupt provided that global interrupts are enabled.

- **Bit 6 – SPE: SPI Enable**

When the SPE bit is set (logic one), the SPI is enabled. This bit must be set to enable any SPI operations.

- **Bit 5 – DORD: Data Order**

When the DORD bit is set (logic one), the LSB of the data word is transmitted first. When the DORD bit is cleared (logic zero), the MSB of the data word is transmitted first.

- **Bit 4 – MSTR: Master/Slave Select**

NOTE: While an input, SPI_UCSN serves as the Slave Select input. When in Master mode, this functionality is disabled and SPI_UCSN is controlled by bit 7 of the BGPPORT register.

This bit selects Master SPI mode when set (logic one) and Slave SPI mode when cleared (logic Zero). This bit changes the direction of the SPI_USI, SPI_USO, SPI_UCK, and SPI_UCSN pins in the following way:

Master/Slave Select			
MSTR	Mode	Input	Output
0	Slave	SPI_USI, SPI_UCK, SPI_UCSN	SPI_USO
1	Master	SPI_USO	SPI_USI, SPI_UCK, SPI_UCSN

Table 20.9: Master/Slave Select

- **Bit 3 – CPOL: Clock Polarity**

When this bit is set (logic one), SCK is high when idle. When CPOL is cleared (logic zero), SCK is low when idle.

CPOL Functionality		
CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

Table 20.10: CPOL Functionality

- **Bit 2 – CPHA: Clock Phase**

This bit, when set, causes data to be transmitted or received on the non-idle (according to CPOL) transition of SCK. When this bit is cleared, data is transmitted and received during the middle of the SCK idle (according to CPOL) period.

CPHA Functionality		
CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

Table 20.11: CPHA Functionality

- **Bits 1..0 – SPR[1:0]: SPI Clock Rate Select**

This value controls the SCK rate of the device configured as a master. SPR[1:0] has no effect on a slave configured device. The relationship between SCK and the Oscillator is shown in the following table:

SPI Clock Frequency		
SPR[1..0] Bits		SCK Frequency
1	0	
0	0	5 MHz
0	1	1.25 MHz
1	0	312.5 kHz
1	1	156.25 kHz

Table 20.12: Step Clock Frequency

Data Modes

There are four combinations of SCK phase and polarity with respect to serial data, which are determined by control bits CPHA and CPOL. The SPI data transfer formats are shown in the figures below.

CPOL and CPHA Functionality

SCK		Leading Edge	Trailing Edge	SPI Mode
COPL	CPHA			
0	0	Sample (Rising)	Setup (Falling)	0
0	1	Setup (Rising)	Sample (Falling)	1
1	0	Sample (Falling)	Setup (Rising)	2
1	1	Setup (Falling)	Sample (Rising)	3

Table 20.13: CPOL and CPHA Functionality

SPI Transfer Format with CPHA = 0

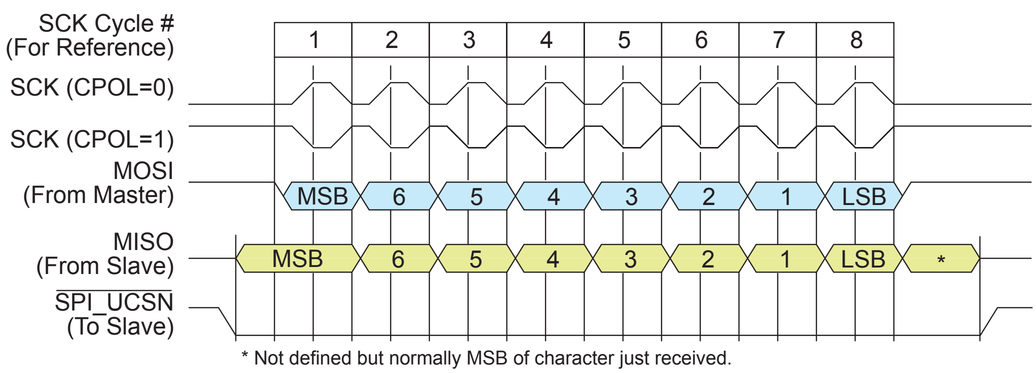


Figure 20.3: SPI Transfer Format with CHPA = 0

SPI Transfer Format with CPHA = 1

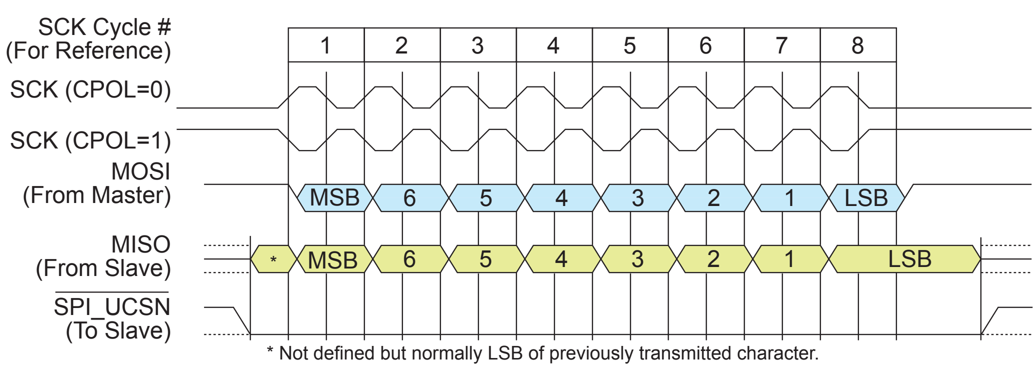


Figure 20.4: SPI Transfer Format with CPHA = 1

Page Intentionally Left Blank

WARRANTY

System Semiconductor ("SSI"), warrants only to the purchaser of the Product from SSI (the "Customer") that the product purchased from SSI (the "Product") will be free from defects and meets the applicable specifications at the time of sale. Customer's exclusive remedy under this Limited Warranty shall be the repair or replacement, at Company's sole option, of the Product, or any part of the Product, determined by SSI to be defective.

This Limited Warranty does not extend to any Product damaged by reason of alteration, accident, abuse, neglect or misuse or improper or inadequate handling; improper or inadequate wiring utilized or installed in connection with the Product; installation, operation or use of the Product not made in strict accordance with the specifications and written instructions provided by SSI; use of the Product for any purpose other than those for which it was designed; ordinary wear and tear; disasters or Acts of God; unauthorized attachments, alterations or modifications to the Product; the misuse or failure of any item or equipment connected to the Product not supplied by SSI; improper maintenance or repair of the Product; or any other reason or event not caused by SSI.

SSI HEREBY DISCLAIMS ALL OTHER WARRANTIES, WHETHER WRITTEN OR ORAL, EXPRESS OR IMPLIED BY LAW OR OTHERWISE, INCLUDING WITHOUT LIMITATION, **ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE**. CUSTOMER'S SOLE REMEDY FOR ANY DEFECTIVE PRODUCT WILL BE AS STATED ABOVE, AND IN NO EVENT WILL THE SSI BE LIABLE FOR INCIDENTAL, CONSEQUENTIAL, SPECIAL OR INDIRECT DAMAGES IN CONNECTION WITH THE PRODUCT.

This Limited Warranty shall be void if the Customer fails to comply with all of the terms set forth in this Limited Warranty. This Limited Warranty is the sole warranty offered by SSI with respect to the Product. SSI does not assume any other liability in connection with the sale of the Product. No representative of SSI is authorized to extend this Limited Warranty or to change it in any manner whatsoever. No warranty applies to any party other than the original Customer.

SSI and its directors, officers, employees, subsidiaries and affiliates shall not be liable for any damages arising from any loss of equipment, loss or distortion of data, loss of time, loss or destruction of software or other property, loss of production or profits, overhead costs, claims of third parties, labor or materials, penalties or liquidated damages or punitive damages, whatsoever, whether based upon breach of warranty, breach of contract, negligence, strict liability or any other legal theory, or other losses or expenses incurred by the Customer or any third party.

System Semiconductor's general policy does not recommend the use of its products in life support or aircraft applications wherein a failure or malfunction of the product may directly threaten life or injury. Per System Semiconductor's terms and conditions of sales, the user of System Semiconductor, products in life support or aircraft applications assumes all risks of such use and indemnifies System Semiconductor, against all damages.

M3000 Motor and Motion
Controller User's Guide Manual
P/N: SS-MAN-3000
Part 1 of 2

SYSTEM SEMICONDUCTOR™

362 North Main Street
Marlborough CT 06447

Phone: 860-295-6170
Fax: 860-295-8318

info@systemsemi.com
www.systemsemi.com

Revision R081307