AUTOMATIONDIRECT.com

DL06 Micro PLC User Manual

Volume 2 of 2

Manual Number: D0-06USER-M

WARNING

Thank you for purchasing automation equipment from **Automationdirect.com®**, doing business as, **AutomationDirect**. We want your new automation equipment to operate safely. Anyone who installs or uses this equipment should read this publication (and any other relevant publications) before installing or operating the equipment.

To minimize the risk of potential safety problems, you should follow all applicable local and national codes that regulate the installation and operation of your equipment. These codes vary from area to area and usually change with time. It is your responsibility to determine which codes should be followed, and to verify that the equipment, installation, and operation is in compliance with the latest revision of these codes.

At a minimum, you should follow all applicable sections of the National Fire Code, National Electrical Code, and the codes of the National Electrical Manufacturer's Association (NEMA). There may be local regulatory or government offices that can also help determine which codes and standards are necessary for safe installation and operation.

Equipment damage or serious injury to personnel can result from the failure to follow all applicable codes and standards. We do not guarantee the products described in this publication are suitable for your particular application, nor do we assume any responsibility for your product design, installation, or operation.

Our products are not fault-tolerant and are not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the product could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). AutomationDirect specifically disclaims any expressed or implied warranty of fitness for High Risk Activities.

For additional warranty and safety information, see the Terms and Conditions section of our catalog. If you have any questions concerning the installation or operation of this equipment, or if you need additional information, please call us at 770-844-4200.

This publication is based on information that was available at the time it was printed. At AutomationDirect we constantly strive to improve our products and services, so we reserve the right to make changes to the products and/or publications at any time without notice and without any obligation. This publication may also discuss features that may not be available in certain revisions of the product.

Trademarks

This publication may contain references to products produced and/or offered by other companies. The product and company names may be trademarked and are the sole property of their respective owners. AutomationDirect disclaims any proprietary interest in the marks and names of others.

AVERTISSEMENT

Nous vous remercions d'avoir acheté l'équipement d'automatisation de **Automationdirect.comMC**, en faisant des affaires comme, **AutomationDirect**. Nous tenons à ce que votre nouvel équipement d'automatisation fonctionne en toute sécurité. Toute personne qui installe ou utilise cet équipement doit lire la présente publication (et toutes les autres publications pertinentes) avant de l'installer ou de l'utiliser.

Afin de réduire au minimum le risque d'éventuels problèmes de sécurité, vous devez respecter tous les codes locaux et nationaux applicables régissant l'installation et le fonctionnement de votre équipement. Ces codes diffèrent d'une région à l'autre et, habituellement, évoluent au fil du temps. Il vous incombe de déterminer les codes à respecter et de vous assurer que l'équipement, l'installation et le fonctionnement sont conformes aux exigences de la version la plus récente de ces codes.

Vous devez, à tout le moins, respecter toutes les sections applicables du Code national de prévention des incendies, du Code national de l'électricité et des codes de la National Electrical Manufacturer's Association (NEMA). Des organismes de réglementation ou des services gouvernementaux locaux peuvent également vous aider à déterminer les codes ainsi que les normes à respecter pour assurer une installation et un fonctionnement sûrs.

L'omission de respecter la totalité des codes et des normes applicables peut entraîner des dommages à l'équipement ou causer de graves blessures au personnel. Nous ne garantissons pas que les produits décrits dans cette publication conviennent à votre application particulière et nous n'assumons aucune responsabilité à l'égard de la conception, de l'installation ou du fonctionnement de votre produit.

Nos produits ne sont pas insensibles aux défaillances et ne sont ni conçus ni fabriqués pour l'utilisation ou la revente en tant qu'équipement de commande en ligne dans des environnements dangereux nécessitant une sécurité absolue, par exemple, l'exploitation d'installations nucléaires, les systèmes de navigation aérienne ou de communication, le contrôle de la circulation aérienne, les équipements de survie ou les systèmes d'armes, pour lesquels la défaillance du produit peut provoquer la mort, des blessures corporelles ou de graves dommages matériels ou environnementaux («activités à risque élevé»). La société **AutomationDirect** nie toute garantie expresse ou implicite d'aptitude à l'emploi en ce qui a trait aux activités à risque élevé.

Pour des renseignements additionnels touchant la garantie et la sécurité, veuillez consulter la section Modalités et conditions de notre documentation. Si vous avez des questions au sujet de l'installation ou du fonctionnement de cet équipement, ou encore si vous avez besoin de renseignements supplémentaires, n'hésitez pas à nous téléphoner au 770-844-4200.

Cette publication s'appuie sur l'information qui était disponible au moment de l'impression. À la société **AutomationDirect**, nous nous efforçons constamment d'améliorer nos produits et services. C'est pourquoi nous nous réservons le droit d'apporter des modifications aux produits ou aux publications en tout temps, sans préavis ni quelque obligation que ce soit. La présente publication peut aussi porter sur des caractéristiques susceptibles de ne pas être offertes dans certaines versions révisées du produit.

Marques de commerce

La présente publication peut contenir des références à des produits fabriqués ou offerts par d'autres entreprises. Les désignations des produits et des entreprises peuvent être des marques de commerce et appartiennent exclusivement à leurs propriétaires respectifs. **AutomationDirect** nie tout intérêt dans les autres marques et désignations.

TABLE OF CONTENTS

Chapter 1: Getting Started

Chapter 2: Installation, Wiring, and Specifications

Chapter 4: System Design and Configuration

Chapter 6: Drum Instruction Programming

VOLUME TWO: TABLE OF CONTENTS

Chapter 7: RLL^{PLUS} Stage Programming

Chapter 9: Maintenance and Troubleshooting

Appendix A: Auxiliary Functions

Appenedix B: DL06 Error codes

Appendix C: Instruction Execution Times

Appendix D: Special Relays

Appendix E: High-speed Input and Pulse Output Features

Appendix K: Introduction to Serial Communications

Index

Notes

Drum Instruction Programming

In This Chapter...

Introduction

Purpose

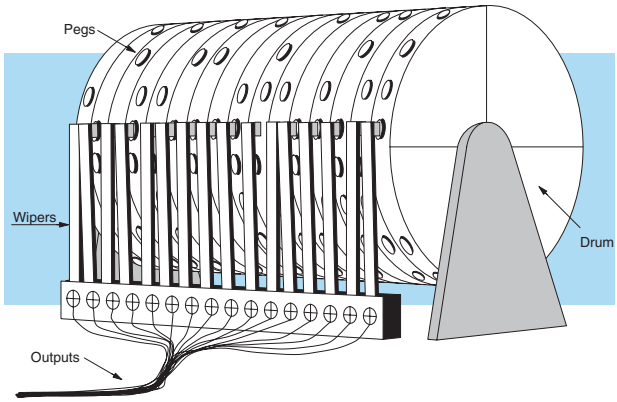
The Event Drum (EDRUM) instruction in the DL06 CPU electronically simulates an electro-mechanical drum sequencer. The instruction offers enhancements to the basic principle, which we describe first.

Drum Terminology

Drum instructions are best suited for repetitive processes that consist of a finite number of steps. They can do the work of many rungs of ladder logic with elegant simplicity. Therefore, drums can save a lot of programming and debugging time.

We introduce some terminology associated with the **drum** instruction by describing the original mechanical drum shown below. The mechanical drum generally has pegs on its curved surface. The pegs are populated in a particular **pattern**, representing a set of desired actions for machine control. A motor or solenoid rotates the drum a precise amount at specific times. During rotation, stationary wipers sense the presence of pegs (present = on, absent = off). This interaction makes or breaks electrical contact with the wipers, creating electrical **outputs** from the drum. The outputs are wired to devices on a machine for On/Off control.

Drums usually have a finite number of positions within one rotation, called **steps**. Each step represents some process step. At powerup, the drum **resets** to a particular step. The drum rotates from one step to the next based on a **timer**, or on some external **event**. During special conditions, a machine operator can manually increment the drum step using a **jog** control on the drum's drive mechanism. The contact closure of each wiper generates a unique on/off pattern called a **sequence**, designed for controlling a specific machine. Because the drum is circular, it automatically repeats the sequence once per rotation. Applications vary greatly, and a particular drum may rotate once per second, or as slowly as once per week.



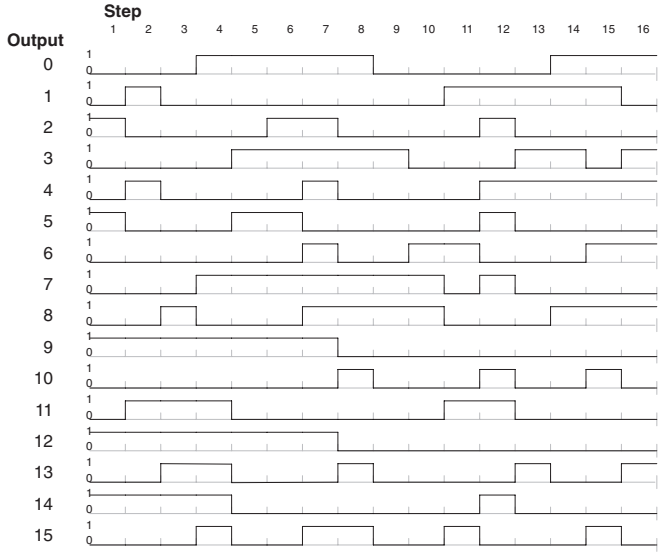
Electronic drums provide the benefits of mechanical drums and more. For example, they have a **preset** feature that is impossible for mechanical drums: The preset function lets you move from the present step *directly* to any other step on command!

Drum Chart Representation

For editing purposes, the electronic drum is presented in chart form in *Direct*SOFT and in this manual. Imagine slicing the surface of a hollow drum cylinder between two rows of pegs, then pressing it flat. Now you can view the drum as a chart as shown below. Each row represents a step, numbered 1 through 16. Each column represents an output, numbered 0 through 15 (to match word bit numbering). The solid circles in the chart represent pegs (On state) in the mechanical drum, and the open circles are empty peg sites (Off state).

STEP	OUTPUTS 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	○	●	○	●	○	○	●	○	○	○	●	○	○	●	○	○
2	○	●	○	●	●	○	●	○	○	○	○	●	○	○	●	○
3	○	●	●	●	●	○	●	●	○	○	○	○	○	○	○	○
4	●	●	●	●	●	○	●	○	●	○	○	○	○	○	○	●
5	○	○	○	●	○	○	●	○	●	○	●	○	●	○	○	●
6	○	○	○	●	○	○	●	○	●	○	●	○	●	●	○	●
7	●	●	○	●	○	○	●	●	●	●	○	●	●	●	○	●
8	●	○	●	○	○	●	○	●	●	○	○	○	●	○	○	●
9	○	○	○	○	○	○	○	●	●	○	○	○	●	○	○	○
10	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	○
11	●	○	○	○	●	○	○	○	○	●	○	○	○	○	●	○
12	○	●	○	○	●	●	○	○	●	○	●	●	○	●	●	○
13	○	○	●	○	○	○	○	○	○	○	○	●	●	○	●	○
14	○	○	○	○	○	○	○	●	○	○	○	●	●	○	●	●
15	●	○	○	○	○	●	○	●	○	●	○	●	○	○	●	●
16	○	○	●	○	○	○	○	●	○	●	○	●	●	○	○	●

The mechanical drum sequencer derives its name from sequences of control changes on its electrical outputs. The following figure shows the sequence of On/Off controls generated by the drum pattern above. Compare the two, and you will find that they are equivalent! If you can see their equivalence, you are well on your way to understanding drum instruction operation.



Step Transitions

Drum Instruction Types

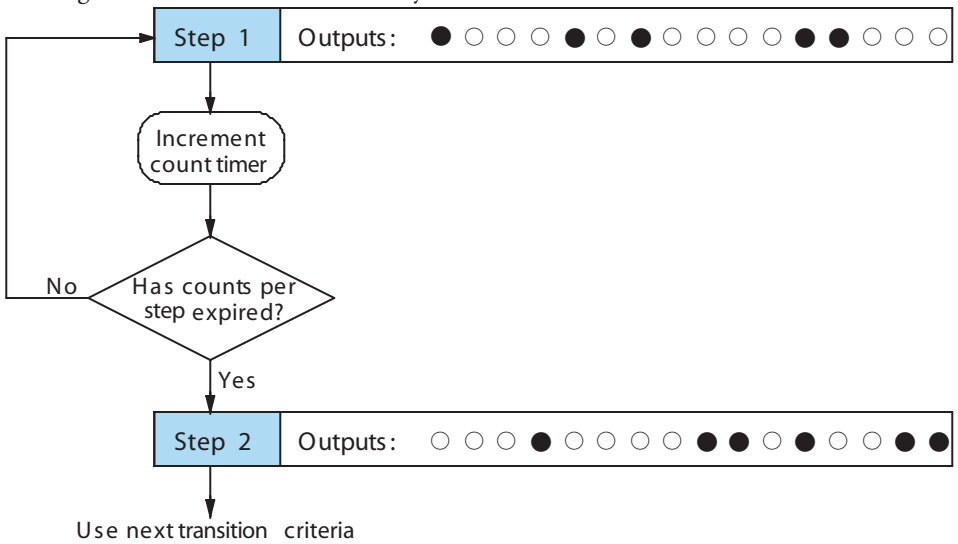
There are two types of Drum instructions in the DL06 CPU:

- Timed Drum with Discrete Outputs (DRUM)
- Time and Event Drum with Discrete Outputs (EDRUM)

The two drum instructions include time-based step transitions, and the EDRUM includes event-based transitions as well. Each drum has 16 steps, and each step has 16 outputs. Refer to the figure below. Each output can be either a Y or C coil, offering programming flexibility. We assign Step 1 an arbitrary unique output pattern.

Timer-Only Transitions

Drums move from one step to another based on time and/or an external event (input). Each step has its own transition condition which you assign during the drum instruction entry. The figure below shows how timer-only transitions work.



The drum stays in Step 1 for a specific duration (user-programmable). The timebase of the timer is programmable, from 0.01 seconds to 99.99 seconds. This establishes the resolution, or the duration of each "tick of the clock." Each step uses the same timebase, but has its own unique counts per step, which you program. When the counts for Step 1 have expired, then the drum moves to Step 2. The outputs change immediately to match the new pattern for Step 2.

The drum spends a specific amount of time in each step, given by the formula:

Time in step = 0.01 seconds X Timebase x Counts per step

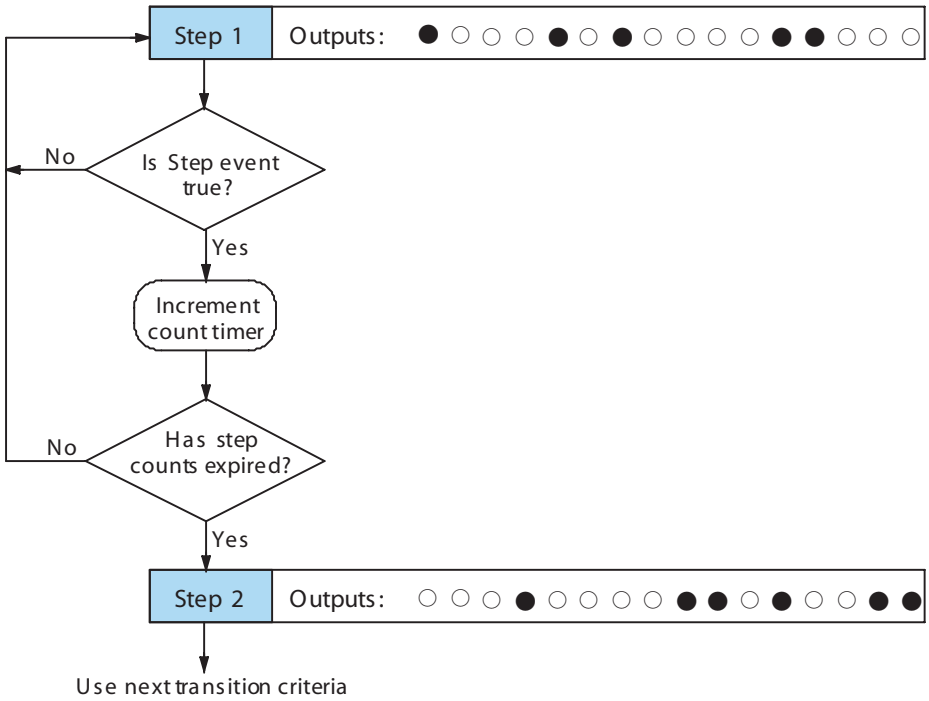
For example, if you program a 5 second time base and 12 counts for Step 1, then the drum will spend 60 seconds in Step 1. The maximum time for any step is given by the formula:

Max Time per step = 0.01 seconds X 9999 X 9999
= 999,800 seconds = 277.7 hours = 11.6 days

***NOTE**: When first choosing the timebase resolution, a good rule of thumb is to make it about 1/10 the duration of the shortest step in your drum. Then you will be able to optimize the duration of that step in 10% increments. Other steps with longer durations allow optimizing by even smaller increments (percentage-wise). Also, note that the drum instruction executes once per CPU scan. Therefore, it is pointless to specify a drum timebase that is much faster than the CPU scan time.*

Timer and Event Transitions

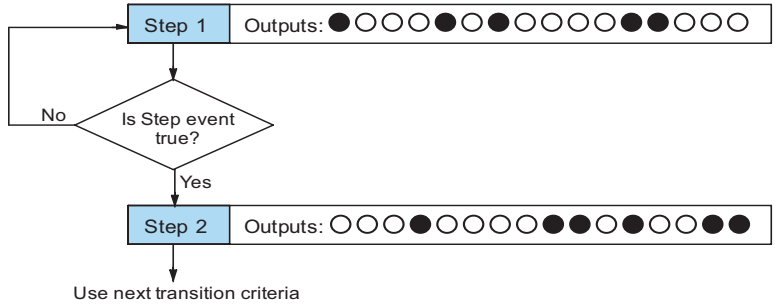
Step transitions may also occur based on time and/or external events. The figure below shows how step transitions work in these cases.



When the drum enters Step 1, it sets the output pattern as shown. Then it begins polling the external input programmed for that step. You can define event inputs as X, Y, or C discrete point types. Suppose we select X0 for the Step 1 event input. If X0 is off, then the drum remains in Step 1. When X0 is On, the event criteria is met and the timer increments. The timer increments as long as the event (X0) remains true. When the counts for Step 1 have expired, then the drum moves to Step 2. The outputs change immediately to match the new pattern for Step 2.

Event-Only Transitions

Step transitions do not require both the event and the timer criteria programmed for each step. You have the option of programming just one of the two, and even mixing transition types among all the steps of the drum. For example, you might want Step 1 to transition on an event, Step 2 to transition on time only, and Step 3 to transition on both time and an event. Furthermore, you may elect to use only part of the 16 steps, and only part of the 16 outputs.



Counter Assignments

Each drum instruction uses the resources of four counters in the CPU. When programming the drum instruction, you select the first counter number. The drum also uses the next three counters automatically. The counter bit associated with the first counter turns on when the drum has completed its cycle, going off when the drum is reset. These counter values and the counter bit precisely indicate the progress of the drum instruction, and can be monitored by your ladder program.

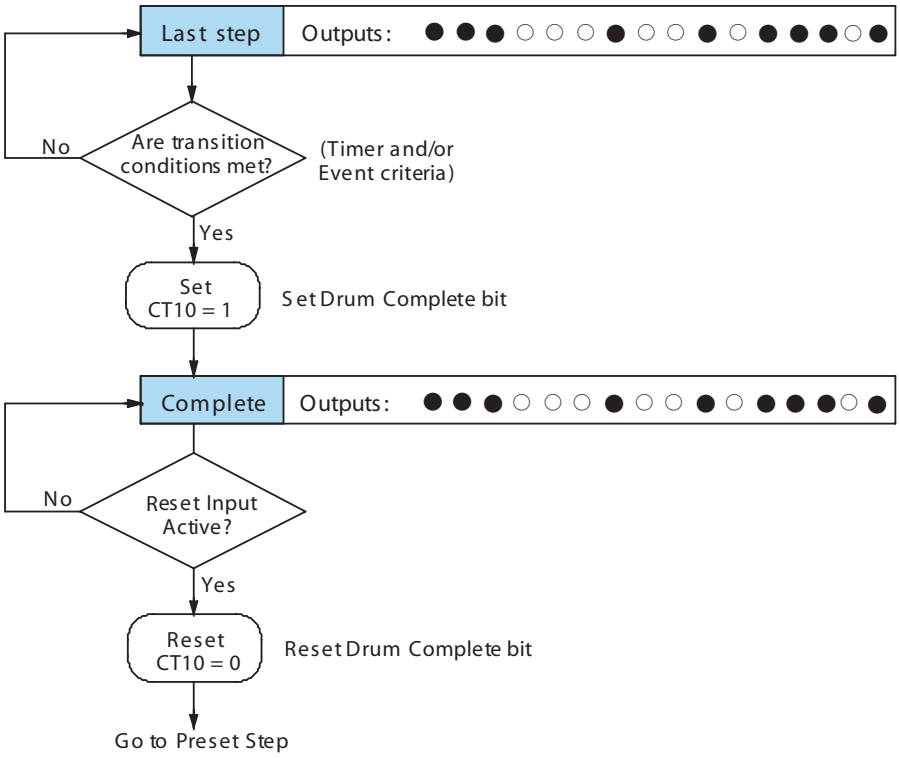
Suppose we program a timer drum to have 8 steps, and we select CT10 for the counter number (remember, counter numbering is in octal). Counter usage is shown to the right. The right column holds typical values, interpreted below.

Counter Assignments			
CT10	Counts in step	V1010	1528
CT11	Timer Value	V1011	0200
CT12	Preset Step	V1012	0001
CT13	Current Step	V1013	0004

CT10 shows that we are at the 1528th count in the current step, which is step 4 (shown in CT13). If we have programmed step 4 to have 3000 counts, then the step is just over half completed. CT11 is the count timer, shown in units of 0.01 seconds. So, each least-significant-digit change represents 0.01 seconds. The value of 200 means that we have been in the current count (1528) for 2 seconds (0.01 x 200). Finally, CT12 holds the preset step value which was programmed into the drum instruction. When the drum's Reset input is active, it presets to step 1 in this case. The value of CT12 changes only if the ladder program writes to it, or the drum instruction is edited and the program is restarted. Counter bit CT10 turns on when the drum cycle is complete, and turns off when the drum is reset.

Last Step Completion

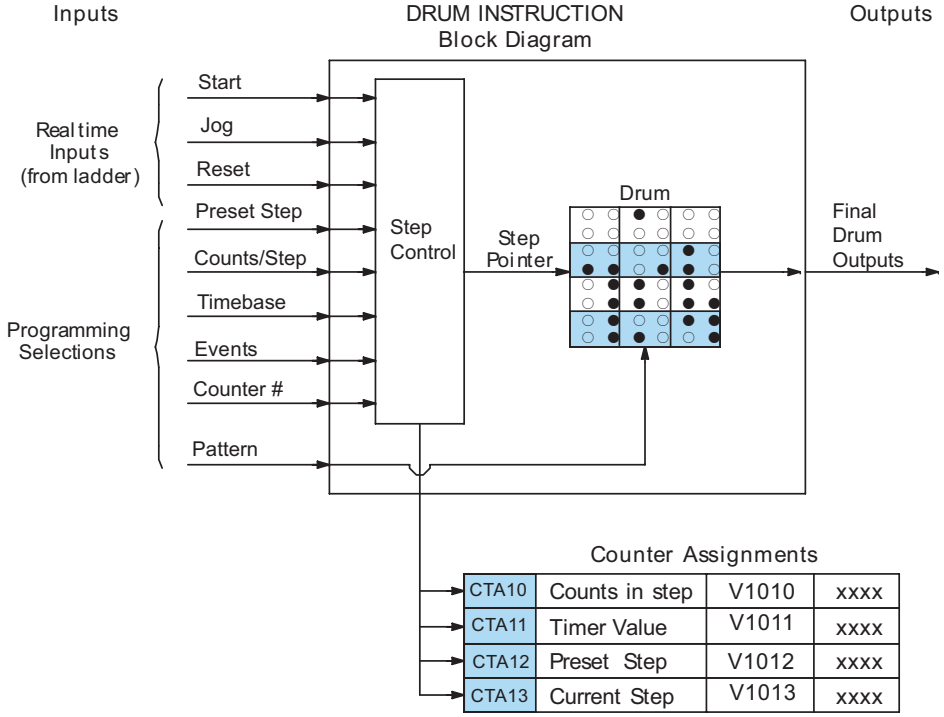
The last step in a drum sequence may be any step number, since partial drums are valid. Refer to the following figure. When the transition conditions of the last step are met, the drum sets the counter bit corresponding to the counter named in the **drum instruction** box (such as CT10). Then it moves to a final "**drum complete**" state. The drum outputs remain in the pattern defined for the last step. Having finished a drum cycle, the Start and Jog inputs have no effect at this point. The drum leaves the "**drum complete**" state when the Reset input becomes active (or on a program-to–run mode transition). It resets the **drum complete** bit (such as CT10), and then goes directly to the appropriate step number defined as the preset step.



Overview of Drum Operation

Drum Instruction Block Diagram

The **drum instruction** utilizes various inputs and outputs in addition to the drum pattern itself. Refer to the figure below.



The **drum instruction** accepts several inputs for step control, the main control of the drum. The inputs and their functions are:

- **Start** – The Start input is effective only when Reset is off. When Start is on, the drum timer runs if it is in a timed transition, and the drum looks for the input event during event transitions. When Start is off, the drum freezes in its current state (Reset must remain off), and the drum outputs maintain their current on/off pattern.
- **Jog** – The jog input is only effective when Reset is off (Start may be either on or off). The jog input increments the drum to the next step on each off-to-on transition (only EDRUM supports the jog input).
- **Reset** – The Reset input has priority over the Start input. When Reset is on, the drum moves to its preset step. When Reset is off, then the Start input operates normally.
- **Preset Step** – A step number from 1 to 16 that you define (typically is step 1). The drum moves to this step whenever Reset is on, and whenever the CPU first enters run mode.

- **Counts/Step** – The number of timer counts the drum spends in each step. Each step has its own counts parameter. However, programming the counts/step is optional.
- **Timer Value** – the current value of the counts/step timer.
- **Counter #** – The counter number specifies the first of four consecutive counters which the drum uses for step control. You can monitor these to determine the drum's progress through its control cycle. The DL06 has 128 counters (CT0 – CT177 in octal).
- **Events** – Either an X, Y, C, S, T, or CT type discrete point serves as step transition inputs. Each step has its own event. However, programming the event is optional.

WARNING: The outputs of a drum are enabled any time the CPU is in Run Mode. The Start Input does not have to be on, and the Reset input does not disable the outputs. Upon entering Run Mode, drum outputs automatically turn on or off according to the pattern of the current step of the drum. This initial step number depends on the counter memory configuration: non-retentive versus retentive.

Powerup State of Drum Registers

The choice of the starting step on powerup and program-to-run mode transitions are important to consider for your application. Please refer to the following chart. If the counter memory is configured as non-retentive, the drum is initialized the same way on every powerup or program-to-run mode transition. However, if the counter memory is configured to be retentive, the drum will stay in its previous state.

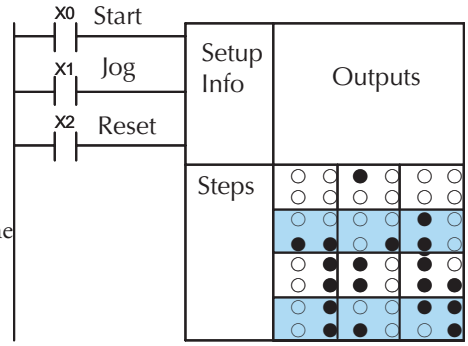
Counter Number	Function	Initialization on Powerup	
		Non-Retentive Case	Retentive Case
CTA(n)	Current Step Count	Initialize = 0	Use Previous (no change)
CTA(n + 1)	Counter Timer Value	Initialize = 0	Use Previous (no change)
CTA(n + 2)	Preset Step	Initialize = Preset Step #	Use Previous (no change)
CTA(n + 3)	Current Step #	Initialize = Preset Step #	Use Previous (no change)

Applications with relatively fast drum cycle times typically will need to be reset on powerup, using the non-retentive option. Applications with relatively long drum cycle times may need to resume at the previous point where operations stopped, using the retentive case. The default option is the retentive case. This means that if you initialize scratchpad V-memory, the memory will be retentive.

Drum Control Techniques

Drum Control Inputs

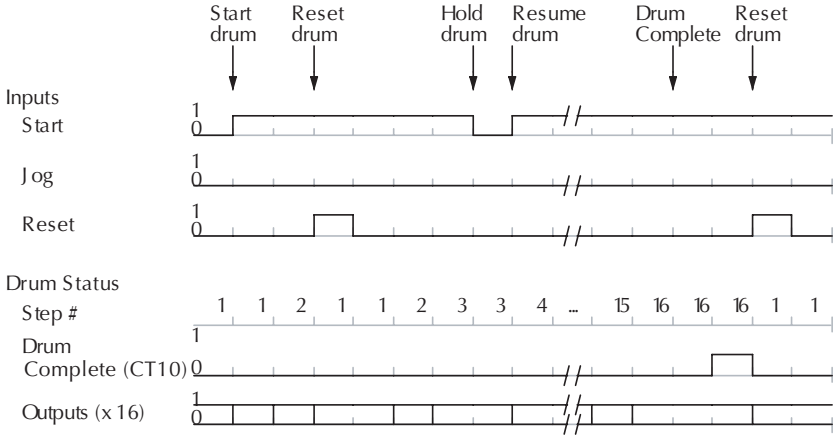
Now we are ready to put together the concepts on the previous pages and demonstrate general control of the drum instruction box. The drawing to the right shows a simplified generic drum instruction. Inputs from ladder logic control the Start, Jog, and Reset Inputs (only the EDRUM instruction supports the Jog Input). The first counter bit of the drum (CT10, for example) indicates the drum cycle is done.



The timing diagram below shows an arbitrary timer drum input sequence and how the drum responds. As the CPU enters Run mode it initializes the step number to the preset step number (typically it is Step 1). When the Start input turns on the drum begins running, waiting for an event and/or running the timer (depends on the setup).

After the drum enters Step 2, Reset turns On while Start is still On. Since Reset has priority over Start, the drum goes to the preset step (Step 1). Note that the drum is *held* in the preset step during Reset, and that step does *not run* (respond to events or run the timer) until Reset turns off.

After the drum has entered step 3, the Start input goes off momentarily, halting the drum's timer until Start turns on again.

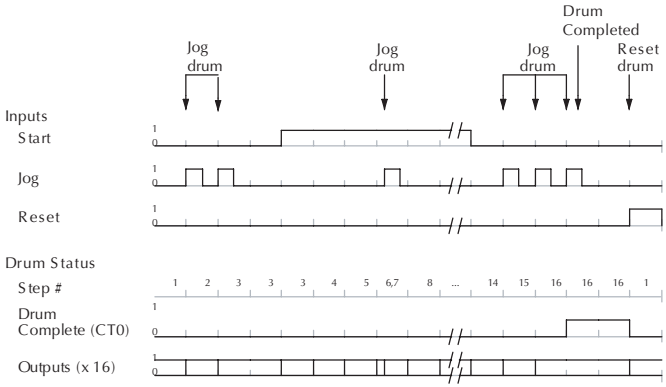


When the drum completes the last step (Step 16 in this example), the Drum Complete bit (CT10) turns on, and the step number remains at 16. When the Reset input turns on, it turns off the Drum Complete bit (CT10), and forces the drum to enter the preset step.

***NOTE**: The timing diagram shows all steps using equal time durations. Step times can vary greatly, depending on the counts/step programmed.*

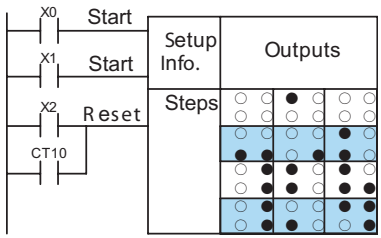
In the figure below, we focus on how the Jog input works on event drums. To the left of the diagram, note that the off-to-on transitions of the Jog input increments the step. Start may be either on or off (however, Reset must be off). Two jogs takes the drum to step three. Next, the Start input turns on, and the drum begins running normally. During step 6 another Jog input signal occurs. This increments the drum to step 7, setting the timer to 0. The drum begins running immediately in step 7, because Start is already on. The drum advances to step 8 normally.

As the drum enters step 14, the Start input turns off. Two more Jog signals moves the drum to step 16. However, note that a third Jog signal is required to move the drum through step 16 to "**drum complete**". Finally, a **Reset** input signal arrives which forces the drum into the preset step and turns off the drum complete bit.



Self-Resetting Drum

Applications often require drums that automatically start over once they complete a cycle. This is easily accomplished, using the drum complete bit. In the figure to the right, the drum instruction setup is for CT10, so we logically OR the drum complete bit (CT10) with the Reset input. When the last step is done, the drum turns on CT10 which resets itself to the preset step, also resetting CT10. Contact X2 still works as a manual reset.



Initializing Drum Outputs

The outputs of a drum are enabled any time the CPU is in run mode. On program-to-run mode transitions, the drum goes to the preset step, and the outputs energize according to the pattern of that step. If your application requires all outputs to be off at powerup, make the preset step in the drum a "**reset step**", with all outputs off.

Using Complex Event Step Transitions

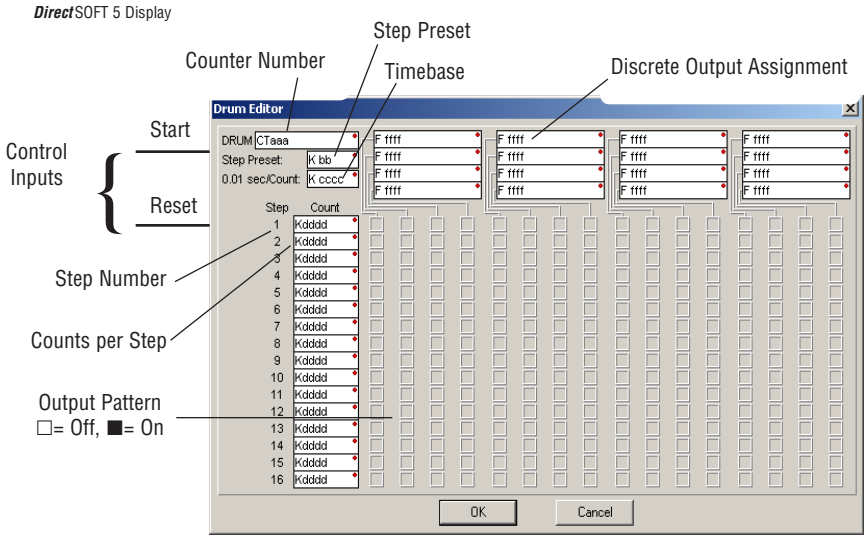
Each event-based transition accepts only one contact reference for the event. However, this does not limit events to just one contact. Just use a control relay contact such as C0 for the step transition event. Elsewhere in ladder logic, you may use C0 as an output coil, making it dependent on many other events (contacts).

Drum Instruction

The DL06 drum instructions may be programmed using *Direct*SOFT, or for the EDRUM instruction only you can use a handheld programmer (firmware version v2.21 or later). This section covers entry using *Direct*SOFT for all instructions plus the handheld mnemonics for the EDRUM instruction.

Timed Drum with Discrete Outputs (DRUM)

The Timed Drum with Discrete Outputs is the most basic of the DL06's drum instructions. It operates according to the principles covered on the previous pages. Below is the instruction in chart form as displayed by *Direct*SOFT.



The Timed Drum features 16 steps and 16 outputs. Step transitions occur only on a timed basis, specified in counts per step. Unused steps must be programmed with "counts per step"=0 (this is the default entry). The discrete output points may be individually assigned as X, Y, or C types, or may be left unused. The output pattern may be edited graphically with *Direct*SOFT.

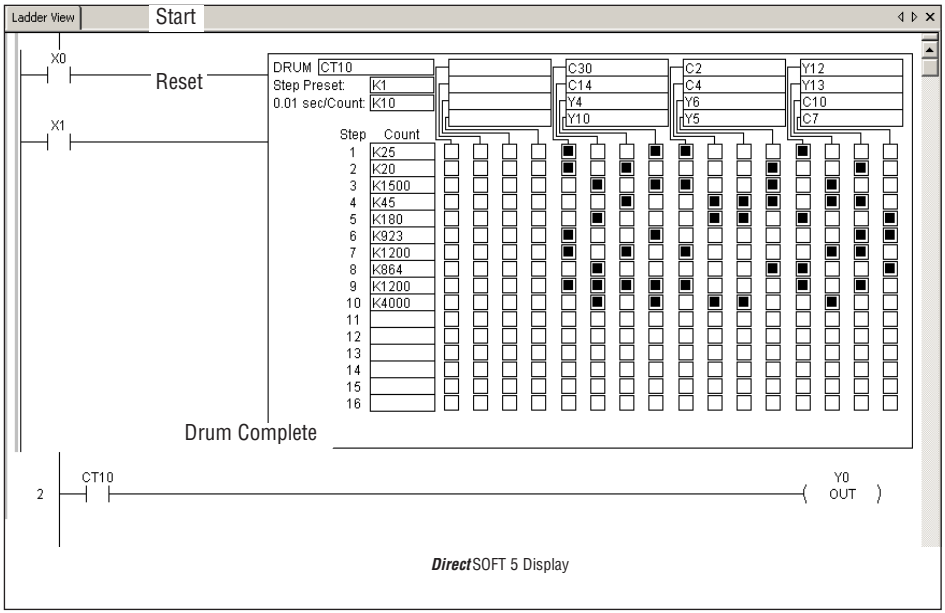
Whenever the Start input is energized, the drum's timer is enabled. It stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa --	0 --174	
Preset Step	bb	K	1 -- 16
Timer base	cccc	K	0 -- 99.99 seconds
Counts per step	dddd	K	0 -- 9999
Discrete Outputs	Fffff	X, Y, C	see memory map

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 -- 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 -- 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 -- 176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 -- 177	Current Step	CT(n+3) = (not used)

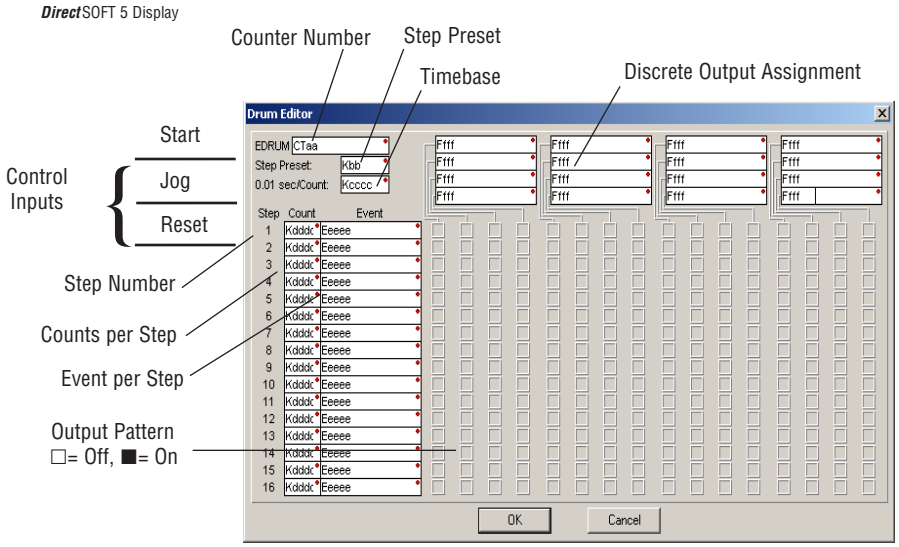
The following ladder program shows the DRUM instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 10 are used, and twelve of the sixteen output points are used. The preset step is step 1. The timebase runs at (K10 x 0.01) = 0.1 second per count. Therefore, the duration of step 1 is (25 x 0.1) = 2.5 seconds. In the last rung, the Drum Complete bit (CT10) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT10.



DirectSOFT 5 Display

Event Drum (EDRUM)

The Event Drum (EDRUM) features time-based and event-based step transitions. It operates according to the general principles of drum operation covered in the beginning of this chapter. Below is the instruction as displayed by *Direct*SOFT.



The Event Drum features 16 steps and 16 discrete outputs. Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps and events must be left blank. The discrete output points may be individually assigned.

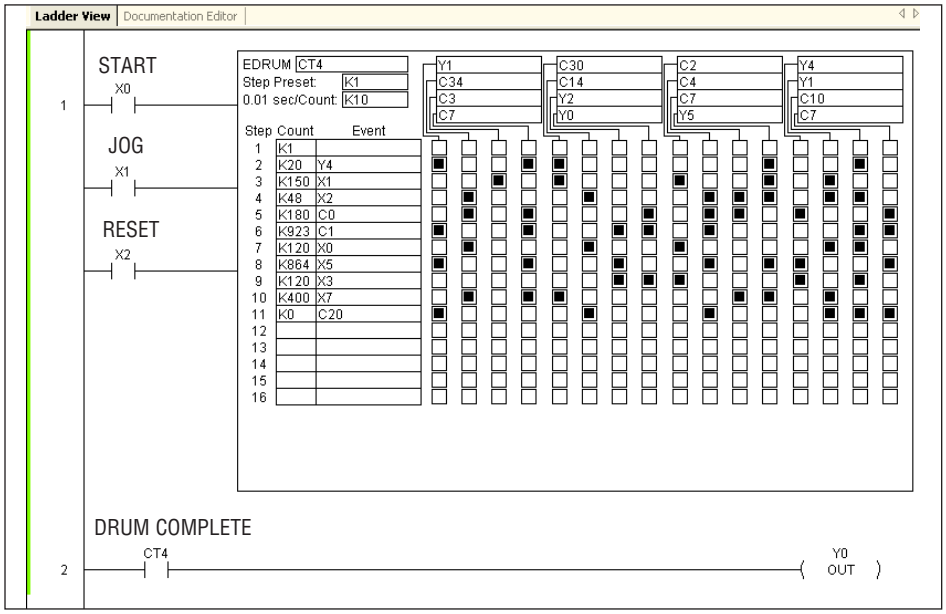
Drum Parameters	Field	Data Types	Ranges
Counter Number	aa	--	0 -- 174
Preset Step	bb	K	1 -- 16
Timer base	cccc	K	0 -- 99.99 seconds
Counts per step	dddd	K	0 -- 9999
Event	Eeeee	X, Y, C, S, T, CT, SP	see memory map
Discrete Outputs	ffff	X, Y, C	see memory map

Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 -- 174	Counts in step	CT(n)= Drum Complete
CTA(n+1)	1 -- 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 -- 176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 -- 177	Current Step	CT(n+3) = (not used)

The following ladder program shows the EDRUM instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all sixteen output points are used. The preset step is step 1. The timebase runs at (K10 x 0.01) = 0.1 second per count. Therefore, the duration of step 1 is (1 x 0.1) = 0.1 second. Note that step 1 is time-based only (event is left blank). And, the output pattern for step 1 programs all outputs off, which is a typically desirable powerup condition. In the last rung, the Drum Complete bit (CT4) turns on output Y0 upon completion of the last step (step 11). A drum reset also resets CT4.

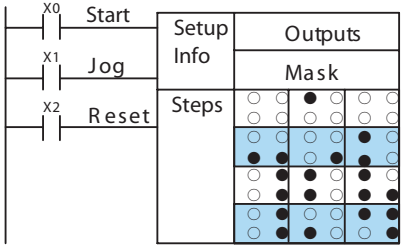


Handheld Programmer Drum Mnemonics

The EDRUM instruction can also be programmed using a handheld programmer. This section explains entry via the handheld programmer.

First, enter Store instructions for the ladder rungs controlling the drum's ladder inputs. In the example to the right, the timer drum's Start, Jog, and Reset inputs are controlled by X0, X1 and X2 respectively. The required keystrokes are listed beside the mnemonic.

These keystrokes precede the EDRUM instruction mnemonic. Note that the ladder rungs for Start, Jog and Reset inputs are not limited to being single–contact rungs.



Handheld Programmer Keystrokes

Store X0 [$ STR] [→] [A 0] [ENT]

(Repeat for Store X1 and Store X2)

Handheld Programmer Keystrokes

EDRUM CNT4 [SHFT] [E 4] [D 3] [R ORN] [U ISG] [M ORST] [→] [E 4] [ENT]

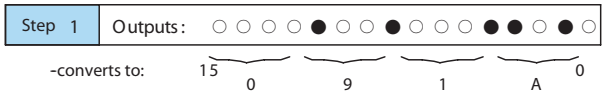
After the Store instructions, enter the EDRUM (using Counter CT0) as shown:

After entering the EDRUM mnemonic as above, the handheld programmer creates an input form for all the drum parameters. The input form consists of approximately fifty or more default mnemonic entries containing DEF (define) statements. The default mnemonics are already input for you, so they appear automatically. Use the NXT and PREV keys to move forward and backward through the form. Only the editing of default values is required, thus eliminating many keystrokes. The entries required for the basic timer drum are in the chart below.

***NOTE**: Default entries for output points and events are "**DEF 0000**", which means they are unassigned. If you need to go back and change an assigned output as unused again, enter "**K0000**". The entry will again show as "**DEF 0000**".*

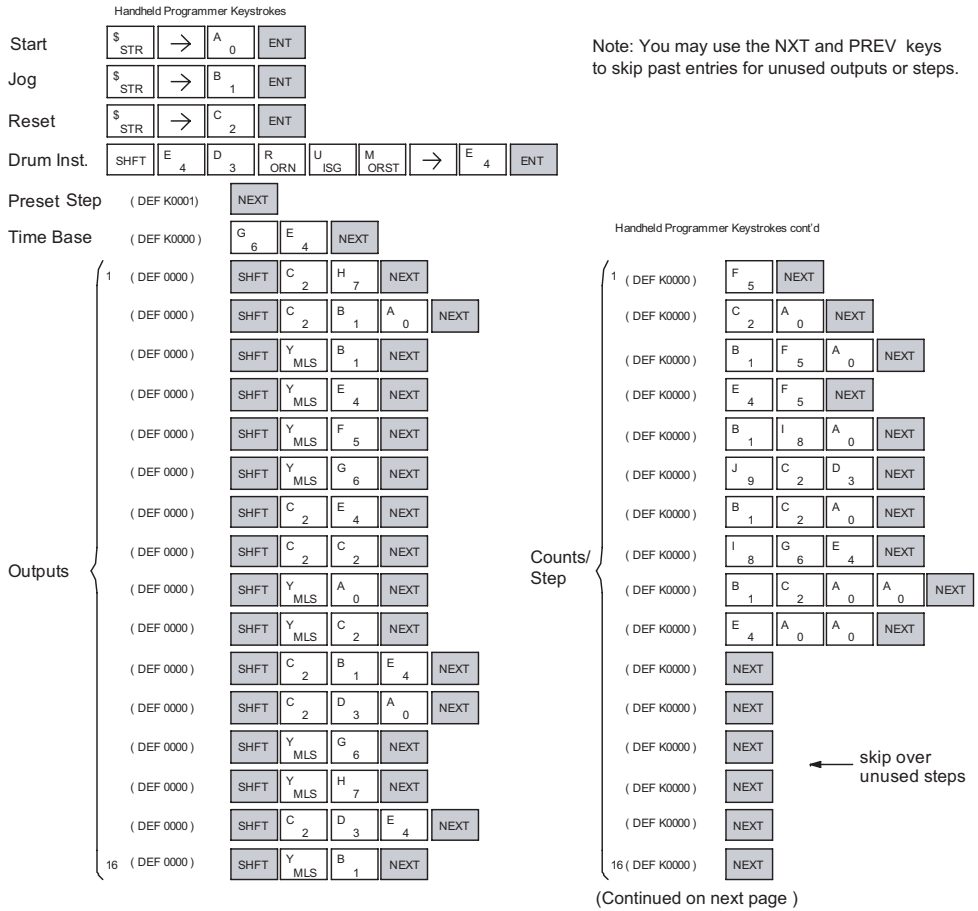
Drum Parameters	Multiple Entries	Mnemonic / Entry	Default Mnemonic	Valid Data Types	Ranges
Start Input	--	STR (plus input rung)	--	--	--
Jog Input	--	STR (plus input rung)	--	--	--
Reset Input	--	STR (plus input rung)	--	--	--
Drum Mnemonic	--	DRUM CNT aa	--	CT	0 -- 174
Preset Step	1	bb	DEF K0000	K	1 -- 16
Timer base	1	cccc	DEF K0000	K	1 -- 9999
Output points	16	ffff	DEF 0000	X, Y, C	see memory map
Counts per step	16	dddd	DEF K0000	K	0 -- 9999
Events	16	dddd	DEF K0000	X, Y, C, S, T, CT, SP	see memory map
Output pattern	16	gggg	DEF K0000	K	0 -- FFFF

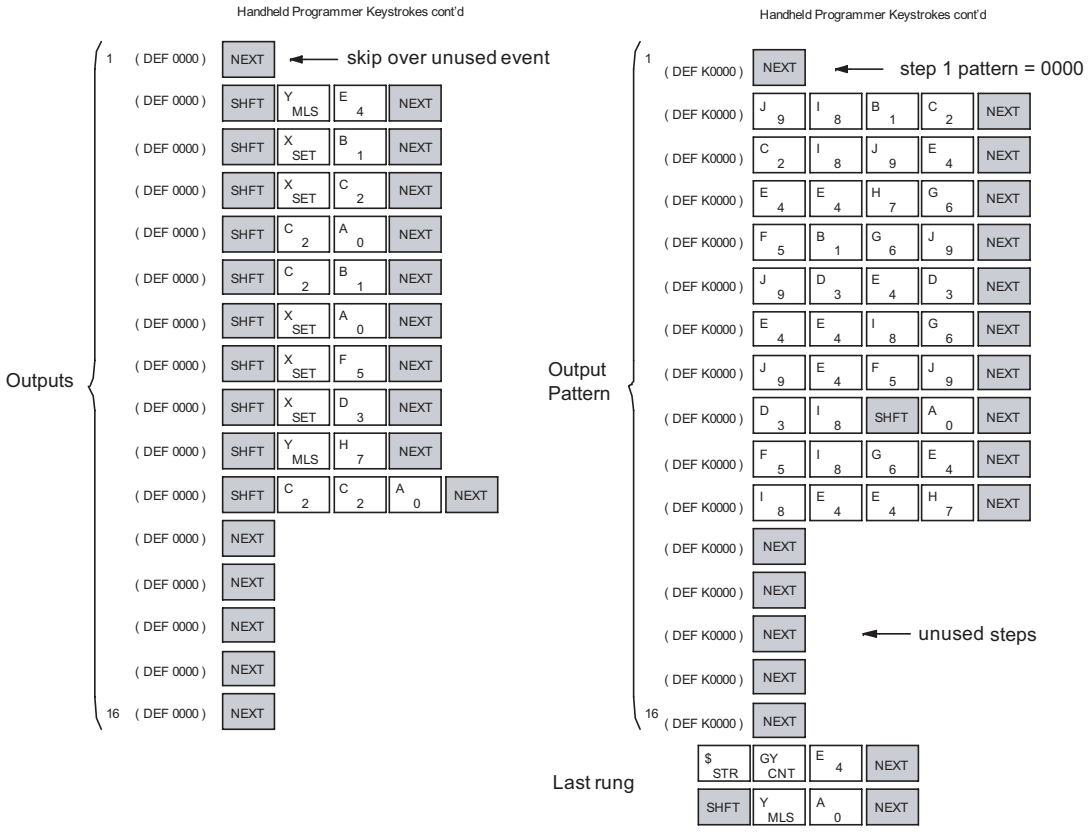
Using the DRUM entry chart (two pages before), we show the method of entry for the basic time/event drum instruction. First, we convert the output pattern for each step to the equivalent hex number, as shown in the following example.



The following diagram shows the method for entering the previous EDRUM example on the HHP. The default entries of the form are in parenthesis. After the drum instruction entry (on the fourth row), the remaining keystrokes over-write the numeric portion of each default DEF statement. **NOTE**: Drum editing requires Handheld Programmer firmware version 2.21 or later.

***NOTE**: You may use the NXT and PREV keys to skip past entries for unused outputs or steps.*



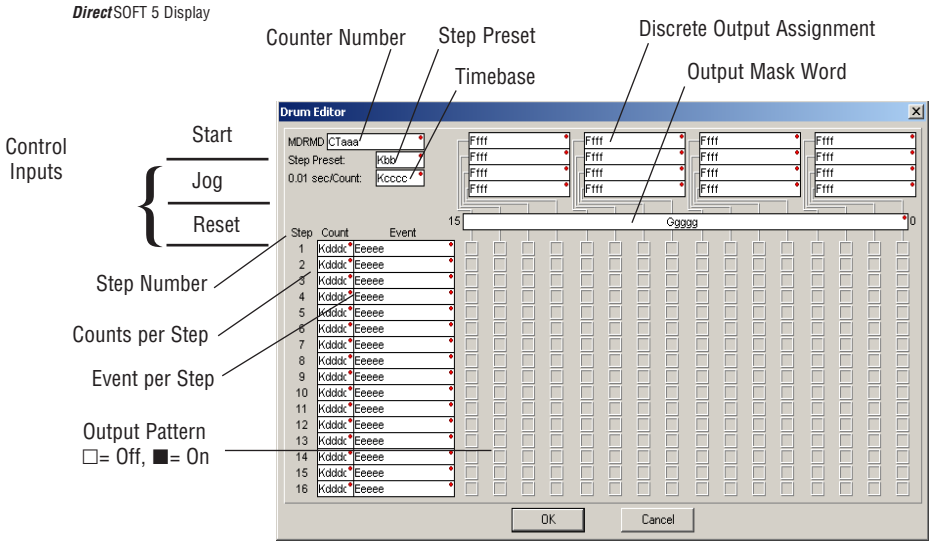


***NOTE**: You may use the NXT and PREV keys to skip past entries for unused outputs or steps.*

***NOTE:** For ease of operation when using the EDRUM instruction, we recommend using **Direct**SOFT over the handheld programmer.*

Masked Event Drum with Discrete Outputs (MDRMD)

The Masked Event Drum with Discrete Outputs has all the features of the basic Event Drum plus final output control for each step. It operates according to the general principles of drum operation covered in the beginning of this section. Below is the instruction in chart form as displayed by *Direct*SOFT.



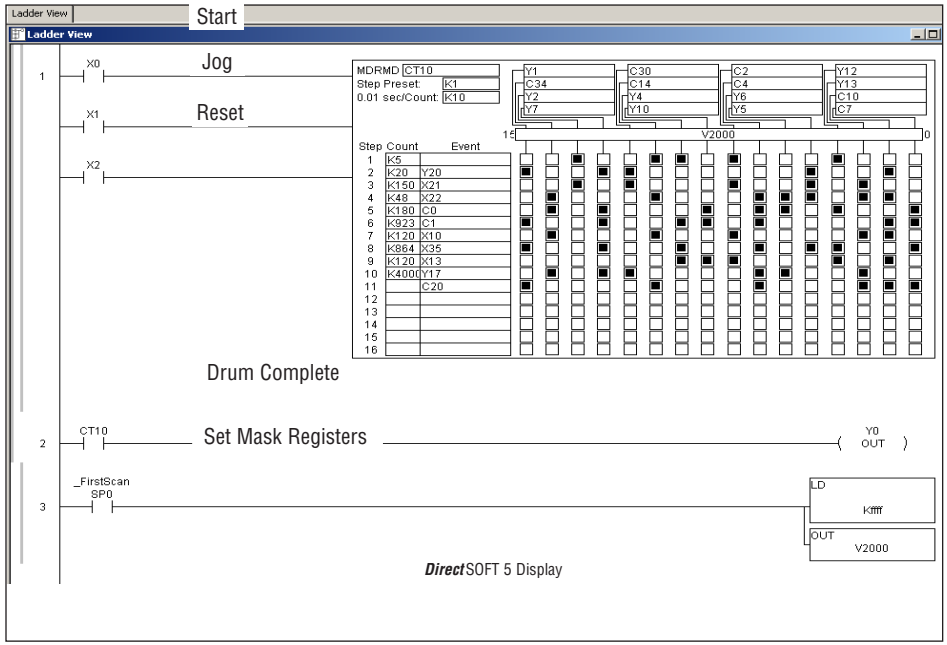
The Masked Event Drum with Discrete Outputs features sixteen steps and sixteen outputs. Drum outputs are logically ANDed bit-by-bit with an output mask word for each step. The Ggggg field specifies the beginning location of the 16 mask words. Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps and events can be left blank (this is the default entry). Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	–	0 – 174
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Event	eeee	X, Y, C, S, T, ST, GX, GY. CT, SP	see memory map
Discrete Outputs	Fffff	X, Y, C, GX, GY	
Output Mask	Ggggg	V	

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 – 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 – 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 –176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 –177	Current Step	CT(n+1) = (not used)

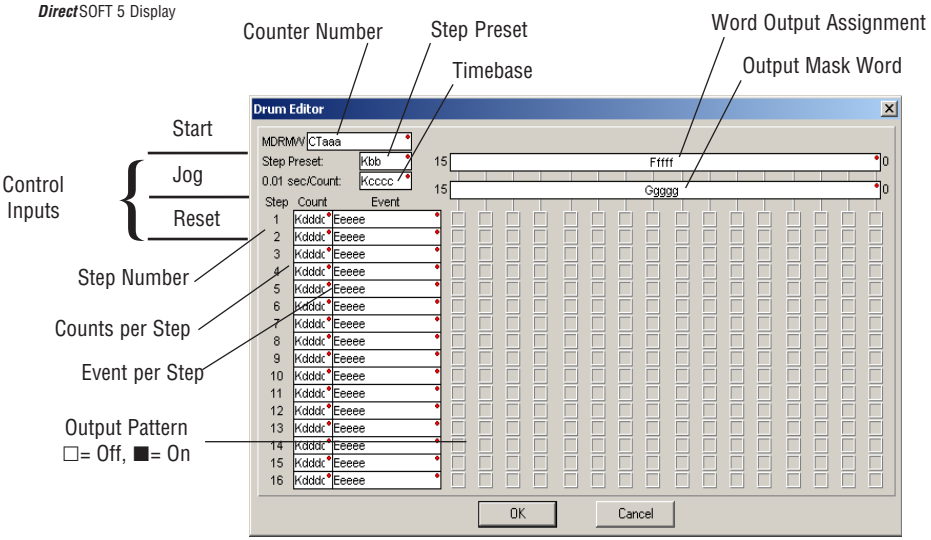
The following ladder program shows the MDRMD instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all 16 output points are used. The output mask word is at V2000. The final drum outputs are shown above the mask word as individual bits. The data bits in V2000 are logically ANDed with the output pattern of the current step in the drum. If you want all drum outputs to be off after powerup, write zeros to V2000 on the first scan. Ladder logic may update the output mask at any time to enable or disable the drum outputs The preset step is step 1. The timebase runs at (K10 x 0.01)=0.1 second per count. Therefore, the duration of step 1 is (5 x 0.1) = 0.5 seconds. Note that step 1 is time-based only (event is left blank). In the last rung, the Drum Complete bit (CT10) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT10.



***NOTE**: The ladder program must load constants in V2000 through V2012 to cover all mask registers for the eleven steps used in this drum.*

Masked Event Drum with Word Output (MDRMW)

The Masked Event Drum with Word Output features outputs organized as bits of a single word, rather than discrete points. It operates according to the general principles of drum operation covered in the beginning of this section. Below is the instruction in chart form as displayed by *Direct*SOFT.



The Masked Event Drum with Word Output features sixteen steps and sixteen outputs. Drum outputs are logically ANDed bit-by-bit with an output mask word for each step. The Ggggg field specifies the beginning location of the 16 mask words, creating the final output (Fffff field). Step transitions occur on timed and/or event basis. The jog input also advances the step on each off-to-on transition. Time is specified in counts per step, and events are specified as discrete contacts. Unused steps and events can be left blank (this is the default entry).

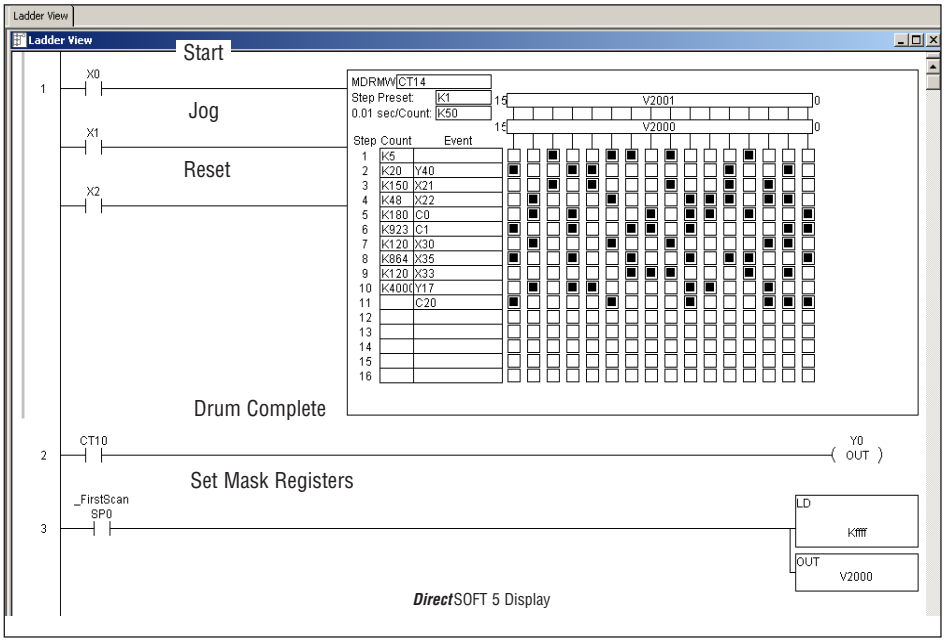
Whenever the Start input is energized, the drum's timer is enabled. As long as the event is true for the current step, the timer runs during that step. When the step count equals the counts per step, the drum transitions to the next step. This process stops when the last step is complete, or when the Reset input is energized. The drum enters the preset step chosen upon a CPU program-to-run mode transition, and whenever the Reset input is energized.

Drum Parameters	Field	Data Types	Ranges
Counter Number	aaa	–	0 – 174
Preset Step	bb	K	1 – 16
Timer base	cccc	K	0 – 99.99 seconds
Counts per step	dddd	K	0 – 9999
Event	eeee	X, Y, C, S, T, ST, GX, GY, SP	see memory map
Word Output	Fffff	V	see memory map
Output Mask	Ggggg	V	see memory map

Drum instructions use four counters in the CPU. The ladder program can read the counter values for the drum's status. The ladder program may write a new preset step number to CTA(n+2) at any time. However, the other counters are for monitoring purposes only.

Counter Number	Ranges of (n)	Function	Counter Bit Function
CTA(n)	0 – 174	Counts in step	CT(n) = Drum Complete
CTA(n+1)	1 – 175	Timer value	CT(n+1) = (not used)
CTA(n+2)	2 –176	Preset Step	CT(n+2) = (not used)
CTA(n+3)	3 –177	Current Step	CT(n+1) = (not used)

The following ladder program shows the MDRMD instruction in a typical ladder program, as shown by ***Direct***SOFT. Steps 1 through 11 are used, and all sixteen output points are used. The output mask word is at V2000. The final drum outputs are shown above the mask word as a word at V2001. The data bits in V2000 are logically ANDed with the output pattern of the current step in the drum, generating the contents of V2001. If you want all drum outputs to be off after powerup, write zeros to V2000 on the first scan. Ladder logic may update the output mask at any time to enable or disable the drum outputs. The preset step is step 1. The timebase runs at (K50 x 0.01)=0.5 seconds per count. Therefore, the duration of step 1 is (5 x 0.5) = 2.5 seconds. Note that step 1 is time-based only (event is left blank). In the last rung, the Drum Complete bit (CT14) turns on output Y0 upon completion of the last step (step 10). A drum reset also resets CT14.



NOTE: *The ladder program must load constants in V2000 through V2012 to cover all mask registers for the eleven steps used in this drum.*

RLL^{PLUS} Stage Programming

In This Chapter...

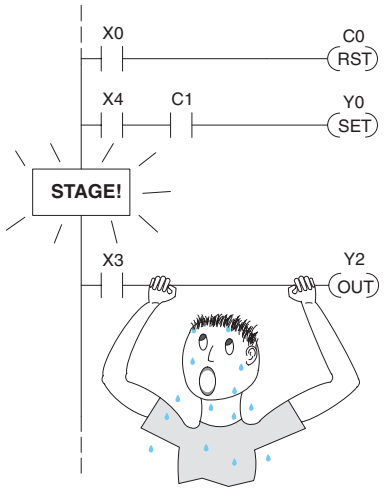
Introduction to Stage Programming

Stage Programming provides a way to organize and program complex applications with relative ease, when compared to purely relay ladder logic (RLL) solutions. Stage programming does not replace or negate the use of traditional boolean ladder programming. This is why Stage Programming is also called RLL^{PLUS} You won't have to discard any training or experience you already have. Stage programming simply allows you to divide and organize an RLL program into groups of ladder instructions called stages. This allows quicker and more intuitive ladder program development than traditional RLL alone provides.

Overcoming "Stage Fright"

Many PLC programmers in the industry have become comfortable using RLL for every PLC program they write, but often remain skeptical or even fearful of learning new techniques such as stage programming. While RLL is great at solving boolean logic relationships, it has disadvantages as well:

- Large programs can become almost unmanageable, because of a lack of structure.
- When a process gets stuck, it is difficult to find the rung where the error occurred.
- Programs become difficult to modify later, because they do not intuitively resemble the application problem they are solving.



It's easy to see that these inefficiencies consume a lot of additional time, and time is money. *Stage programming overcomes these obstacles!* We believe a few moments of studying the stage concept is one of the greatest investments in programming speed and efficiency a PLC programmer can make!

So, we encourage you to study stage programming and add it to your toolbox of programming techniques. This chapter is designed as a self-paced tutorial on stage programming. For best results:

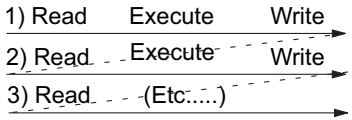
- Start at the beginning and do not skip over any sections.
- Study each stage programming concept by working through each example. The examples build progressively on each other.
- Read the Stage Questions and Answers at the end of the chapter for a quick review.

Learning to Draw State Transition Diagrams

Introduction to Process States

Those familiar with ladder program execution know that the CPU must scan the ladder program repeatedly, over and over. Its three basic steps are:

1. Read the inputs
2. Execute the ladder program
3. Write the outputs



The benefit is that a change at the inputs can affect the outputs in just a few milliseconds.

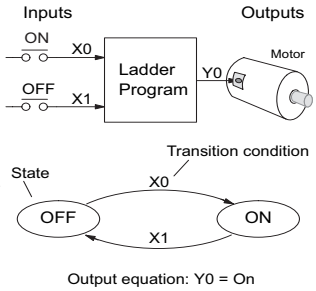
Most manufacturing processes consist of a series of activities or conditions, each lasting for several seconds, minutes, or even hours. We might call these *process states*, which are either active or inactive at any particular time. A challenge for RLL programs is that a particular input event may last for just a brief instant. We typically create latching relays in RLL to preserve the input event in order to maintain a process state for the required duration.

We can organize and divide ladder logic into sections called **stages**, representing process states. But before we describe stages in detail, we will reveal **the secret to understanding stage programming:** state transition diagrams.

The Need for State Diagrams

Sometimes we need to forget about the scan nature of PLCs, and focus our thinking toward the states of the process we need to identify. Clear thinking and concise analysis of an application gives us the best chance at writing efficient, bug-free programs. *State diagrams are just a tool to help us draw a picture of our process!* You'll discover that if we can get the picture right, **our program will also be right!**

A 2–State Process



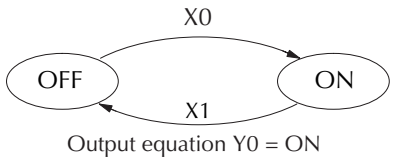
Consider the simple process shown to the right, which controls an industrial motor. We will use a green momentary SPST pushbutton to turn the motor on, and a red one to turn it off. The machine operator will press the appropriate pushbutton for just a second or so. The two states of our process are ON and OFF.

The next step is to draw a state transition diagram, as shown to the right. It shows the two states OFF and ON, with two transition lines in-between. When the event X0 is true, we transition from OFF to ON. When X1 is true, we transition from ON to OFF.

If you're following along, you are very close to grasping the concept and the problem-solving power of state transition diagrams. The output of our controller is Y0, which is true any time we are in the ON state. In a boolean sense, Y0=ON state.

Next, we will implement the state diagram first as RLL, then as a stage program. This will help you see the relationship between the two methods in problem solving.

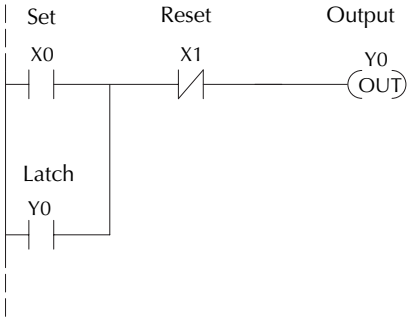
The state transition diagram to the right is a picture of the solution we need to create. The beauty of it is this: it expresses the problem independently of the programming language we may use to realize it. In other words, *by drawing the diagram we have already solved the control problem!*



First, we'll translate the state diagram to traditional RLL. Then, we'll show how easy it is to translate the diagram into a stage programming solution.

RLL Equivalent

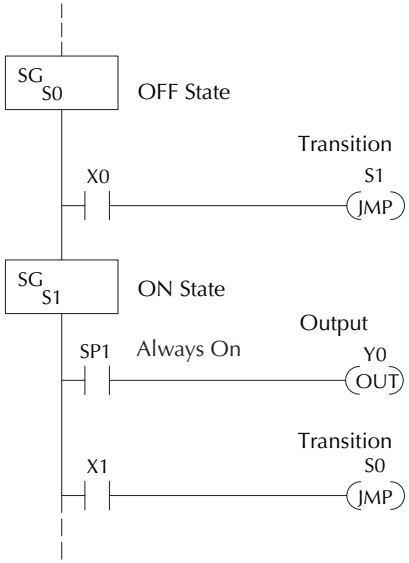
The RLL solution is shown to the right. Output control relay, Y0, has a dual purpose. It turns the motor on and off and acts as a latching relay. When the On pushbutton (X0) is pressed, output coil Y0 turns on and the Y0 contact on the second row latches itself on. So, X0 turns on the motor output Y0 which now has power flow and **sets the latch** Y0. It will remain on after the X0 contact opens.



When the Off pushbutton (X1) is pressed, it opens the normally-closed X1 contact, which turns off moter output Y0 and also **resets the latch**.

Stage Equivalent

The stage program solution is shown to the right. The two inline stage boxes S0 and S1 correspond to the two states OFF and ON. The ladder rung(s) below each stage box belong to each respective stage. This means that the PLC only has to scan those rungs when the corresponding stage is active!



For now, let's assume we begin in the OFF State, so stage S0 is active. When the On pushbutton (X0) is pressed, a stage transition occurs. The JMP S1 instruction executes, which simply turns off the Stage bit S0 and turns on Stage bit S1. So on the next PLC scan, the CPU will not execute Stage S0, but will execute stage S1!

In the On State (Stage S1), we want the motor to always be on. The special relay contact SP1 is defined as always on, so Y0 turns the motor on.

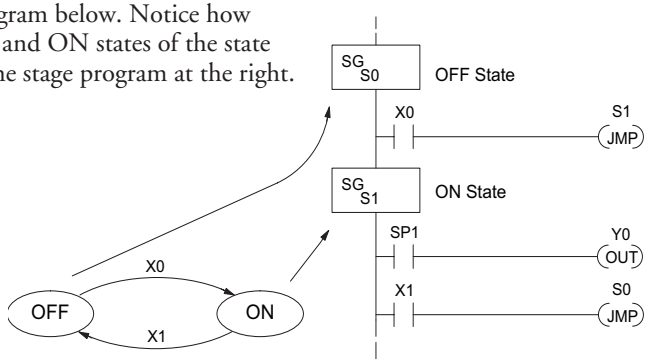
When the Off pushbutton (X1) is pressed, a transition back to the Off State occurs. The JMP S0 instruction executes, which simply turns off the Stage bit S1 and turns on Stage bit S0. On the next PLC scan, the CPU will not execute Stage S1, so the motor output Y0 will turn off. The Off state (Stage 0) will be ready for the next cycle.

Let's Compare

Right now, you may be thinking, "I don't see the big advantage to Stage Programming ... in fact, the stage program is longer than the plain RLL program." Well, now is the time to exercise a bit of faith. As control problems grow in complexity, stage programming quickly out-performs RLL in simplicity, program size, etc.

For example, consider the diagram below. Notice how easy it is to correlate the OFF and ON states of the state transition diagram below to the stage program at the right.

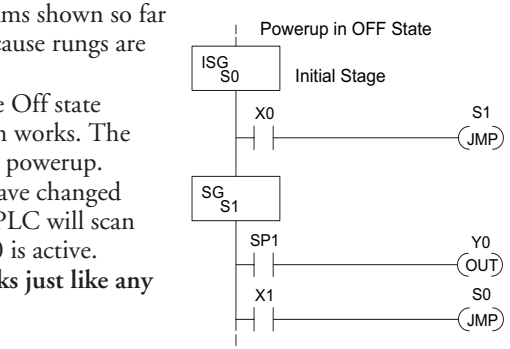
Now, we challenge anyone to easily identify the same states in the RLL program on the previous page!



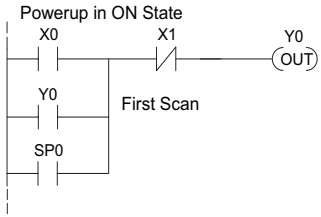
Initial Stages

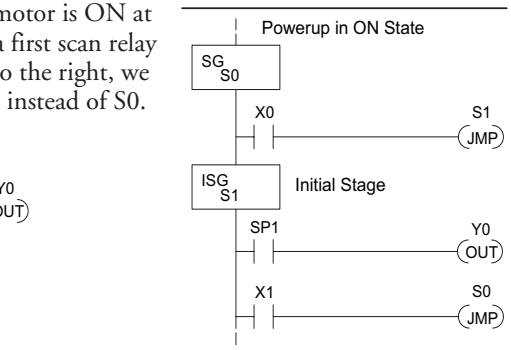
At powerup and Program-to-Run Mode transitions, the PLC always begins with all normal stages (SG) off. So, the stage programs shown so far have actually had no way to get started (because rungs are not scanned unless their stage is active).

Assume that we want to always begin in the Off state (motor off), which is how the RLL program works. The Initial Stage (ISG) is defined to be active at powerup. In the modified program to the right, we have changed stage S0 to the ISG type. This ensures the PLC will scan contact X0 after powerup, because Stage S0 is active. **After powerup, an Initial Stage (ISG) works just like any other stage!**



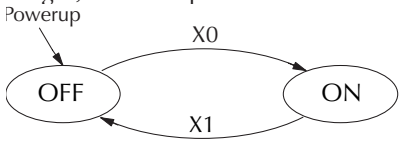
We can change both programs so that the motor is ON at powerup. In the RLL below, we must add a first scan relay SP0, latching Y0 on. In the stage example to the right, we simply make Stage S1 an initial stage (ISG) instead of S0.





NOTE: *If the ISG is within the retentive range for stages, the ISG will remain in the state it was in before power down and will NOT turn itself on during the first scan.*

We can mark our desired powerup state as shown to the right, which helps us remember to use the appropriate Initial Stages when creating a stage program. It is permissible to have as many initial stages as the process requires.

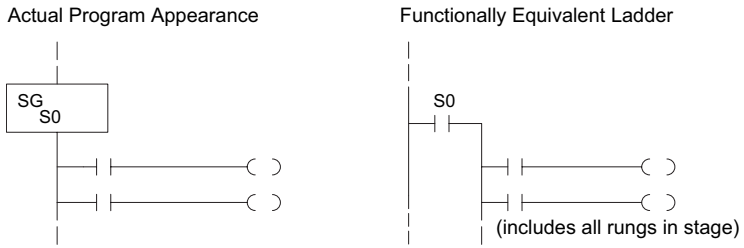


What Stage Bits Do

You may recall that a stage is just a section of ladder program which is either active or inactive at a given moment. All stage bits (S0 to 1777) reside in the PLC's image register as individual status bits. Each stage bit is either a boolean 0 or 1 at any time.

Program execution always reads ladder rungs from top to bottom, and from left to right. The drawing below shows the effect of stage bit status. The ladder rungs below the stage instruction continuing until the next stage instruction or the end of program belong to stage 0. Its equivalent operation is shown on the right. When S0 is true, the two rungs have power flow.

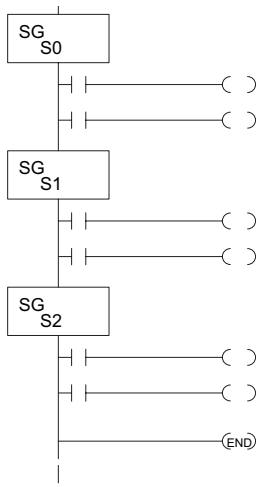
- If Stage bit S0 = 0, its ladder rungs *are not scanned* (executed).
- If Stage bit S0 = 1, its ladder rungs *are scanned* (executed).



Stage Instruction Characteristics

The inline stage boxes on the left power rail divide the ladder program rungs into stages. Some stage rules are:

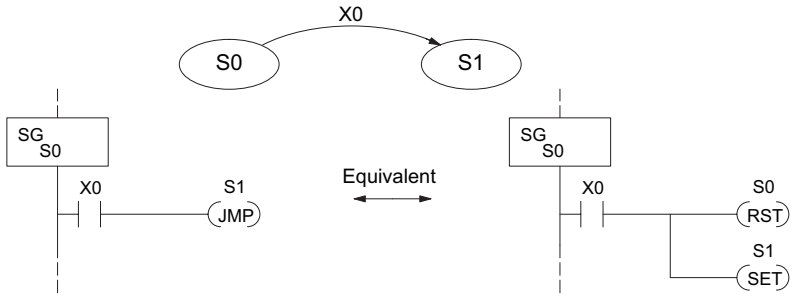
- **Execution** – Only logic in active stages are executed on any scan.
- **Transitions** – Stage transition instructions take effect on the next occurrence of the stages involved.
- **Octal numbering** – Stages are numbered in octal, like I/O points, etc. So "S8" is not valid.
- **Total Stages** – The DL06 offers up to 1024 stages (S0 to 1777 in octal).
- **No duplicates** –Each stage number is unique and can be used just once.
- **Any order** – You can skip numbers and sequence the stage numbers in any order.
- **Last Stage** – The last stage in the ladder program includes all rungs from its stage box until the end coil.



Using the Stage Jump Instruction for State Transitions

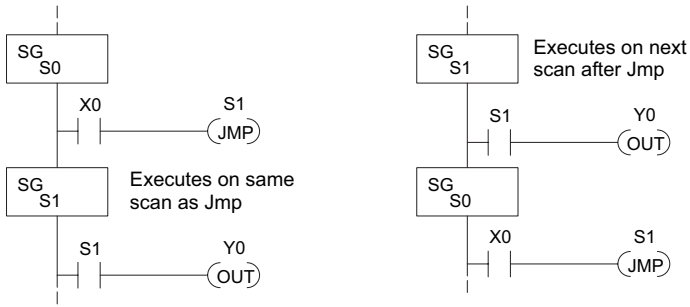
Stage Jump, Set, and Reset Instructions

The Stage JMP instruction we have used deactivates the stage in which the instruction occurs, while activating the stage in the JMP instruction. Refer to the state transition shown below. When contact X0 energizes, the state transition from S0 to S1 occurs. The two stage examples shown below are equivalent. So, the Stage Jump instruction is equal to a Stage Reset of the current stage, plus a Stage Set instruction for the stage to which we want to transition.



Please Read Carefully – The jump instruction is easily misunderstood. The "jump" does not occur immediately like a GOTO or GOSUB program control instruction when executed. Here's how it works:

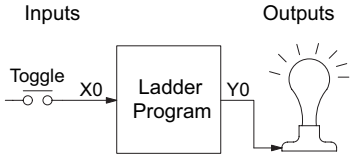
- The jump instruction resets the stage bit of the stage in which it occurs. All rungs in the stage still finish executing during the current scan, *even if there are other rungs in the stage below the jump instruction!*
- The reset will be in effect on the following scan, so the stage that executed the jump instruction previously will be inactive and bypassed.
- The stage bit of the stage named in the Jump instruction will be set immediately, so the stage will be executed on its next occurrence. In the left program shown below, stage S1 executes during the same scan as the JMP S1 occurs in S0. In the example on the right, Stage S1 executes on the next scan after the JMP S1 executes, because stage S1 is located above stage S0.



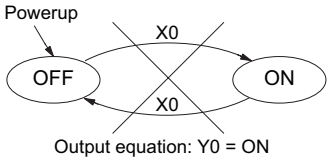
***NOTE**: Assume we start with Stage 0 active and Stage 1 inactive for both examples.*

Stage Program Example: Toggle On/Off Lamp Controller

A 4–State Process



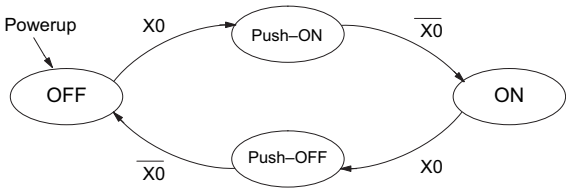
In the process shown to the right, we use an ordinary momentary pushbutton to control a light bulb. The ladder program will latch the switch input, so that we will push and release to turn on the light, push and release again to turn it off (sometimes called toggle function). Sure, we could just buy a mechanical switch with the alternate on/off action built in... However, this example is educational and also fun! Next we draw the state transition diagram.



A typical first approach is to use X0 for both transitions (like the example shown to the right). However, *this is incorrect* (please keep reading).

Note that this example differs from the motor example, because now we have just one pushbutton. When we press the pushbutton, both transition conditions are met. We would just transition around the state diagram at top speed. If implemented in Stage, this solution would flash the light on or off each scan (obviously undesirable)!

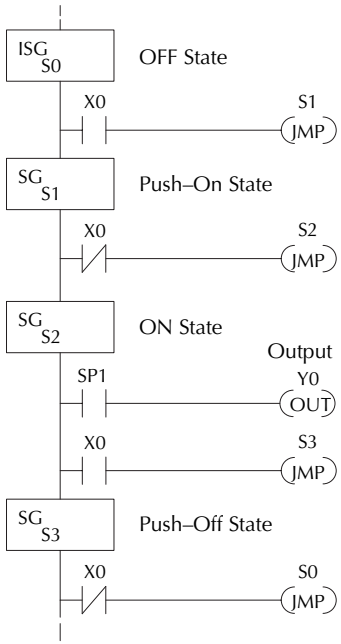
The solution is to make the push and the release of the pushbutton separate events. Refer to the new state transition diagram below. At powerup we enter the OFF state. When switch X0 is pressed, we enter the Press-ON state. When it is released, we enter the ON state. Note that X0 with the bar above it denotes X0 NOT.



When in the ON state, another push and release cycle similarly takes us back to the OFF state. Now we have two unique states (OFF and ON) used when the pushbutton is released, which is what was required to solve the control problem.

The equivalent stage program is shown to the right. The desired powerup state is OFF, so we make S0 an initial stage (ISG). In the ON state, we add special relay contact SP1, which is always on.

Note that even as our programs grow more complex, it is still easy to correlate the state transition diagram with the stage program!



Four Steps to Writing a Stage Program

By now, you've probably noticed that we follow the same steps to solve each example problem. The steps will probably come to you automatically if you work through all the examples in this chapter. It's helpful to have a checklist to guide us through the problem solving. The following steps summarize the stage program design procedure:

1. Write a Word Description of the application.

Describe all functions of the process in your own words. Start by listing what happens first, then next, etc. If you find there are too many things happening at once, try dividing the problem into more than one process. Remember, you can still have the processes communicate with each other to coordinate their overall activity.

2. Draw the Block Diagram.

Inputs represent all the information the process needs for decisions, and outputs connect to all devices controlled by the process.

- Make lists of inputs and outputs for the process.
- Assign I/O point numbers (X and Y) to physical inputs and outputs.

3. Draw the State Transition Diagram.

The state transition diagram describes the central function of the block diagram, reading inputs and generating outputs.

- Identify and name the states of the process.
- Identify the event(s) required for each transition between states.
- Ensure the process has a way to re-start itself, or is cyclical.
- Choose the powerup state for your process.
- Write the output equations.

4. Write the Stage Program.

Translate the state transition diagram into a stage program.

- Make each state a stage. Remember to number stages in octal. Up to 1024 total stages are available in the DL06, numbered 0 to 1777 in octal.
- Put transition logic inside the stage which originates each transition (the stage each arrow points away from).
- Use an initial stage (ISG) for any states that must be active at powerup.
- Place the outputs or actions in the appropriate stages.

You'll notice that Steps 1 through 3 just prepare us to write the stage program in Step 4. However, the program virtually writes itself because of the preparation beforehand. Soon you'll be able to start with a word description of an application and create a stage program in one easy session!

Stage Program Example: A Garage Door Opener

Garage Door Opener Example

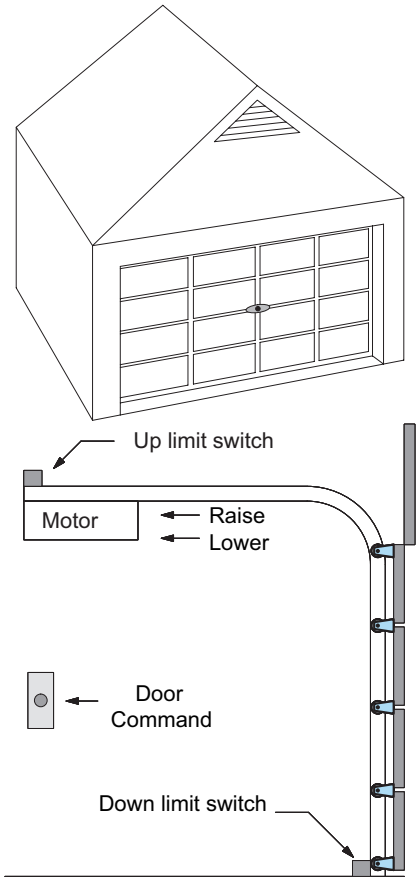
In this next stage programming example, we'll create a garage door opener controller. Hopefully, most readers are familiar with this application, and we can have fun, too!

The first step we must take is to describe how the door opener works. We will start by achieving the basic operation, waiting to add extra features later. Stage programs are very easy to modify.

Our garage door controller has a motor which raises or lowers the door on command. The garage owner pushes and releases a momentary pushbutton once to raise the door. After the door is up, another push-release cycle will lower the door.

In order to identify the inputs and outputs of the system, it's sometimes helpful to sketch its main components, as shown in the door side view to the right. The door has an up limit and a down limit switch. Each limit switch closes only when the door has reach the end of travel in the corresponding direction. In the middle of travel, neither limit switch is closed.

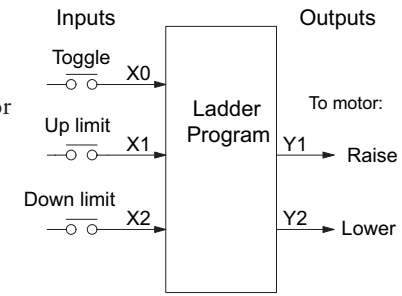
The motor has two command inputs: raise and lower. When neither input is active, the motor is stopped. The door command is just a simple pushbutton. Whether wall-mounted as shown, or a radio-remote control, all door control commands logical OR together as one pair of switch contacts.



Draw the Block Diagram

The block diagram of the controller is shown to the right. Input X0 is from the pushbutton door control. Input X1 energizes when the door reaches the full up position. Input X2 energizes when the door reaches the full down position. When the door is positioned between fully up or down, both limit switches are open.

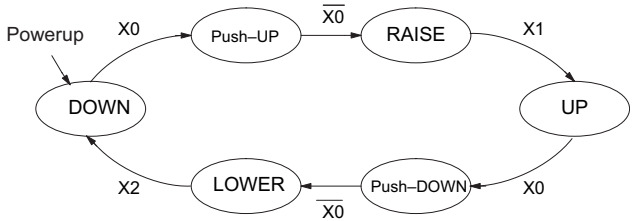
The controller has two outputs to drive the motor. Y1 is the up (raise the door) command, and Y2 is the down (lower the door) command.



Draw the State Diagram

Now we are ready to draw the state transition diagram. Like the previous light bulb controller example, this application also has just one switch for the command input. Refer to the figure below.

- When the door is down (DOWN state), nothing happens until X0 energizes. Its push and release brings us to the RAISE state, where output Y1 turns on and causes the motor to raise the door.
- We transition to the UP state when the up limit switch (X1) energizes, and turns off the motor.
- Then nothing happens until another X0 press-release cycle occurs. That takes us to the LOWER state, turning on output Y2 to command the motor to lower the door. We transition back to the DOWN state when the down limit switch (X2) energizes.

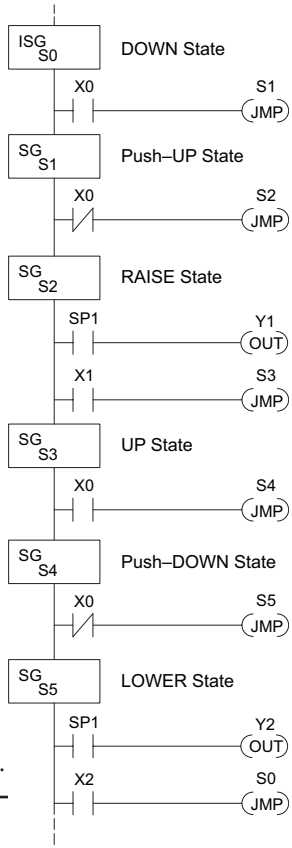


Output equations: Y1 = Raise Y2 = Lower

The equivalent stage program is shown to the right. For now, we will assume the door is down at powerup, so the desired powerup state is DOWN. We make S0 an initial stage (ISG). Stage S0 remains active until the door control pushbutton activates. Then we transition (JMP) to Push-UP stage, S1.

A push-release cycle of the pushbutton takes us through stage S1 to the RAISE stage, S2. We use the always-on contact SP1 to energize the motor's raise command, Y1. When the door reaches the fully-raised position, the up limit switch X1 activates. This takes us to the UP Stage S3, where we wait until another door control command occurs.

In the UP Stage S3, a push-release cycle of the pushbutton will take us to the LOWER Stage S5, where we activate Y2 to command the motor to lower the door. This continues until the door reaches the down limit switch, X2. When X2 closes, we transition from Stage S5 to the DOWN stage S0, where we began.

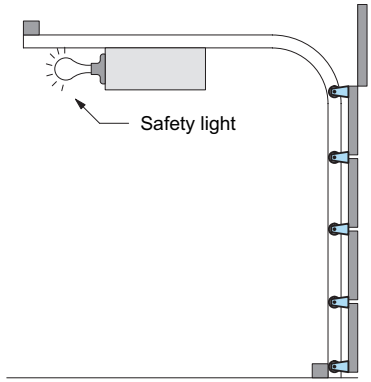


***NOTE**: The only special thing about an initial stage (ISG) is that it is automatically active at powerup. Afterwards, it is just like any other.*

Add Safety Light Feature

Next we will add a safety light feature to the door opener system. It's best to get the main function working first as we have done, then adding the secondary features.

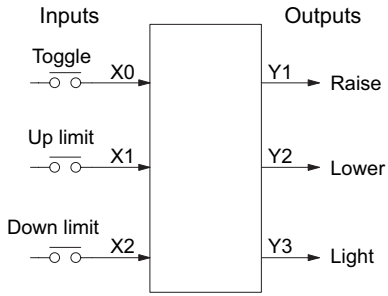
The safety light is standard on many commercially-available garage door openers. It is shown to the right, mounted on the motor housing. The light turns on upon any door activity, remaining on for approximately 3 minutes afterwards.



This part of the exercise will demonstrate the use of parallel states in our state diagram. Instead of using the JMP instruction, we'll use the set and reset commands.

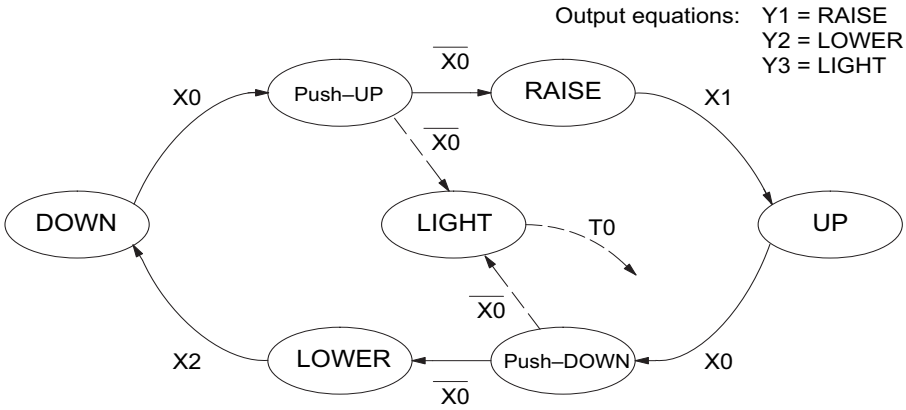
Modify the Block Diagram and State Diagram

To control the light bulb, we add an output to our controller block diagram, shown to the right, Y3 is the light control output.



In the diagram below, we add an additional state called "LIGHT". Whenever the garage owner presses the door control switch and releases, the RAISE or LOWER state is active *and the LIGHT state is simultaneously active*. The line to the Light state is dashed, because it is not the primary path.

We can think of the Light state as a parallel process to the raise and lower state. The paths to the Light state are not a transition (Stage JMP), but a State Set command. In the logic of the Light stage, we will place a three-minute timer. When it expires, timer bit T0 turns on and resets the Light stage. The path out of the Light stage goes nowhere, indicating the Light stage just becomes inactive, and the light goes out!



Using a Timer Inside a Stage

The finished modified program is shown to the right. The shaded areas indicate the program additions.

In the Push-UP stage S1, we add the Set Stage Bit S6 instruction. When contact X0 closes, we transition from S1 and go to two new active states: S2 and S6. In the Push-DOWN state S4, we make the same additions. So, any time someone presses the door control pushbutton, the light turns on.

Most new stage programmers would be concerned about where to place the Light Stage in the ladder, and how to number it. The good news is that it doesn't matter!

- Just choose an unused Stage number, and use it for the new stage and as the reference from other stages.
- Placement in the program is not critical, so we place it at the end

You might think that each stage has to be directly under the stage that transitions to it. While it is good practice, it is not required (that's good, because our two locations for the Set S6 instruction make that impossible). Stage numbers and how they are used determines the transition paths.

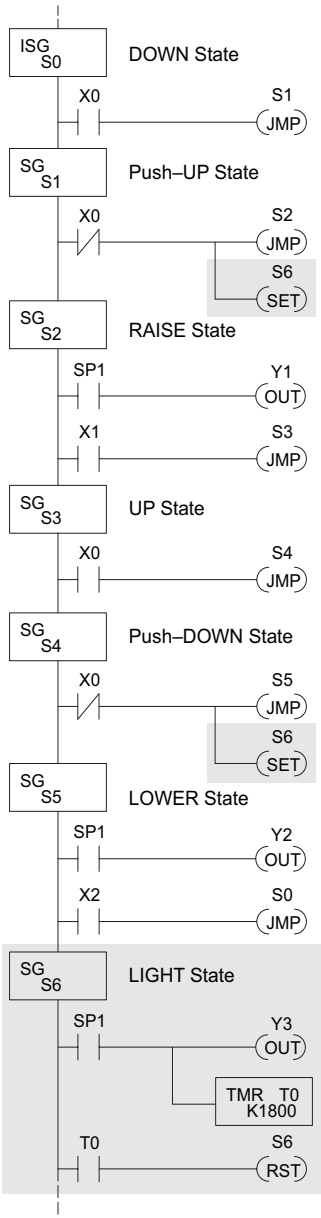
In stage S6, we turn on the safety light by energizing Y3. Special relay contact SP1 is always on. Timer T0 times at 0.1 second per count. To achieve 3 minutes time period, we calculate:

$$K = \frac{3 \text{ min.} \times 60 \text{ sec/min}}{0.1 \text{ sec/count}}$$

$$K = 1800 \text{ counts}$$

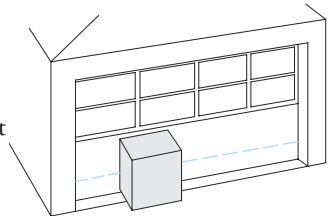
The timer has power flow whenever stage S6 is active. The corresponding timer bit T0 is set when the timer expires. So three minutes later, T0=1 and the instruction Reset S6 causes the stage to be inactive.

While Stage S6 is active and the light is on, stage transitions in the primary path continue normally and independently of Stage 6. That is, the door can go up, down, or whatever, but the light will be on for precisely 3 minutes.

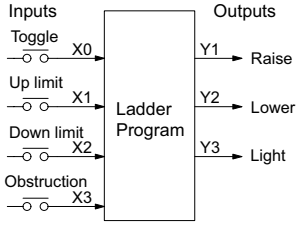


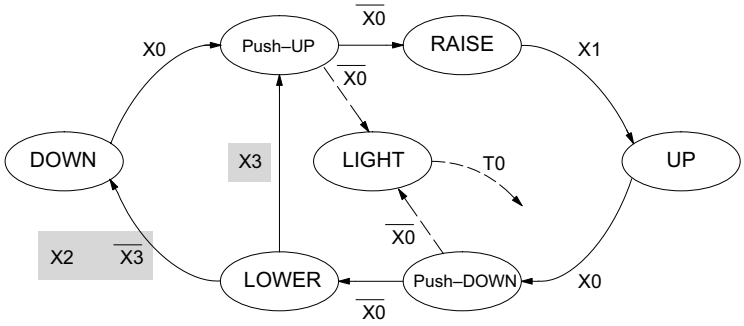
Add Emergency Stop Feature

Some garage door openers today will detect an object under the door. This halts further lowering of the door. Usually implemented with a photocell (electric-eye), a door in the process of being lowered will halt and begin raising. We will define our safety feature to work in this way, adding the input from the photocell to the block diagram as shown to the right. X3 will be on if an object is in the path of the door.



Next, we make a simple addition to the state transition diagram, shown in shaded areas in the figure below. Note the new transition path at the top of the LOWER state. If we are lowering the door and detect an obstruction (X3), we then jump to the Push-UP State. We do this instead of jumping directly to the RAISE state, to give the Lower output Y2 one scan to turn off, before the Raise output Y1 energizes.

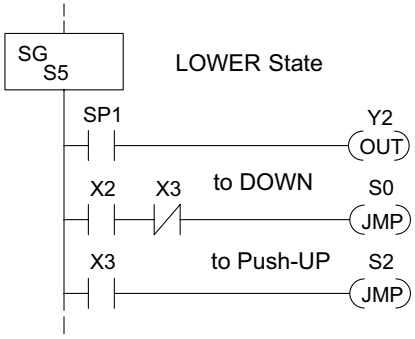




Exclusive Transitions

It is theoretically possible that the down limit (X2) and the obstruction input (X3) could energize at the same moment. In that case, we would **jump** to the Push-UP and DOWN states simultaneously, which does not make sense.

Instead, we give priority to the obstruction by changing the transition condition to the DOWN state to [X2 AND NOT X3]. This ensures the obstruction event has the priority. The modifications we must make to the LOWER Stage (S5) logic are shown to the right. The first rung remains unchanged. The second and third rungs implement the transitions we need. Note the opposite relay contact usage for X3, which ensures the stage will execute only one of the JMP instructions.

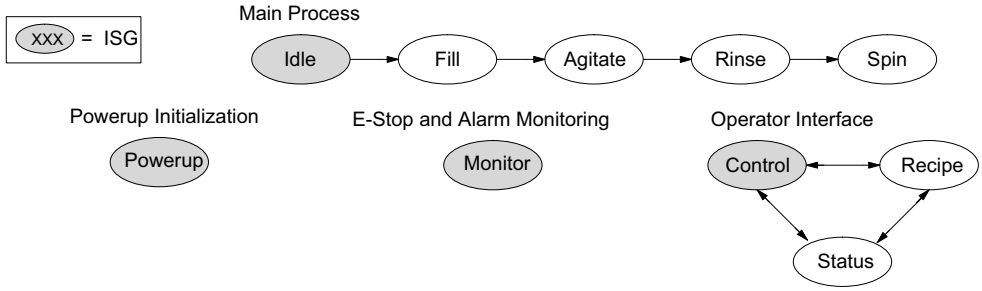


Stage Program Design Considerations

Stage Program Organization

The examples so far in this chapter used one self-contained state diagram to represent the main process. However, we can have multiple processes implemented in stages, all in the same ladder program. New stage programmers sometimes try to turn a stage on and off each scan, based on the false assumption that only one stage can be on at a time. For ladder rungs that you want to execute each scan, just put them in a stage that is always on.

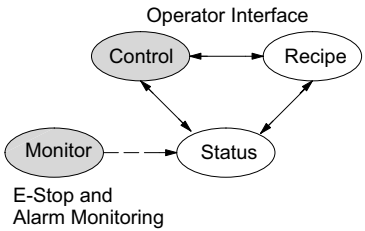
The following figure shows a typical application. During operation, the primary manufacturing activity Main Process, Powerup Initialization, E-Stop and Alarm Monitoring, and Operator Interface are all running. At powerup, three initial stages shown begin operation.



In a typical application, the separate stage sequences above operate as follows:

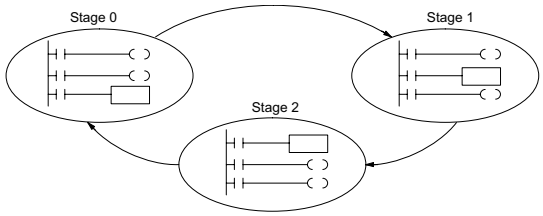
- **Powerup Initialization** – This stage contains ladder rung tasks done just once at powerup. Its last rung resets the stage, so this stage is only active for one scan (or only as many scans that are required).
- **Main Process** –This stage sequence controls the heart of the process or machine. One pass through the sequence represents one part cycle of the machine, or one batch in the process.
- **E-Stop and Alarm Monitoring** –This stage is always active because it is watching for errors that could indicate an alarm condition or require an emergency stop. It is common for this stage to reset stages in the main process or elsewhere, in order to initialize them after an error condition.
- **Operator Interface** –This is another task that must always be active and ready to respond to an operator. It allows an operator interface to change modes, etc., independently of the current main process step.

Although we have separate processes, there can be coordination among them. For example, in an error condition, the Status Stage may want to automatically switch the operator interface to the status mode to show error information as shown to the right. The monitor stage could set the stage bit for Status and Reset the stages Control and Recipe.



How Instructions Work Inside Stages

We can think of states or stages as simply dividing up our ladder program as depicted in the figure below. Each stage contains only the ladder rungs which are needed for the corresponding state of the process. The logic for transitioning out of a stage is contained within that stage. It's easy to choose which ladder rungs are active at powerup by using an **initial** stage type (ISG).



Most all instructions work just like they do in standard RLL. You can think of a stage just like a miniature RLL program which is either active or inactive.

Output Coils – As expected, output coils in active stages will turn on or off outputs according to power flow into the coil. However, note the following:

- Outputs work as usual, provided each output reference, such as "Y3", is used in only one stage.
- An output can be referenced from more than one stage, as long as only one of the stages is active at a time.
- If an output coil is controlled by more than one stage simultaneously, the active stage nearest the bottom of the program determines the final output status during each scan. Therefore, use the OROUT instruction instead when you want multiple stages to have a logical OR control of an output.

One-Shot or PD coils – Use care if you must use a Positive Differential coil in a stage. Remember that the input to the coil must make a 0–1 transition. If the coil is already energized on the first scan when the stage becomes active, the PD coil will not work. This is because the 0–1 transition did not occur.

PD coil alternative: If there is a task which you want to do only once (on 1 scan), it can be placed in a stage which transitions to the next stage on the same scan.

Counter – In using a counter inside a stage, the stage must be active for one scan before the input to the counter makes a 0–1 transition. Otherwise, there is no real transition and the counter will not count.

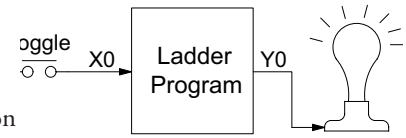
The ordinary Counter instruction does have a restriction inside stages: it may not be reset from other stages using the RST instruction for the counter bit. However, the special Stage counter provides a solution (see next paragraph).

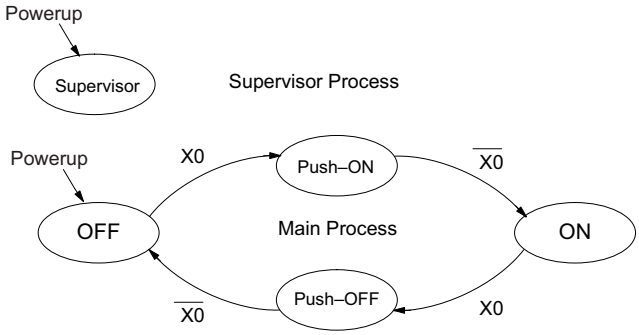
Stage Counter – The Stage Counter has the benefit that its count may be globally reset from other stages by using the RST instruction. It has a count input, but no reset input. This is the only difference from a standard counter.

Drum – Realize that the drum sequencer is its own process, and is a different programming method than stage programming. If you need to use a drum with stages, be sure to place the drum instruction in an ISG stage that is always active.

Using a Stage as a Supervisory Process

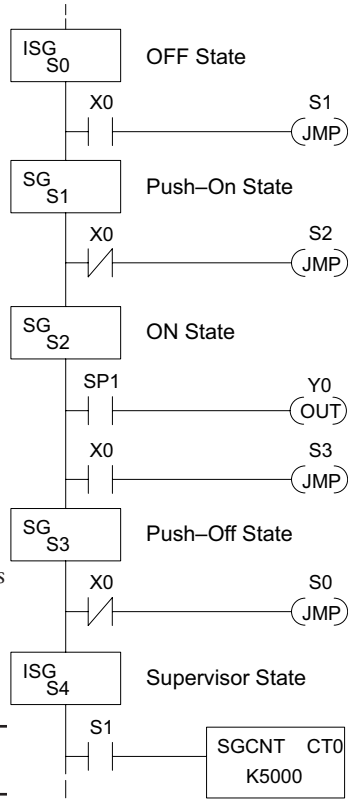
You may recall the light bulb on-off controller example from earlier in this chapter. For the purpose of illustration, suppose we want to monitor the *productivity* of the lamp process, by counting the number of on-off cycles which occurs. This application will require the addition of a simple counter, but the key decision is in where to put the counter.





New stage programming students will typically try to place the counter inside one of the stages of the process they are trying to monitor. The problem with this approach is that the stage is active only part of the time. In order for the counter to count, the count input must transition from off to on at least one scan after its stage activates. Ensuring this requires extra logic that can be tricky.

In this case, we only need to add another supervisory stage as shown above, to *watch* the main process. The counter inside the supervisor stage uses the stage bit S1 of the main process as its count input. *Stage bits used as a contact let us monitor a process!*



***NOTE**: Both the **Supervisor** stage and the **OFF** stage are initial stages. The supervisor stage remains active indefinitely.*

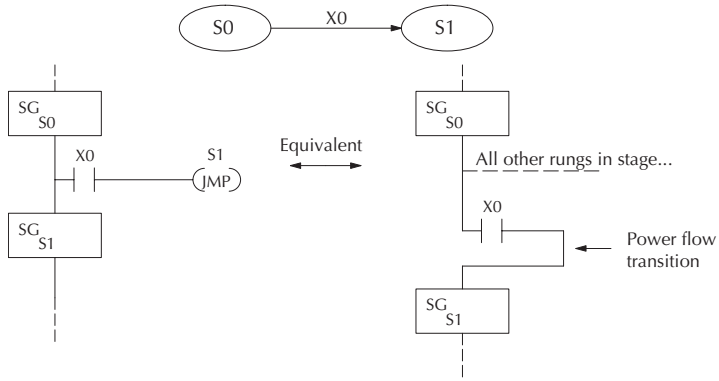
Stage Counter

The counter in the above example is a special Stage Counter. Note that it does not have a reset input. The count is reset by executing a Reset instruction, naming the counter bit (CT0 in this case). The Stage Counter has the benefit that its count may be globally reset from other stages. The standard Counter instruction does not have this global reset capability. You may still use a regular Counter instruction inside a stage... however, the reset input to the counter is the only way to reset it.

Power Flow Transition Technique

Our discussion of state transitions has shown how the Stage JMP instruction makes the current stage inactive and the next stage (named in the JMP) active. As an alternative way to enter this in *Direct*SOFT, you may use the power flow method for stage transitions.

The main requirement is that the current stage be located directly above the next (jump-to) stage in the ladder program. This arrangement is shown in the diagram below, by stages S0 and S1, respectively.

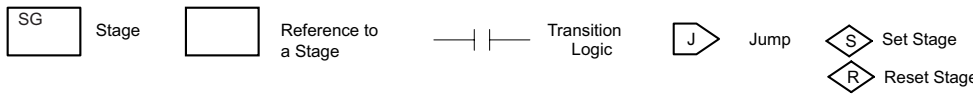


Remember that the Stage JMP instruction may occur anywhere in the current stage, and the result is the same. However, power flow transitions, as shown above, must occur as the last rung in a stage. All other rungs in the stage will precede it. The power flow transition method is also achievable on the handheld programmer, by simply following the transition condition with the Stage instruction for the next stage.

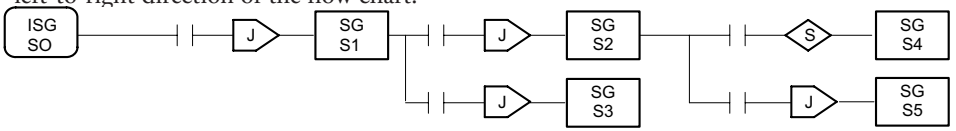
The power flow transition method does eliminate one Stage JMP instruction, its only advantage. However, it is not as easy to make program changes as using the Stage JMP. Therefore, we advise using Stage JMP transitions for most programmers.

Stage View in *Direct*SOFT

The Stage View option in *Direct*SOFT will let you view the ladder program as a flow chart. The figure below shows the symbol convention used in the diagrams. You may find the stage view useful as a tool to verify that your stage program has faithfully reproduced the logic of the state transition diagram you intend to realize.



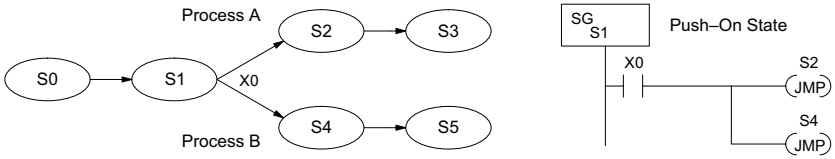
The following diagram is a typical stage view of a ladder program containing stages. Note the left-to-right direction of the flow chart.



Parallel Processing Concepts

Parallel Processes

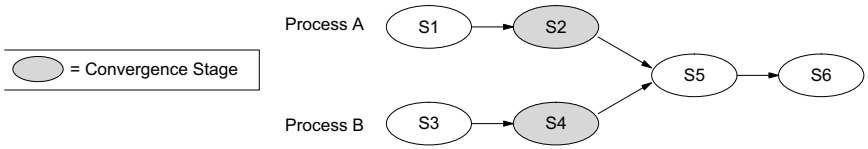
Previously in this chapter we discussed how a state may transition to either one state or another, called an *exclusive transition*. In other cases, we may need to branch simultaneously to two or more parallel processes, as shown below. It is acceptable to use all JMP instructions as shown, or we could use one JMP and a Set Stage bit instruction(s) (at least one must be a JMP, in order to leave S1). Remember that all instructions in a stage execute, even when it transitions (the JMP is not a GOTO).



Note that if we want Stages S2 and S4 to energize exactly on the same scan, both stages must be located below or above Stage S1 in the ladder program (see the explanation at the bottom of page 7–7). Overall, parallel branching is easy!

Converging Processes

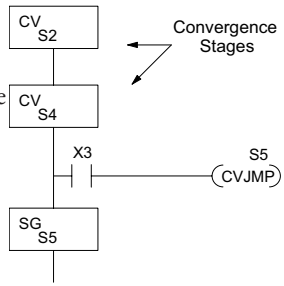
Now, we consider the opposite case of parallel branching, which is converging processes. This simply means we stop doing multiple things and continue doing one thing at a time. In the figure below, processes A and B converge when stages S2 and S4 transition to S5 at some point in time. So, S2 and S4 are *Convergence Stages.*



Convergence Stages (CV)

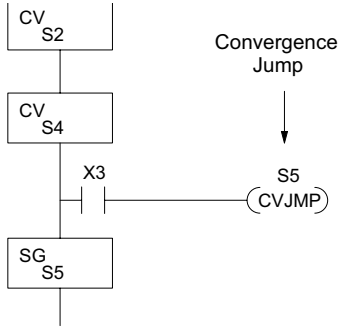
While the converging principle is simple enough, it brings a new complication. As parallel processing completes, the multiple processes almost never finish at the same time. In other words, how can we know whether Stage S2 or S4 will finish last? This is an important point, because we have to decide how to transition to Stage S5.

The solution is to coordinate the transition condition out of convergence stages. We accomplish this with a stage type designed for this purpose: the Convergence Stage (type CV). In the example to the right, convergence stages S2 and S4 are required to be grouped together as shown. **No logic is permitted between CV stages!** The transition condition (X3 in this case) must be located in the last convergence stage. The transition condition only has power flow when all convergence stages in the group are active.



Convergence Jump (CVJMP)

Remember, the last convergence stage only has power flow when all CV stages in the group are active. To complement the convergence stage, we need a new jump instruction. The Convergence Jump (CVJMP) shown to the right will transition to Stage S5 when X3 is active (as one might expect), but it also *automatically resets all convergence stages in the group*. This makes the CVJMP jump a very powerful instruction. Note that this instruction may only be used with convergence stages.



Convergence Stage Guidelines

The following summarizes the requirements in the use of convergence stages, including some tips for their effective application:

- A convergence stage is to be used as the last stage of a process which is running in parallel to another process or processes. A transition to the convergence stage means that a particular process is finished and represents a waiting point until all other parallel processes also finish.
- The maximum number of convergence stages which make up one group is 16. In other words, a maximum of 16 stages can converge into one stage.
- Convergence stages of the same group must be placed together in the program, connected on the power rail without any other logic in between.
- Within a convergence group, the stages may occur in any order, top to bottom. It does not matter which stage is last in the group, because all convergence stages have to be active before the last stage has power flow.
- The last convergence stage of a group may have ladder logic within the stage. However, this logic will not execute until all convergence stages of the group are active.
- The convergence jump (CVJMP) is the intended method to be used to transition from the convergence group of stages to the next stage. The CVJMP resets all convergence stages of the group, and energizes the stage named in the jump.
- The CVJMP instruction must only be used in a convergence stage, as it is invalid in regular or initial stages.
- Convergence Stages or CVJMP instructions may not be used in subroutines or interrupt routines.

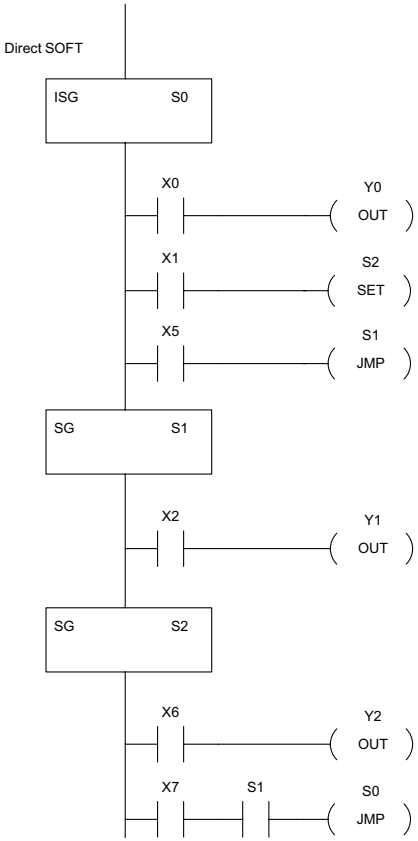
RLL^{PLUS} (Stage) Instructions

Stage (SG)

The Stage instructions are used to create structured RLL^{PLUS} programs. Stages are program segments which can be activated by transitional logic, a jump or a set stage that is executed from an active stage. Stages are deactivated one scan after transitional logic, a jump, or a reset stage instruction is executed.

Operand Data Type	DL06 Range
	aaa
Stage S	0–1777

The following example is a simple RLL^{PLUS} program. This program utilizes an initial stage, stage, and jump instructions to create a structured program.

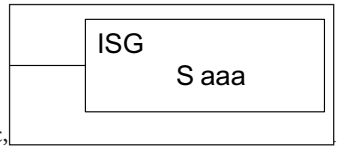


Handheld Programmer Keystrokes

U ISG	→	A 0	ENT		
$ STR	→	A 0	ENT		
GX OUT	→	A 0	ENT		
$ STR	→	B 1	ENT		
X SET	→	SHFT	S RST	C 2	ENT
$ STR	→	F 5	ENT		
K JMP	→	B 1	ENT		
2 SG	→	B 1	ENT		
$ STR	→	C 2	ENT		
GX OUT	→	B 1	ENT		
2 SG	→	C 2	ENT		
$ STR	→	G 6	ENT		
GX OUT	→	C 2	ENT		
$ STR	→	H 7	ENT		
V AND	→	SHFT	S RST	B 1	ENT
K JMP	→	A 0	ENT		

Initial Stage (ISG)

The Initial Stage instruction is normally used as the first segment of an RLL^{PLUS} program. Multiple Initial Stages are allowed in a program. They will be active when the CPU enters the Run mode allowing for a starting point in the program. Initial Stages are also activated by transitional logic, jump or a set stage executed from an active stage.



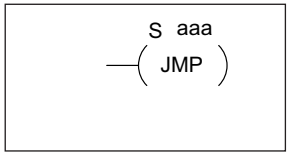
Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

NOTE: *If the ISG is within the retentive range for stages, the ISG will remain in the state it was in before power down and will NOT turn itself on during the first scan.*

7

Jump (JMP)

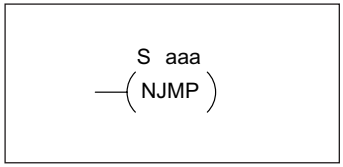
The Jump instruction allows the program to transition from an active stage containing the jump instruction to another stage (specified in the instruction). The jump occurs when the input logic is true. The active stage containing the Jump will deactivate 1 scan later.



Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

Not Jump (NJMP)

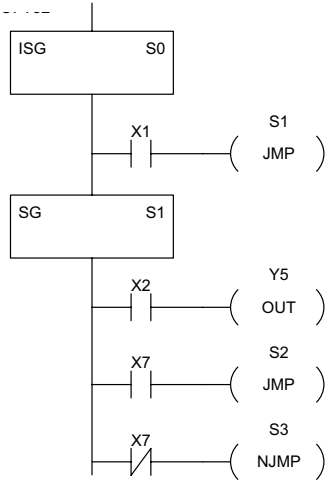
The Not Jump instruction allows the program to transition from an active stage which contains the jump instruction to another which is specified in the instruction. The jump will occur when the input logic is off. The active stage that contains the Not Jump will be deactivated 1 scan after the Not Jump instruction is executed.



Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

In the following example, only stage ISG0 will be active when program execution begins. When X1 is on, program execution will jump from Initial Stage 0 to Stage 1.

DirectSOFT



Handheld Programmer Keystrokes

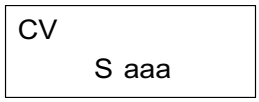
U ISG	→	A 0	ENT				
$ STR	→	B 1	ENT				
K JMP	→	B 1	ENT				
2 SG	→	B 1	ENT				
$ STR	→	C 2	ENT				
GX OUT	→	F 5	ENT				
$ STR	→	H 7	ENT				
K JMP	→	C 2	ENT				
SHFT	N TMR	SHFT	K JMP	→	D 3	ENT	

7

Converge Stage (CV) and Converge Jump (CVJMP)

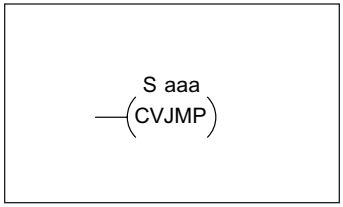
The Converge Stage instruction is used to group certain stages together by defining them as Converge Stages.

When all of the Converge Stages within a group become active, the CVJMP instruction (and any additional logic in the final CV stage) will be executed. All preceding CV stages must be active before the final CV stage logic can be executed. All Converge Stages are deactivated one scan after the CVJMP instruction is executed.



Additional logic instructions are only allowed following the last Converge Stage instruction and before the CVJMP instruction. Multiple CVJMP instructions are allowed.

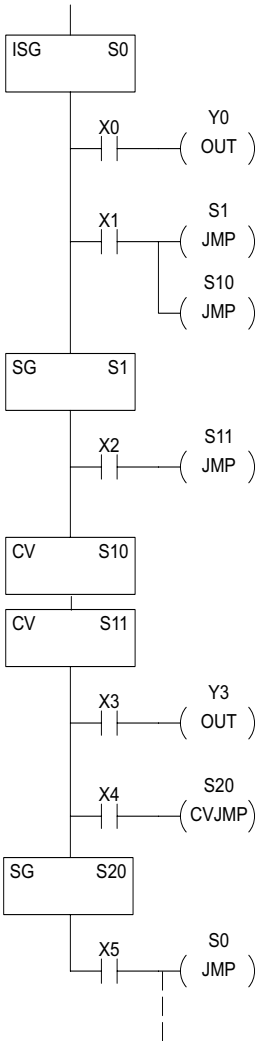
Converge Stages must be programmed in the main body of the application program. This means they cannot be programmed in Subroutines or Interrupt Routines.



Operand Data Type		DL06 Range
		aaa
Stage	S	0–1777

In the following example, when Converge Stages S10 and S11 are both active the CVJMP instruction will be executed when X4 is on. The CVJMP will deactivate S10 and S11, and activate S20. Then, if X5 is on, the program execution will jump back to the initial stage, S0.

DirectSOFT

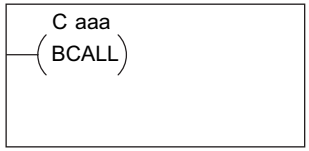


Handheld Programmer Keystrokes

U ISG	→	A 0	ENT					
$ STR	→	A 0	ENT					
GX OUT	→	A 0	ENT					
$ STR	→	B 1	ENT					
K JMP	→	B 1	ENT					
K JMP	→	B 1	A 0	ENT				
2 SG	→	B 1	ENT					
$ STR	→	C 2	ENT					
K JMP	→	B 1	B 1	ENT				
SHFT	C 2	V AND	→	B 1	A 0	ENT		
SHFT	C 2	V AND	→	B 1	B 1	ENT		
$ STR	→	D 3	ENT					
GX OUT	→	D 3	ENT					
$ STR	→	E 4	ENT					
SHFT	C 2	V AND	SHFT	K JMP	→	C 2	A 0	ENT
2 SG	→	C 2	A 0	ENT				
$ STR	→	F 5	ENT					
K JMP	→	A 0	ENT					

Block Call (BCALL)

The stage block instructions are used to activate a block of stages. The Block Call, Block, and Block End instructions must be used together. The BCALL instruction is used to activate a stage block. There are several things you need to know about the BCALL instruction.

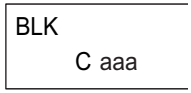


- Uses CR Numbers — The BCALL appears as an output coil, but does not actually refer to a Stage number as you might think. Instead, the block is identified with a Control Relay (Caaa). This control relay cannot be used as an output anywhere else in the program.
- Must Remain Active — The BCALL instruction actually controls all the stages between the BLK and the BEND instructions even after the stages inside the block have started executing. The BCALL must remain active or all the stages in the block will automatically be turned off. If either the BCALL instruction, or the stage that contains the BCALL instruction goes off, then the stages in the defined block will be turned off automatically.
- Activates First Block Stage — When the BCALL is executed it automatically activates the first stage following the BLK instructions.

Operand Data Type		DL06 Range
		aaa
Control Relay	S	0–1777

Block (BLK)

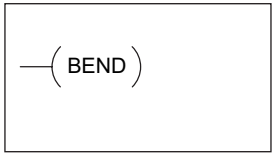
The Block instruction is a label which marks the beginning of a block of stages that can be activated as a group. A Stage instruction must immediately follow the Start Block instruction. Initial Stage instructions are not allowed in a block. The control relay (Caaa) specified in Block instruction must not be used as an output any where else in the program.



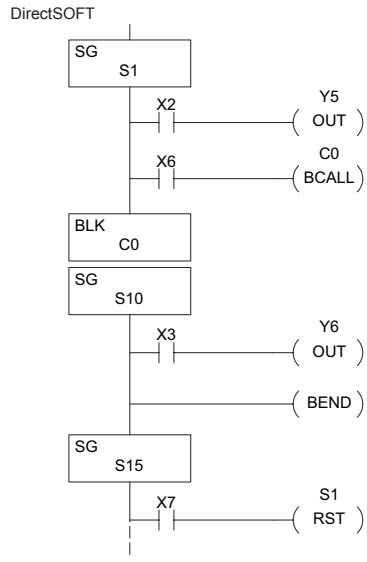
Operand Data Type		DL06 Range
		aaa
Control Relay	S	0–1777

Block End (BEND)

The Block End instruction is a label used with the Block instruction. It marks the end of a block of stages. There is no operand with this instruction. Only one Block End is allowed per Block Call.



In this example, the Block Call is executed when stage 1 is active and X6 is on. The Block Call then automatically activates stage S10, which immediately follows the Block instruction.
This allows the stages between S10 and the Block End instruction to operate as programmed.
the BCALL instruction is turned off, or
the stage containing the BCALL instruction turned off, then all stages between the BLK BEND instructions are automatically turned
If you examine S15, you will notice that could reset Stage S1, which would disable BCALL, thus resetting all stages within the block.

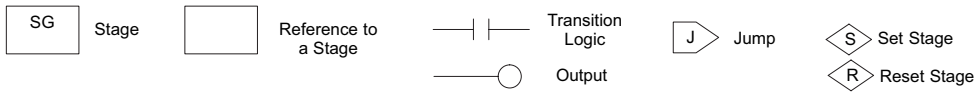


Handheld Programmer Keystrokes

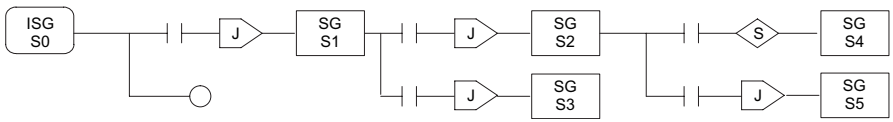
SG	→	S(SG)	1	ENT					
STR	→	X(IN)	2	ENT					
OUT	→	Y(OUT)	5	ENT					
STR	→	X(IN)	6	ENT					
SHFT	B	C	A	L	L	→	C(CR)	0	ENT
SHFT	B	L	K	→	C(CR)	0	ENT		
SG	→	S(SG)	1	0	ENT				
STR	→	X(IN)	3	ENT					
OUT	→	Y(OUT)	6	ENT					
SHFT	B	E	N	D	ENT				
SG	→	S(SG)	1	5	ENT				
STR	→	X(IN)	7	ENT					
RST	→	S(SG)	1	ENT					

Stage View in *Direct*SOFT

The Stage View option in *Direct*SOFT will let you view the ladder program as a flow chart. The figure below shows the symbol convention used in the diagrams. You may find the stage view useful as a tool to verify that your stage program has faithfully reproduced the logic of the state transition diagram you intend to realize.



The following diagram is a typical stage view of a ladder program containing stages. Note the left-to-right direction of the flow chart.



Questions and Answers about Stage Programming

We include the following commonly-asked questions about Stage Programming as an aid to new students. All question topics are covered in more detail in this chapter.

Q. What does stage programming do that I can't do with regular RLL programs?

A. Stages allow you to identify all the states of your process before you begin programming. This approach is more organized, because you divide up a ladder program into sections. As stages, these program sections are active only when they are actually needed by the process. Most processes can be organized into a sequence of stages, connected by event-based transitions.

Q. What are Stage Bits?

A. A stage bit is just a single bit in the CPU's image register, representing the active/inactive status of the stage in real time. For example, the bit for Stage 0 is referenced as "S0". If S0 = 0, then the ladder rungs in Stage 0 are bypassed (not executed) on each CPU scan. If S0 = 1, then the ladder rungs in Stage 0 are executed on each CPU scan. Stage bits, when used as contacts, allow one part of your program to monitor another part by detecting stage active/inactive status.

Q. How does a stage become active?

A. There are three ways:

- If the Stage is an initial stage (ISG), it is automatically active at powerup.
- Another stage can execute a Stage JMP instruction naming this stage, which makes it active upon its next occurrence in the program.
- A program rung can execute a Set Stage Bit instruction (such as Set S0).

Q. How does a stage become inactive?

A. There are three ways:

- Standard Stages (SG) are automatically inactive at powerup.
- A stage can execute a Stage JMP instruction, resetting its Stage Bit to 0.
- Any rung in the program can execute a Reset Stage Bit instruction (such as Reset S0).

Q. What about the power flow technique of stage transitions?

A. The power flow method of connecting adjacent stages (directly above or below in the program) actually is the same as the Stage Jump instruction executed in the stage above, naming the stage below. Power flow transitions are more difficult to edit in *Direct*SOFT, we list them separately from two preceding questions.

Q. Can I have a stage which is active for only one scan?

A. Yes, but this is not the intended use for a stage. Instead, just make a ladder rung active for 1 scan by including a stage Jump instruction at the bottom of the rung. Then the ladder will execute on the last scan before its stage jumps to a new one.

Q. Isn't a Stage JMP just like a regular GOTO instruction used in software?

A. No, it is very different. A GOTO instruction sends the program execution immediately to the code location named by the GOTO. A Stage JMP simply resets the Stage Bit of the current stage, while setting the Stage Bit of the stage named in the JMP instruction. Stage bits are 0 or 1, determining the inactive/active status of the corresponding stages. A stage JMP has the following results:

- When the JMP is executed, the remainder of the current stage's rungs are executed, even if they reside past(under) the JMP instruction. On the following scan, that stage is not executed, because it is inactive.
- The Stage named in the Stage JMP instruction will be executed upon its next occurrence. If located past (under) the current stage, it will be executed on the same scan. If located before (above) the current stage, it will be executed on the following scan.

Q. How can I know when to use stage JMP, versus a Set Stage Bit or Reset Stage Bit?

A. These instructions are used according to the state diagram topology you have derived:

- Use a Stage JMP instruction for a state transition ... moving from one state to another.
- Use a Set Stage Bit instruction when the current state is spawning a new parallel state or stage sequence, or when a supervisory state is starting a state sequence under its command.
- Use a Reset Bit instruction when the current state is the last state in a sequence and its task is complete, or when a supervisory state is ending a state sequence under its command.

Q. What is an initial stage, and when do I use it?

A. An initial stage (ISG) is automatically active at powerup. Afterwards, it works just like any other stage. You can have multiple initial stages, if required. Use an initial stage for ladder that must always be active, or as a starting point.

Q. Can I place program ladder rungs outside of the stages, so they are always on?

A. It is possible, but it's not good software design practice. Place ladder that must always be active in an initial stage, and do not reset that stage or use a Stage JMP instruction inside it. It can start other stage sequences at the proper time by setting the appropriate Stage Bit(s).

Q. Can I have more than one active stage at a time?

A. Yes, and this is a normal occurrence for many programs. However, it is important to organize your application into separate processes, each made up of stages. And a good process design will be mostly sequential, with only one stage on at a time. However, all the processes in the program may be active simultaneously.

PID Loop Operation

In This Chapter...

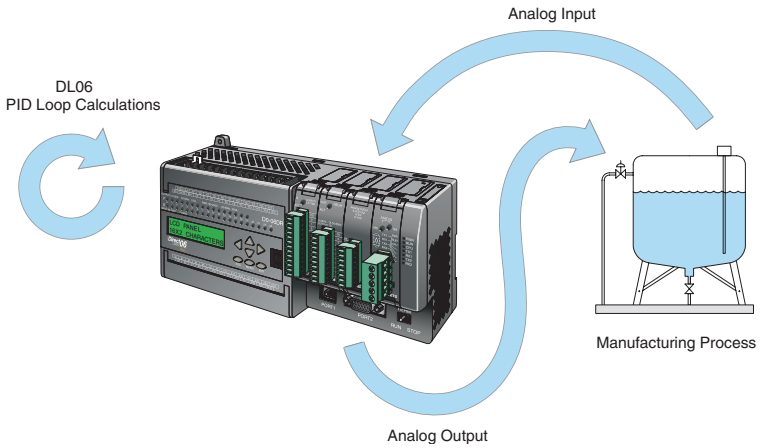
DL06 PID Control

DL06 PID Control Features

Along with control functions discussed in this manual, the DL06 PLC features PID process control capability. The DL06 PID process control loops offer the same features offered in much larger PLCs. The primary features are:

- Up to 8 PID loops, individual programmable sample rates
- Manual, Automatic and Cascade loop operation modes
- Two types of bumpless transfer available
- Full-featured alarms
- Ramp/soak generator with up to 16 segments
- Auto Tuning

The DL06 CPU has process control loop capability in addition to ladder program execution. You can select and configure up to eight loops. All sensor and actuator wiring connects directly to DL06 analog modules. All process variables, gain values, alarm levels, etc., associated with each loop reside in a Loop Variable Table in the CPU. The DL06 CPU reads process variable (PV) inputs during each scan. Then, it makes PID loop calculations during a dedicated time slice on each PLC scan, updating the control output value. The control loops use a Proportional-Integral-Derivative (PID) algorithm to generate the control output. This chapter describes how the loops operate, and how to configure and tune the loops.



*Direct*SOFT programming software, release 5, or later, is used for configuring analog control loops in the DL06. *Direct*SOFT uses dialog boxes to help you set up the individual loops. After completing the setup, you can use *Direct*SOFT's PID Trend View to tune each loop. The configuration and tuning selections you make are stored in the DL06's V-memory (RAM). The loop parameters also may be saved to disk for recall later.

PID Loop Feature	Specifications
Number of loops	Selectable, 8 maximum
CPU V-memory needed	32 words (V locations) per loop selected, 64 words if using ramp/soak
PID algorithm	Position or Velocity form of the PID equation
Control Output polarity	Selectable direct-acting or reverse-acting
Error term curves	Selectable as linear, square root of error, and error squared
Loop update rate (time between PID calculation)	0.05 to 99.99 seconds, user programmable
Minimum loop update rate	0.05 seconds for 1 to 4 loops 0.1 seconds for 5 to 8 loops
Loop modes	Automatic, Manual (operator control), or Cascade control
Ramp/Soak Generator	Up to 8 ramp/soak steps (16 segments) per loop with indication of ramp/soak step number
PV curves	Select standard linear, or square-root extract (for flow meter input)
Set Point Limits	Specify minimum and maximum setpoint values
Process Variable Limits	Specify minimum and maximum Process Variable values
Proportional Gain	Specify gains of 0.01 to 99.99
Integrator (Reset)	Specify reset time of 0.1 to 99.99 in units of seconds or minutes
Derivative (Rate)	Specify the derivative time from 0.01 to 99.99 seconds
Rate Limits	Specify derivative gain limiting from 1 to 20
Bumpless Transfer I	Automatically sets the bias equal to the control output and the setpoint equal to the process variable when control switches from manual to automatic.
Bumpless Transfer II	Automatically sets the bias equal to the control output when control switches from manual to automatic.
Step Bias	Provides proportional bias adjustment for large setpoint changes
Anti-windup (Freeze Bias)	For position form of PID, this inhibits integrator action when the control output reaches 0% or 100 % (speeds up loop recovery when output recovers from saturation)
Error Deadband	Specify a tolerance (plus and minus) for the error term (SP–PV), so that no change in control output value is made

Alarm Feature	Specifications
PV Alarm Hysteresis	Specify 1 to 200 (word/binary) does not affect all alarms, such as PV Rate-of-Change Alarm
PV Alarm Points	Select PV alarm settings for Low–low, Low, High, and High-high conditions
PV Deviation	Specify alarms for two ranges of PV deviation from the setpoint value
Rate of Change	Detect when PV exceeds a rate of change limit you specify

Introduction to PID Control

What is PID Control?

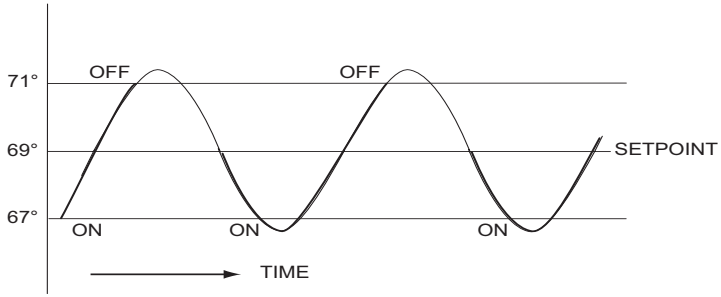
In this discussion, we will explain why PID control is used in process control instead of trying to provide control by simply using an analog input and a discrete output.

There are many types of analog controllers available, and the proper selection will depend upon the particular application. There are two types of analog controllers that are used throughout industry:

1. The ON-OFF controller, sometimes referred to as an open loop controller.
2. The PID controller, sometimes called a closed loop controller.

Regardless of type, analog controllers require input signals from electronic sensors such as pressure, differential pressure, level, flow meter or thermocouples. As an example, one of the most common analog control applications is located in your house for controlling either heat or air conditioning, the thermostat.

You wish for your house to be at a comfortable temperature so you set a thermostat to a desired temperature (setpoint). You then select the comfort mode, either heat or A/C. A temperature sensing device, normally a thermistor, is located within the thermostat. If the thermostat is set for heat and the setpoint is set for 69°, the furnace will be turned on to provide heat at, normally, 2° below the setpoint. In this case, it would turn on at 67°. When the temperature reaches 71°, 2° above setpoint, the furnace will turn off. In the opposite example, if the thermostat is set for A/C (cooling), the thermostat will turn the A/C unit on/off opposite the heat setting. For instance, if the thermostat is set to cool at 76°, the A/C unit will turn on when the sensed temperature reaches 2° above the setpoint or 78°, and turn off when the temperature reaches 74°. This would be considered to be an ON-OFF controller. The waveform below shows the action of the heating cycle. Note that there is a slight overshoot at the turn-off point, also a slight undershoot at the turn-on point.



The ON-OFF controller is used in some industrial control applications, but is not practical in the majority of industrial control processes.

The most common process controller that is used in industry is the PID controller.

The PID controller controls a continuous feedback loop that keeps the process output (control variable) flowing normally by taking corrective action whenever there is a deviation from the desired value (setpoint) of the process variable (PV) such as, rate of flow, temperature, voltage, etc. An *error* occurs when an operator manually changes the setpoint or when an event (valve opened, closed, etc.) or a disturbance (cold water, wind, etc.) changes the load, thus causing a change in the process variable.

The PID controller receives signals from sensors and computes corrective action to the actuator from a computation based on the error (Proportional), the sum of all previous errors (Integral) and the rate of change of the error (Derivative).

We can look at the PID controller in more simple terms. Take the cruise control on an automobile as an example. Let's say that we are cruising on an interstate highway in a car equipped with cruise control. The driver decides to engage the cruise control by turning it ON, then he manually brings the car to the desired cruising speed, say 70 miles per hour. Once the cruise speed is reached, the SET button is pushed fixing the speed at 70 mph, the setpoint. Now, the car is cruising at a steady 70 mph until it comes to a hill to go up. As the car goes up the hill, it tends to slow down. The speed sensor senses this and causes the throttle to increase the fuel to the engine. The vehicle speeds up to maintain 70 mph without jerking the car and it reaches the top at the set speed. When the car levels out after reaching the top of the hill it will speed up. The speed sensor senses this and signals the throttle to provide less fuel to the engine, thus, the engine slows down allowing the car to maintain the 70 mph speed. How does this application apply to PID control? Lets look at the function of P, I and D terms:

- **Proportional** - is commonly referred to as Proportional Gain. The proportional term is the corrective action which is proportional to the error, that is, the change of the manipulated variable is equal to the proportional gain multiplied by the error (the activating signal). In mathematical terms:

  Proportional action = proportional gain X error
  Error = Setpoint (SP) - Process Variable (PV)

  Applying this to the cruise control, the speed was set at 70 mph which is the Setpoint. The speed sensor senses the actual speed of the car and sends this signal to the cruise controller as the Process Variable (PV). When the car is on a level highway, the speed is maintained at 70 mph, thus, no error since the error would be SP - PV = 0. When the car goes up the hill, the speed sensor detected a slow down of the car, SP-PV = error. The proportional gain would cause the output of the speed controller to bring the car back to the setpoint of 70 mph. This would be the Controlled Output.

- **Integral** - this term is often referred to as Reset action. It provides additional compensation to the control output, which causes a change in proportion to the value of the error over a period of time. In other words, the reset term is the integral sum of the error values over a period of time.

- **Derivative** - this term is referred to as rate. The Rate action adds compensation to the control output, which causes a change in proportion to the rate of change of error. Its job is to anticipate the probable growth of the error and generate a contribution to the output in advance.

Introducing DL06 PID Control

The DL06 is capable of controlling a process variable such as those already mentioned. As previously mentioned, the control of a variable, such as temperature, at a given level (setpoint) as long as there are no disturbances (cold water) in the process.

The DL06 PLC has the ability to directly accept signals from electronic sensors, such as thermocouples, pressure, VFDs, etc. These signals may be used in mathematically derived control systems.

In addition, the DL06 has built-in PID control algorithms that can be implemented. The basic function of PID closed loop process control is to maintain certain process characteristics at desired setpoints. As a rule, the process deviates from the desired setpoint reference as a result of load material changes and interaction with other processes. During this control, the actual condition of the process characteristics (liquid level, temperature, motor control, etc.) is measured as a *process variable* (PV) and compared with the target setpoint (SP). When deviations occur, an error is generated by the difference between the process variable (actual value) and the setpoint (desired value). Once an error is detected, the function of the control loop is to modify the control output in order to force the error to zero.

The DL06 PID control provides feedback loops using the PID algorithm. The control output is computed from the measured process variable as follows:

Let:

K_c = proportional gain

T_i = Reset or integral time

T_d = Derivative time or rate

SP = Setpoint

PV(t) = Process Variable at time "t"

e(t) = SP-PV(t) = PV deviation from setpoint at time "t" or PV error.

Then:

M(t) = Control output at time "t"

$$M(t) = Kc \left[e(t) + 1/T_i \int_0^t e(x)\, dx + T_d\, d/dt\, e(t) \right] + M_o$$

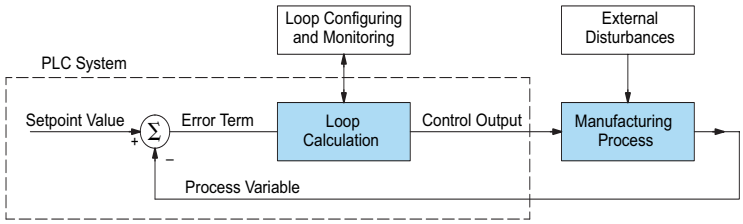
The analog input module receives the process variable in analog form along with an operator entered setpoint; the CPU computes the error. The error is used in the algorithm computation to provide corrective action at the control output. The function of the control action is based on an output control, which is proportional to the instantaneous error value. The *integral control action* (reset action) provides additional compensation to the control output, which causes a change in proportion to the value of the change of error over a period of time. The *derivative control action* (rate change) adds compensation to the control output, which causes a change in proportion to the rate of change of error. These three modes are used to provide the desired control action in Proportional (P), Proportional-Integral (PI), or Proportional-Integral-Derivative (PID) control fashion.

Standard DL05/06 analog input modules are used to interface to field transmitters to obtain the PV. These transmitters normally provide a 4-20mA current or an analog voltage of various ranges for the control loop.

For temperature control, thermocouple or RTD can be connected directly to the appropriate module. The PID control algorithm, residing in the CPU memory, receives information from the user program, primarily control parameters and setpoints. Once the CPU makes the PID calculation, the result may be used to directly control an actuator connected to a 4-20mA current output module to control a valve.

With *Direct*SOFT, additional ladder logic programming, both time proportioning (eg., heaters for temperature control) and position actuator (eg., reversible motor on a valve) type of control schemes can be easily implemented. This chapter will explain how to set up the PID control loop, how to implement the software and how to tune the loop.

The following block diagram shows the key parts of a PID control loop. The path from the PLC to the manufacturing process and back to the PLC is the closed loop control.



Process Control Definitions

Manufacturing Process – the set of actions that adds value to raw materials. The process can involve physical changes and/or chemical changes to the material. The changes render the material more useful for a particular purpose, ultimately used in a final product.

Process Variable – The controlled variable part of the process that you wish to control. It may be temperature, pressure, level, flow, composition, density, the ratio of two streams, etc. Also known as the actual value.

Setpoint – This is the target for the process variable. When all conditions of the process are correct, the process variable will equal the setpoint.

Control Output – The result of the loop calculation, which becomes a command for the process (such as the heater level in an oven). This is sometimes referred to as control variable.

Error Term – The algebraic difference between the process variable and the setpoint. This is the control loop error, and is equal to zero when the process variable is equal to the setpoint (desired) value. A well-behaved control loop is able to maintain a small error term magnitude.

Manipulated Variable – This is what is used to effect the controlled variable. For example, the fuel used in a furnace might be manipulated in order to control the temperature.

Disturbance – Something in the system that changes such that corrective action is required. For instance, when controlling a flow and the upstream pressure drops, the control valve must open wider in order to keep flow constant. The drop in upstream pressure is the disturbance.

Final Control Element – This is the physical device used to control the manipulated variable. Valves are probably the most widely used final control element.

Lag Time – The time it takes for the process to respond to a change in manipulated variable. This is also known as the capacitance of the system. When you're in the shower and you turn up the hot water a little, the time it takes before the water gets hot is the lag time.

Dead Time – The time it takes for a change in the process to be recognized. Composition analyzers and quality control are usually sources of significant dead time.

Loop Configuring – Operator-initiated selections which set up and optimize the performance of a control loop. The loop calculation function uses the configuration parameters in real time to adjust gains, offsets, etc.

Loop Monitoring – The function which allows an operator to observe the status and performance of a control loop. This is used in conjunction with the loop configuring to optimize the performance of a loop (minimize the error term).

PID Loop Operation

The Proportional–Integral–Derivative (PID) algorithm is widely used in process control. The PID method of control adapts well to electronic solutions, whether implemented in analog or digital (CPU) components. The DL06 CPU implements the PID equations digitally by solving the basic equations in software. I/O modules serve only to convert electronic signals into digital form (or vice versa).

The DL06 uses two types of PID controls: "position" and "velocity". These terms usually refer to motion control situations, but here we use them in a different sense:

- PID *Position* Algorithm – The control output is calculated so it responds to the displacement (position) of the PV from the SP (error term).
- PID *Velocity* Algorithm – The control output is calculated to represent the rate of change (velocity) for the PV to become equal to the SP.

Position Form of the PID Equation

Referring to the control output equation on page 8-6, the DL06 CPU approximates the output M(t) using a discrete position form of the PID equation.

Let:

Ts = Sample rate
Kc = Proportional gain
Ki = Kc * (Ts/Ti) = Coefficient of integral term
Kr = Kc * (Td/Ts) = Coefficient of derivative term
Ti = Reset or integral time
Td = Derivative time or rate
SP = Setpoint
PV_n = Process variable at n^{th} sample
en = SP – PV_n = Error at n^{th} sample
M_o = Value to which the controller output has been initialized

Then:

M_n = Control output at n^{th} sample

$$M_n = Kc * e_n + Ki \sum_{i=1}^{n} e_i + Kr\ (e_n - e_{n-1}) + M_o$$

This form of the PID equation is referred to as the position form since the actual actuator position is computed. The velocity form of the PID equation computes the change in actuator position. The CPU modifies the standard equation slightly to use the derivative of the process variable instead of the error as follows:

$$M_n = Kc * e_n + Ki \sum_{i=1}^{n} e_i + Kr\ (PV_n - PV_{n-1}) + M_o$$

These two forms are equivalent unless the setpoint is changed. In the original equation, a large step change in the setpoint will cause a correspondingly large change in the error resulting in a bump to the process due to derivative action. This bump is not present in the second form of the equation.

The DL06 also combines the integral sum and the initial output into a single term called the bias (Mx). This results in the following set of equations:

$$Mx_o = M_o$$

$$Mx = Ki * e_n + Mx_{n-1}$$

$$M_n = Kc * e_n - Kr(PV_n - PV_{n-1}) + Mx_n$$

The DL06 by default will keep the normalized output M in the range of 0.0 to 1.0. This is done by clamping M to the nearer of 0.0 or 1.0 whenever the calculated output falls outside this range. The DL06 also allows you to specify the minimum and maximum output limit values (within the range 0 to 4095 in BCD if using 12 bit unipolar).

NOTE: *The equations and algorithms, or parts of, in this chapter, are only for references. Analysis of these equations can be found in most good text books about process control.*

Reset Windup Protection

Reset windup can occur if reset action (integral term) is specified and the computation of the bias term Mx is:

$$Mx = Ki * e_n + Mx_{n-1}$$

For example, assume the output is controlling a valve and the PV remains at some value greater than the setpoint. The negative error (e_n) will cause the bias term (Mx) to constantly decrease until the output M goes to 0 closing the valve. However, since the error term is still negative, the bias will continue to decrease becoming ever more negative. When the PV finally does come back down below the SP, the valve will stay closed until the error is positive for long enough to cause the bias to become positive again. This will cause the process variable to undershoot.

One way to solve the problem is to simply clamp the normalized bias between 0.0 and 1.0. The DL06 CPU does this. However, if this is the only thing that is done, then the output will not move off 0.0 (thus opening the valve) until the PV has become less than the SP. This will also cause the process variable to undershoot.

The DL06 CPU is programmed to solve the overshoot problem by either freezing the bias term, or by adjusting the bias term.

Freeze Bias

If the "Freeze Bias" option is selected when setting up the PID loop (discussed later) then the CPU simply stops changing the bias (Mx) whenever the computed normalized output (M) goes outside the interval 0.0 to 1.0.

$$Mx = Ki * e_n + Mx_{n-1}$$

$$M = Kc * e_n - Kr(PV_n - PV_{n-1}) + Mx$$

$$M_n = 0 \quad \text{if } M < 0$$
$$M_n = M \quad \text{if } 0 \leqslant M \leqslant 1$$
$$M_n = 1 \quad \text{if } M > 1$$

$$Mx_n = Mx \quad \text{if } 0 \leqslant M \leqslant 1$$
$$Mx_n = Mx_{n-1} \quad \text{otherwise}$$

Thus in this example, the bias will probably not go all the way to zero so that, when the PV does begin to come down, the loop will begin to open the valve sooner than it would have if the bias had been allowed to go all the way to zero. This action has the effect of reducing the amount of overshoot.

8

Adjusting the Bias

The normal action of the CPU is to adjust the bias term when the output goes out of range as shown below.

$$Mx = Ki * e_n + Mx_{n-1}$$
$$M = Kc * e_n - Kr(PV_n - PV_{n-1}) + Mx$$

$$M_n = 0 \quad \text{if } M < 0$$
$$M_n = M \quad \text{if } 0 \leqslant M \leqslant 1$$
$$M_n = 1 \quad \text{if } M > 1$$

$$Mx_n = Mx \quad \text{if } 0 \leqslant M \leqslant 1$$
$$Mx_n = M_n - Kc * e_n - Kr(PV_n - PV_{n-1}) \quad \text{otherwise}$$

By adjusting the bias, the valve will begin to open as soon as the PV begins to come down. If the loop is properly tuned, overshoot can be eliminated entirely. If the output went out of range due to a setpoint change, then the loop probably will oscillate because we must wait for the bias term to stabilize again.

The choice of whether to use the default loop action or to freeze the bias is dependent on the application. **If large, step changes to the setpoint are anticipated, then it is probably better to select the freeze bias option** (see page 8-34).

Step Bias Proportional to Step Change in SP

This feature reduces oscillation caused by a step change in setpoint when the adjusting bias feature is used.

$Mx = Mx * SP_n / SP_{n-1}$ if the loop is direct acting

$Mx = Mx * SP_{n-1} / SP_n$ if the loop is reverse acting

$Mx_n = 0$ if $Mx < 0$

$Mx_n = Mx$ if $0 \leq Mx \leq 1$

$Mx_n = 1$ if $M > 1$

Eliminating Proportional, Integral or Derivative Action

It is not always necessary to run a full three mode PID control loop. Most loops require only the PI terms or just the P term. Parts of the PID equation may be eliminated by choosing appropriate values for the gain (Kc), reset (Ti) and rate (Td) yielding a P, PI, PD, I and even an ID and a D loop.

Eliminating Integral Action The effect of integral action on the output may be eliminated by setting Ti = 9999. When this is done, the user may then manually control the bias term (Mx) to eliminate any steady-state offset.

Eliminating Derivative Action The effect of derivative action on the output may be eliminated by setting Td = 0 (most loops do not require a D parameter; it may make the loop unstable).

Eliminating Proportional Action Although rarely done, the effect of proportional term on the output may be eliminated by setting Kc = 0. Since Kc is also normally a multiplier of the integral coefficient (Ki) and the derivative coefficient (Kr), the CPU makes the computation of these values conditional on the value of Kc as follows:

$Ki = Kc * (Ts / Ti)$ if Kc 0

$Ki = Ts / Ti$ if Kc = 0 (I or ID only)

$Kr = Kc * (Td / Ts)$ if Kc 0

$Kr = Td / Ts$ if Kc = 0 (ID or D only)

Velocity Form of the PID Equation

The standard position form of the PID equation computes the actual actuator position. An alternative form of the PID equation computes the change in actuator position. This form of the equation is referred to as the velocity PID equation and is obtained by subtracting the equation at time "n" from the equation at time "n-1".

The velocity equation is given by:

$\Delta M_n = M - M_{n-1}$

$\Delta M_n = Kc * (e_n - e_{n-1}) + Ki * (PV_n - 2 * PV_{n-1} + PV_{n-2})$

Bumpless Transfer

The DL06 loop controller provides for bumpless mode changes. A bumpless transfer from manual mode to automatic mode is achieved by preventing the control output from changing immediately after the mode change.

When a loop is switched from Manual mode to Automatic mode, the setpoint and Bias are initialized as follows:

Position PID Algorithm	Velocity PID Algorithm
SP = PV	SP = PV
Mx = M	

The bumpless transfer feature of the DL06 is available in two types: Bumpless I and Bumpless II (see page 8-26). The transfer type is selected when the loop is set up.

Loop Alarms

The DL06 allows the user to specify alarm conditions that are to be monitored for each loop. Alarm conditions are reported to the CPU by setting up the alarms in ***Direct***SOFT using the PID setup alarm dialog when the loop is setup. The alarm features for each loop are:

- **PV Limit** – Specify up to four PV alarm points.

High-High	PV rises above the programmed High-High Alarm Limit.
High	PV rises above the programmed High Alarm Limit.
Low	PV fails below the Low Alarm Limit.
Low-Low	PV fails below the Low-Low Limit.

- **PV Deviation Alarm** – Specify an alarm for High and Low PV deviation from the setpoint (Yellow Deviation). An alarm for High High and Low Low PV deviation from the setpoint (Orange Deviation) may also be specified. When the PV is further from the setpoint than the programmed Yellow or Orange Deviation Limit the corresponding alarm bit is activated.
- **Rate of Change** – This alarm is set when the PV changes faster than a specified rate-of-change limit.
- **PV Alarm Hysteresis** – The PV Limit Alarms and PV Deviation Alarms are programmed using threshold values. When the absolute value or deviation exceeds the threshold, the alarm status becomes true. Real-world PV signals have some noise on them, which can cause some fluctuation in the PV value in the CPU. As the PV value crosses an alarm threshold, its fluctuations will cause the alarm to be intermittent and annoy process operators. The solution is to use the PV Alarm Hysteresis feature.

Loop Operating Modes

The DL06 loop controller operates in one of three modes, either *Manual , Automatic* or *Cascade*.

Manual

In manual mode, the control output is determined by the operator, not the loop controller. While in manual mode, the loop controller will still monitor all of the alarms including High-High, High, Low, Low-Low, Yellow deviation, Orange deviation and Rate-of-Change.

Automatic

In automatic mode, the loop controller computes the control output based on the programmed parameters stored in V-memory. All alarms are monitored while in automatic.

Cascade

Cascade mode is an option with the DL06 PLC and is used in special control applications. If the cascade feature is used, the loop will operate as it would if in automatic mode except for the fact that a cascaded loop has a setpoint which is the control output from another loop.

Special Loop Calculations

Reverse Acting Loop

Although the PID algorithm is used in a direct, or forward, acting loop controller, there are times when a reverse acting control output is needed. The DL06 loop controller allows a loop to operate as reverse acting. With a reverse acting loop, the output is driven in the opposite direction of the error. For example, if SP > PV, then a reverse acting controller will decrease the output to increase the PV.

$$Mx = -Ki * e_n + Mx_{n-1}$$

$$M = -Kc * e_n + Kr(PV_n - PV_{n-1}) + Mx_n$$

Square Root of the Process Variable

Square root is selected whenever the PV is from a device such as an orifice meter which requires this calculation.

Error Squared Control

Whenever error squared control is selected, the error is calculated as:

$$e_n = (SP - PV_n) * ABS(SP - PV_n)$$

A loop using the error squared is less responsive than a loop using just the error, however, it will respond faster with a large error. The smaller the error, the less responsive the loop. Error squared control would typically be used in a PH control application.

Error Deadband Control

With error deadband control, no control action is taken if the PV is within the specified deadband area around the setpoint. The error deadband is the same above and below the setpoint.

Once the PV is outside of the error deadband around the setpoint, the entire error is used in the loop calculation.

$e_n = 0$ SP - Deadband_Below_SP < PV < SP - Deadband_Above_SP
$e_n = P - PV_n$ otherwise

The error will be squared first if both Error Squared and Error Deadband is selected.

Derivative Gain Limiting

When the coefficient of the derivative term, Kr, is a large value, noise introduced into the PV can result in erratic loop output. This problem is corrected by specifying a derivative gain limiting coefficient, Kd. Derivative gain limiting is a first order filter applied to the derivative term computation, Y_n, as shown below.

$$Y_n = Y_{n-1} + \frac{Ts}{Ts + (\frac{Td}{Kd})} * (PV_n - Y_{n-1})$$

Position Algorithm

$Mx = Ki * e_n + Mx_{n-1}$

$M = Kc * e_n - Kr * (Y_n - Y_{n-1}) + Mx$

Velocity Algorithm

$\Delta M = Kc * (e_n - e_{n-1}) + Ki * e_n - Kr * (Y_n - 2 * Y_{n-1} + Y_{n-2})$

Ten Steps to Successful Process Control

Controllers such as the DL06 PLC provide sophisticated process control features. Automated control systems can be difficult to debug, because a given symptom can have many possible causes. We recommend a careful, step-by-step approach to bringing new control loops online:

Step 1: Know the Recipe

The most important is – how to produce your product. This knowledge is the foundation for designing an effective control system. A good process *recipe* will do the following:

- Identify all relevant Process Variables, such as temperature, pressure, or flow rates, etc., which need precise control.
- Plot the desired Setpoint values for each process variables for the duration of one process cycle.

Step 2: Plan Loop Control Strategy

This simply means choosing the method the machine will use to maintain control over the Process Variables to follow their Setpoints. This involves many issues and trade-offs, such as energy efficiency, equipment costs, ability to service the machine during production, and more. You must also determine how to generate the Setpoint value during the process, and whether a machine operator can change the SP.

Step 3: Size and Scale Loop Components

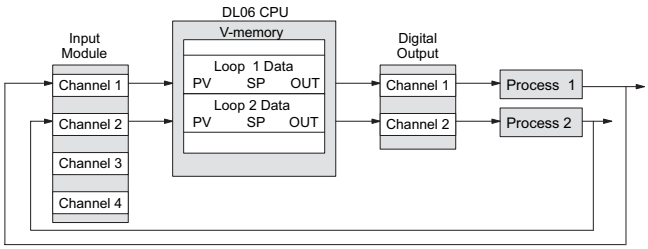
Assuming the control strategy is sound, it is still crucial to *properly size the actuator and properly scale the sensors.*

- Choose an actuator (heater, pump. etc.) which matches the size of the load. An oversized actuator will have an overwhelming effect on your process after a SP change. However, an undersized actuator will allow the PV to lag or drift away from the SP after a SP change or process disturbance.
- Choose a PV sensor which matches the range of interest (and control) for our process. Decide the resolution of control you need for the PV (such as within 2°C), and make sure the sensor input value provides the loop with at least 5 times that resolution (at LSB level). However, an over-sensitive sensor can cause control oscillations, etc. The DL06 provides 12–bit and 15–bit unipolar and bipolar data format options, and a 16–bit unipolar option. This selection affects SP, PV, Control Output and Integrator sum.

Step 4: Select I/O Modules

After deciding the number of loops, PV variables to measure, and SP values, you can choose the appropriate I/O module. Refer to the figure on the next page. In many cases, you will be able to share input or output modules, or use an analog I/O combination module, among several control loops. The example shown sends the PV and Control Output signals for two loops through the same set of modules.

Automationdirect offers DL06 analog input modules with 4 channels per module that accept 0 – 20mA or 4 – 20mA signals. Also, analog input and output combination modules are now available. Thermocouple and RTD modules can also be used to maintain temperatures to a 10th of a degree. Refer to the sales catalog for further information on these modules, or find the modules on our website, **www.automationdirect.com.**



Step 5: Wiring and Installation

After selection and procurement of all loop components and I/O module(s), you can perform the wiring and installation. Refer to the wiring guidelines in Chapter 2 of this manual, and to the **D0–OPTIONS–M** manual. The most common wiring errors when installing PID loop controls are:

- Reversing the polarity of sensor or actuator wiring connections.
- Incorrect signal ground connections between loop components.

Step 6: Loop Parameters

After wiring and installation, choose the loop setup parameters. The easiest method for programming the loop tables is using ***Direct***SOFT (5.0 or later). This software provides PID Setup using dialog boxes to simplify the task. **Note: It is important to understand the meaning of all loop parameters mentioned in this chapter before choosing values to enter.**

Step 7: Check Open Loop Performance

With the sensor and actuator wiring done, and loop parameters entered, we must manually and carefully check out the new control system using the Manual mode.

- Verify that the PV value from the sensor is correct.
- If it is safe to do so, gradually increase the control output up above 0%, and see if the PV responds (and moves in the correct direction!).

Step 8: Loop Tuning

If the Open Loop Test (page 8–40) shows the PV reading is correct and the control output has the proper effect on the process; you can follow the closed loop tuning procedure (see page 8–45). In this step, the loop is tuned so the PV automatically follows the SP.

Step 9: Run Process Cycle

If the closed loop test shows the PV will follow small changes in the SP, consider running an actual process cycle. You will need to have completed the programming which will generate the desired SP in real time. In this step, you may want to run a small test batch of product through the machine, watching the SP change according to the recipe.

WARNING: Be sure the Emergency Stop and power-down provision is readily accessible, in case the process goes out of control. Damage to equipment and/or serious injury to personnel can result from loss of control of some processes.

Step 10: Save Parameters

When the loop tests and tuning sessions are complete, be sure to save all loop setup parameters to disk.

PID Loop Setup

Some Things to Do and Know Before Starting

Have your analog module installed and operational before beginning the loop setup (refer to the DL05/06 Option Modules User Manual, D0-OPTIONS-M). The DL06 PLC gets its PID loop processing instructions from V-memory tables. There isn't a PID instruction that can be used in RLL, such as a block, to setup the PID loop control. Instead, the CPU reads the setup parameters from system V-memory locations. These locations are shown in the table below for reference only; they can be used in a RLL program if needed.

Address	Setup Parameter	Data type	Ranges	Read/Write
V7640	Loop Parameter Table Pointer	Octal	V1200 – V7340 V10000-V17740	write
V7641	Number of Loops	BCD	1 – 8	write
V7642	Loop Error Flags	BITS	0 or 1	read

NOTE: *The V-memory data is stored in SRAM memory. If power is removed from the CPU for an extended period of time, the PID Setup Parameters will be lost. It is recommended to use the optional battery backup to retain the memory in SRAM. Another option is to use the MOV instruction, which places the data in non-volatile memory, when setting up the parameters in the ladder program.*

8

PID Error Flags

The CPU reports any programming errors of the setup parameters in V7640 and V7641. It does this by setting the appropriate bits in V7642 on program-to-run mode transitions.

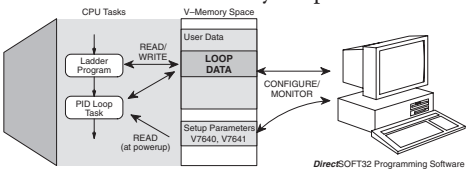
If you use the ***Direct***SOFT loop setup dialog box, its automatic range checking prohibits possible setup errors. However, the setup parameters may be written using other methods such as RLL, so the error flag register may be helpful in those cases. The following table lists the errors reported in V7642.

As a quick check, if the CPU is in Run mode and V7642=0000, there are no programming errors.

Bit	Error Description (0 = no error, 1 = error)
0	The starting address (in V7640) is out of the lower V-memory range.
1	The starting address (in V7640) is out of the upper V-memory range.
2	The number of loops selected (in V7641) is greater than 8.
3	The loop table extends past (straddles) the boundary at V7377. Use an address closer to V1200.

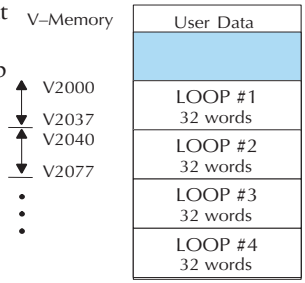
Establishing the Loop Table Size and Location

On a PROGRAM-to-RUN mode transition, the CPU reads the loop setup parameters as pictured below. At that moment, the CPU learns the location of the loop table and the number of loops it configures. Then during the ladder program scan, the PID Loop task uses the loop data to perform calculations, generate alarms, and so on. There are some loop table parameters the CPU will read or write on every loop calculation.



***NOTE**: The DL06 CPU's PID algorithm requires **Direct**SOFT Version 5.0 (or later) and firmware version 2.1 (or later). See our website for more information: **www.automationdirect.com**.*

The Loop Table contains data for only the number of loops that are selected. The address for the table is stored in V7641. Each loop configuration occupies 32 words (0 to 37 octal) in the loop table.



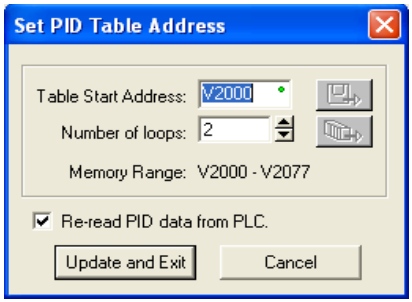
For example, consider an application with 4 loops, and V2000 has been chosen as the starting location. The Loop Parameter will occupy V2000 – V2037 for loop 1, V2040 – V2077 for loop 2 and so on. Loop 4 occupies V2140 - V2177.

Determine the block of V-memory to be used for each PID loop. Besides being the beginning of the PID parameter memory block, the first address will be the start of loop 1 parameters. Remember, there are 32 words (0 to 37 octal) needed for each loop. Once you have determined the beginning V-memory address to be used, you can setup and store the PID parameters either directly in your RLL program or by the using PID Setup in *Direct*SOFT.

NOTE:** Whether one or more loops are being setup, this block of V-memory will only be used for the PID loop parameters, **do not use this block of memory for anything else in your program.

8

Using *Direct*SOFT is the simplest way to setup the parameters. To setup the PID parameters, the DL06 must be powered up and connected to the programming computer. The parameters can only be entered in PID setup when the PLC is in the Program mode. Once the parameters have been entered and saved for each loop, changes made through the PID setup can be made, but only in Program Mode. You can type the beginning address in the PID Table Address dialog found when the PID Setup is opened in *Direct*SOFT. This can be seen in the diagram below. After the address has been entered, the memory range will appear. Also, entering the number of PID loops (1 to 8) will set the total V-memory range for the number of loops entered. After the V-memory address has been entered, the necessary PID parameters for a basic loop operation for each loop can be setup with the dialogs made available.



***NOTE:** Have an edited program open, then click on PLC > Setup > PID to access the Setup PID dialog.*

Loop Table Word Definitions

These are the loop parameters associated with each of the four loops available in the DL06. The parameters are listed in the following table. The address offset is in octal, to help you locate specific parameters in the loop table. For example, if a table begins at V2000, then the location of the reset (integral) term is Addr+11, or V2011. Do not use the Word # (in the first column) to calculate addresses.

Word #	Address+Offset	Description	Format	Read on-the-fly***
1	Addr + 0	PID Loop Mode Setting 1	bits	Yes
2	Addr + 1	PID Loop Mode Setting 2	bits	Yes
3	Addr + 2	Setpoint Value (SP)	word/binary	Yes
4	Addr + 3	Process Variable (PV)	word/binary	Yes
5	Addr + 4	Bias (Integrator) Value	word/binary	Yes
6	Addr + 5	Control Output Value	word/binary	Yes
7	Addr + 6	Loop Mode and Alarm Status	bits	–
8	Addr + 7	Sample Rate Setting	word/BCD	Yes
9	Addr + 10	Gain (Proportional) Setting	word/BCD	Yes
10	Addr + 11	Reset (Integral) Time Setting	word/BCD	Yes
11	Addr + 12	Rate (Derivative) Time Setting	word/BCD	Yes
12	Addr + 13	PV Value, Low-low Alarm	word/binary	No*
13	Addr + 14	PV Value, Low Alarm	word/binary	No*
14	Addr + 15	PV Value, High Alarm	word/binary	No*
15	Addr + 16	PV Value, High-high Alarm	word/binary	No*
16	Addr + 17	PV Value, deviation alarm (YELLOW)	word/binary	No*
17	Addr + 20	PV Value, deviation alarm (RED)	word/binary	No*
18	Addr + 21	PV Value, rate-of-change alarm	word/binary	No*
19	Addr + 22	PV Value, alarm hysteresis setting	word/binary	No*
20	Addr + 23	PV Value, error deadband setting	word/binary	Yes
21	Addr + 24	PV low-pass filter constant	word/BCD	Yes
22	Addr + 25	Loop derivative gain limiting factor setting	word/BCD	No**
23	Addr + 26	SP value lower limit setting	word/binary	Yes
24	Addr + 27	SP value upper limit setting	word/binary	Yes
25	Addr + 30	Control output value lower limit setting	word/binary	No**
26	Addr + 31	Control output value upper limit setting	word/binary	No**
27	Addr + 32	Remote SP Value V-Memory Address Pointer	word/hex	Yes
28	Addr + 33	Ramp/Soak Setting Flag	bit	Yes
29	Addr + 34	Ramp/Soak Programming Table Starting Address	word/hex	No**
30	Addr + 35	Ramp/Soak Programming Table Error Flags	bits	No**
31	Addr + 36	PV auto transfer, channel number	word/hex	Yes
32	Addr + 37	Control output auto transfer, channel number	word/hex	Yes

* Read data only when alarm enable bit transitions from 0 to 1.
** Read data only on PLC Mode change.
*** Read on-the-fly means that the content of V-memory can be changed while the PID loop is in operation.

PID Mode Setting 1 Bit Descriptions (Addr + 00)

The individual bit definitions of the PID Mode Setting 1 word (Addr+00) are listed in the following table.

Bit	PID Mode Setting 1 Description	Read/Write	Bit=0	Bit=1
0	Manual Mode Loop Operation request	write	–	01 request
1	Automatic Mode Loop Operation request	write	–	01 request
2	Cascade Mode Loop Operation request	write	–	01 request
3	Bumpless Transfer select	write	Mode I	Mode II
4	Direct or Reverse-Acting Loop select	write	Direct	Reverse
5	Position / Velocity Algorithm select	write	Position	Velocity
6	PV Linear / Square Root Extract select	write	Linear	Sq. root
7	Error Term Linear / Squared select	write	Linear	Squared
8	Error Deadband enable	write	Disable	Enable
9	Derivative Gain Limit select	write	Off	On
10	Bias (Integrator) Freeze select	write	Off	On
11	Ramp/Soak Operation select	write	Off	On
12	PV Alarm Monitor select	write	Off	On
13	PV Deviation alarm select	write	Off	On
14	PV rate-of-change alarm select	write	Off	On
15	Loop mode is independent from CPU mode when set	write	Loop with CPU mode	Loop Independent of CPU mode

PID Mode Setting 2 Bit Descriptions (Addr + 01)

The individual bit definitions of the PID Mode Setting 2 word (Addr+01) are listed in the following table.

Bit	PID Mode 2 Word Description	Read/Write	Bit=0	Bit=1
0	Input (PV) and Control Output Range Unipolar/Bipolar select (See Notes 2 and 3)	write	unipolar	bipolar
1	Input/Output Data Format select (See Notes 2 and 3)	write	12 bit	15 bit
2	Analog Input filter	write	off	on
3	SP Input limit enable	write	disable	enable
4	Integral Gain (Reset) units select	write	seconds	minutes
5	Select Auto tune PID algorithm	write	closed loop	open loop
6	Auto tune selection	write	PID	PI only (rate = 0)
7	Auto tune start (See Note 1)	read/write	auto tune cancel/done	force start
8	PID Scan Clock (internal use)	read	–	–
9	Input/Output Data Format 16-bit select (See Notes 1 and 2)	write	not 16 bit	select 16 bit
10	Select separate data format for input and output (See Notes 2, and 3)	write	same format	separate formats
11	Control Output Range Unipolar/Bipolar select See Notes 2, and 3)	write	unipolar	bipolar
12	Output Data Format select (See Notes 2, and 3)	write	12 bit	15 bit
13	Output data format 16-bit select (See Notes 2, and 3)	write	not 16 bit	select16 bit
14–15	Reserved for future use	–	–	–

NOTE 1: *Bit 7 can be used to cancel Autotune mode by setting it to 0.*

NOTE 2: *If the value in bit 9 is 0, then the values in bits 0 and 1 are read. If the value in bit 9 is 1, then the values in bits 0 and 1 are not read, and bit 9 defines the data format (the range is automatically unipolar).*

NOTE 3: *If the value in bit 10 is 0, then the values in bits 0, 1 and 9 define the input and output ranges and data formats (the values in bits 11, 12, and 13 are not read). If the value in bit 10 is 1, then the values in bits 0, 1, and 9 define only the input range and data format, and bits 11, 12, and 13 are read and define the output range and data format.*

NOTE 4: *If bit 10 has a value of 1 and bit 13 has a value of 0, then bits 11 and 12 are read and define the output range and data format. If bit 10 and bit 13 each have a value of 1, then bits 11 and 12 are not read, and bit 13 defines the data format, (the output range is automatically unipolar).*

Mode/Alarm Monitoring Word (Addr + 06)

The individual bit definitions of the Mode / Alarm monitoring (Addr+06) word are listed in the following table.

Bit	Mode/Alarm Bit Description	Read/Write	Bit=0	Bit=1
0	Manual Mode Indication	read	–	Manual
1	Automatic Mode Indication	read	–	Auto
2	Cascade Mode Indication	read	–	Cascade
3	PV Input LOW–LOW Alarm	read	Off	On
4	PV Input LOW Alarm	read	Off	On
5	PV Input HIGH Alarm	read	Off	On
6	PV Input HIGH–HIGH Alarm	read	Off	On
7	PV Input YELLOW Deviation Alarm	read	Off	On
8	PV Input RED Deviation Alarm	read	Off	On
9	PV Input Rate-of-Change Alarm	read	Off	On
10	Alarm Value Programming Error	read	–	Error
11	Loop Calculation Overflow/Underflow	read	–	Error
12	Loop in Auto-Tune indication	read	Off	On
13	Auto-Tune error indication	read	–	Error
14–15	Reserved for Future Use	–	–	–

Ramp/Soak Table Flags (Addr + 33)

The individual bit definitions of the Ramp/Soak Table Flag (Addr+33) word are listed in the following table.

Bit	Ramp/Soak Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Start Ramp / Soak Profile	write	–	01 Start
1	Hold Ramp / Soak Profile	write	–	01 Hold
2	Resume Ramp / soak Profile	write	–	01 Resume
3	Jog Ramp / Soak Profile	write	–	01 Jog
4	Ramp / Soak Profile Complete	read	–	Complete
5	PV Input Ramp / Soak Deviation	read	Off	On
6	Ramp / Soak Profile in Hold	read	Off	On
7	Reserved	read	–	–
8–15	Current Step in R/S Profile	read	decode as byte (hex)	

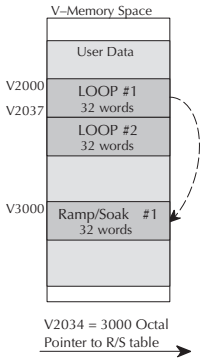
Bits 8–15 must be read as a byte to indicate the current segment number of the Ramp/Soak generator in the profile. This byte will have the values 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, and 10, which represent segments 1 to 16 respectively. If the byte=0, then the Ramp/Soak table is not active.

Ramp/Soak Table Location (Addr + 34)

Each loop that you configure has the option of using a built-in Ramp/Soak generator dedicated to that loop. This feature generates SP values that follow a profile. To use the Ramp Soak feature, you must program a separate table of 32 words with appropriate values. The *Direct*SOFT dialog box makes this easy to do.

In the loop table, the Ramp/Soak Table Pointer at Addr+34 must point to the start of the ramp/soak data for that loop. This may be anywhere in user memory, and does not have to adjoin to the Loop Parameter table, as shown to the left. Each R/S table requires 32 words, regardless of the number of segments programmed.

The ramp/soak table parameters are defined in the table below. Further details are in the section on Ramp/Soak Operation in this chapter.



Addr Offset	Step	Description	Addr Offset	Step	Description
+ 00	1	Ramp End SP Value	+ 20	9	Ramp End SP Value
+ 01	1	Ramp Slope	+ 21	9	Ramp Slope
+ 02	2	Soak Duration	+ 22	10	Soak Duration
+ 03	2	Soak PV Deviation	+ 23	10	Soak PV Deviation
+ 04	3	Ramp End SP Value	+ 24	11	Ramp End SP Value
+ 05	3	Ramp Slope	+ 25	11	Ramp Slope
+ 06	4	Soak Duration	+ 26	12	Soak Duration
+ 07	4	Soak PV Deviation	+ 27	12	Soak PV Deviation
+ 10	5	Ramp End SP Value	+ 30	13	Ramp End SP Value
+ 11	5	Ramp Slope	+ 31	13	Ramp Slope
+ 12	6	Soak Duration	+ 32	14	Soak Duration
+ 13	6	Soak PV Deviation	+ 33	14	Soak PV Deviation
+ 14	7	Ramp End SP Value	+ 34	15	Ramp End SP Value
+ 15	7	Ramp Slope	+ 35	15	Ramp Slope
+ 16	8	Soak Duration	+ 36	16	Soak Duration
+ 17	8	Soak PV Deviation	+ 37	16	Soak PV Deviation

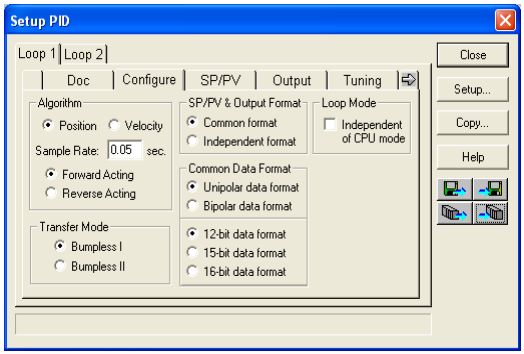
Ramp/Soak Table Programming Error Flags (Addr + 35)

The individual bit definitions of the Ramp/Soak Table **Programming Error Flags** word (Addr+35) are listed in the following table. Further details are given in the PID Loop Mode section and in the PV Alarm section later in chapter 8.

Bit	R/S Error Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Starting Addr out of lower V-memory range	read	–	Error
1	Starting Addr out of upper V-memory range	read	–	Error
2–3	Reserved for Future Use	–	–	–
4	Starting Addr in System Parameter V-memory Range	read	–	Error
5–15	Reserved for Future Use	–	–	–

Configure the PID Loop

Once the PID table is established in V-memory, configuring the PID loop continues with the *Direct*SOFT PID setup configuration dialog. You will need to check and fill in the data required to control the PID loop. Select Configure and the following dialog will appear for this process.

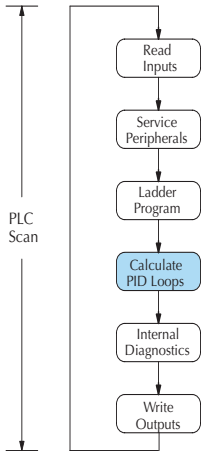


Select the Algorithm Type

Chose either *Position* or *Velocity*. The default algorithm is Position. This is the choice for most applications which include heating and cooling loops as well as most position and level control loops. A typical velocity control will consist of a process variable such as a flow totalizer in a flow control loop.

Enter the Sample Rate

The main tasks of the CPU fall into categories as shown to the right. The list represents the tasks done when the CPU is in Run Mode, on each PLC scan. Note that PID loop calculations occur after the ladder logic task.



The *sample rate* of a control loop is simply the frequency of the PID calculation. Each calculation generates a new control output value. With the DL06 CPU, you can set the sample rate of a loop from 50 ms to 99.99 seconds. Most loops do not require a fresh PID calculation on every PLC scan. Some loops may need to be calculated only once in 1000 scans.

Enter 0.05 sec., or the sample rate of your choice, for each loop, and the CPU automatically schedules and executes PID calculations on the appropriate scans.

NOTE: *If more than 4 loops are programmed, enter a minimum of 0.1 second.*

Select Forward/Reverse

It is important to know which direction the control output will respond to the error (SP-PV), either *forward* or *reverse.* A forward (direct) acting control loop means that whenever the control output increases, the process variable will also increase. The control outputs of most PID loops are forward acting, such as a heating control loop. An increase in heat applied will increase the PV (temperature).

A reverse acting control loop is one where an increase in the control output results in a decrease in the PV. A common example of this would be a refrigeration system, where an increase in the cooling input causes a decrease in the PV (temperature).

The Transfer Mode

Choose either Bumpless I or Bumpless II to provide a smooth transition of the control output from Manual Mode to Auto Mode. Choosing Bumpless I will set the SP equal to the PV when the control output is switched from Manual to Auto. If this is not desired, choose BumplessII.

The characteristics of Bumpless I and II transfer types are listed in the chart below. Note that their operation also depends on which PID algorithm you are using, the position or velocity form of the PID equation. Note that you must use Bumpless Transfer type I when using the velocity form of the PID algorithm.

Transfer Type	Transfer Select Bit 3	PID Algorithm	Manual-to-Auto Transfer Action	Auto-to-Cascade Transfer Action
Bumpless Transfer I	0	Position	Forces Bias = Control Output Forces SP = PV	Forces Major Loop Output = Minor Loop PV
		Velocity	Forces SP = PV	Forces Major Loop Output = Minor Loop PV
Bumpless Transfer II	1	Position	Forces Bias = Control Output	none
		Velocity	none	none

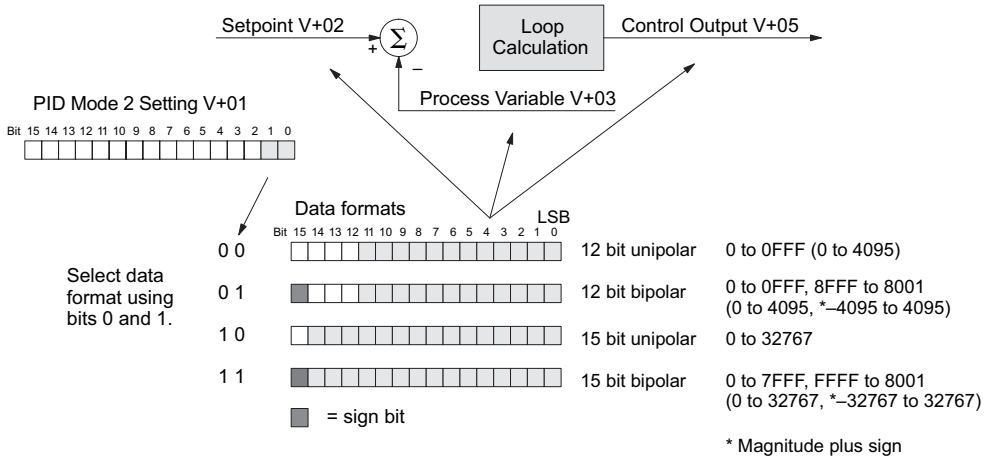
The transfer type can also be selected in an RLL program by setting bit 3 of PID Mode 1, V+00 setting as shown.

SP/PV & Output Format

This block allows you to select either *Common format* or *Independent format*. Common format is the default and is most commonly used. With this format, both SP/PV and Output will have the same data structure. Both will have the same number of bits and either bipolar or unipolar. If Independent format is selected, the data structure selections will be grayed out. The reason for this is that they become independently selectable in the *SP/PV* and the *Output* dialogs.

Common Data Format

Select either *Unipolar data format* (which is positive data only) in 12 bit (0 to 4095), 15 bit (0 to 32767), or 16 bit (0 to 65535) format, or *Bipolar data format,* which ranges from negative to positive (-4095 to 4095 or -32767 to 32767) and requires a sign bit. Bipolar selection displays input/output as magnitude plus sign, not two's complement. The bipolar selection is not available when 16-bit data format is selected.



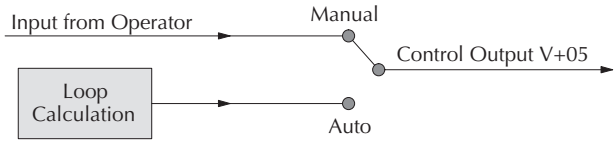
The data format determines the numerical interface between the PID loop and the PV sensorand the control output device. This selects the data format for both the SP and the PV.

Loop Mode

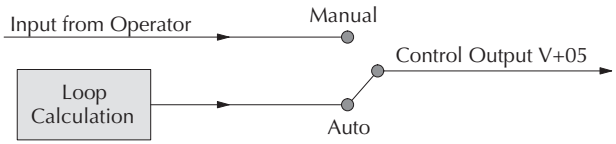
Loop Mode is a special feature that allows the PID loop controller to perform closed-loop control while the CPU is in the Program Mode. Careful thought must be taken before using this feature called *Independent of CPU mode* in the dialog. Before continuing with the PID setup, a knowledge of the three PID loop modes will be helpful.

The DL06 provides the three standard control modes: *Manual, Automatic,* and *Cascade*. The sources of the three basic variables SP, PV and control output are different for each mode.

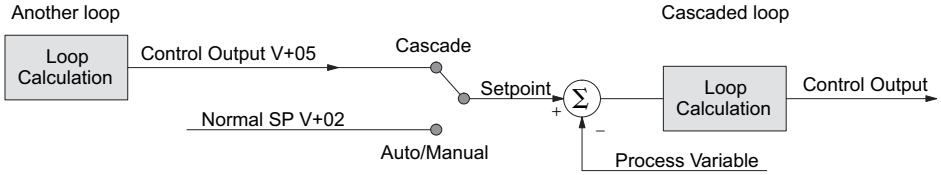
In Manual Mode, the loop is not executing PID calculations (however, loop alarms are still active). With regard to the loop table, the CPU stops writing values to location V+05 (control output) for that loop. *It is expected that an operator or other intelligent source* is manually controlling the output by observing the PV and writing data to the control output as necessary to keep the process under control. The drawing below shows the equivalent schematic diagram of manual mode operation.



In Automatic Mode, the loop operates normally and generates new control output values. It calculates the PID equation and writes the result in location V+05 every sample period of that loop. The equivalent schematic diagram is shown below.

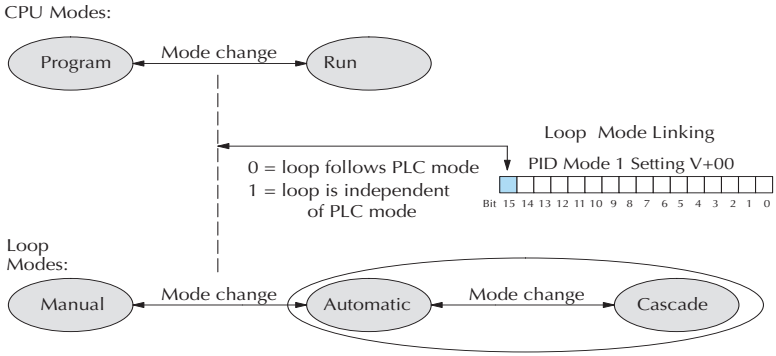


In Cascade Mode, the loop operates as it does in Automatic Mode, with one important difference. The data source for the SP changes from its normal location at V+02 to using the control output value, V+05, from another loop. So in Auto or Manual modes, the loop calculation uses the data at V+02. In Cascade Mode, the loop calculation reads the control output from another loop's parameter table, V+05.



As pictured below, a loop can be changed from one mode to another, but *cannot go from Manual Mode directly to Cascade, or vice versa*. This mode change is prohibited because a loop would be changing two data sources at the same time, and could cause a loss of control.

Once the CPU is operating in the Run Mode, the normal operation of the PID loop controller is to read the loop data and perform calculations on each scan of the RLL program. When the CPU is placed in the Program Mode, the RLL program halts operation and all PID loops are automatically put into the Manual Mode. The PID parameters can then be changed if desired. Similarly, by placing the CPU in the Run mode, the PID loops are returned to the operational mode which they were previously in, i.e., Manual, Automatic and Cascade. With this selection you automatically affect the modes by changing the CPU mode.



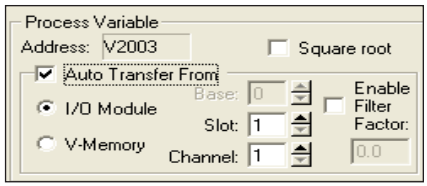
If bit 15 is set to one, then the loops will run independently of the CPU mode. It is like having two independent processors in the CPU... one is running the RLL program and the other is running the process loops.

Having the ability to run loops independently of the RLL program makes it feasible to make a ladder logic change while the process is still running. This is especially beneficial for large-mass continuous processes that are difficult or costly to interrupt. The independent of CPU is the feature used for this.

If you need to operate the PID loops while the RLL program is halted, in Program Mode, either select the Independent of CPU mode in the dialog or edit your program to set and reset bit 15 of PID Mode 1 word (V+00) in your RLL program. If the bit is set to a zero, the loop will follow the CPU mode; then, when the CPU is placed in the Program Mode, all loops will be forced into the Manual Mode.

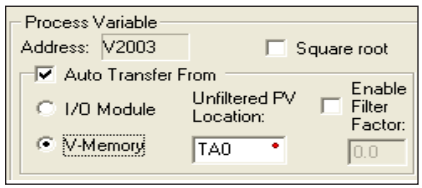
When Independent of CPU mode is used, you should also set the PV to be read directly from an analog input module. This can easily be done in the PID setup dialog, SP/PV.

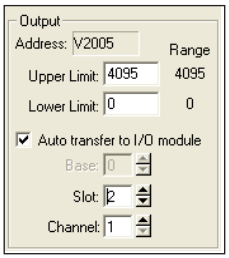
The SP/PV dialog has a block entitled *Process Variable*. There is a block within this block called *Auto Transfer From* (from analog input) with the information grayed out. Checking the box to the left of the Auto Transfer From will highlight the information. Select *I/O Module* then enter the slot number in which the input module resides. Next, select the analog input channel of your choice.



The second choice is *V-Memory*. When this is selected, the V-memory address from where the PV is transferred must be specified.

Whichever method of auto transfer is used, it is recommended to check the *Enable Filter Factor* (a low pass filter) and specify the coefficient.

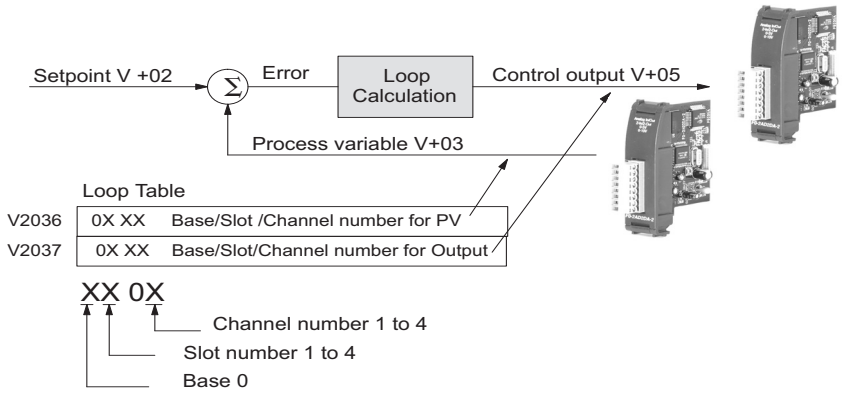




You should also select the analog output for the control output to be transferred to. This is done in the PID setup *Output* dialog shown here. The block of information in this dialog is grayed-out until the box next to *Auto transfer to I/O module* is checked. Once checked, enter the slot number where the output module is residing and then enter the analog output channel number.

NOTE: *To make changes to any loop table parameters, the PID loop must be in Manual mode and the PLC must be stopped. If you have selected to operate the PID loop independent of the CPU mode, then you must take certain steps to make it possible to make loop parameter changes. You can temporarily make the loops follow the CPU mode by changing bit 15 to 0. Then, you will be able to place the loop into Manual Mode using* ***Direct****SOFT. After you change the loop's parameter settings, restore bit 15 to a value of 1 to re-establish PID operation independent of CPU.*

You may optionally configure each loop to access its analog I/O (PV and control output) by placing proper values in the associated loop table registers in your RLL program. The following figure shows the loop table parameters at V+36 and V+37 and their auto transfer role to access the analog values directly.

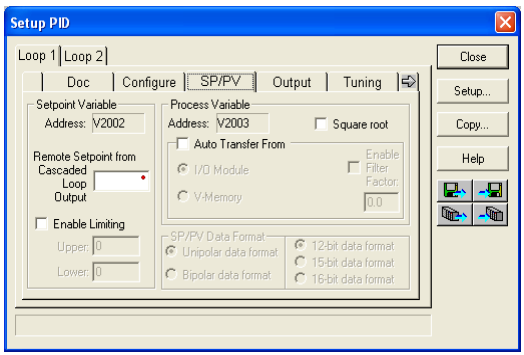


When these loop table parameters are programmed directly, a value of **0102** in register V2036 directs the loop controller to read the PV data from channel 1 of the analog input. A value of **0000** in either register tells the loop controller not to access the corresponding analog value directly. In that case, ladder logic must be used to transfer the value between the analog input and the loop table.

NOTE: *When auto transfer to/from I/O is used, the analog data for all of the channels on the analog module cannot be accessed by any other method, i.e., pointer or multiplex.*

SP/PV Addresses

An SP/PV dialog will be made available to setup how the setpoint (SP) and the process variable (PV) will be used in the loop. If this loop is the minor loop of a cascaded pair, enter that control output address in the *Remote SP from Cascaded Loop Output* area. It is sometimes desirable to limit the range of setpoint values allowed to be entered. To activate this feature, check the box next to *Enable Limiting*. This will activate the *Upper* and *Lower* fields for the values to be entered. Set the limits around the SP value to prevent an operator from entering

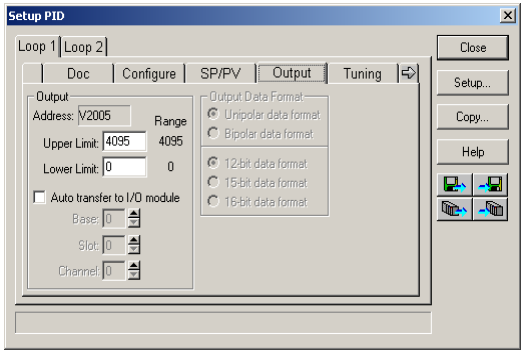


a setpoint value outside of a safe range. The *Square root* box is only checked for certain PID loops, such as a flow control loop. If the *Auto transfer from I/O module* is selected, a first-order low-pass filter can be used by checking the *Enable Filter* box. The filter coefficient is user specified. The use of this filter is recommended during closed loop auto-tuning. If the Independent format had been checked previously, make the Data format selections here.

NOTE: *The SP/PV dialog can be left as it first appears for basic PID operation.*

Set Control Output Limits

Another dialog that will be available in the PID setup will be the Output dialog. The control output address, V+05, (determined by the PID loop table beginning address) will be in view. Enter the output range limits, *Upper Limit* and *Lower Limit,* that will meet the requirement of the process and which will agree with the data format that has been selected. For a basic PID operation using a 12-bit output module, set the Upper Limit to 4095 and leave the

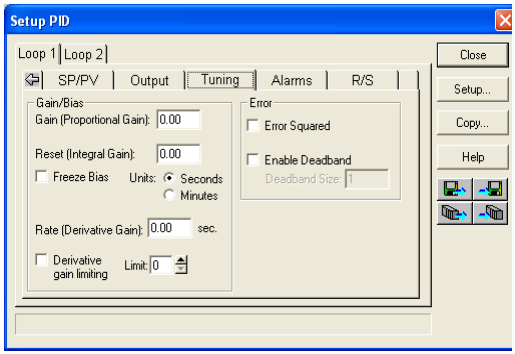


Lower Limit set to 0. Check the box next for *Auto transfer to I/O module* if there is a need to send the control output to a certain analog output module, as in the case of using the Loop Mode independent of CPU Mode; otherwise, the PID output signal cannot control the analog output when the PLC is not in RUN Mode. If the *Auto transfer to I/O module* feature is checked, all channels of the module must be used for PID control outputs. If Independent format has been previously chosen, the *Output Data Format* will need to be setup here, that is, select Unipolar or Bipolar format and the bit structure. This area is not available and is grayed out if *Common format* has been chosen (see page 8-26).

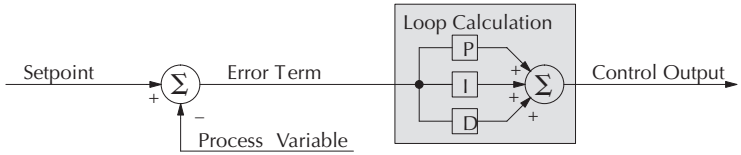
WARNING: If the Upper Limit is set to zero, the output will never get above zero. In effect, there will be no control output.

Enter PID Parameters

Another PID setup dialog, Tuning, is for entering the PID parameters shown as: Gain (Proportional Gain), Reset (Integral Gain) and Rate (Derivative Gain).



Recall the position and velocity forms of the PID loop equations which were introduced earlier. The equations basically show the three components of the PID calculation: Proportional Gain (P), Integral Gain (I) and Derivative Gain (D). The following diagram shows a form of the PID calculation in which the control output is the sum of the proportional gain, integral gain and derivative gain. With each calculation of the loop, each term receives the same error signal value.



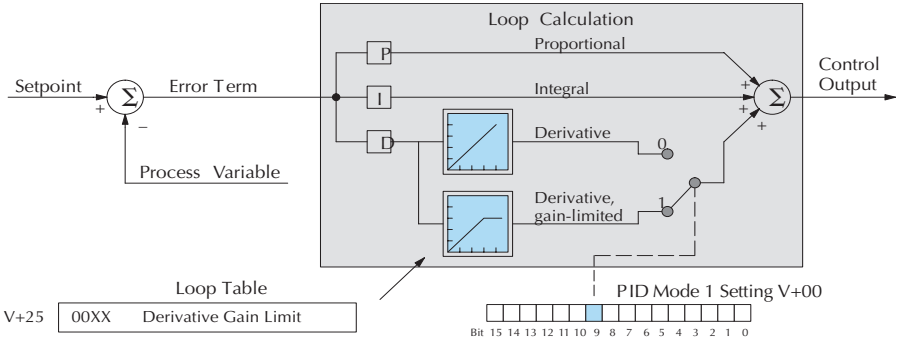
The P, I and D gains are 4-digit BCD numbers with values from 0000 to 9999. They contain an implied decimal point in the middle, so the values are actually 00.00 to 99.99. Some gain values have units – Proportional gain has no unit, Integral gain may be selected in seconds or in minutes, and Derivative gain is in seconds.

Gain (Proportional Gain) – This is the most basic gain of the three. Values range from 0000 to 9999, but they are used internally as xx.xx. An entry of "0000" effectively removes the proportional term from the PID equation. This accommodates applications which need integral-only loops.

Reset (Integral Gain) – Values range from 0001 to 9998, but they are used internally as xx.xx. An entry of "0000" or "9999" causes the integral gain to be "infinity", effectively removing the integrator term from the PID equation. This accommodates applications which need proportional-only loops. The units of integral gain may be either seconds or minutes, as shown in the above dialog.

Rate (Derivative Gain) – Values which can be entered range from 0001 to 9999, but they are used internally as XX.XX. An entry of "0000" allows removal of the derivative term from the PID equation (a common practice). This accommodates applications which require only proportional and/or integral loops. Most control loops will operate as a PI loop.

NOTE: *You may elect to leave the tuning dialog blank and enter the tuning parameters in the* ***Direct****SOFT PID View.*



Derivative Gain Limiting

The derivative gain (rate) has an optional gain-limiting feature. This is provided because the derivative gain reacts badly to PV signal noise or other causes of sudden PV fluctuations. The function of the gain-limiting is shown in the diagram below.

The gain limit can be particularly useful during loop tuning. Most loops can tolerate only a little derivative gain without going into uncontrolled oscillations.

If this option is checked, a *Limit* from 0 to 20 must also be entered.

NOTE*: When first configuring a loop, it's best to use the standard error term until after the loop is tuned. Once the loop is tuned, you will be able to tell if these functions will enhance control. The Error Squared and/or Enable Deadband can be selected later in the PID setup. Also, values are not required to be entered in the Tuning dialog, but they can set later in the* ***Direct****SOFT PID View.*

Error Term Selection

The error term is internal to the CPUs PID loop controller, and is generated again in each PID calculation. Although its data is not directly accessible, you can easily calculate it by subtracting: Error = (SP–PV). If the PV square-root extract is enabled, then: Error = $\sqrt{}$PV In any case, the size of the error and algebraic sign determine the next change of the control output for each PID calculation.

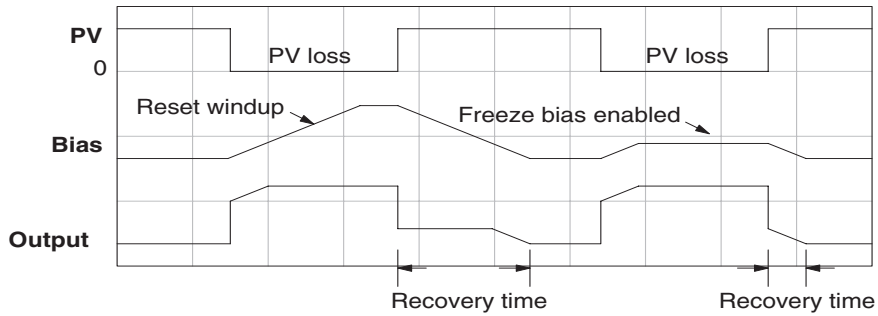
Error Squared – When selected, the squared error function simply squares the error term (but preserves the original algebraic sign), which is used in the calculation. This affects the Control Output by diminishing its response to smaller error values, but maintaining its response to larger errors. Some situations in which the error squared term might be useful:

- Noisy PV signal – using a squared error term can reduce the effect of low-frequency electrical noise on the PV, which will make the control system jittery. A squared error maintains the response to larger errors.
- Non-linear process – some processes (such as chemical pH control) require non-linear controllers for best results. Another application is surge tank control, where the Control Output signal must be smooth.

Enable Deadband – When selected, the enable deadband function takes a range of small error values near zero, and simply substitutes zero as the value of the error. If the error is larger than the deadband range, then the error value is used normally.

Freeze Bias

The term *reset windup* refers to an undesirable characteristic of integrator behavior which occurs naturally under certain conditions. Refer to the figure below. Suppose the PV signal becomes disconnected, and the PV value goes to zero. While this is a serious loop fault, it is made worse by reset windup. Notice the bias (reset) term keeps integrating normally during the PV disconnect, until its upper limit is reached. When the PV signal returns, the bias value is saturated (windup) and takes a long time to return to normal. The loop output consequently has an extended recovery time. Until recovery, the output level is wrong and causes further problems.



In the second PV signal loss episode in the figure, the freeze bias feature is enabled. It causes the bias value to freeze when the control output goes out of bounds. Much of the reset windup is thus avoided, and the output recovery time is much less.

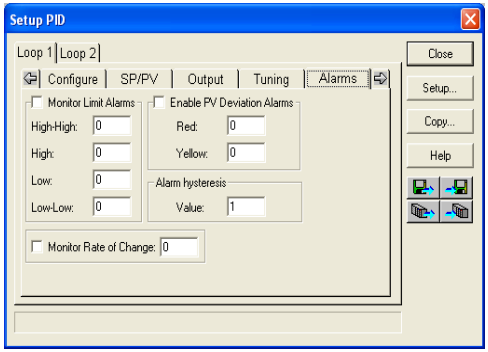
For most applications, the freeze bias feature will work with the loop as described above. It is suggested to enable this feature by selecting it in the dialog. Bit 10 of PID Mode 1 Setting (V+00) word can also be set in RLL.

NOTE: *The freeze bias feature stops the bias term from changing when the control output reaches the end of the data range. If you have set limits on the control output other than the range (i.e, 0–4095 for a unipolar/12-bit loop), the bias term still uses the end of range for the stopping point and bias freeze will not work.*

Setup the PID Alarms

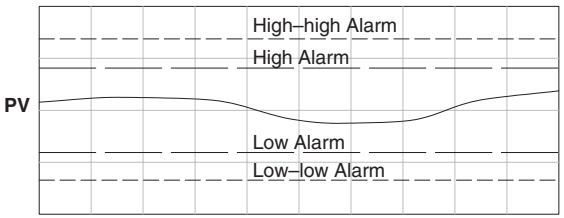
Although the setup of the PID alarms is optional, you surely would not want to operate a process without monitoring it. The performance of a process control loop may generally be measured by how closely the process variable matches the setpoint. Most process control loops in industry operate continuously, and will eventually lose control of the PV due to an error condition. Process alarms are vital in early discovery of a loop error condition and can alert plant personnel to manually control a loop or take other measures until the error condition has been repaired.

The alarm thresholds are fully programmable, and each type of alarm may be independently enabled and monitored. The following diagram shows the Alarm dialog in the PID setup which simplifies the alarm setup.



Monitor Limit Alarms

Checking this box will allow all of the PV limit alarms to be monitored once the limits are entered. The PV absolute value alarms are organized as two upper and two lower alarms. The alarm status is false as long as the PV value remains in the region between the upper and lower alarms, as shown below. The alarms nearest the safe zone are named *High Alarm* and *Low Alarm.* If the loop loses control, the PV will cross one of these thresholds first. Therefore, you can program the appropriate alarm threshold values in the loop table locations shown below to the right. The data format is the same as the PV and SP (12-bit or 15-bit). The threshold values for these alarms should be set to give an operator an early warning if the process loses control.



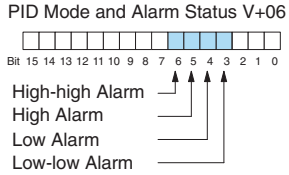
Loop Table

V+16	XXXX	High-high Alarm
V+15	XXXX	High Alarm
V+14	XXXX	Low Alarm
V+13	XXXX	Low-low Alarm

NOTE: *The Alarm dialog can be left as it first appears, without alarm entries. The alarms can then be setup in the* ***Direct****SOFT PID View.*

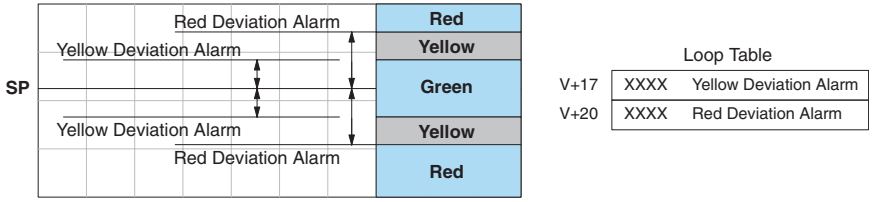
If the process remains out of control for some time, the PV will eventually cross one of the outer alarm thresholds, named High-high alarm and Low-low alarm. Their threshold values are programmed using the loop table registers listed above. A High-high or Low-low alarm indicates a serious condition exists, and needs the immediate attention of the operator.

The PV Absolute Value Alarms are reported in the four bits in the PID Mode and Alarm Status word in the loop table, as shown to the right. We highly recommend using ladder logic to monitor these bits. The bit-of-word instructions make this easy to do. Additionally, you can monitor PID alarms using *Direct*SOFT.



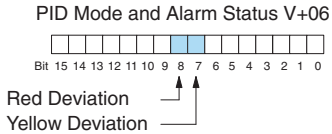
PV Deviation Alarms

The PV Deviation Alarms monitor the PV deviation with respect to the SP value. The deviation alarm has two programmable thresholds, and each threshold is applied equally above and below the current SP value. In the figure below, the smaller deviation alarm is called the Y**ellow Deviation**, indicating a cautionary condition for the loop. The larger deviation alarm is called the **Red Deviation**, indicating a strong error condition for the loop. The threshold values use the loop parameter table locations V+17 and V+20 as shown.



The thresholds define zones, which fluctuate with the SP value. The green zone which surrounds the SP value represents a safe (no alarm) condition. The yellow zones lie outside the green zone, and the red zones are beyond those.

The PV Deviation Alarms are reported in the two bits in the PID Mode and Alarm Status word in the loop table, as shown to the right. We highly recommend using ladder logic to monitor these bits. The bit-of-word instructions make this easy to do. Additionally, you can monitor PID alarms using *Direct*SOFT.



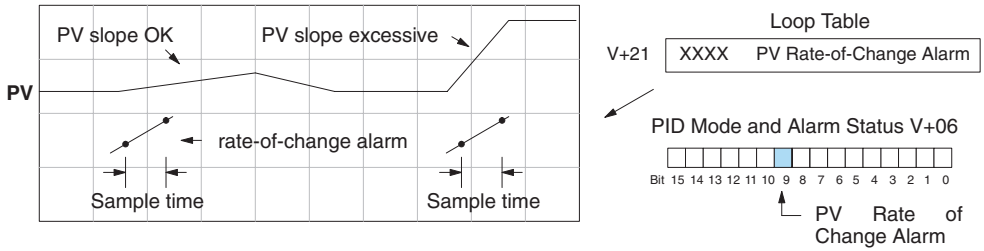
The PV Deviation Alarm can be independently enabled and disabled from the other PV alarms, using bit 13 of the PID Mode 1 Setting V+00 word.

Remember the alarm hysteresis feature works in conjunction with both the deviation and absolute value alarms, and is discussed at the end of this section.

PV Rate-of-Change Alarm

An excellent way to get an early warning of a process fault is to monitor the *rate-of-change* of the PV. Most batch processes have large masses and slowly-changing PV values. A relatively fast-changing PV will result from a broken signal wire for either the PV or control output, a SP value error, or other causes. If the operator responds to a PV Rate-of-Change Alarm quickly and effectively, the PV absolute value will not reach the point where the material in process would be ruined.

The DL06 loop controller provides a programmable PV Rate-of-Change Alarm, as shown below. The rate-of-change is specified in PV units change per loop sample time. This value is programmed into the loop table location V+21.



As an example, suppose the PV is the temperature for your process, and you want an alarm whenever the temperature changes faster than 15 degrees/minute. The PV counts per degree and the loop sample rate must be known. Then, suppose the PV value (in V+03 location) represents 10 counts per degree, and the loop sample rate is 2 seconds. Use the formula below to convert our engineering units to counts/sample period:

$$\text{Alarm Rate-of-Change} = \frac{15 \text{ degrees}}{1 \text{ minute}} \times \frac{10 \text{ counts / degree}}{30 \text{ loop samples / min.}} = \frac{150}{30} = 5 \text{ counts / sample period}$$

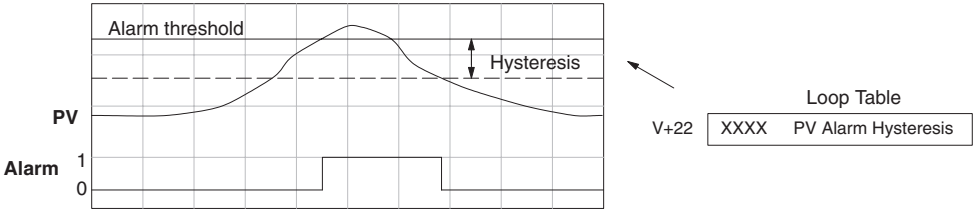
From the calculation result, you would program the value 5 in the loop table for the rate-of-change. The PV Rate-of-Change Alarm can be independently enabled and disabled from the other PV alarms, using bit 14 of the PID Mode 1 Setting V+00 word.

The alarm hysteresis feature (discussed next) does not affect the Rate-of-Change Alarm.

PV Alarm Hysteresis

The PV Absolute Value Alarm and PV Deviation Alarm are programmed using threshold values. When the absolute value or deviation exceeds the threshold, the alarm status becomes true. Real-world PV signals have some noise on them, which can cause some fluctuation in the PV value in the CPU. As the PV value crosses an alarm threshold, its fluctuations cause the alarm to be intermittent and annoy process operators. The solution is to use the PV Alarm Hysteresis feature.

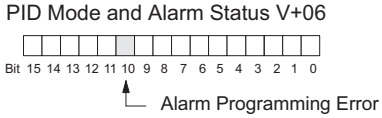
The PV Alarm Hysteresis amount is programmable from 1 to 200 (binary/decimal). When using the PV Deviation Alarm, the programmed hysteresis amount must be less than the programmed deviation amount. The figure below shows how the hysteresis is applied when the PV value goes past a threshold and descends back through it.



The hysteresis amount is applied after the threshold is crossed, and toward the safe zone. In this way, the alarm activates immediately above the programmed threshold value. It delays turning off until the PV value has returned through the threshold by the hysteresis amount.

Alarm Programming Error

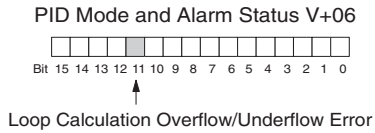
The PV Alarm threshold values must have certain mathematical relationships to be valid. The requirements are listed below. If not met, the Alarm Programming Error bit will be set, as indicated to the right.



- PV Absolute Alarm value requirements:
  Low-low < Low < High < High-high
- PV Deviation Alarm requirements:
  Yellow < Red

Loop Calculation Overflow/Underflow Error

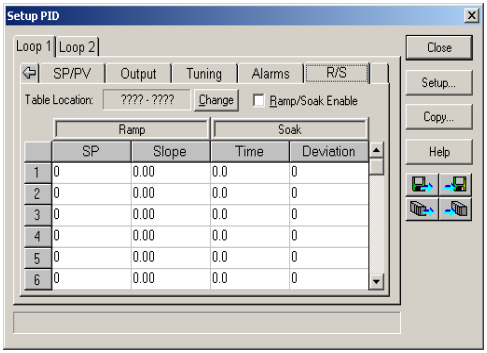
This error occurs whenever the output reaches its upper or lower limit and the PV does not reach the setpoint. A typical example might be when a valve is stuck, the output is at its limit, but the PV has not reached setpoint.



NOTE: *Overflow/underflow can be alarmed in PID View. The optional C-more operator interface panel (see the automationdirect.com website) can also be setup to read these error bits using the PID Faceplate templates.*

Ramp/Soak

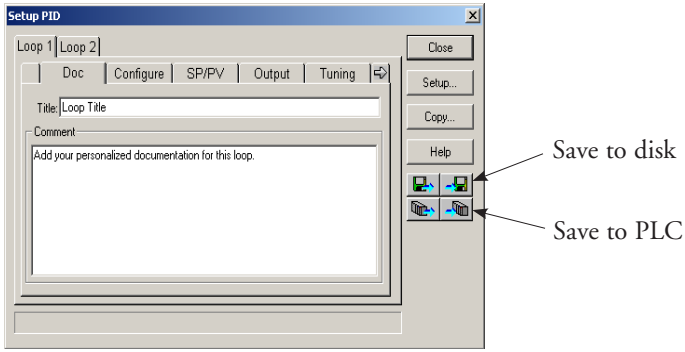
R/S (Ramp/Soak) is the last dialog available in the PID setup. The basic PID does not require any entries to be made in order to operate the PID loop. Ramp/Soak will be discussed in another section.



Complete the PID Setup

Once you have filled in the necessary information for the basic PID setup, the configuration should be saved. The icons on the Setup PID dialog will allow you to save the configuration to the PLC and to disk. The save to icons have the arrow pointing to the PLC and disk. The read from icons have the arrows pointing away from the PLC and disk.

An optional feature is available with the Doc tab in the Setup PID window. You enter a name and description for the loop. This is useful if there is more than one PID loop in your application.



NOTE: *It is good practice to save your project after setting up the PID loop by selecting* ***File*** *from the menu toolbar, then* ***Save project > to disk****. In addition to saving your entire project, all the PID parameters are also saved.*

PID Loop Tuning

Once you have set up a PID loop, it must be tuned in order for it to work. The goal of loop tuning is to adjust the loop gains so the loop has optimal performance in dynamic conditions. The quality of a loop's performance may generally be judged by how well the PV follows the SP after an SP step change. It is important to keep in mind that understanding the process is fundamental to getting a well designed control loop. Sensors must be in appropriate locations and valves must be sized correctly with appropriate trim. **PID control does not have *typical* values**. There isn't one control process that is identical to another.

Manual Tuning vs. Auto Tuning

You may enter the PID gain values to tune your loops (manual tuning), or you can rely on the PID processing engine in the CPU to automatically calculate the gain values (auto tuning). Most experienced process engineers will have a favorite method; the DL06 will accommodate either preference. The use of auto tuning can eliminate much of the trial-and-error of the manual tuning approach, especially if you do not have a lot of loop tuning experience. However, performing the auto tuning procedure will get the gains close to optimal values, but additional manual tuning can get the gain values to their optimal values.

WARNING: Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tune procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL06 is not intended to be used as a replacement for your process knowledge.

Open-Loop Test

Whether you use manual or auto tuning, it is very important to verify basic characteristics of a newly-installed process before attempting to tune it. With the loop in Manual Mode, verify the following items for each new loop.

- **Setpoint** – verify that the SP source can generate a setpoint. Put the PLC in Run Mode and leave the loop in Manual Mode, then monitor the loop table location V+02 to see the SP value(s). (If you are using the ramp/soak generator, test it now).
- **Process Variable** – verify that the PV value is an accurate measurement, and the PV data arriving in the loop table location V+03 is correct. If the PV signal is very noisy, consider filtering the input either through hardware (RC low-pass filter), or using the filter in this chapter.
- **Control Output** – if it is safe to do so, manually change the output a small amount (perhaps 10%) and observe its affect on the process variable. Verify the process is direct-acting or reverse acting, and check the setting for the control output (inverted or non-inverted). Make sure the control output upper and lower limits are not equal to each other.
- **Sample Rate** – while operating open-loop, this is a good time to find the ideal sample rate (see Configure the PID Loop on page 8-25). However, if you are going to use auto tuning, the auto tuning procedure will automatically calculate the sample rate in addition to the PID gains.

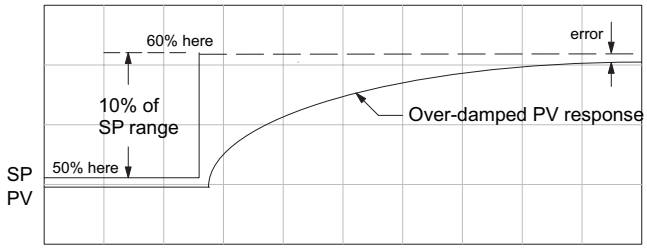
Manual Tuning Procedure

It is not necessary to try to obtain the best values for the P, I and D parameters in the PID loop by trial and error. Following is a typical procedure for tuning a temperature control loop which you may use to tune your loop.

Monitor the values of SP, PV and CV with a loop trending instrument or use the PID View feature in *Direct*SOFT (see page 8-49).

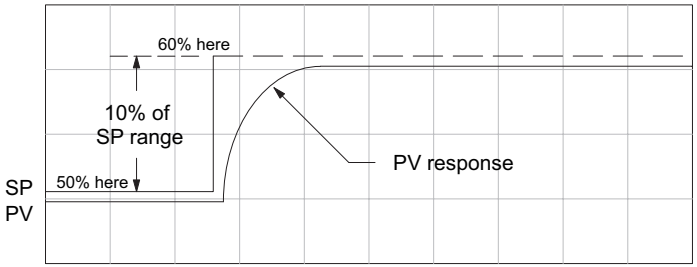
NOTE: *We recommend using the PID View Tuning and Trending window to select manual for the vertical scale feature, for both SP/PV area and Bias/Control Output areas. The auto scaling feature would otherwise change the vertical scale on the process parameters and add confusion to the loop tuning process.*

- Adjust the gains so the Proportional Gain = 0.5 or 1.0 (1.0 is a good value based on experience), Integral Gain = 9999 (this basically eliminates reset) and Derivative Gain = 0000. This disables the integrator and derivative terms, and provides some proportional gain.
- Check the bias value in the PID View and set it to zero.
- Set the SP to a value equal to 50% of the full range.
- Now, select Auto Mode. If the loop will not stay in Auto Mode, check the troubleshooting tips at the end of this chapter. Allow the PV to stabilize around the 50% point of the range.

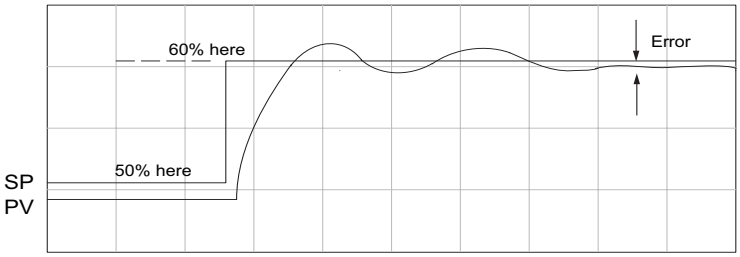


- Change the SP to the 60% point of the range.

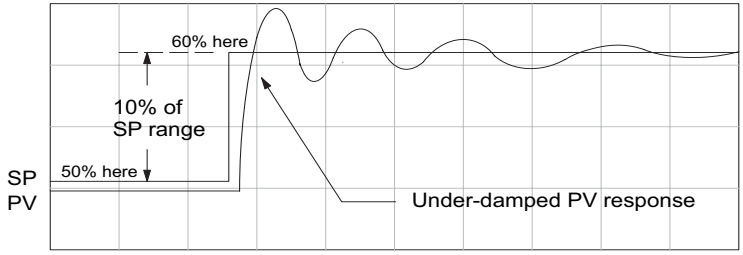
The response may take awhile, but you will see that there isn't any oscillation. This response is not desirable since it takes a long time to correct the error; also, there is a difference between the SP and the PV.



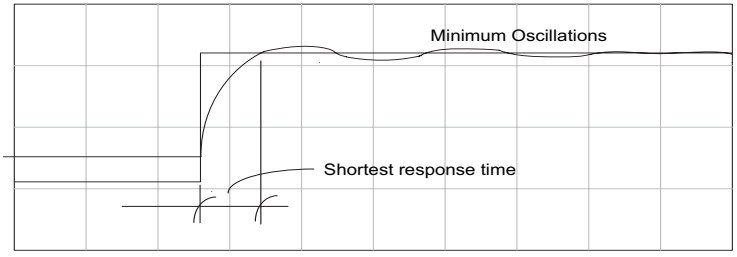
- Increase the Proportional gain, for example to 2.0. The control output will be greater and the response time will be quicker. The trend should resemble the figure below.
- Increase the Proportional gain in small increments, such as 4, 6, 7, etc., until the control output response begins to oscillate. This is the Proportional gain that should be recorded.



- Now, return the Proportional gain to the stable response; for example, 9.7. The error, SP-PV, should be small, but not at zero.
- Next, add a small amount of Integral gain (reset) in order for the error to reach zero. Begin by using 80 seconds (adjust in minutes if necessary). The error should get smaller.
- Set the Integral gain to a lower value, such as 50 for a different response. If there is no response, continue to decrease the reset value until the response becomes unstable. See the figure below.



- For discussion, let us say that a reset value of 35 made the control output unstable. Return the reset value to the stable value, such as 38. **Be careful with this adjustment since the oscillation can destroy the process.**
- The control output response should be optimal now, without a Derivative gain. The example recorded values are: Proportional gain = 9.7 and Integral gain = 38 seconds. Note that the error has been minimized.



The foregone method is the most common method used to tune a PID loop. Derivative gain is almost never used in a temperature control loop. This method can also be used for other control loops, but other parameters may need to be added for a stable control output.

Test your loop for a high PV of 80% and again for a low PV of 20%, and correct the values if necessary. Small adjustments of the parameters can make the control output more precise or more unstable. It is sometimes acceptable to have a small overshoot to make the control output react quicker.

The derivative gain can be helpful for those control loops which are not controlling temperature. For these loops, try adding a value of 0.5 for the derivative gain and see if this improves the control output. If there is little or no response, increase the derivative by increments of 0.5 until there is an improvement to the output trend. Recall that the derivative gain reacts with a rate of change of the error.

Alternative Manual Tuning Procedures by Others

The following tuning procedures have been extracted from various publications about PID process control. These procedures are for comparison to the procedure in this manual.

Tuning PID Controllers

Two-Mode Simple Method - – for P-I controllers

1. Turn off reset and set the gain to a small value (0.5 - 1.0).
2. Increase gain until cycling starts, then decrease gain slightly.
3. Make setpoint changes to observe offset (error).
4. Increase reset to eliminate offset (error).
5. Repeat steps 2 through 4 until you obtain the largest gain and reset consistent with the criteria of the control desired, i.e., offset, overshoot, stability.

Zeigler-Nichols Method– "Quarter amplitude decay"

1. Turn off reset and rate; set the proportional gain to a fairly large value.
2. Make a small setpoint change and observe how the controlled variable cycles.
3. Adjust the gain until the cycle is self-sustaining, and of constant amplitude; this value is the ultimate gain (Gu).
4. Measure the period of cycling in minutes. This is the ultimate period (Pu).
5. Calculate the controller adjustments as follows:

P only:	G = Gu/2	
P & I :	G = Gu/2.2	
	Ti = 1.2/Pu	(repeats/minute)
P-I-D:	G = Gu/1.6	
	Ti = 2.0/Pu	(repeats/minute)
	Td = Pu/8.0	(minutes)

Pessen Method

1. Follow the procedure described above (Zeigler-Nichols) to determine the ultimate gain and ultimate period.
2. Apply the formulas below.

For no overshoot during startup:

G = Gu/5.0	
Ti = 3/Pu	(repeats/minute)
Td = Pu/2	(minutes)

For some overshoot, but better response to disturbances:

G = Gu/3	
Ti = 3/Pu	(repeats/minute)
Td = Pu/3	(minutes)

Auto Tuning Procedure

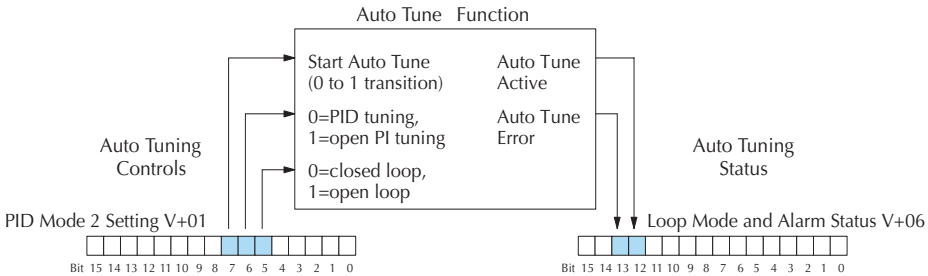
The auto tuning feature for the DL06 loop controller will only run once each time it is enabled in the PID table. Therefore, auto tuning does not run continuously during operation (this would be *adaptive* control). Whenever there is a substantial change in loop dynamics, such as mass of process, size of actuator, etc., the tuning process will need to be repeated in order to derive new gains required for optimal control.

WARNING: Only authorized personnel fully familiar with all aspects of the process should make changes that affect the loop tuning constants. Using the loop auto tuning procedures will affect the process, including inducing large changes in the control output value. Make sure you thoroughly consider the impact of any changes to minimize the risk of injury to personnel or damage to equipment. The auto tune in the DL06 is not intended to be used as a replacement for your process knowledge.

Once the physical loop components are connected to the PLC, auto tuning can be initiated within ***Direct***SOFT (see the ***Direct***SOFT Programming Software Manual), and it can be used to establish initial PID parameter values. Auto tuning is the best "guess" the CPU can do after some trial tests.

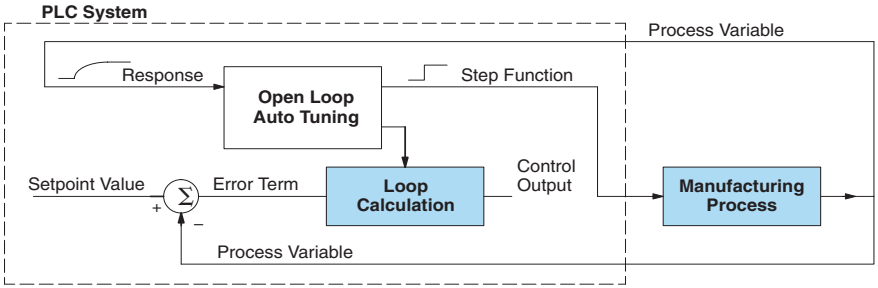
The loop controller offers both closed-loop and open-loop methods. The following sections describe how to use the auto tuning feature, and what occurs in open and closed-loop auto tuning.

The controls for the auto tuning function use three bits in the PID Mode 2 word V+01, as shown below. ***Direct***SOFT will manipulate these bits automatically when you use the auto tune feature within ***Direct***SOFT. Or, you may have your ladder logic access these bits directly for allowing control from another source such as a dedicated operator interface. The individual control bits allow you to start the auto tune procedure, select PID or PI tuning and select closed-loop or open-loop tuning. If you select PI tuning, the auto tune procedure leaves the derivative gain at 0. The Loop Mode and Alarm Status word V+06 reports the auto tune status as shown. Bit 12 will be on (1) during the auto tune cycle, automatically returning to off (0) when done.



Open-Loop Auto Tuning

During an open-loop auto tuning cycle, the loop controller operates as shown in the diagram below. Before starting this procedure, place the loop in Manual Mode and ensure the PV and control output values are in the middle of their ranges (away from the end points).

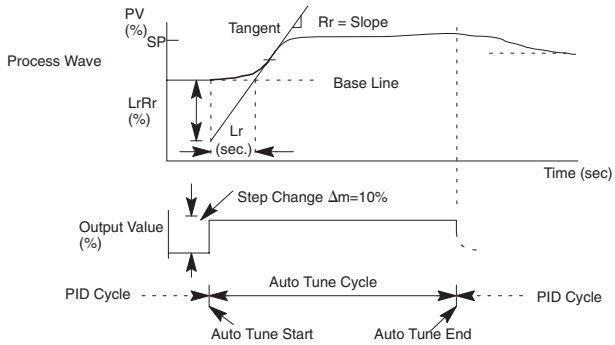


***NOTE**: In theory, the SP value does not matter in this case, because the loop is not closed. However, the requirement of the firmware is that the SP value must be more than 5% of the PV range from the actual PV before starting the auto tune cycle (for the DL06, 12 bit PV should be 205 counts or more below the SP for forward-acting loops, or 205 counts or more above the SP for reverse-acting loops).*

When auto tuning, the loop controller induces a step change on the output and simply observes the response of the PV. From the PV response, the auto tune function calculates the gains and the sample time. It automatically places the results in the corresponding registers in the loop table.

The following timing diagram shows the events which occur in the open-loop auto tuning cycle. The auto tune function takes control of the control output and induces a 10%-of-span step change. If the PV change which the loop controller observes is less than 2%, then the step change on the output is increased to 20%-of-span.

* When Auto Tune starts, step change output m=10%
* During Auto Tune, the controller output reached the full scale positive limit. Auto Tune stopped and the Auto Tune Error bit in the Alarm word bit turned on.
* When PV change is under 2%, output is changed at 20%. Open Loop Auto Tune Cycle Wave: Step Response Method



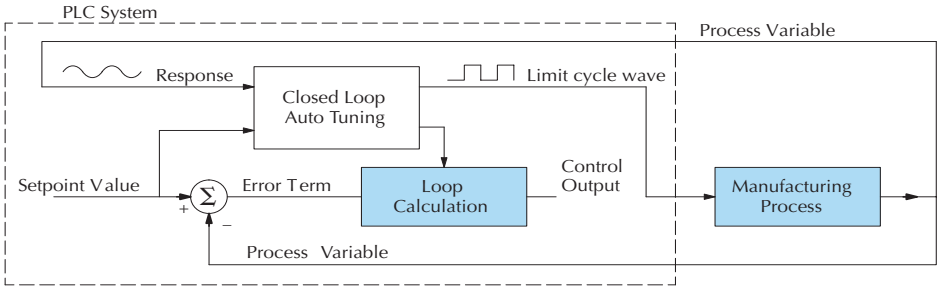
When the loop tuning observations are complete, the loop controller computes Rr (maximum slope in %/sec.) and Lr (dead time in sec). The auto tune function computes the gains according to the Zeigler-Nichols equations, shown below:

PID Tuning	PI Tuning
P=1.2*Δm/LrRr	P=0.9*Δm/LrRr
I=2.0* Lr	I=3.33* Lr
D=0.5* Lr	D=0
Sample Rate = 0.056* Lr	Sample Rate = 0.12*Lr
Δm = Output step change (10% = 0.1, 20% = 0.2)	

We highly recommend using *Direct*SOFT for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of the process. A slowly-changing PV will result in a longer auto tune cycle time. When the auto tuning is complete, the proportional, integral, and derivative gain values are automatically updated in loop table locations V+10, V+11, and V+12 respectively. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to do this are in the section on the manual tuning procedure (located prior to this auto tuning section).

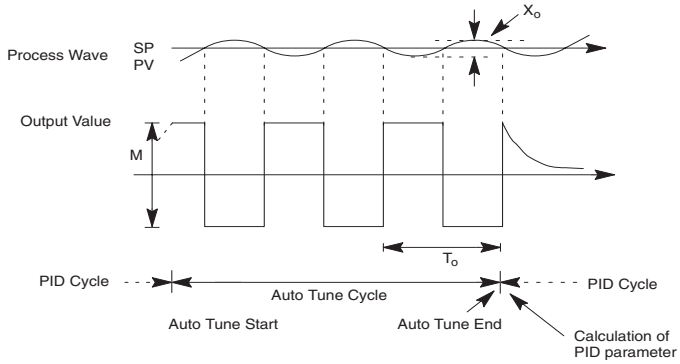
Closed-Loop Auto Tuning

During a closed-loop auto tuning cycle the loop controller operates as shown in the diagram below.



When auto tuning, the loop controller imposes a square wave on the output. Each transition of the output occurs when the PV value crosses over/under the SP value. Therefore, the frequency of the limit cycle is roughly proportional to the mass of the process. From the PV response, the auto tune function calculates the gains and the sample time. It automatically places the results in the corresponding registers in the loop table.

The following timing diagram shows the events which occur in the closed-loop auto tuning cycle. The auto tune function examines the direction of the offset of the PV from the SP. The auto tune function then takes control of the control output and induces a full-span step change in the opposite direction. Each time the sign of the error (SP – PV) changes, the output changes full-span in the opposite direction. This proceeds through three full cycles.



*Mmax = Output Value upper limit setting. Mmin = Output Value lower limit setting.

* This example is direct–acting.

When set to reverse–acting, the output will be inverted. When the loop tuning observations are complete, the loop controller computes To (bump period) and Xo (amplitude of the PV). Then it uses these values to compute Kpc (sensitive limit) and Tpc (period limit). From these values, the loop controller auto tune function computes the PID gains and the sample rate according to the Zeigler-Nichols equations shown below:

Kpc = 4M / (π *X0) Tpc = 0	
M = Amplitude of output	
PID Tuning	**PI Tuning**
P = 0.45*Kpc	P = 0.30*Kpc
I = 0.60*Tpc	I = 1.00*Tpc
D = 0.10*Tpc	D = 0
Sample Rate = 0.014*Tpc	Sample Rate = 0.03*Tpc

Auto tuning error

In open-loop tuning, if the auto tune error bit (bit 13 of loop Mode/Alarm status word V+06) is on, please verify the PV and SP values are within 5% of full scale difference, as required by the auto tune function.

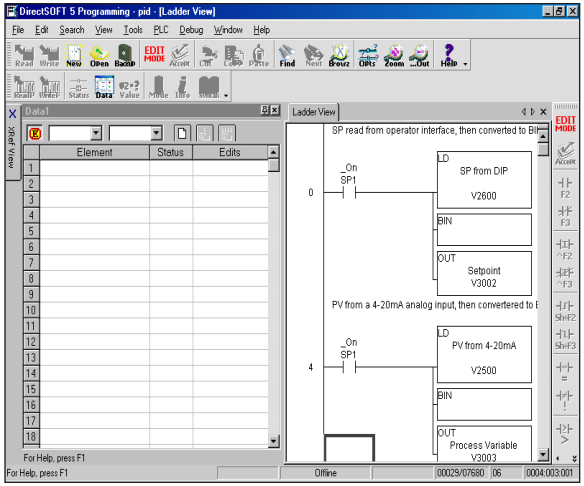
***NOTE**: If your PV fluctuates rapidly, you probably need to use the built-in analog filter (see page 8–55) or create a filter in ladder logic (see example on page 8–56).*

Use *Direct*SOFT Data View with PID View

The **Data View** window is a very useful tool which can be used to help tune your PID loop. You can compare the variables in the **PID View** with the actual values in the V-memory location with Data View.

Open a New Data View Window

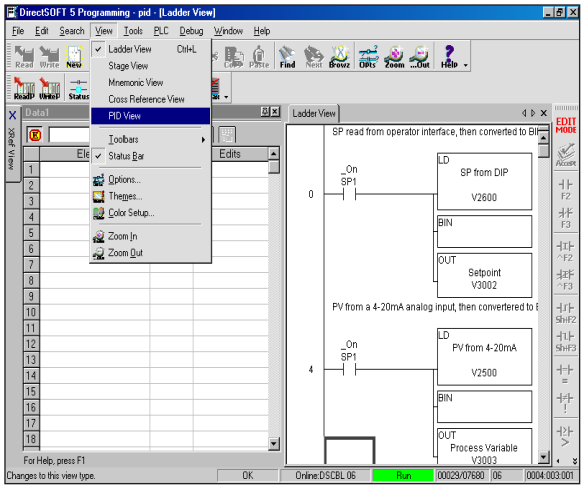
A new Data View window can be opened in any one of three ways; the menu bar **Debug > Data View > New**, the keyboard shortcut **Ctrl + Shift + F3** or the **Data** button on the Status toolbar. By default, the Data View window is assigned Data1 as the default name. This name can be changed for the current view using the Options dialog. The following diagram is an example of a newly opened Data View. The window will open next to the Ladder View by default.



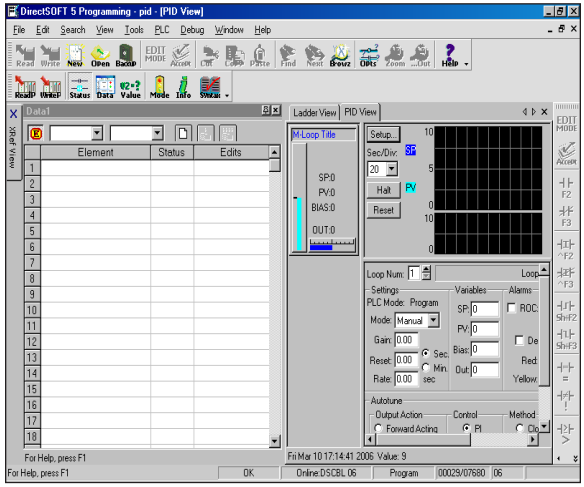
The Data View window can be used just as it is shown above for troubleshooting your PID logic, and it can be most useful when tuning the PID loop.

Open PID View

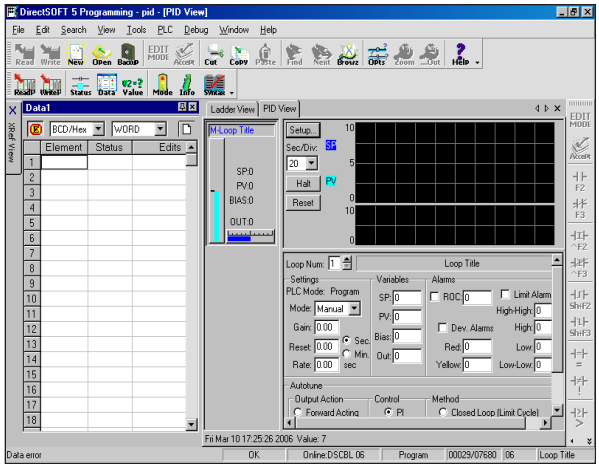
The PID View can only be opened after a loop has been setup in your ladder program. PID View is opened by selecting it from the View submenu on the Menu bar, **View > PID View**. The PID View can also be opened by clicking on the PID View button from the PLC Setup toolbar if it is in view.



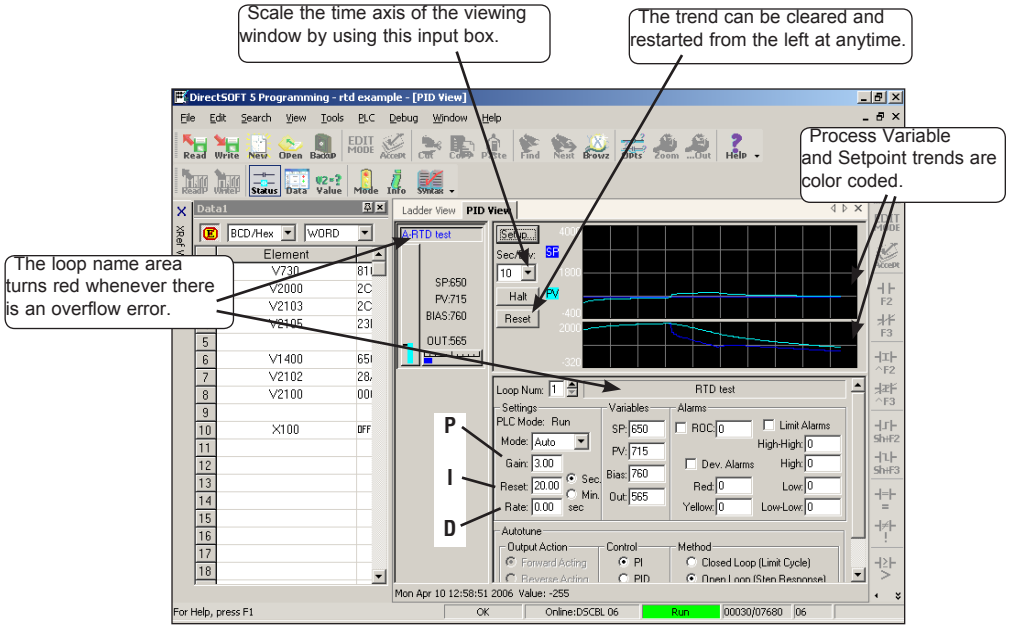
The PID View will open and appear over the Ladder View which can be brought into view by clicking on its tab. When using the Data View and the PID View together, each view can be sized for better use as shown on the facing page.



The two views are now ready to be used to tune your loop. You will be able to see where the PID values have been set and see the process that it is controlling.



The diagram below illustrates how the to use the views to see the current SP, PV and Output values, along with the other PID addresses. Refer to the Loop Table Definitions page 8-20 for details of each word in the table. This is also a good data type reference for each word in the table.

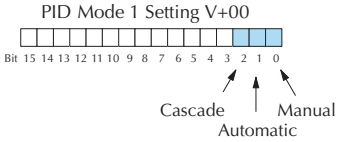


With both windows positioned in this manner, you are able to see where the PID values have been set and see the process that it is controlling. In the diagram below, you can see the current SP, PV and Output values, along with the other PID addresses. Refer to the Loop Table Word Definitions (page 8-20) for details for each word in the table. This is also a good data type reference for each word in the table.

Using the Special PID Features

It's a good idea to understand the special features of the DL06 and how to use them. You may want to incorporate some of these features for your PID.

How to Change Loop Modes

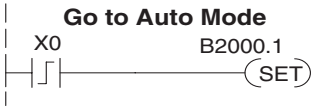


The first three bits of the PID Mode 1 word (V+00) request the operating mode of the corresponding loop. Note: these bits are mode change *requests*, not commands (certain conditions can prohibit a particular mode change – see next page).

The normal state of these mode request bits is "000." To request a mode change, you must SET the corresponding bit to a "**1**" using a one-shot. The PID loop controller automatically resets the bits back to "000" after it reads the mode change request. Methods of requesting mode changes are:

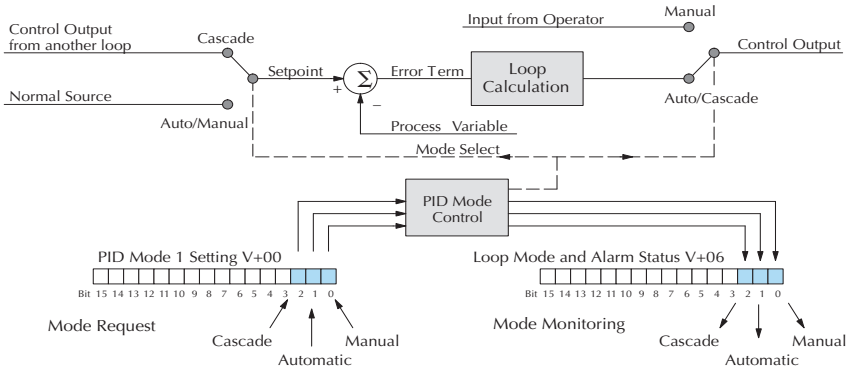
- ***Direct***SOFT's **PID View** – this is the easiest method. Use the pull-down menu, or click on one of the radio buttons if using older *Direct*SOFT versions, and the appropriate bit will get set.
- **Ladder program**– ladder logic can request any loop mode when the PLC is in Run Mode. This will be necessary after application startup if mode changes are part of the application.

Use the program shown to the right to SET the mode bit (do not use an OUT coil). On a 0–1 transition of X0, the rung sets the Auto bit equal to 1. The loop controller resets it.



- **Operator panel** – interface the operator's panel to ladder logic using standard methods, then use the logic to the right to set the mode bit.

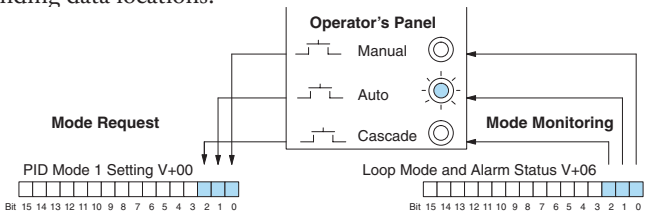
Since mode changes can only be *requested*, the PID loop controller will decide when to permit mode changes and provide the loop mode status. It reports the current mode on bits 0, 1, and 2 of the Loop Mode/Alarm Status word, location V+06 in the loop table. The parallel request/monitoring functions are shown in the figure below. The figure also shows the two possible mode-dependent SP sources, and the two possible Control Output sources.



Operator Panel Control of PID Modes

Since the modes Manual, Auto and Cascade are the most fundamental and important PID loop controls, you may want to hard-wire mode control switches to an operator's panel. Most applications will need only Manual and Auto selections (Cascade is used in special applications). Remember that mode controls are really *mode request* bits, and the actual loop mode is indicated elsewhere.

The following figure shows an operator's panel using momentary push-buttons to request PID mode changes. The panel's mode indicators do not connect to the switches, but interface to the corresponding data locations.



PLC Modes Effect on Loop Modes

If you have selected the option for the loops to follow the PLC mode, the PLC modes (Program, Run) interact with the loops as a group. The following summarizes this interaction:

- When the PLC is in Program Mode, all loops are placed in Manual Mode and no loop calculations occur. However, note that output modules (including analog outputs) turn off in PLC Program Mode. So, actual manual control is not possible when the PLC is in Program Mode.
- The only time the CPU will allow a loop mode change is during PLC Run Mode operation. As such, the CPU records the modes of all 4 loops as the desired mode of operation. If power failure and restoration occurs during PLC Run Mode, the CPU returns all loops to their prior mode (which could be Manual, Auto, or Cascade).
- On a Program-to-Run mode transition, the CPU forces each loop to return to its prior mode recorded during the last PLC Run Mode.
- You can add and configure new loops only when the PLC is in Program Mode. New loops automatically begin in Manual Mode.

Loop Mode Override

In normal conditions the mode of a loop is determined by the request to V+00, bits 0, 1, and 2. However, some conditions exist which will prevent a requested mode change from occurring:

- A loop that is not set independent of PLC mode cannot change modes when the PLC is in Program mode.
- A major loop of a cascaded pair of loops cannot go from Manual to Auto until its minor loop is in Cascade mode.

In other situations, the PID loop controller will automatically change the mode of the loop to ensure safe operation:

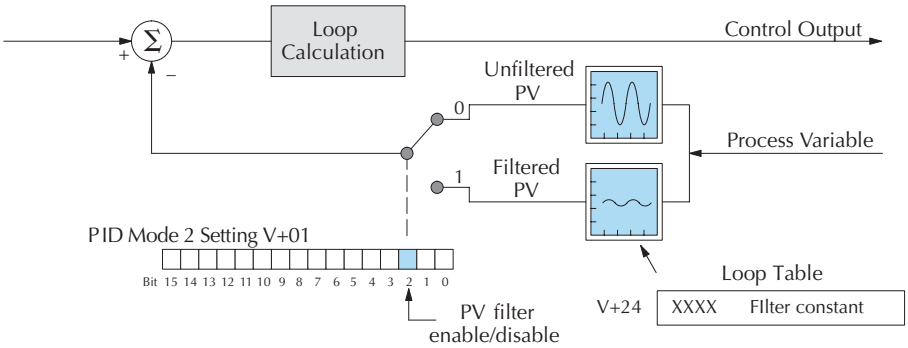
- A loop which develops an error condition automatically goes to Manual.
- If the minor loop of a cascaded pair of loops leaves Cascade Mode for any reason, its major loop automatically goes to Manual Mode.

PV Analog Filter

A noisy PV signal can make tuning difficult and can cause the control output to be more extreme than necessary, as the output tries to respond to the peaks and valleys of the PV. There are two equivalent methods of filtering the PV input to make the loop more stable. The first method is accomplished using the DL06's built-in filter. The second method achieves a similar result using ladder logic.

The DL06 Built-in Analog Filter

The DL06 provides a selectable first-order low-pass PV input filter. **We only recommend the use of a filter during auto tuning or PID control if there is noise on the input signal.** You may disable the filter after auto tuning is complete, or continue to use it if the PV input signal is noisy.



Bit 2 of PID Mode Setting 2 provides the enable/disable control for the low-pass PV filter (0=disable, 1=enable). The roll-off frequency of the single-pole low-pass filter is controlled by using register V+24 in the loop parameter table, the filter constant. The data format of the filter constant value is BCD, with an implied decimal point 00X.X, as follows:

- The filter constant has a valid range of 000.1 to 001.0. **The smaller the filter value, the greater the filtering performed; for example, the value 001.0 provides no filtering.**
- ***Direct***SOFT converts values above the valid range to 001.0 and values below this range to 000.1
- Values close to 001.0 result in higher roll-off frequencies, while values closer to 000.1 result in lower roll-off frequencies.

We highly recommend using ***Direct***SOFT for the auto tuning interface. The duration of each auto tuning cycle will depend on the mass of your process. A slowly-changing PV will result in a longer auto tune cycle time.

When the auto tuning is complete, the proportional and integral gain values are automatically updated in loop table locations V+10 and V+11 respectively. The derivative is calculated if you autotune for PID and updated in loop table location V+12. The sample time in V+07 is also updated automatically. You can test the validity of the values the auto tuning procedure yields by measuring the closed-loop response of the PV to a step change in the output. The instructions on how to do this are in the section on the manual tuning procedure.

The algorithm which the built-in filter follows is:

yi = k (xi – yi–1) + yi–1

yi is the current output of the filter

xi is the current input to the filter

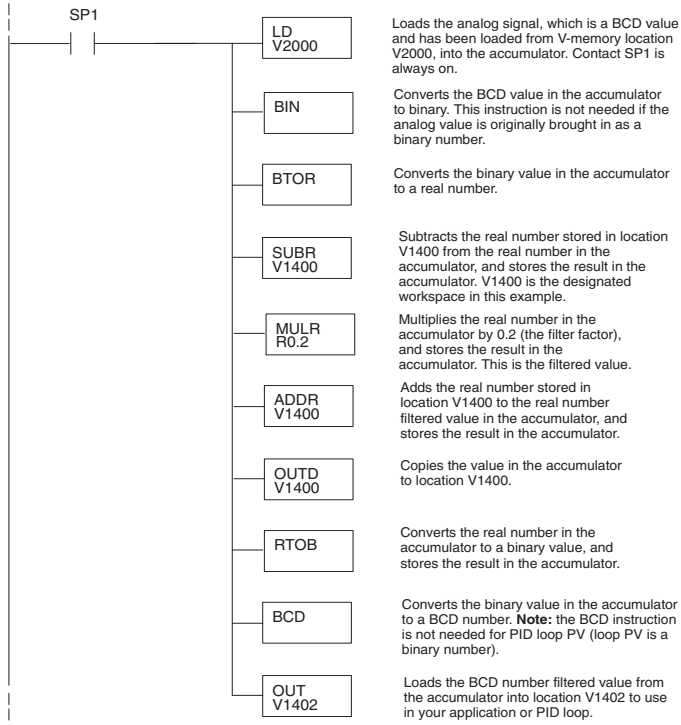
yi–1 is the previous output of the filter

k is the PV Analog Input Filter Factor

Creating an Analog Filter in Ladder Logic

A similar algorithm can be built in your ladder program. Your analog inputs can be filtered effectively using either method. The following programming example describes the ladder logic you will need. Be sure to change the example memory locations to those that fit your application.

Filtering can induce a 1 part in 1000 error in your output because of "rounding." If your process cannot tolerate a 1 part in 1000 error, do not use filtering. Because of the rounding error, you should not use zero or full scale as alarm points. Additionally, the smaller the filter constant the greater the smoothing effect, but the slower the response time. Be sure a slower response is acceptable in controlling your process.



Use the *Direct*SOFT 5 Filter Intelligent Box Instruction

For those who are using *Direct*SOFT 5, you have the opportunity to use Intelligent Box (IBox) instruction IB-402, Filter Over Time in Binary (decimal). This IBox will perform a first-order filter on the Raw Data on a defined time interval. The equation is,

New = Old + [(Raw - Old) / FDC] where

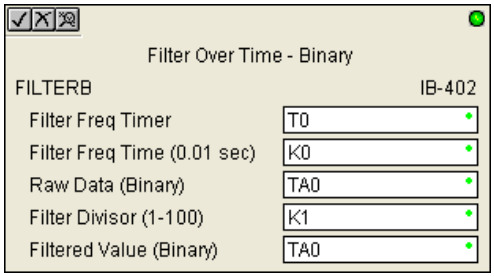
New =New Filtered Value

Old = Old Filtered Value

FDC = Filter Divisor Constant

Raw = Raw Data

The Filter Divisor Constant is an integer in the range K1 to K100, such that if it equaled K1, then no filtering is performed.

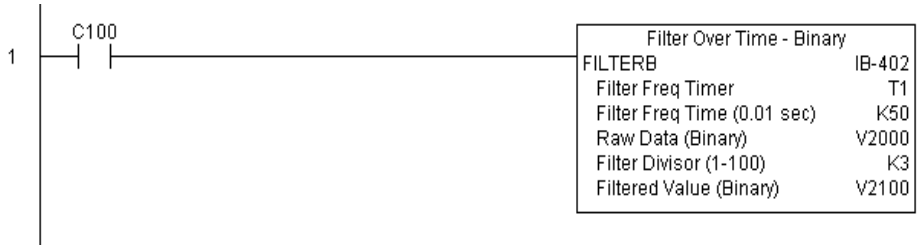


The rate at which the calculation is performed is specified by time in hundredths of a second (0.01 seconds) as the Filter Freq Time parameter. Note that this Timer instruction is embedded in the IBox and must NOT be used any other place in your program. Power flow controls whether the calculation is enabled. If it is disabled, the Filter Value is not updated. On the first scan from Program to Run mode, the Filter Value is initialized to 0 to give the calculation a consistent starting point.

Since the following binary filter example does not write directly to the PID PV location, the BCD filter could be used with BCD values and then converted to BIN.

FilterB Example

Following is an example of how the FilterB IBox is used in a ladder program. The instruction is used to filter a binary value that is in V2000. Timer (T1) is set to 0.5 seconds, the rate at which the filter calculation will be performed. The filter constant is set to 3.0. A larger value will increase the smoothing effect of the filter. A value of 1 results with no filtering. The filtered value will be placed in V2100.

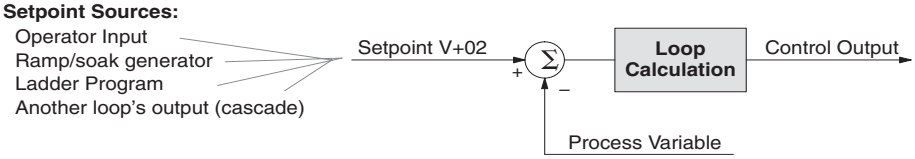


NOTE: *See Chapter 5, page 242, for more detailed information.*

Ramp/Soak Generator

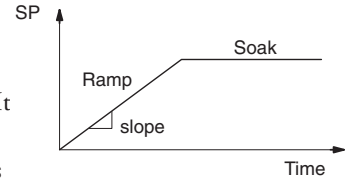
Introduction

Our discussion of basic loop operation noted the setpoint for a loop will be generated in various ways, depending on the loop operating mode and programming preferences. In the figure below, the ramp/soak generator is one of the ways the SP may be generated. It is the responsibility of your ladder program to ensure only one source attempts to write the SP value at V+02 at any particular time.



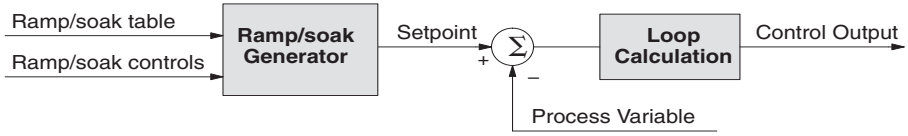
If the SP for your process rarely changes or can tolerate step changes, you probably will not need to use the ramp/soak generator. However, some processes require precisely-controlled SP value changes. *The ramp/soak generator can greatly reduce the amount of programming required for these applications.*

The terms **ramp** and **soak** have special meanings in the process control industry, and refer to desired setpoint (SP) values in temperature control applications. In the figure to the right, the setpoint increases during the ramp segment. It remains steady at one value during the soak segment.



Complex SP profiles can be generated by specifying a series of ramp/soak segments. The ramp segments are specified in SP units per second time. The soak time is also programmable in minutes.

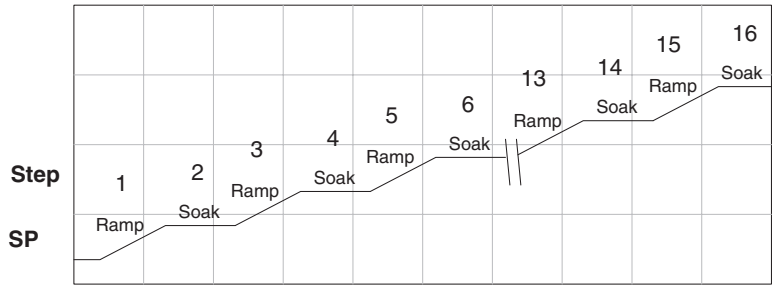
It is instructive to view the ramp/soak generator as a dedicated function to generate SP values, as shown below. It has two categories of inputs which determine the SP values generated. The ramp/soak table must be programmed in advance, containing the values that will define the ramp/soak profile. The loop reads from the table during each PID calculation as necessary. The ramp/soak controls are bits in a special loop table word that control the real-time start/stop functionality of the ramp/soak generator. The ladder program can monitor the status of the ramp soak profile (current ramp/segment number).



Now that we have described the general ramp/soak generator operation, we list its specific features:

- Each loop has its own ramp/soak generator (use is optional).
- You may specify up to eight ramp/soak steps (16 segments).
- The ramp soak generator can run any time the PLC is in Run mode. Its operation is independent of the loop mode (Manual or Auto).
- Ramp/soak real-time controls include Start, Hold, Resume, and Jog.
- Ramp/soak monitoring includes Profile Complete, Soak Deviation (SP minus PV), and current ramp/soak step number.

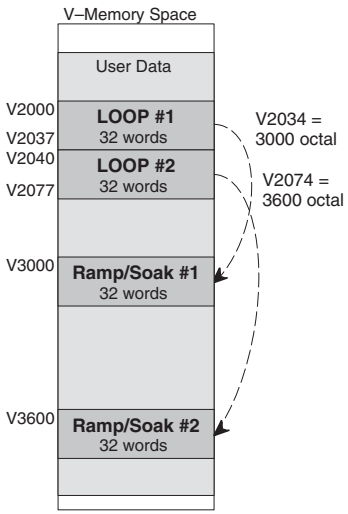
The following figure shows an SP profile consisting of ramp/soak segment pairs. The segments are individually numbered as steps from 1 to 16. The slope of each of the ramp segments may be either increasing or decreasing. The ramp/soak generator automatically knows whether to increase or decrease the SP based on the relative values of a ramp's end points. These values come from the ramp/soak table.



Ramp/Soak Table

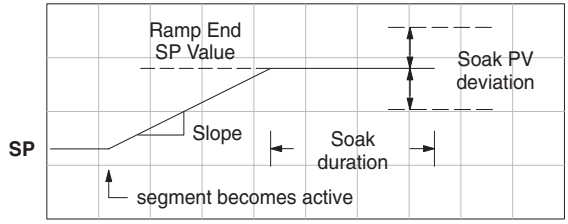
The parameters which define the ramp/soak profile for a loop are in a ramp/soak table. Each loop may have its own ramp/soak table, but it is optional. Recall the Loop Parameter table consists of a 32-word block of memory for each loop, and together they occupy one contiguous memory area. However, the ramp/soak table for a loop is individually located, because it is optional for each loop. An address pointer in location V+34 in the loop table specifies the starting location of the ramp/soak table.

In the example to the right, the loop parameter tables for Loop #1 and #2 occupy contiguous 32-word blocks as shown. Each has a pointer to its ramp/soak table, independently located elsewhere in user V-memory. Of course, you may locate all the tables in one group, as long as they do not overlap.



The parameters in the ramp/soak table must be user-defined. the most convenient way is to use *Direct*SOFT, which features a special editor for this table. Four parameters are required to define a ramp and soak segment pair, as pictured below.

- **Ramp End Value** – specifies the destination SP value for the end of the ramp. Use the same data format for this number as you use for the SP. It may be above or below the beginning SP value, so the slope could be up or down (we don't have to know the starting SP value for ramp #1).
- **Ramp Slope** – specifies the SP increase in counts (units) per second. It is a BCD number from 00.00 to 99.99 (uses implied decimal point).
- **Soak Duration** – specifies the time for the soak segment in minutes, ranging from 000.1 to 999.9 minutes in BCD (implied decimal point).
- **Soak PV Deviation** – (optional) specifies an allowable PV deviation above and below the SP value during the soak period. A PV deviation alarm status bit is generated by the ramp/soak generator.



Ramp/Soak Table

V+00	XXXX	Ramp End SP Value
V+01	XXXX	Ramp Slope
V+02	XXXX	Soak Duration
V+03	XXXX	Soak PV Deviation

The ramp segment becomes active when the previous soak segment ends. If the ramp is the first segment, it becomes active when the ramp/soak generator is started, and automatically assumes the present SP as the starting SP.

Offset	Step	Description	Offset	Step	Description
+ 00	1	Ramp End SP Value	+ 20	9	Ramp End SP Value
+ 01	1	Ramp Slope	+ 21	9	Ramp Slope
+ 02	2	Soak Duration	+ 22	10	Soak Duration
+ 03	2	Soak PV Deviation	+ 23	10	Soak PV Deviation
+ 04	3	Ramp End SP Value	+ 24	11	Ramp End SP Value
+ 05	3	Ramp Slope	+ 25	11	Ramp Slope
+ 06	4	Soak Duration	+ 26	12	Soak Duration
+ 07	4	Soak PV Deviation	+ 27	12	Soak PV Deviation
+ 10	5	Ramp End SP Value	+ 30	13	Ramp End SP Value
+ 11	5	Ramp Slope	+ 31	13	Ramp Slope
+ 12	6	Soak Duration	+ 32	14	Soak Duration
+ 13	6	Soak PV Deviation	+ 33	14	Soak PV Deviation
+ 14	7	Ramp End SP Value	+ 34	15	Ramp End SP Value
+ 15	7	Ramp Slope	+ 35	15	Ramp Slope
+ 16	8	Soak Duration	+ 36	16	Soak Duration
+ 17	8	Soak PV Deviation	+ 37	16	Soak PV Deviation

Many applications do not require all 16 R/S steps. Use all zeros in the table for unused steps. The R/S generator ends the profile when it finds ramp slope = 0.

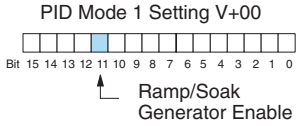
Ramp/Soak Table Flags

The individual bit definitions of the Ramp/Soak Table Flag (Addr+33) word is listed in the following table.

Bit	Ramp/Soak Flag Bit Description	Read/Write	Bit=0	Bit=1
0	Start Ramp / Soak Profile	write	–	01 Start
1	Hold Ramp / Soak Profile	write	–	01 Hold
2	Resume Ramp / Soak Profile	write	–	01 Resume
3	Jog Ramp / Soak Profile	write	–	01 Jog
4	Ramp / Soak Profile Complete	read	–	Complete
5	PV Input Ramp / Soak Deviation	read	Off	On
6	Ramp / Soak Profile in Hold	read	Off	On
7	Reserved	read	Off	On
8–15	Current Step in R/S Profile	read	decode as byte (hex)	

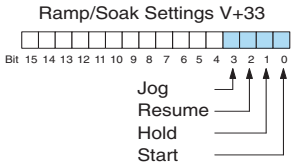
Ramp/Soak Generator Enable

The main enable control to permit ramp/soak generation of the SP value is accomplished with bit 11 in the PID Mode 1 Setting V+00 word, as shown to the right. The other ramp/soak controls in V+33 shown in the table above will not operate unless this bit=1 during the entire ramp/soak process.



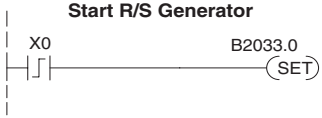
Ramp/Soak Controls

The four main controls for the ramp/soak generator are in bits 0 to 3 of the ramp/soak settings word in the loop parameter table. *Direct*SOFT controls these bits directly from the ramp/soak settings dialog. However, you must use ladder logic to control these bits during program execution. We recommend using the bit-of-word instructions.



Ladder logic must set a control bit to a 1 to command the corresponding function. When the loop controller reads the ramp/soak value, it automatically turns off the bit for you. Therefore, a reset of the bit is not required, when the CPU is in Run Mode.

The example program rung to the right shows how an external switch X0 can turn on, and the PD contact uses the leading edge to set the proper control bit to start the ramp soak profile. This uses the Set Bit-of-word instruction.

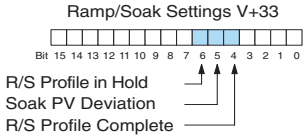


The normal state for the ramp/soak control bits is all zeros. Ladder logic must set only one control bit at a time.

- **Start** – a 0 to 1 transition will start the ramp soak profile. The CPU must be in Run Mode, and the loop can be in Manual or Auto Mode. If the profile is not interrupted by a Hold or Jog command, it finishes normally.
- **Hold** – a 0 to 1 transition will stop the ramp/soak profile in its current state, and the SP value will be frozen.
- **Resume** – a 0 to 1 transition cause the ramp/soak generator to resume operation if it is in the hold state. The SP values will resume from their previous value.
- **Jog** – a 0 to 1 transition will cause the ramp/soak generator to truncate the current segment (step), and go to the next segment.

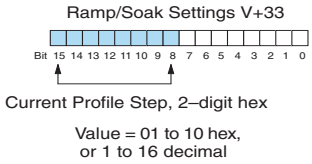
Ramp/Soak Profile Monitoring

You can monitor the Ramp/Soak profile status using other bits in the Ramp/Soak Settings V+33 word, shown to the right.



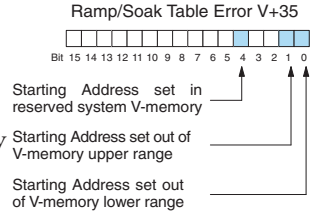
- R/S Profile Complete – =1 when the last programmed step is done.
- Soak PV Deviation – =1 when the error (SP–PV) exceeds the specified deviation in the R/S table.
- R/S Profile in Hold – =1 when the profile was active but is now in hold. Ramp/Soak Settings V+33

The number of the current step is available in the upper 8 bits of the Ramp/Soak Settings V+33 word. The bits represent a 2-digit hex number, ranging from 1 to 10. Ladder logic can monitor these to synchronize other parts of the program with the ramp/soak profile. Load this word to the accumulator and shift right 8 bits, and you have the step number.



Ramp/Soak Programming Errors

The starting address for the ramp/soak table must be a valid location. If the address points outside the range of user V-memory, one of the bits to the right will turn on when the ramp/soak generator is started. We recommend using *Direct*SOFT to configure the ramp/soak table. It automatically range checks the addresses for you.



Testing Your Ramp/Soak Profile

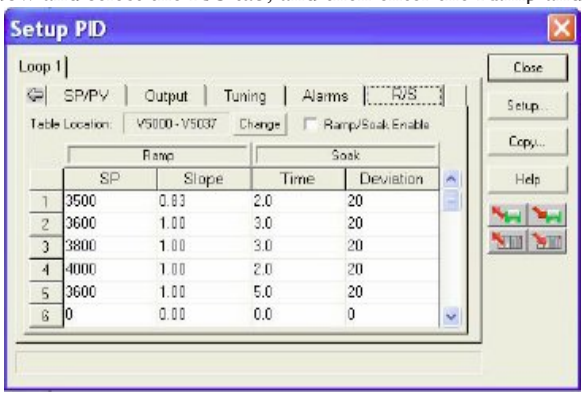
It's a good idea to test your ramp/soak profile before using it to control the process. This is easy to do, because the ramp/soak generator will run even when the loop is in Manual Mode. Using *Direct*SOFT's PID View will be a real time-saver, because it will draw the profile on-screen for you. Be sure to set the trending timebase slow enough to display completed ramp-soak segment pairs in the waveform window.

*Direct*SOFT Ramp/Soak Example

The following example will step you through the Ramp/Soak setup.

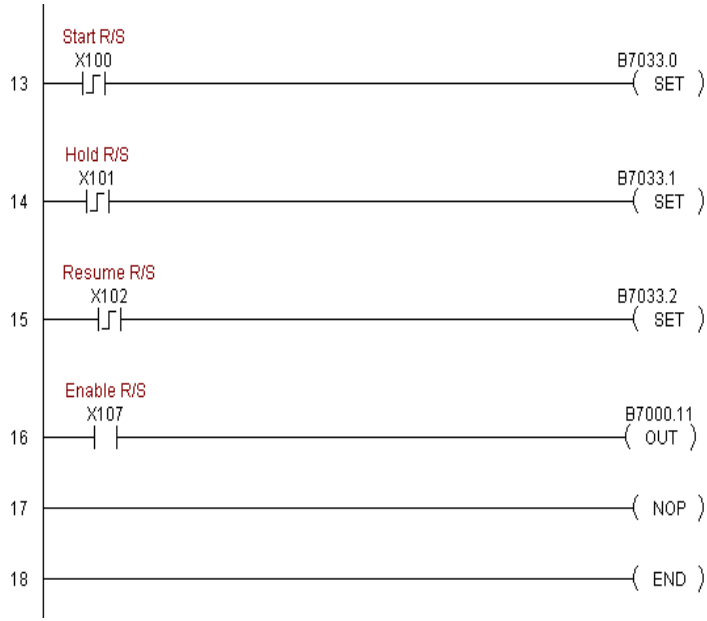
Setup the Profile in PID Setup

The first step is to use Setup PID in *Direct*SOFT to set the profile of your process. Open the Setup PID window and select the R/S tab, and then enter the Ramp and Soak data.



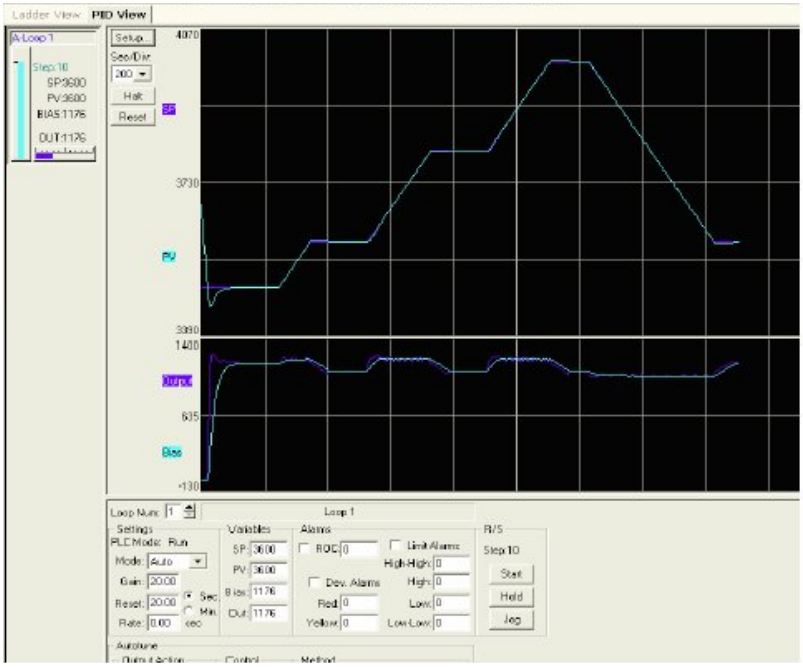
Program the Ramp/Soak Control in Relay Ladder

Refer to the Ramp/Soak Flag Bit Description table on page 8-60 when adding the control rungs to your program similar to the ladder rungs below.



Test the Profile

Test your profile using PID View.



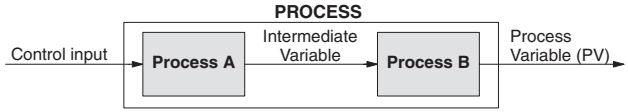
Cascade Control

Introduction

Using cascaded loops is an advanced control technique, superior to individual loop control in certain situations. As the name implies, cascade means that one loop is connected to another loop. In addition to Manual (open loop) and Auto (closed loop) Modes, the DL06 also provides Cascaded Mode.

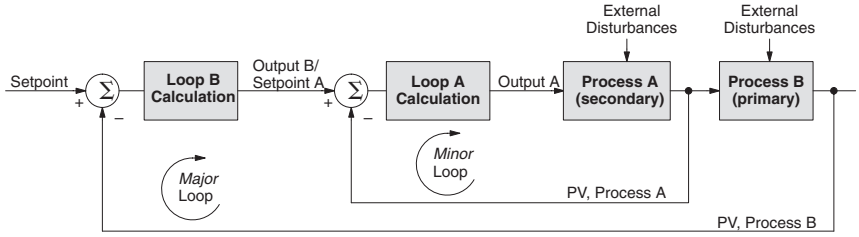
***NOTE**: Using cascaded loops is an advanced process control technique; therefore, we recommend their use only for experienced process control engineers.*

When a manufacturing process is complex and contains a lag time from control input to process variable output, even the most perfectly tuned single loop around the process may yield slow and inaccurate control. It may be the actuator operates on one physical property, which eventually affects the process variable, measured by a different physical property. Identifying the intermediate variable allows us to divide the process into two parts as shown in the following figure.



The principle of cascaded loops is simply that we add another process loop to more precisely control the intermediate variable! This separates the source of the control lag into two parts, as well.

The diagram below shows a cascade control system, showing that it is simply one loop nested inside another. The inside loop is called the minor loop, and the outside loop is called the major loop. For overall stability, the minor loop must be the fastest responding loop of the two (try a factor of 10 for a better response time). We do have to add the additional sensor to measure the intermediate variable (PV for process A). Notice the setpoint for the minor loop is automatically generated for us, by using the output of the major loop. Once the cascaded control is programmed and debugged, we only need to deal with the original setpoint and process variable at the system level. The cascaded loops behave as one loop, but with improved performance over the previous single-loop solution.



One of the benefits to cascade control can be seen by examining its response to external disturbances. Remember the minor loop is faster acting than the major loop. Therefore, if a disturbance affects process A in the minor loop, the Loop A PID calculation can correct the resulting error before the major loop sees the effect.

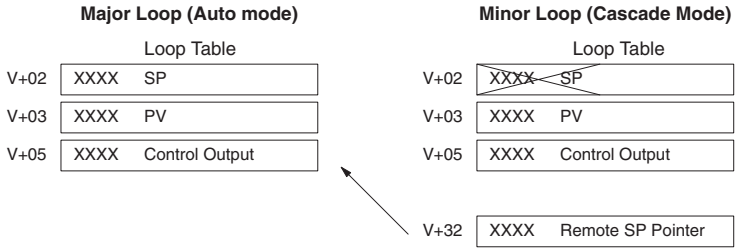
Cascaded Loops in the DL06 CPU

In the use of the term cascaded loops, we must make an important distinction. Only the minor loop will actually be in the Cascade Mode. In normal operation, the major loop must be in Auto Mode. If you have more than two loops cascaded together, the outer-most (major) loop must be in Auto Mode during normal operation, and all inner loops in Cascade Mode.

***NOTE**: Technically, both major and minor loops are cascaded in strict process control terminology. Unfortunately, we are unable to retain this convention when controlling loop modes. Remember that all minor loops will be in Cascade Mode, and only the outer-most (major) loop will be in Auto Mode.*

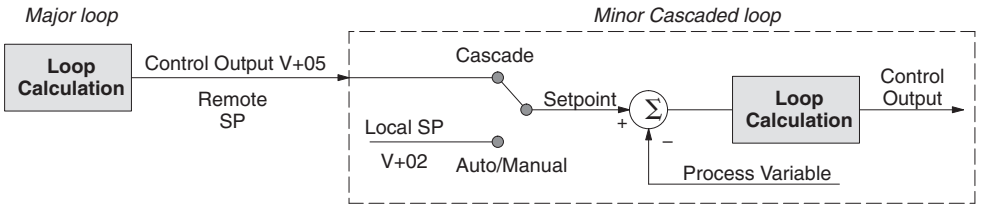
You can cascade together as many loops as necessary on the DL06, and you may have multiple groups of cascaded loops. For proper operation on cascaded loops you must use the same data range (12/15 bit) and unipolar/bipolar settings on the major and minor loop.

To prepare a loop for Cascade Mode operation as a minor loop, you must program its remote Setpoint Pointer in its loop parameter table location V+32, as shown below. The pointer must be the address of the V+05 location (control output) of the major loop. In Cascade Mode, the minor loop will ignore the its local SP register (V+02), and read the major loop's control output as its SP instead.



When using *Direct*SOFT's PID View to watch the SP value of the minor loop, *Direct*SOFT automatically reads the major loop's control output and displays it for the minor loop's SP. The minor loop's normal SP location, V+02, remains unchanged.

Now, we use the loop parameter arrangement above and draw its equivalent loop diagram, shown below.



Remember that a major loop goes to Manual Mode automatically if its minor loop is taken out of Cascade Mode.

Tuning Cascaded Loops

In tuning cascaded loops, you will need to de-couple the cascade relationship and tune the loops individually, using one of the loop tuning procedures previously covered.

1. If you are not using auto tuning, then find the loop sample rate for the minor loop, using the method discussed earlier in this chapter. Then set the sample rate of the major loop slower than the minor loop by a factor of 10. Use this as a starting point.

2. Tune the minor loop first. Leave the major loop in Manual Mode, and you will need to generate SP changes for the minor loop manually as described in the loop tuning procedure.

3. Verify the minor loop gives a critically-damped response to a 10% SP change while in Auto Mode. Then we are finished tuning the minor loop.

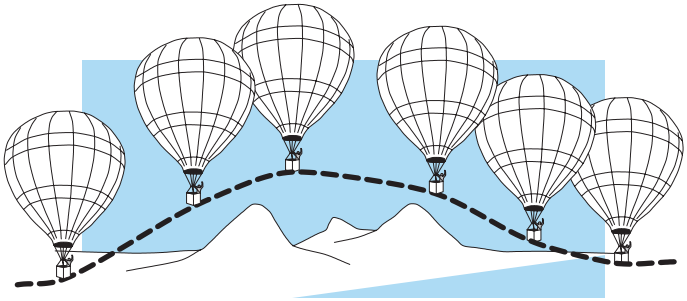
4. In this step, you will need to get the minor loop in Cascade Mode, and then the Major loop in Auto Mode. We will be tuning the major loop with the minor loop treated as a series component its overall process. Therefore, do not go back and tune the minor loop again while tuning the major loop.

5. Tune the major loop, following the standard loop tuning procedure in this section. The response of the major loop PV is actually the overall response of the cascaded loops together.

Time-Proportioning Control

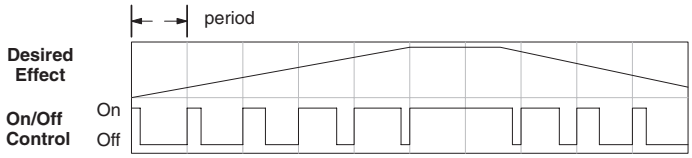
The PID loop controller in the DL06 CPU generates a smooth control output signal across a numerical range. The control output value is suitable to drive an analog output module, which connects to the process. In the process control field, this is called *continuous control*, because the output is on (at some level) continuously.

While continuous control can be smooth and robust, the cost of the loop components (such as actuator, heater amplifiers) can be expensive. A simpler form of control is called *time-proportioning control.* This method uses actuators which are either on or off (no in-between). Loop components for on/off-based control systems are lower cost than their continuous control counterparts.



In this section, we will show you how to convert the control output of a loop to time-proportioning control for the applications that need it. Let's take a moment to review how alternately turning a load on and off can control a process. The diagram below shows a hot-air balloon following a path across some mountains. The desired path is the *setpoint*. The balloon pilot turns the burner on and off alternately, which is his *control output*. The large mass of air in the balloon effectively averages the effect of the burner, converting the bursts of heat into a continuous effect: slowly changing balloon temperature and ultimately the altitude, which is the *process variable*.

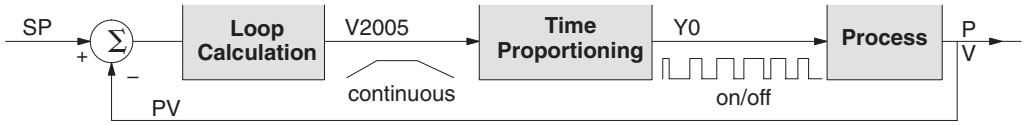
Time-proportioning control approximates continuous control by virtue of its duty-cycle – the ratio of ON time to OFF time. The following figure shows an example of how duty-cycle approximates a continuous level when it is averaged by a large process mass.



If we were to plot the on/off times of the burner in the hot-air balloon, we would probably see a very similar relationship to its effect on balloon temperature and altitude.

On/Off Control Program Example

The following ladder segment provides a time proportioned on/off control output. It converts the continuous output in V2005 to on/off control using the output coil, Y0.

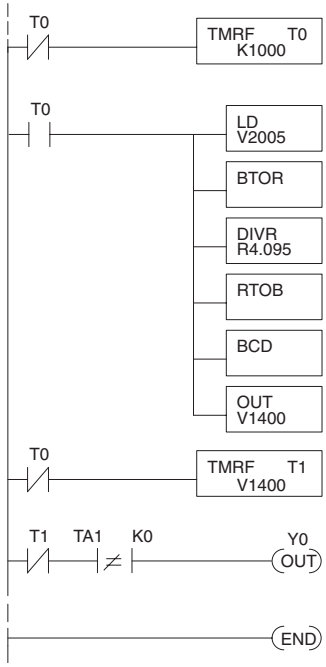


The example program uses two timers to generate On/Off control. It makes the following **assumptions**, which you can alter to fit your application:

- The loop table starts at V2000, so the control output is at V2005.
- The data format of the control output is 12-bit, unipolar (0 – FFF).
- The time base (one full cycle) for the On/Off waveform is 10 seconds. We use a fast timer (0.01 sec/tick), counting to 1000 ticks (10 seconds).
- The On/Off control output is Y0.

The time proportioning program must match the resolution of the output (1 part in 1000) to the resolution of the time base of T0 (also 1 part in 1000).

***NOTE**: Some processes change too fast for time proportioning control. Consider the speed of your process when you choose this control method. Use continuous control for processes that change too fast for time proportioning control. Also, consider using a solid state switch for a longer switch life instead of a relay.*



A fast timer (0.01 sec. timebase) establishes the primary time interval. The constant, K1000, sets the preset at 10 seconds (1,000 ticks). The N.C. enabling contact, T0, makes the timer self-resetting. T0 is on for one scan each 10 seconds, when it resets itself and T1.

At the end of the 10 second period, T0 turns on, and loads the control output value (binary) from the loop table V+05 location (V2005).

The BTOR instruction changes the number in the accumulator to a real number.

Dividing the control output by 4.095, converts the 0 – 4095 range to 0 – 1000, which "matchs" the number of ticks in the 10 second timer range.

This instruction converts the real number back to binary. This step prepares the number for conversion to BCD. There is no real-to-BCD instruction.

Convert the number in the accumulator to BCD format. This satisfies the timer preset format requirement.

Output the result to V1400. In our example, this is the location of the timer preset for the second timer.

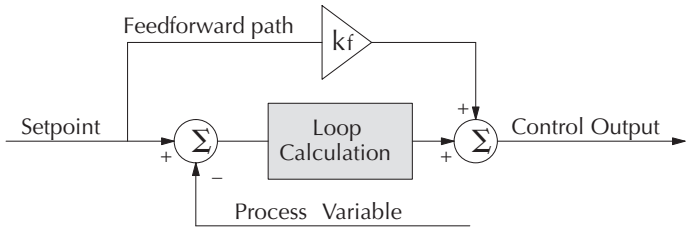
The second fast timer also counts in increments of .01 seconds, so its range is variable from 0 to a maximum of 1000 ticks, or 10 seconds. This timer's output, T1, turns off the output coil, Y0, when the preset is reached.

The N.C. T1 contact, inverts the T1 timer output. The control output is on at the beginning of the 10-second time interval. Y0 turns off when T1 times out. The STRNE contact prevents Y0 from energizing during the one scan when T0 resets T1. Y0 is the actual control output.

END coil marks the end of the main program.

Feedforward Control

Feedforward control is an enhancement to standard closed-loop control. It is most useful for diminishing the effects of a *quantifiable and predictable* loop disturbance or sudden change in setpoint. Use of this feature is an option available to you on the DL06. However, it's best to implement and tune a loop without feedforward, and adding it only if better loop performance is still needed. The term *feedforward* refers to the control technique involved, shown in the diagram below. The incoming setpoint value is fed forward around the PID equation, and summed with the output.



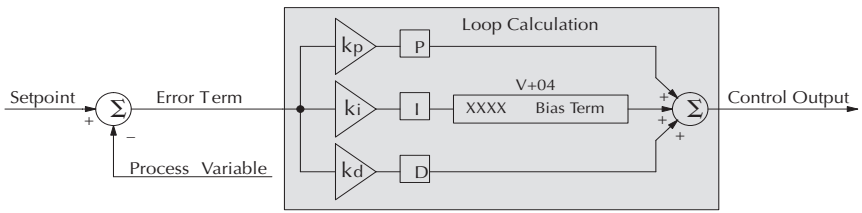
In the previous section on the bias term, we said that "the bias term value establishes a working region or operating point for the control output. *When the error fluctuates around its zero point, the output fluctuates around the bias value.*" Now, when there is a change in setpoint, an error is generated and the output must change to a new operating point. This also happens if a disturbance introduces a new offset in the loop. The loop does not really know its way to the new operating point... the integrator (bias) must increment/decrement until the error disappears, and then the bias has found the new operating point.

Suppose that we are able to know a sudden setpoint change is about to occur (common in some applications). We can avoid much of the resulting error in the first place, if we can quickly change the output to the new operating point. If we know (from previous testing) what the operating point (bias value) will be after the setpoint change, we can artificially change the output directly (which is feedforward). The benefits of using feedforward are:

- The SP–PV error is reduced during predictable setpoint changes or loop offset disturbances.
- Proper use of feedforward will allow us to reduce the integrator gain. Reducing integrator gain gives us an even more stable control system.

Feedforward is very easy to use in the DL06 loop controller, as shown below. The bias term has been made available to the user in a special read/write location, at PID Parameter Table location V+04.

Parameter Table location V+04.

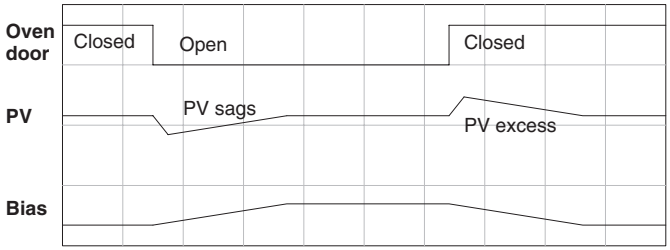


To change the bias (operating point), ladder logic only has to write the desired value to V+04. The PID loop calculation first reads the bias value from V+04 and modifies the value based on the current integrator calculation. Then it writes the result back to location V+04. This arrangement creates a sort of transparent bias term. All you have to do to implement feed forward control is write the correct value to the bias term at the right time (see the following example).

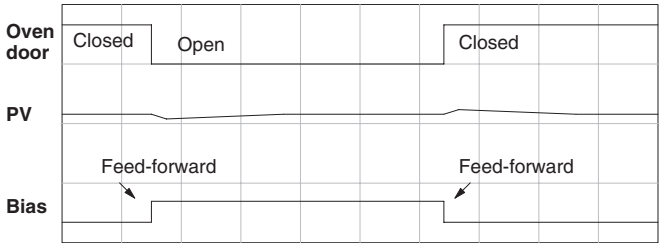
***NOTE**: When writing the bias term, one must be careful to design ladder logic to write the value only once, at the moment when the new bias operating point is to occur. If ladder logic writes the bias value on every scan, the loop's integrator is effectively disabled.*

Feedforward Example

How do we know when to write to the bias term, and what value to write? Suppose we have an oven temperature control loop, and we have already tuned the loop for optimal performance. Refer to the figure below. We notice that when the operator opens the oven door, the temperature sags a bit while the loop bias adjusts to the heat loss. Then, when the door closes, the temperature rises above the SP until the loop adjusts again. Feedforward control can help diminish this effect.



First, we record the amount of bias change the loop controller generates when the door opens or closes. Then, we write a ladder program to monitor the position of an oven door limit switch. When the door opens, our ladder program reads the current bias value from V+04, adds the desired change amount, and writes it back to V+04. When the door closes, we duplicate the procedure, but subtracting desired change amount instead. The following figure shows the results.



The step changes in the bias are the result of our two feed-forward writes to the bias term. We can see the PV variations are greatly reduced. The same technique may be applied for changes in setpoint.

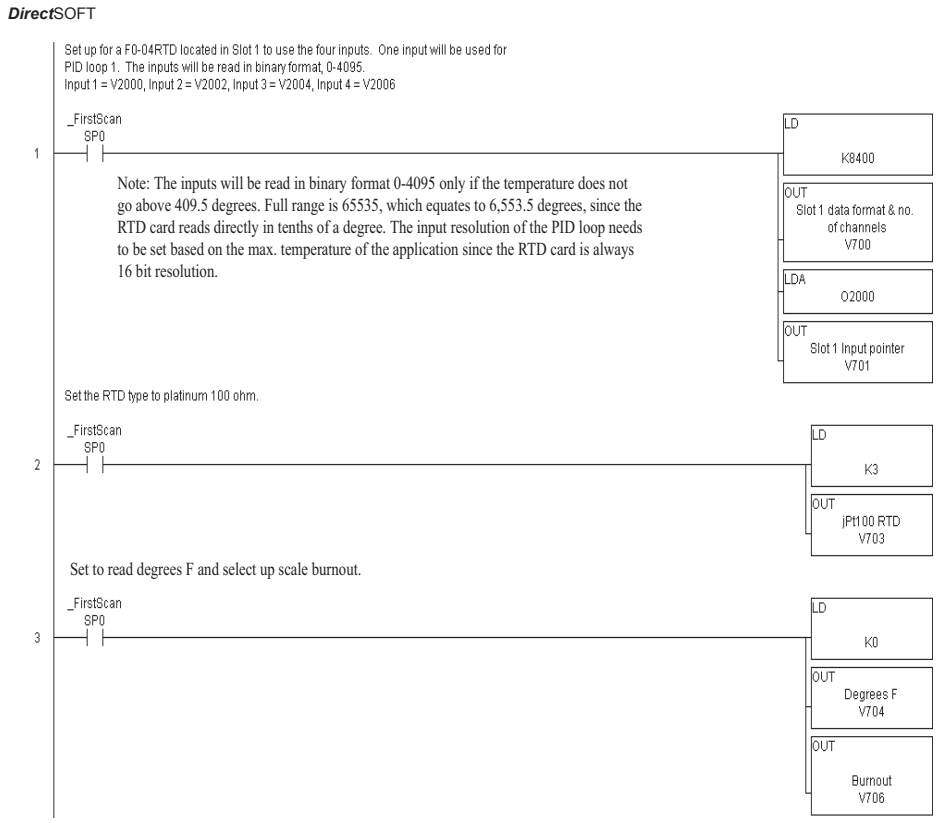
PID Example Program

Program Setup for the PID Loop

After setting up the PID loop or loops, with *Direct*SOFT, you will need to edit your RLL program to include the rungs needed to setup the analog I/O module to be used by the PID loop(s).

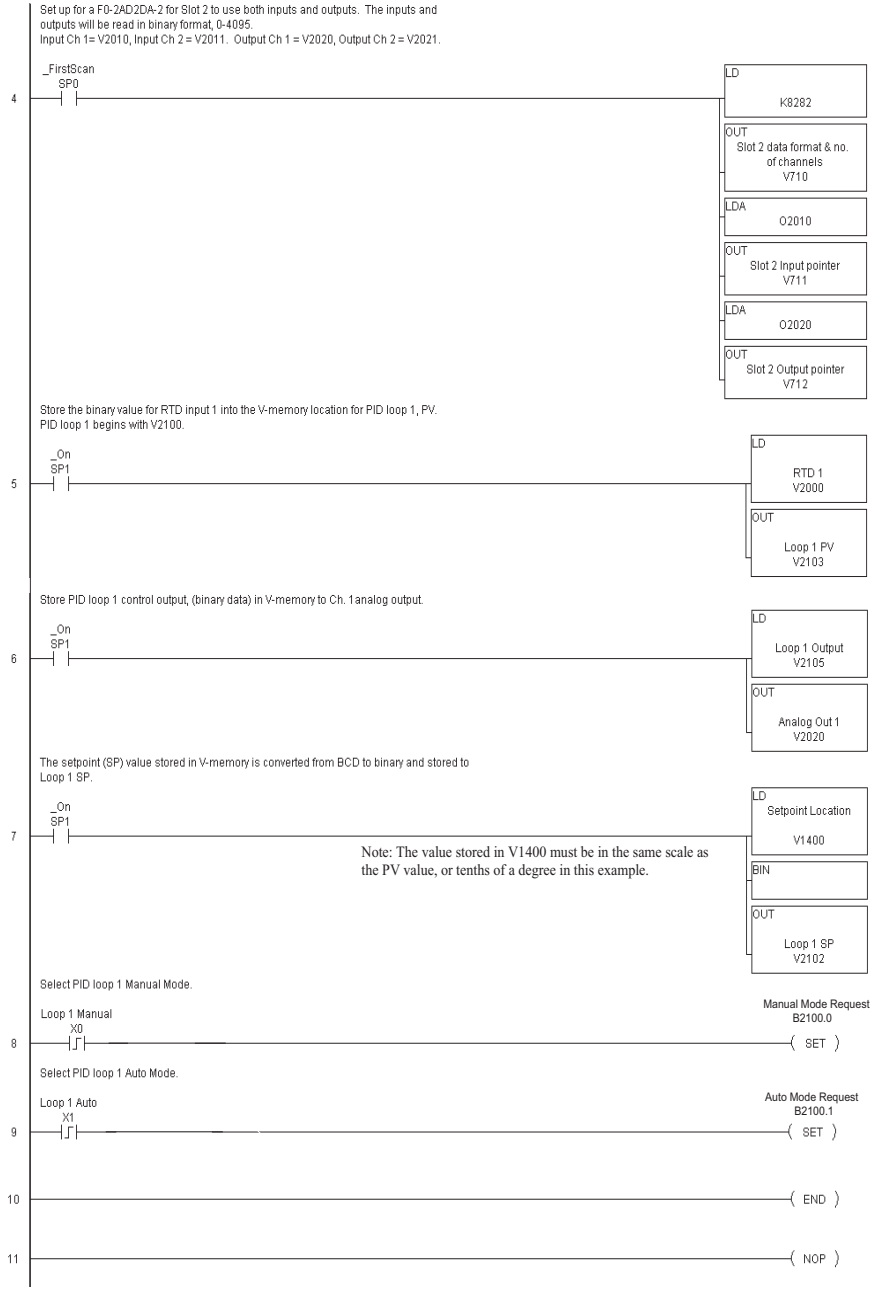
The following example program shows how an RTD module, F0-04RTD, and an analog combination module, F0-2AD2DA-2, are used and setup for a PID loop. This example assumes that the PID table for loop 1 has a beginning address of V2100.

All of the analog I/O modules used with the DL06 are setup in a similar manner. Refer to the DL05/DL06 Options Manual for the setup information for the particular module that you will be using.



Program continued on next page

Example program continued



Note that the modules used in the PID loop example program were set up for binary format. They could have been set up for BCD format. In the latter case, the BCD data would have to be converted to binary format before being stored to the setpoint and process variable, and the control output would have to be converted from BCD to binary before being stored to the analog output.

By following the steps outlined in this chapter, you should be able to set up workable PID control loops. The *Direct*SOFT Programming Software Manual provides more information for the use of PID View.

For a step-by-step tutorial, go to the Technical Support section located on our website, **www.automationdirect.com**. Once you are at the website, click on **Technical Support Home**. After this page opens, find and select **Guided Tutorials** located under the **Using Your Products** column. An **Animated Tutorial** page will open. Under **Available Tutorials**, find **PID Trainer** and select **View the Powerpoint slide show** and begin viewing the tutorial. The Powerpoint Viewer can be downloaded if your computer does not have Powerpoint installed.

Troubleshooting Tips

Q. The loop will not go into Automatic Mode.

A. Check the following for possible causes:

- A PV alarm exists, or a PV alarm programming error exists.
- The loop is the major loop of a cascaded pair, and the minor loop is not in Cascade Mode.

Q. The Control Output stays at zero constantly when the loop is in Automatic Mode.

A. Check the following for possible causes:

- The Control Output upper limit in loop table location V+31 is zero.
- The loop is driven into saturation, because the error never goes to zero value and changes (algebraic) sign.

Q. The Control Output value is not zero, but it is incorrect.

A. Check the following for possible causes:

- The gain values are entered improperly. Remember, gains are entered in the loop table in BCD, while the SP and PV are in binary. If you are using *Direct*SOFT, it displays the SP, PV, Bias and Control output in decimal (BCD), converting it to binary before updating the loop table.

Q. The Ramp/Soak Generator does not operate when I activate the Start bit.

A. Check the following for possible causes:

- The Ramp/Soak enable bit is off. Check the status of bit 11 of loop parameter table location V+00. It must be set =1.
- The hold bit or other bits in the Ramp/Soak control are on.
- The beginning SP value and the first ramp ending SP value are the same, so first ramp segment has no slope and consequently has no duration. The ramp/soak generator moves quickly to the soak segment, giving the illusion the first ramp is not working.
- The loop is in Cascade Mode, and is trying to get the SP remotely.
- The SP upper limit value in the loop table location V+27 is too low.
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode, and manually start the ramp/soak generator.

Q. The PV value in the table is constant, even though the analog module receives the PV signal.

A. Your ladder program must read the analog value from the module successfully and write it into the loop table V+03 location. Verify the analog module is generating the value, and the ladder is working.

Q. The Derivative gain doesn't seem to have any affect on the output.

A. The derivative limit is probably enabled (see section on derivative gain limiting).

Q. The loop Setpoint appears to be changing by itself.

A. Check the following for possible causes:

- The Ramp/Soak generator is enabled, and is generating setpoints.
- If this symptom occurs on loop Manual-to-Auto Mode changes, the loop is in Bumpless Transfer 1.
- Check your ladder program to verify it is not writing to the SP location (V+02 in the loop table). A quick way to do this is to temporarily place an end coil at the beginning of your program, then go to PLC Run Mode.

Q. The SP and PV values I enter with *Direct*SOFT work okay, but these values do not work properly when the ladder program writes the data.

A. The PID View in *Direct*SOFT lets you enter SP, PV, and Bias values in decimal, and displays them in decimal for your convenience. For example, when the data format is 12 bit unipolar, the values range from 0 to 4095. However, the loop table actually requires these in hex, so *Direct*SOFT converts them for you. The values in the table range from 0 to FFF, for 12-bit unipolar format.

Q. The loop seems unstable and impossible to tune, no matter what gains I use.

A. Check the following for possible causes:

- The loop sample time is set too long. Refer to the section near the front of this chapter on selecting the loop update time.
- The gains are too high. Start out by reducing the derivative gain to zero. Then reduce the integral gain, and the proportional gain if necessary.
- There is too much transfer lag in your process. This means the PV reacts sluggishly to control output changes. There may be too much “distance” between actuator and PV sensor, or the actuator may be weak in its ability to transfer energy into the process.
- There may be a process disturbance that is over-powering the loop. Make sure the PV is relatively steady when the SP is not changing.

Glossary of PID Loop Terminology

Automatic Mode An operational mode of a loop, in which it makes PID calculations and updates the loop's control output.

Bias Freeze A method of preserving the bias value (operating point) for a control output, by inhibiting the integrator when the output goes out-of-range. The benefit is a faster loop recovery.

Bias Term In the position form of the PID equation, it is the sum of the integrator and the initial control output value.

Bumpless Transfer A method of changing the operation mode of a loop while avoiding the usual sudden change in control output level. This consequence is avoided by artificially making the SP and PV equal, or the bias term and control output equal at the moment of mode change.

Cascaded Loops A cascaded loop receives its setpoint from the output of another loop. Cascaded loops have a major/minor relationship, and work together to ultimately control one PV.

Cascade Mode An operational mode of a loop, in which it receives its SP from another loop's output.

Continuous Control Control of a process done by delivering a smooth (analog) signal as the control output.

Control Output The numerical result of a PID equation which is sent by the loop with the intention of nulling out the current error.

Derivative Gain A constant that determines the magnitude of the PID derivative term in response to the current error.

Direct-Acting Loop A loop in which the PV increases in response to a control output increase. In other words, the process has a positive gain.

Error The difference in value between the SP and PV, Error = SP – PV

Error Deadband An optional feature which makes the loop insensitive to errors when they are small. You can specify the size of the deadband.

Error Squared An optional feature which multiplies the error by itself, but retains the original algebraic sign. It reduces the effect of small errors, while magnifying the effect of large errors.

Feedforward A method of optimizing the control response of a loop when a change in setpoint or disturbance offset is known and has a quantifiable effect on the bias term.

Integral Gain A constant that determines the magnitude of the PID integral term in response to the current error.

Major Loop In cascade control, it is the loop that generates a setpoint for the cascaded loop.

Manual Mode An operational mode of a loop, it which the PID calculations are stopped. The operator must manually control the loop by writing to the control output value directly.

Minor Loop In cascade control, the minor loop is the subordinate loop that receives its SP from the major loop.

On/Off Control A simple method of controlling a process, through on/off application of energy into the system. The mass of the process averages the on/off effect for a relatively smooth PV. A simple ladder program can convert the DL06's continuous loop output to on/off control.

PID Loop A mathematical method of closed-loop control involving the sum of three terms based on proportional, integral, and derivative error values. The three terms have independent gain constants, allowing one to optimize (tune) the loop for a particular physical system.

Position Algorithm The control output is calculated so it responds to the displacement (position) of the PV from the SP (error term)

Process A manufacturing procedure which adds value to raw materials. Process control particularly refers to inducing chemical changes to the material in process.

Process Variable (PV) A quantitative measurement of a physical property of the material in process, which affects final product quality and is important to monitor and control.

Proportional Gain A constant that determines the magnitude of the PID proportional term in response to the current error.

PV Absolute Alarm A programmable alarm that compares the PV value to alarm threshold values.

PV Deviation Alarm A programmable alarm that compares the difference between the SP and PV values to a deviation threshold value.

Ramp/Soak Profile A set of SP values called a profile, which is generated in real time upon each loop calculation. The profile consists of a series of ramp and soak segment pairs, greatly simplifying the task of programming the PLC to generate such SP sequences.

Rate Also called differentiator, the rate term responds to the changes in the error term.

Remote Setpoint The location where a loop reads its setpoint when it is configured as the minor loop in a cascaded loop topology.

Reset Also called integrator, the reset term adds each sampled error to the previous, maintaining a running total called the bias.

Reset Windup A condition created when the loop is unable to find equilibrium, and the persistent error causes the integrator (reset) sum to grow excessively (windup). Reset windup causes an extra recovery delay when the original loop fault is remedied.

Reverse-Acting Loop A loop in which the PV increases in response to a control output decrease. In other words, the process has a negative gain.

Sampling time The time between PID calculations. The CPU method of process control is called a sampling controller, because it samples the SP and PV only periodically.

Setpoint (SP) The desired value for the process variable. The setpoint (SP) is the input command to the loop controller during closed loop operation.

Soak Deviation The soak deviation is a measure of the difference between the SP and PV during a soak segment of the Ramp/Soak profile, when the Ramp/Soak generator is active.

Step Response The behavior of the process variable in response to a step change in the SP (in closed loop operation), or a step change in the control output (in open loop operation)

Transfer To change from one loop operational mode to another (between Manual, Auto, or Cascade). The word "transfer" probably refers to the transfer of control of the control output or the SP, depending on the particular mode change.

Velocity Algorithm The control output is calculated to represent the rate of change (velocity) for the PV to become equal to the SP.

Bibliography

Fundamentals of Process Control Theory, Second Edition Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1–55617–297–4	Application Concepts of Process Control Author: Paul W. Murrill Publisher: Instrument Society of America ISBN 1–55617–080–7
PID Controllers: Theory, Design, and Tuning, 2nd Edition Author: K. Astrom and T Hagglund Publisher: Instrument Society of America ISBN 1–55617–516–7	Fundamentals of Temperature, Pressure, and Flow Measurements, Third edition Author: Robert P. Benedict Publisher: John Wiley and Sons ISBN 0–471–89383–8
Process / Industrial Instruments & Controls Handbook, Fourth Edition Author (Editor-in-Chief): Douglas M. Considine Publisher: McGraw-Hill, Inc ISBN 0-07-012445-0	pH Measurement and Control, Second Edition Author: Gregory K. McMillan Publisher: Instrument Society of America ISBN 1–55617–483–7
Programmable Controllers Concepts and Applications, First Edition Authors: C.T. Jones and L.A. Bryan Publisher: International Programmable Controls ISBN 0-915425-00-9	Fundamentals of Programmable Logic Controllers, Sensors, and Communications Author: Jon Stenerson Publisher: Prentice Hall ISBN 0-13-726860-2
Process Control, Third Edition Instrument Engineer's Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0–8019–8242–1	Process Measurement and Analysis, Third Edition Instrument Engineer's Handbook Author (Editor-in-Chief): Bela G. Liptak Publisher: Chilton ISBN 0–8019–8197–2

Notes

Maintenance and Troubleshooting

In This Chapter...

Hardware System Maintenance

Standard Maintenance

No regular or preventative maintenance is required for this product (there are no internal batteries); however, a routine maintenance check (about every one or two months) of your PLC and control system is good practice, and should include the following items:

- Air Temperature – Monitor the air temperature in the control cabinet, so the operating temperature range of any component is not exceeded.
- Air Filter – If the control cabinet has an air filter, clean or replace it periodically as required.
- Fuses or breakers – Verify that all fuses and breakers are intact.
- Cleaning the Unit – Check that all air vents are clear. If the exterior case needs cleaning, disconnect the input power, and carefully wipe the case using a damp cloth. Do not let water enter the case through the air vents and do not use strong detergents because this may discolor the case.

Diagnostics

Diagnostics

Your DL06 Micro PLC performs many pre-defined diagnostic routines with every CPU scan. The diagnostics can detect various errors or failures in the PLC. The two primary error classes are *fatal and non-fatal.*

Fatal Errors

Fatal errors are errors which may cause the system to function improperly, perhaps introducing a safety problem. The CPU will automatically switch to Program Mode if it is in Run Mode. (Remember, in Program Mode all outputs are turned off.) If the fatal error is detected while the CPU is in Program Mode, the CPU will not allow you to transition to Run Mode until the error has been corrected.

Some examples of fatal errors are:

- Power supply failure
- Parity error or CPU malfunction
- Particular programming errors

Non-fatal Errors

Non-fatal errors are errors that need your attention, but should not cause improper operation. They do not cause or prevent any mode transitions of the CPU. The application program can use special relay contacts to detect non-fatal errors, and even take the system to an orderly shutdown or switch the CPU to Program Mode if desired. An example of a non-fatal error is:

- Particular programming errors - The programming devices will notify you of an error if one occurs while online.
- *Direct*SOFT provides the error number and an error message.
- The handheld programmer displays error numbers and short descriptions of the error.

Appendix B has a complete list of error messages in order by error number. Many error messages point to supplemental V-memory locations which contain related information. Special relays (SP contacts) also provide error indications (refer to Appendix D).

V-memory Error Code Locations

The following table names the specific memory locations that correspond to certain types of error messages.

Error Class	Error Category	Diagnostic V-memory
User-Defined	Error code used with FAULT instruction	V7751
System Error	Fatal Error code	V7755
	Major Error code	V7756
	Minor Error code	V7757
Grammatical	Address where syntax error occurs	V7763
	Error Code found during syntax check	V7764
CPU Scan	Number of scans since last Program to Run Mode transition	V7765
	Current scan time (ms)	V7775
	Minimum scan time (ms)	V7776
	Maximum scan time (ms)	V7777

Special Relays (SP) Corresponding to Error Codes

The special relay table also includes status indicators which can indicate errors. For a more detailed description of each of these special relays refer to Appendix D.

CPU Status Relays	
SP11	Forced Run mode
SP12	Terminal Run mode
SP13	Test Run mode
SP15	Test stop mode
SP16	Terminal Program mode
SP17	Forced stop
SP20	STOP instruction was executed
SP22	Interrupt enabled
System Monitoring Relays	
SP36	Override setup
SP37	Scan control error
SP40	Critical error
SP41	Non-critical error
SP42	Diagnostics error
SP44	Program memory error
SP45	I/O error
SP46	Communications error
SP50	Fault instruction was executed
SP51	Watchdog timeout
SP52	Syntax error
SP53	Cannot solve the logic
SP54	Communication error
SP56	Table instruction overrun

Accumulator Status Relays	
SP60	Acc. is less than value
SP61	Acc. is equal to value
SP62	Acc. is greater than value
SP63	Acc. result is zero
SP64	Half borrow occurred
SP65	Borrow occurred
SP66	Half carry occurred
SP67	Carry occurred
SP70	Result is negative (sign)
SP71	Pointer reference error
SP73	Overflow
SP75	Data is not in BCD
SP76	Load zero

DL06 Micro PLC Error Codes

These errors can be generated by the CPU or by the Handheld Programmer, depending on the actual error. Appendix B provides a more complete description of the error codes.

The errors can be detected at various times. However, most of them are detected at power-up, on entry to Run Mode, or when a Handheld Programmer key sequence results in an error or an illegal request.

Error Code	Description
E003	Software time-out
E004	Invalid instruction(RAM parity error in the CPU)
E104	Write failed
E151	Invalid command
E311	Communications error 1
E312	Communications error 2
E313	Communications error 3
E316	Communications error 6
E320	Time out
E321	Communications error
E360	HP Peripheral port time-out
E501	Bad entry
E502	Bad address
E503	Bad command
E504	Bad reference / value
E505	Invalid instruction
E506	Invalid operation
E520	Bad operation – CPU in Run
E521	Bad operation – CPU in Test Run
E523	Bad operation – CPU in Test Program
E524	Bad operation – CPU in Program
E525	Mode Switch not in Term position

Error Code	Description
E526	Unit is offline
E527	Unit is online
E528	CPU mode
E540	CPU locked
E541	Wrong password
E542	Password reset
E601	Memory full
E602	Instruction missing
E604	Reference missing
E620	Out of memory
E621	EEPROM Memory not blank
E622	No Handheld Programmer EEPROM
E624	V memory only
E625	Program only
E627	Bad write operation
E628	Memory type error (should be EEPROM)
E640	Mis-compare
E650	Handheld Programmer system error
E651	Handheld Programmer ROM error
E652	Handheld Programmer RAM error

Program Error Codes

The following table lists program syntax and runtime error codes. Error detection occurs during a Program-to-Run mode transition, or when you use AUX 21 – Check Program. The CPU will also turn on SP52 and store the error code in V7755. Appendix B provides a more complete description of the error codes.

Error Code	Description
E4**	No Program in CPU
E401	Missing END statement
E402	Missing LBL
E403	Missing RET
E404	Missing FOR
E405	Missing NEXT
E406	Missing IRT
E412	SBR / LBL >64
E421	Duplicate Stage reference
E422	Duplicate SBR/LBL reference
E423	Nested Loop
E431	Invalid ISG/SG address
E433	Invalid ISG / SG address
E434	Invalid RTC
E435	Invalid RT
E436	Invalid INT address
E437	Invalid IRTC

Error Code	Description
E438	Invalid IRT address
E440	Invalid Data Address
E441	ACON/NCON
E451	Bad MLS/MLR
E453	Missing T/C
E454	Bad TMRA
E455	Bad CNT
E456	Bad SR
E461	Stack Overflow
E462	Stack Underflow
E463	Logic Error
E464	Missing Circuit
E471	Duplicate Coil reference
E472	Duplicate TMR reference
E473	Duplicate CNT reference
E499	Print Instruction

CPU Indicators

The DL06 Micro PLCs have indicators on the front to help you determine potential problems with the system. In normal runtime operation only, the RUN and PWR indicators are on. The table below is a quick reference to potential problems.

Indicator Status	Potential Problems
PWR (Green LED off)	System voltage incorrect
	PLC power supply faulty
RUN (Green LED off)	CPU programming error
	CPU in program mode
RUN (Green LED flashing)	CPU in firmware upgrade mode
CPU (Red LED on)	Electrical noise interference
	Internal CPU defective
CPU (Blinking Red LED)	Low backup battery (refer to page 3-8)

PWR Indicator

In general there are three reasons for the CPU power status LED (PWR) to be OFF:

1. Power to the unit is incorrect or is not applied.
2. PLC power supply is faulty.
3. Other component(s) have the power supply shut down.

If the voltage to the power supply is not correct, the PLC may not operate properly or may not operate at all. Use the following guidelines to correct the problem.

WARNING: To minimize the risk of electrical shock, always disconnect the system power before inspecting the physical wiring.

1. First, disconnect the external power.
2. Verify that all external circuit breakers or fuses are still intact.
3. Check all incoming wiring for loose connections. If you're using a separate termination block, check those connections for accuracy and integrity.
4. If the connections are acceptable, reconnect the system power and verify the voltage at the DL06 power input is within specification. If the voltage is not correct, shut down the system and correct the problem.
5. If all wiring is connected correctly and the incoming power is within the specifications, the PLC internal supply may be faulty.

The best way to check for a faulty PLC is to substitute a known good one to see if this corrects the problem. The removable connectors on the DL06 make this relatively easy. If there has been a major power surge, it is possible the PLC internal power supply has been damaged. If you suspect this is the cause of the power supply damage, consider installing an AC line conditioner to attenuate damaging voltage spikes in the future.

RUN Indicator

If the CPU will not enter the Run mode (the RUN indicator is off), the problem is usually in the application program, unless the CPU has a fatal error. If a fatal error has occurred, the CPU LED should be on. (You can use a programming device to determine the cause of the error.)

Both of the programming devices, Handheld Programmer and *Direct*SOFT, will return an error message describing the problem. Depending on the error, there may also be an AUX function you can use to help diagnose the problem. The most common programming error is "Missing END Statement". All application programs require an END statement for proper termination. A complete list of error codes can be found in Appendix B.

The RUN indicator will flash (blink) whenever the CPU is in the firmware upgrade mode.

CPU Indicator

If the CPU indicator is on, a fatal error has occurred in the CPU. Generally, this is not a programming problem but an actual hardware failure. You can power cycle the system to clear the error. If the error clears, you should monitor the system and determine what caused the problem. You will find this problem is sometimes caused by high frequency electrical noise introduced into the CPU from an outside source. Check your system grounding and install electrical noise filters if the grounding is suspected. If power cycling the system does not reset the error, or if the problem returns, you should replace the CPU.

If the CPU indicator is blinking, the backup battery is low (refer to page 3-8).

Communications Problems

If you cannot establish communications with the CPU, check these items.

- The cable is disconnected.
- The cable has a broken wire or has been wired incorrectly.
- The cable is improperly terminated or grounded.
- The device connected is not operating at the correct baud rate (9600 baud).
- The device connected to the port is sending data incorrectly, or another application is running on the device.
- A grounding difference exists between the two devices.
- Electrical noise is causing intermittent errors.
- The PLC has a bad communication port and should be replaced.

For problems in communicating with *Direct*SOFT on a personal computer, refer to the *Direct*SOFT programming user manual. It includes a troubleshooting section that can help you diagnose PC problems in communications port setup, address or interrupt conflicts, etc.

I/O Point Troubleshooting

Possible Causes

If you suspect an I/O error, there are several things that could be causing the problem.

- High-Speed I/O configuration error
- A blown fuse in your machine or panel (the DL06 does not have internal I/O fuses)
- A loose terminal block
- The auxiliary 24 VDC supply has failed
- The Input or Output Circuit has failed

Some Quick Steps

When troubleshooting the DL06 Micro PLCs, please be aware of the following facts which may assist you in quickly correcting an I/O problem.

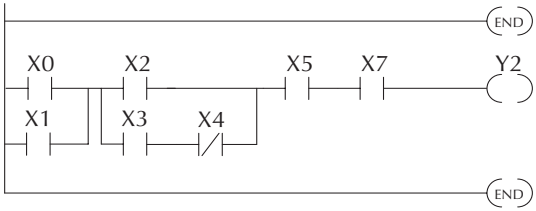
- HSIO configuration errors are commonly mistaken for I/O point failure during program development. If the I/O point in question is in X0–X2, or Y0–Y1, check all parameter locations listed in Chapter 3 that apply to the HSIO mode you have selected.
- The output circuits cannot detect shorted or open output points. If you suspect one or more faulty points, measure the voltage drop from the common to the suspect point. Remember when using a Digital Volt Meter, leakage current from an output device such as a triac or a transistor must be considered. A point which is off may appear to be on if no load is connected the point.
- The I/O point status indicators are logic-side indicators. This means the LED which indicates the on or off status reflects the status of the point with respect to the CPU. On an output point the status indicators could be operating normally while the actual output device (transistor, triac etc.) could be damaged. With an input point, if the indicator LED is on, the input circuitry is probably operating properly. Verify the LED goes off when the input signal is removed.
- Leakage current can be a problem when connecting field devices to an I/O point. False input signals can be generated when the leakage current of an output device is great enough to turn on the connected input device. To correct this install a resistor in parallel with the input or output of the circuit. The value of this resistor will depend on the amount of leakage current and the voltage applied but usually a 10K to 20K resistor will work. Verify the wattage rating of the resistor is correct for your application.
- Because of the removable terminal blocks on the DL06, the easiest method to determine if an I/O circuit has failed is to replace the unit if you have a spare. However, if you suspect a field device is defective, that device may cause the same failure in the replacement PLC as well. As a point of caution, you may want to check devices or power supplies connected to the failed I/O circuit before replacing the unit with a spare.

Output points can be set on or off in the DL06 series CPUs. If you want to do an I/O check-out independent of the application program, follow the procedure below:

Step	Action
1	Use a handheld programmer or ***Direct***SOFT to communicate online to the PLC.
2	Change to Program Mode.
3	Go to address 0.
4	Insert an "END" statement at address 0. (This will cause program execution to occur only at address 0 and prevent the application program from turning the I/O points on or off).
5	Change to Run Mode.
6	Use the programming device to set (turn) on or off the points you wish to test.
7	When you finish testing I/O points delete the "END" statement at address 0.

WARNING: Depending on your application, forcing I/O points may cause unpredictable machine operation that can result in a risk of personal injury or equipment damage. Make sure you have taken all appropriate safety precautions prior to testing any I/O points.

Handheld Programmer Keystrokes Used to Test an Output Point



Insert an END statement at the beginning of the program. This disables the remainder of the program.

From a clear display, use the following keystrokes

STAT ENT

```
16P STATUS
BIT REF   X
```

Use the PREV or NEXT keys to select the Y data type

NEXT A 0 ENT

```
      Y 10          Y0
□□□□□□□□□□□□□□□□
```

Use arrow keys to select point, then use ON and OFF to change the status

← ← SHFT ON INS

Y2 is now on

```
      Y 10          Y0
□□□□□□□□□□□□□■□□
```

Noise Troubleshooting

Electrical Noise Problems

Noise is one of the most difficult problems to diagnose. Electrical noise can enter a system in many different ways and falls into one of two categories, conducted or radiated. It may be difficult to determine how the noise is entering the system but the corrective actions for either of the types of noise problems are similar.

- Conducted noise is when the electrical interference is introduced into the system by way of an attached wire, panel connection ,etc. It may enter through an I/O circuit, a power supply connection, the communication ground connection, or the chassis ground connection.
- Radiated noise is when the electrical interference is introduced into the system without a direct electrical connection, much in the same manner as radio waves.

Reducing Electrical Noise

While electrical noise cannot be eliminated it can be reduced to a level that will not affect the system.

- Most noise problems result from improper grounding of the system. A good earth ground can be the single most effective way to correct noise problems. If a ground is not available, install a ground rod as close to the system as possible. Ensure all ground wires are single point grounds and are not daisy chained from one device to another. Ground metal enclosures around the system. A loose wire can act as a large antenna, introducing noise into the system, so, tighten all connections in your system. Loose ground wires are more susceptible to noise than the other wires in your system. Review Chapter 2 Installation, Wiring, and Specifications if you have questions regarding how to ground your system.
- Electrical noise can enter the system through the power source for the PLC and I/O circuits. Installing an isolation transformer for all AC sources can correct this problem. DC sources should be well-grounded good quality supplies.
- Separate input wiring from output wiring. Never run low-voltage I/O wiring close to high voltage wiring.

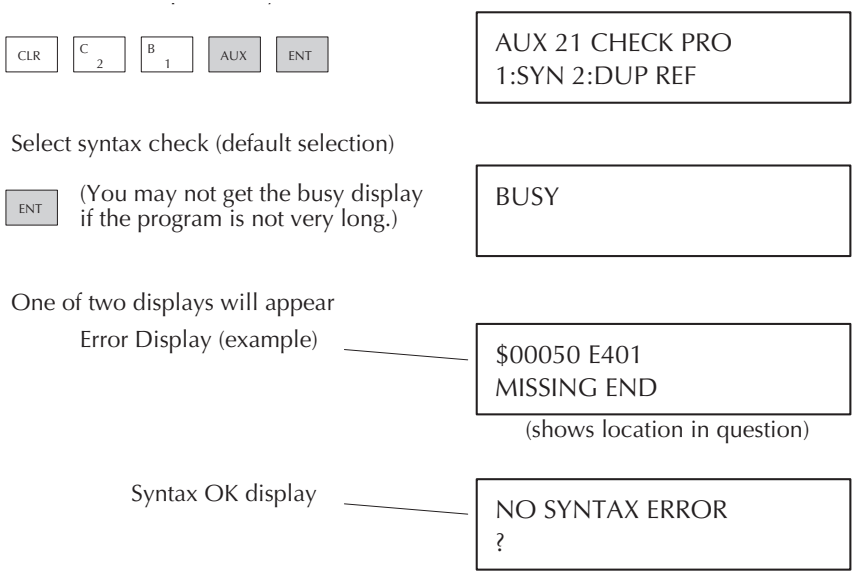
Machine Startup and Program Troubleshooting

The DL06 Micro PLCs provide several features that can help you debug your program before and during machine startup. This section discusses the following topics which can be very helpful.

- Program Syntax Check
- Duplicate Reference Check
- Special Instructions
- Run Time Edits
- Forcing I/O Points

Syntax Check

Even though the Handheld Programmer and *Direct*SOFT provide error checking during program entry, you may want to check a program that has been modified. Both programming devices offer a way to check the program syntax. For example, you can use AUX 21, CHECK PROGRAM to check the program syntax from a Handheld Programmer, or you can use the PLC Diagnostics menu option within *Direct*SOFT. This check will find a wide variety of programming errors. The following example shows how to use the syntax check with a Handheld Programmer.



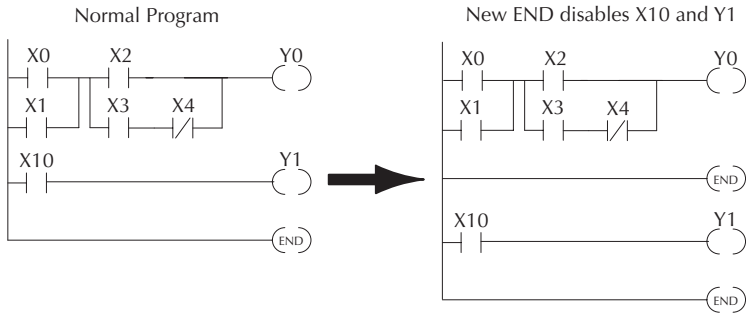
See the Error Codes Section for a complete listing of programming error codes. If you get an error, just press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Syntax check until the NO SYNTAX ERROR message appears.

Special Instructions

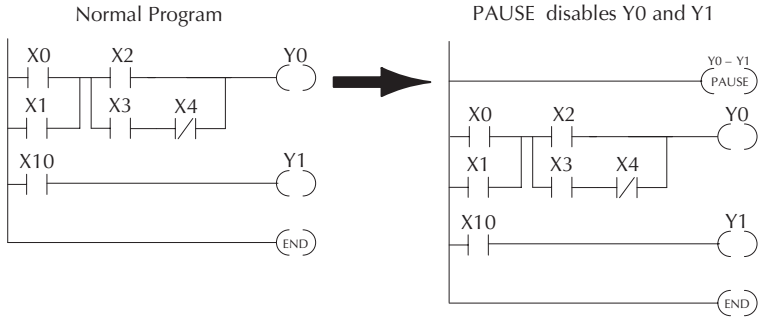
There are several instructions that can be used to help you debug your program during machine startup operations.

- END
- PAUSE
- STOP

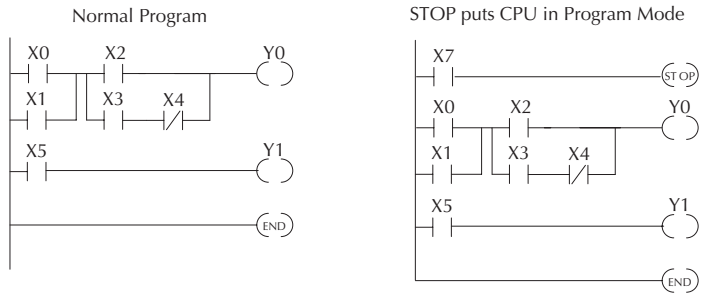
END Instruction: If you need a way to quickly disable part of the program, just insert an END statement prior to the portion that should be disabled. When the CPU encounters the END statement, it assumes that is the end of the program. The following diagram shows an example.



PAUSE Instruction: This instruction provides a quick way to allow the inputs (or other logic) to operate while disabling selected outputs. The output image register is still updated, but the output circuits are not. For example, you could make this conditional by adding an input contact or CR to control the instruction with a switch or a programming device. Or, you could just add the instruction without any conditions so the selected outputs would be disabled at all times.



STOP Instruction: Sometimes during machine startup you need a way to quickly turn off all the outputs and return to Program Mode. You can use the STOP instruction. When this instruction is executed the CPU automatically exits Run Mode and enters Program Mode. Remember, all outputs are turned off during Program Mode. The following diagram shows an example of a condition that returns the CPU to Program Mode.



In the example shown above, you could trigger X7 which would execute the STOP instruction. The CPU would enter Program Mode and all outputs would be turned off.

Duplicate Reference Check

You can also check for multiple uses of the same output coil. Both programming devices offer a way to check for this condition.. For example, you can AUX 21, CHECK PROGRAM to check for duplicate references from a Handheld Programmer, or you can use the PLC Diagnostics menu option within ***Direct***SOFT. The following example shows how to perform the duplicate reference check with a Handheld Programmer.

If you get an error, just press CLR and the Handheld will display the instruction where the error occurred. Correct the problem and continue running the Duplicate Reference check until no duplicate references are found.

Use AUX 21 to perform syntax check

CLR C 2 B 1 AUX ENT

```
AUX 21 CHECK PRO
1:SYN 2:DUP REF
```

Select duplicate reference check

→ ENT (You may not get the busy display if the program is not very long.)

```
BUSY
```

One of two displays will appear

Error Display (example)
(shows location in question)

```
$00024 E471
DUP COIL REF
```

Syntax OK display

```
NO DUP REFS
?
```

***NOTE**: You can use the same coil in more than one location, especially in programs containing Stage instructions and / or OROUT instructions. The Duplicate Reference check will find occurrences, even though they are acceptable.*

Run Time Edits

The DL06 Micro PLC allows you to make changes to the application program during Run Mode. These edits are not "bumpless." Instead, CPU scan is momentarily interrupted (and the outputs are maintained in their current state) until the program change is complete. This means if the output is off, it will remain off until the program change is complete. If the output is on, it will remain on.

WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Changes during Run Mode become effective immediately. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment. There are some important operational changes during Run Time Edits.
1. If there is a syntax error in the new instruction, the CPU will not enter the Run Mode.
2. If you delete an output coil reference and the output was on at the time, the output will remain on until it is forced off with a programming device.
3. Input point changes are not acknowledged during Run Time Edits, so, if you're using a high-speed operation and a critical input comes on, the CPU may not see the change.

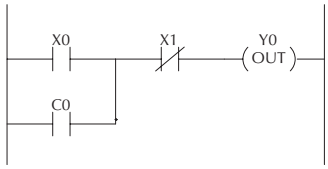
Not all instructions can be edited during a Run Time Edit session. The following list shows the instructions that can be edited.

Mnemonic	Description
TMR	Timer
TMRF	Fast timer
TMRA	Accumulating timer
TMRAF	Accumulating fast timer
CNT	Counter
UDC	Up / Down counter
SGCNT	Stage counter
STR, STRN	Store, Store not (Boolean)
AND, ANDN	And, And not (Boolean)
OR, ORN	Or, Or not (Boolean)
STRE, STRNE	Store equal, Store not equal
ANDE, ANDNE	And equal, And not equal
ORE, ORNE	Or equal, Or not equal
STR, STRN	Store greater than or equal Store less than (Comparative Boolean)
AND, ANDN	And greater than or equal And less than (Comparative Boolean)

Mnemonic	Description
OR, ORN	Or greater than or equal or less than (Comparative Boolean)
LD	Load data (constant)
LDD	Load data double (constant)
ADDD	Add data double (constant)
SUBD	Subtract data double (constant)
MUL	Multiply (constant)
DIV	Divide (constant)
CMPD	Compare accumulator (constant)
ANDD	And accumulator (constant)
ORD	Or accumulator (constant)
XORD	Exclusive or accumulator (constant)
LDF	Load discrete points to accumulator
OUTF	Output accumulator to discrete points
SHFR	Shift accumulator right
SHFL	Shift accumulator left
NCON	Numeric constant

Run Time Edit Example

We'll use the program logic shown to describe ho process works. In the example, we'll change X0 to the example assumes you have already placed the Run Mode.



Use the MODE key to select Run Time Edits

MODE NEXT NEXT ENT

```
*MODE CHANGE*
RUN TIME EDIT?
```

Press ENT to confirm the Run Time Edits

ENT (Note, the RUN LED on the D2–HPP Handheld starts flashing to indicate Run Time Edits are enabled.)

```
*MODE CHANGE*
RUNTIME EDITS
```

Find the instruction you want to change (X0)

SHFT X SET A 0 SHFT FD REF FIND

```
$00000 STR X0
```

Press the arrow key to move to the X. Then enter the new contact (C10).

→ → SHFT C 2 B 1 A 0 ENT

```
RUNTIME EDIT?
STR C10
```

Press ENT to confirm the change.

ENT (Note, once you press ENT , the next address is displayed.

```
OR C0
```

Forcing I/O Points

There are many times, especially during machine startup and troubleshooting, that you need the capability to force an I/O point to be either on or off. Before you use a programming device to force any data type, it is important to understand how the DL06 CPUs process the forcing requests.

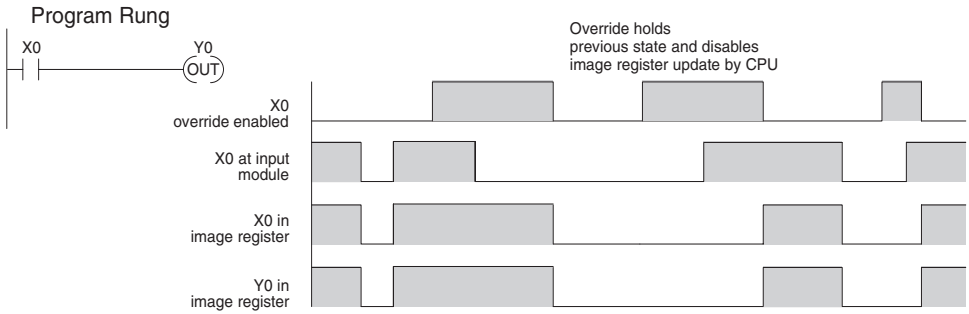
WARNING: Only authorized personnel fully familiar with all aspects of the application should make changes to the program. Make sure you thoroughly consider the impact of any changes to minimize the risk of personal injury or damage to equipment.

There are two types of forcing available with the DL06 CPUs. (Chapter 3 provides a detailed description of how the CPU processes each type of forcing request).

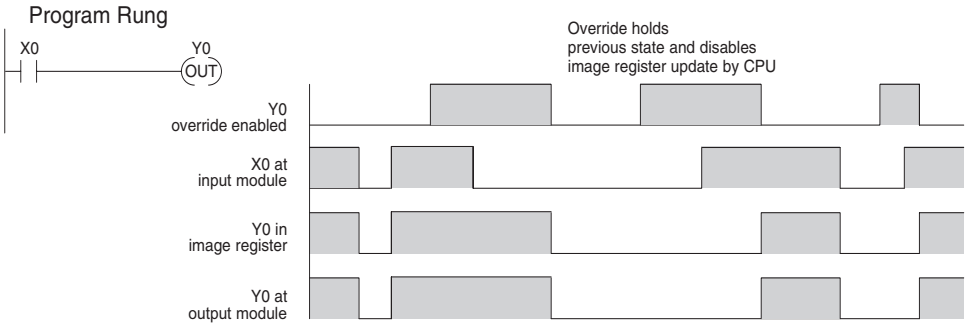
- Regular Forcing: This type of forcing can temporarily change the status of a discrete bit. For example, you may want to force an input on, even though it is really off. This allows you to change the point status that was stored in the image register. This value will be valid until the image register location is written to during the next scan. This is primarily useful during testing situations when you need to force a bit on to trigger another event.
- Bit Override: Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or by a menu option in ***Direct***SOFT. You can use Bit Override with X, Y, C, T, CT, and S data types. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. Therefore, if you used X1 in the program, it would always be evaluated as **off**, in this case. If X1 was on when the bit override was enabled, then X1 would always be evaluated as **on**.

There is an advantage available when you use the Bit Override feature. The Regular Forcing is not disabled because the Bit Override is enabled. For example, if you enabled the Bit Override for Y0 and it was off at the time, the CPU would not change the state of Y0. However, you can still use a programming device to change the status. If you use the programming device to force Y0 on, it will remain on and the CPU will not change the state of Y0. If you then force Y0 off, the CPU will maintain Y0 as off. The CPU will never update the point with the results from the application program or from the I/O update until the Bit Override is removed from the point.

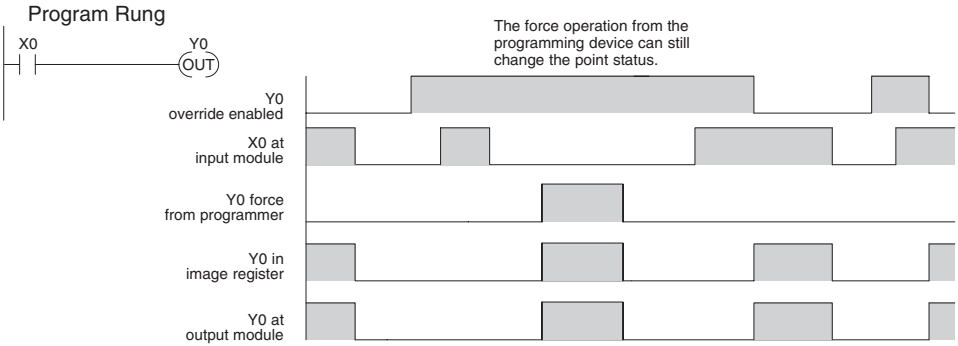
The following diagrams show how the bit override works for both input and output points. The example uses a simple rung, but the concepts are similar for any type of bit memory.



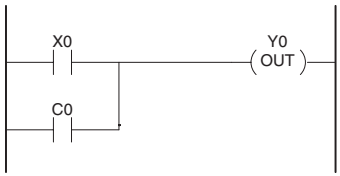
The following diagram shows how the bit override works for an output point. Notice the bit override maintains the output in the current state. If the output is on when the bit override is enabled, then the output stays on. If it is off, then the output stays off.



The following diagram shows how you can use a programming device in combination with the bit override to change the status of the point. Remember, bit override only disables CPU changes. You can still use a programming device to force the status of the point. Plus, since bit override maintains the current status, this enables true forcing. The example shown is for an output point, but you can also use the other bit data types.



The following diagrams show a brief example of how you could use the DL06 Handheld Programmer to force an I/O point. Remember, if you are using the Bit Override feature, the CPU will retain the forced value until you disable the Bit Override or until you remove the force. The image register will not be updated with the status from the input module. Also, the solution from the application program will not be used to update the output image register. The example assumes you have already placed the CPU into Run Mode.



From a clear display, use the following keystrokes

STAT ENT

```
16P STATUS
BIT REF   X
```

Use the PREV or NEXT keys to select the Y data type. Once the Y appears, press 0 to start at Y0.

NEXT A 0 ENT

```
   Y  10          Y    0
□□□□□□□□□□□□□□□□
```

Use arrow keys to select point, then use ON and OFF to change the status

← ← SHFT ON INS

Y2 is now on

```
   Y  10          Y    0
□□□□□□□□□□□□■□□
```

Regular Forcing with Direct Access

From a clear display, use the following keystrokes to force Y10 ON. Solid fill indicates point is on.

SHFT Y MLS B 1 A 0 SHFT ON INS

Solid fill indicates point is on.

```
BIT FORCE
Y10
```

From a clear display, use the following keystrokes to force Y10 OFF. No fill indicates point is off.

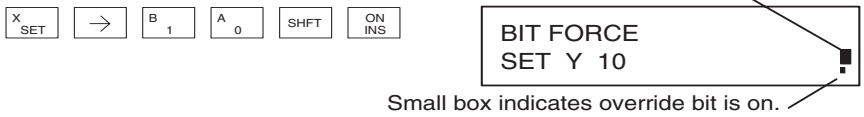
SHFT Y MLS B 1 A 0 SHFT OFF DEL

No fill indicates point is off.

```
BIT FORCE
Y10
```

Bit Override Forcing

From a clear display, use the following keystrokes to turn on the override bit for Y10.



Note, at this point you can use the PREV and NEXT keys to move to adjacent memory locations and use the SHFT ON keys to set the override bit on.

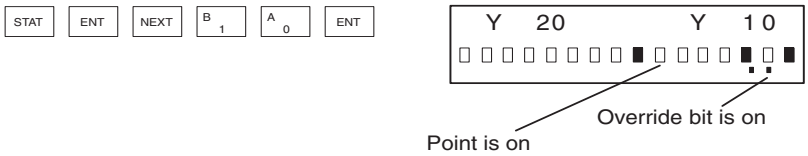
From a clear display, use the following keystrokes to turn off the override bit for Y10. Solid fill indicates point is on.

Like the example above, you can use the PREV and NEXT keys to move to adjacent memory locations and use the SHFT OFF keys to set the override bit off.

Bit Override Indicators

Override bit indicators are also shown on the handheld programmer status display. Below are the keystrokes to call the status display for Y10 – Y20.

From a clear display, use the following keystrokes to display the status of Y10 – Y20.

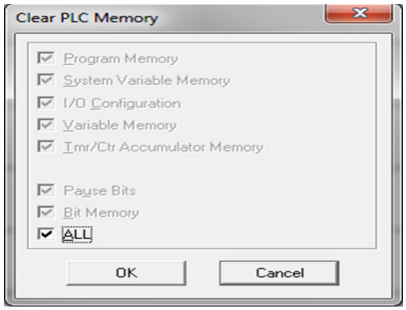


Reset the PLC to Factory Defaults

***NOTE**: Resetting to factory defaults will not clear any password stored in the PLC.*

Resetting a DirectLogic PLC to Factory Defaults is a two-step process. Be sure to have a verified backup of your program using "Save Project to Disk" from the File menu before performing this procedure. Please be aware that the program as well as any settings will be erased and not all settings are stored in the project. In particular you will need to write down any settings for Secondary Communications Ports and manually set the ports up after resetting the PLC to factory defaults.

Step One – While connected to the PLC with DirectSoft, go to the PLC menu and select; "Clear PLC Memory". Check the "ALL" box at the bottom of the list and press "OK".

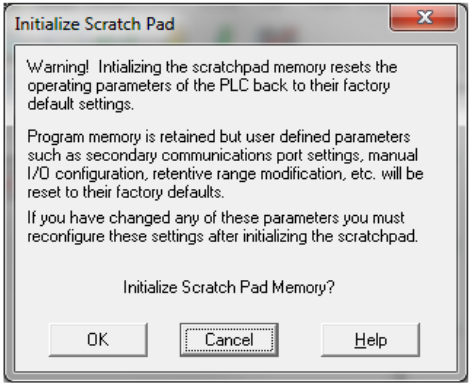


Step Two – While connected with DirectSoft, go the PLC menu and then to the "Setup" submenu and select; "Initialize Scratch Pad" and press "Ok".

***NOTE**: All configurable communications ports will be reset to factory default state. If you are connected via Port 2 or another configurable port, you may be disconnected when this operation is complete.*

***NOTE**: Retentive ranges will be reset to the factory settings..*

***NOTE**: Manually addressed IO will be reset to factory default settings..*



The PLC has now been reset to factory dafaults and you can proceed to program the PLC.

9

Notes:

LCD Display Panel

In This Chapter...

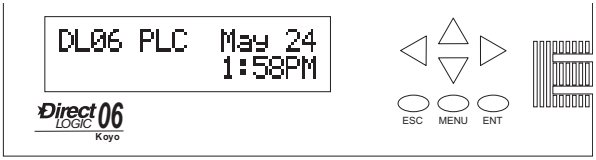
Introduction to the DL06 LCD Display Panel

The DL06 LCD Display Panel is a 16 character, two row display that mounts directly on the face of the DL06 PLC. The LCD is backlit for easy readability in most lighting situations. There are multiple ways of interacting with the LCD Display Panel:

- Built-in keypad
- LCD ladder instruction
- Using ladder instructions to write bit status changes to specified memory locations

The seven function keys on the face of the LCD Display Panel give the user access to clock and calendar setup, V-memory data values or I/O status, etc. Individuals with password authorization can:

- Change clock or calendar settings or formats
- Monitor or change V-memory values (including DWord values)
- Force individual bits on or off (up to 16 per screen)
- Review error code history
- Set or change the password
- Turn the back light or buzzer on or off

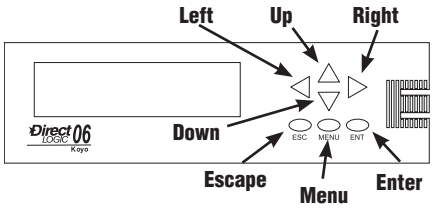


The potential uses for the DL06's LCD display vary widely. An operator can change values for setting up batch processes or machine timing for manufacturing different products. Maintenance personnel can interface in the control cabinet to identify machine problems. LCD messages can be preprogrammed for process events or alarms. The LCD can satisfy these and many other needs.

10

Keypad

The LCD Display Panel keypad has seven keys you can use to navigate through the menu hierarchy. Each screen displayed has a specific set of active keys associated with it. All other keys (those not associated with the current screen) are inactive.

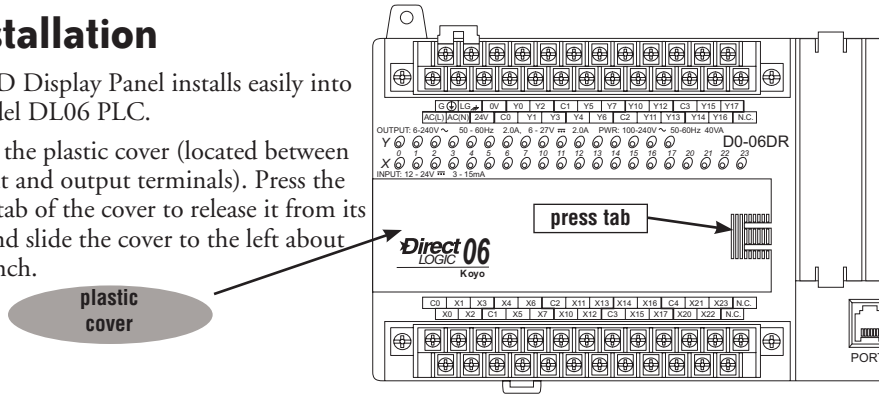


Function Keys		
Name	**Label**	**Function**
Up arrow	none	Move to selection above or increase value
Down arrow	none	Move to selection below or decrease value
Left arrow	none	Move to next digit to the left
Right arrow	none	Move to next digit to the right
Escape	ESC	Return to previous screen or next level up in the menu hierarchy
Menu	MENU	Scroll down through main menu or sub-menu selections
Enter	ENT	Enter the domain of the menu screen selected or save new value

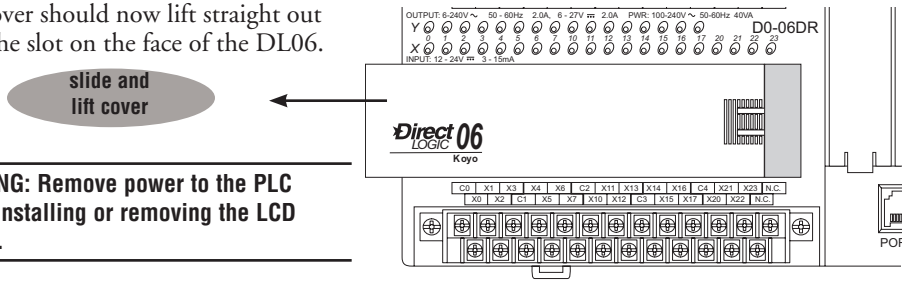
Snap-in installation

The LCD Display Panel installs easily into any model DL06 PLC.

Remove the plastic cover (located between the input and output terminals). Press the locking tab of the cover to release it from its catch, and slide the cover to the left about 3/8ths inch.

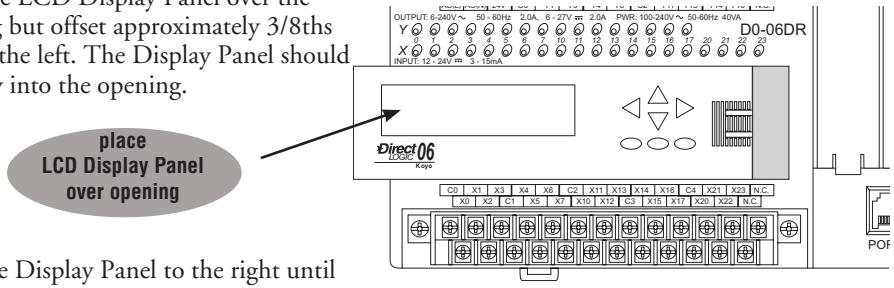


The cover should now lift straight out from the slot on the face of the DL06.

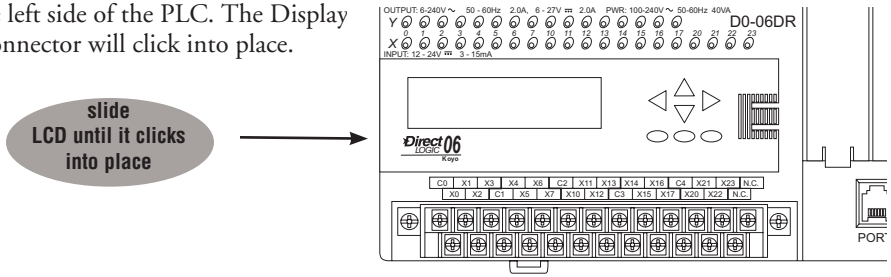


WARNING: Remove power to the PLC before installing or removing the LCD display.

Place the LCD Display Panel over the opening but offset approximately 3/8ths inch to the left. The Display Panel should fit easily into the opening.



Slide the Display Panel to the right until the left side of the Display Panel is flush with the left side of the PLC. The Display Panel connector will click into place.



Display Priority

The LCD Display Panel will show one of the following (unless power is removed from the PLC):

- Default screen (user defined or factory default)
- Menu selection
- Message from ladder program
- Error message

The built-in keypad allows you to navigate through these message displays.

On power-up the default message is normally displayed. The default message is set at the factory but can be customized by the user. Loading a custom default message is described later in this chapter.

If a system error occurs, the error message supercedes the default message (or other current display screen), and the appropriate error code is displayed for diagnostic purposes.

Menu Navigation

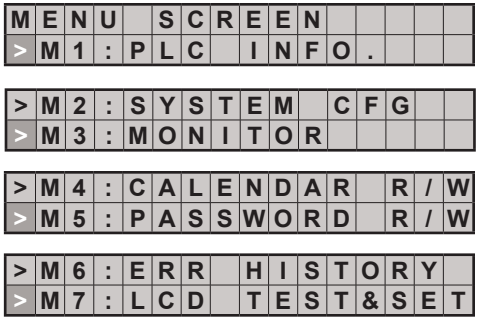
Beginning at the default screen, each time you press the MENU key the display will scroll to the next menu option. The up arrow and down arrow keys also scroll through the list of menus (in the direction indicated by the arrow), but ***you must initially press the MENU key (at the default screen) to activate the up and down arrow keys.***

There are seven built-in menus selections. Some of the menu items have sub-menus. The menus and sub-menus are described in this chapter. Each menu selection requires that you press the ENT key to view or change settings or values within the domain of that main menu selection.

Seven Menu Choices

Pressing and holding the MENU key will cause the display to scroll through the following menu options:

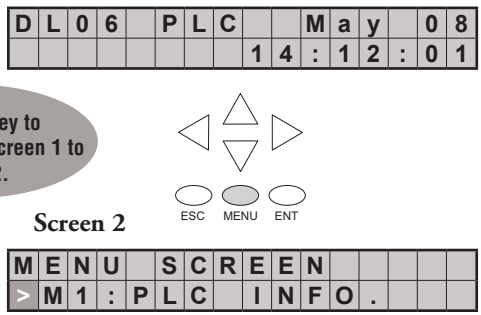
- **M1 : PLC information**
- **M2 : System configuration**
- **M3 : Monitor**
- **M4 : Calendar read/write**
- **M5 : Password read/write**
- **M6 : Error history read**
- **M7 : LCD test and set**



In this section we use illustrations of the LCD Display Panel keypad and display area to show how to navigate through the menu hierarchy. The example below shows the factory default screen as Screen 1 and the main menu entry screen as Screen 2.

The illustration of the keypad between the display screens indicates that pressing the MENU key causes a transition from Screen 1 to Screen 2. This type of representation is used throughout this section. When inside the menu hierarchy, the ESC key returns the display to the previous screen.

Screen 1 - factory default

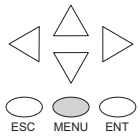


Confirm PLC Type, Firmware Revision Level, Memory Usage, Etc.

Menu 1, M1:PLC INFO.

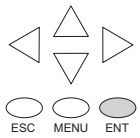
From the default screen, press the MENU key one time to arrive at the PLC INFO menu option.

Default screen

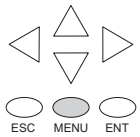


Step 1.1

Press ENT to enter this menu selection. The first screen inside the PLC INFO selection is M1:PLC TYPE. This selection displays the model number of the PLC.

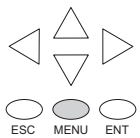


Step 1.2



Step 1.3

Press MENU again to sequence to PLC MODE. The PLC mode is either RUN, STOP (for Stop or Program Mode), TEST-STOP (for Test Stop Mode), or TEST-RUN (for Test Run Mode). You can put the DL06 in the Test Run Mode from the Test Stop Mode.

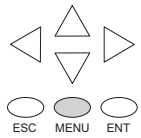


***NOTE**: The menu screen examples shown in this section assume the password/lock feature is not turned on. If the password/lock feature is turned on, the user will be prompted by a message on the Display Panel to enter the password at the appropriate time. Users without password authorization will have access to a limited number of screens.*

Step 1.4

Press MENU again to sequence to FIRMWARE REV.

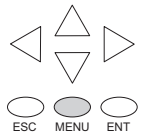
M1:FIRMWARE REV.
V1.000



Step 1.5

Press MENU again to sequence to LADDER MEMORY USED. The number of words used and the total number available in the PLC are displayed.

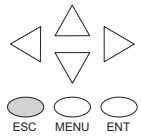
M1:LADDER MEMORY
USED 21 / 7680



Step 1.6

Press MENU again to sequence to LADDER PASSWORD, ACTIVATED OR NOT ACTIVATED. This is the last screen of the PLC INFO menu and is self-explanatory.

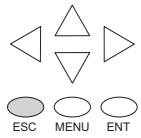
M1:LADDER PASSWD
NOT ACTIVATED



Return to Step 1.1

Press ESC to exit the M1 menu and return to the main menu.

MENU SCREEN
> M1:PLC INFO.



Default screen

Press ESC once more to return to the default screen.

DL06 PLC May 08
14:22:11

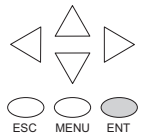
Examining Option Slot Contents

Menu 2, M2:SYSTEM CFG.

From the default screen, press MENU twice to arrive at the M2:SYSTEM CFG. (System Configuration) menu option.

Step 2.1

>	M	1	:	P	L	C		I	N	F	O	.			
>	M	2	:	S	Y	S	T	E	M		C	F	G	.	

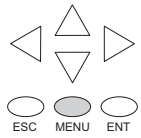


Step 2.2

Press ENT to enter the SYSTEM CFG. menu selection.

M	2	:	O	P	T	I	O	N		S	L	O	T		1
		D	0	-	D	E	V	N	E	T	S				

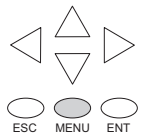
NOTE: *This is an example only and may not represent the contents of this or any option slot on your system.*



Step 2.3

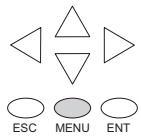
Pressing the MENU key four times will cycle through the four option slots. The model number of the option card in each slot is shown on line 2 or there is an indication that the slot is empty.

M	2	:	O	P	T	I	O	N		S	L	O	T		2
		E	M	P	T	Y		I	/	O		S	L	O	T



Step 2.4

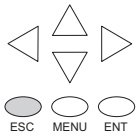
M	2	:	O	P	T	I	O	N		S	L	O	T		3
		F	0	-	0	4	A	D	-	1					



Press the ESC key twice to return to the default screen.

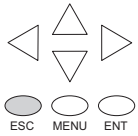
Step 2.5

M	2	:	O	P	T	I	O	N		S	L	O	T		4
			E	M	P	T	Y			I	/	O		S	L	O	T



Return to Step 2.1

>	M	1	:	P	L	C		I	N	F	O	.			
>	M	2	:	S	Y	S	T	E	M		C	F	G	.	



Return to default screen

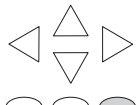
D	L	0	6		P	L	C			M	a	y		0	8
								1	4	:	5	7	:	2	1

Monitoring and Changing Data Values

Menu 3, M3:MONITOR

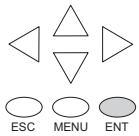
From the default screen, press MENU three times to arrive at the M3:MONITOR menu option.

Step 3.1



The M3:MONITOR sub-menu contains the data monitor and the bit monitor. The data monitor allows you to examine the contents of memory registers or pointers to determine their contents. The default format is BCD/HEX, but the format can be changed to decimal by setting bit 8 of V7742. Please refer to the DL06 Memory Map for ranges.

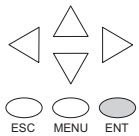
Step 3.2



Data Monitor

Data type = V for V-memory or P for pointer. Press MENU to change data type, or press ENT to designate the register whose data you want to view or change.

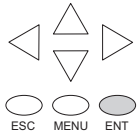
Step 3.3



V-memory values

Use the right or left arrow key to move the cursor to the digit you want to change. Use the up or down arrow key to change the digit. The V-memory address is expressed as an octal number so you will not see 8's or 9's.

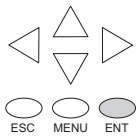
Step 3.4



This screen allows you to view two adjacent V-memory locations in BCD format. The lower word is to the right. Pressing ENT makes it possible to change the value in the **lower word**. At this level of the menu hierarchy, you can also use the up and down arrow keys to scroll to other memory locations.

Step 3.5

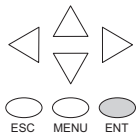
The data values on this screen will be four digits in length for BCD/HEX unless bit 8 of V7742 is set. Bit 8 of V7742 changes the data format to decimal (five digits).



Step 3.6

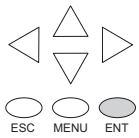
M	3	:	D	A	T	A				V				0
	C	H	G	=		0	0	0	0			0	0	0	0

Use the right or left arrow key to move the cursor to the digit you want to change. Use the up or down arrow key to move to another digit. The V-memory value is expressed as a BCD number so you will see values (in the range: 0 - F) available for each digit. The data format can be changed to decimal by setting bit 8 of V7742.



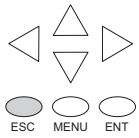
Step 3.7

M	3	:	D	A	T	A				V				0
	C	H	G	=		A	F	0	6			0	0	0	0



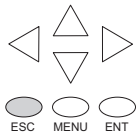
Step 3.8

M	3	:	V				1		V					0
V	A	L			0	0	0	0				A	F	0	6



Returns to Step 1.1

M	3	:	D	A	T	A		T	Y	P	E			V
A	D	D	R	E	S	S					0	0	0	0	0



Returns to default screen

Push the ESC key five (5) times to return to the default screen.

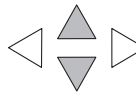
D	L	0	6		P	L	C			M	a	y		0	8
								1	5	:	0	2	:	1	3

Pointer values

Press ESC twice to return to the Step 3.3 screen with the cursor on the V, as shown. Use the up or down arrow key to change the V to P. Now, the pointer information is displayed.

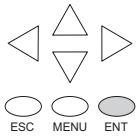
Return to Step 3.3

M 3 : D A T A		T Y P E			V
A D D R E S S			0 0 0 0	0	



Step 3.4a

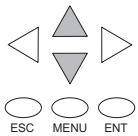
M 3 : D A T A		T Y P E			P
A D D R E S S			0 0 0 0	0	



Use the up or down arrow keys to change the value of the current digit. Use the left or right arrow keys to move from one digit to the next.

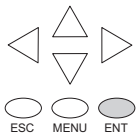
Step 3.5a

M 3 : D A T A		T Y P E			P
A D D R E S S			0 0 0 0	0	



Step 3.6a

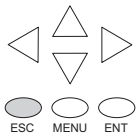
M 3 : D A T A		T Y P E			P
A D D R E S S			1 0 0 0	0	



Step 3.7a

At Step 3.7a, the up and down arrow keys can be used to cycle through data words. Each time you press the up or down arrow key, the address increments or decrements by one 16-bit word (addresses are expressed in octal).

M 3 : D A T A			P 1 0 0 0 0
(V 0 0 0 0 0 0)			2 0 0 0



Return to Step 3.3

To change from the data monitor to the bit monitor, press ESC three times to return to Step 3.2 (five times to return to the default screen).

M 3 : > D A T A		M O N I T O R
> B I T		M O N I T O R

Bit Monitor

Bit status

From Step 3.3, press the up or down arrow key, then the ENT key. You will see one of eleven bit data types displayed. The data type that appears on the display is the last data type accessed. The address shown is also the last address accessed for that particular data type.

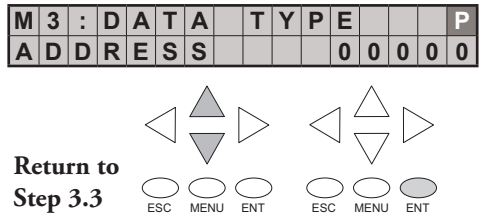
Press ENT to change the address.

Use the arrow keys to change the address as necessary.

Press ENT to view the selected bits.

Use the left and right arrow keys to select a bit whose status you want to change. Press ENT once to see the change status screen. Press ENT again to change the status from OFF to ON or ON to OFF.

Return to Step 3.3

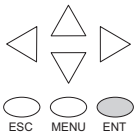


Changing Date and Time

Menu 4, M4 : CALENDAR R/W

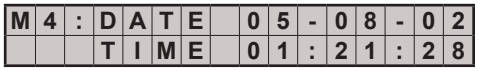
From the default screen, press the MENU key four times to arrive at Step 4.1. Press ENT(Enter) to select M4: Calendar R/W

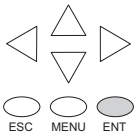
Step 4.1



4.2 While viewing the current Date and Time Settings press ENT to change the date or time.

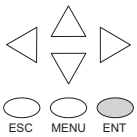
Step 4.2





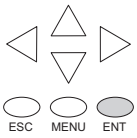
4.3 Press ENT to select > Change Date/ Time.

Step 4.3



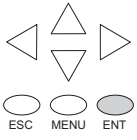
Step 4.4

At Step 4.4, use the up and down arrows to change the value for month, day, or year. Use the left and right arrow keys to move between the different digits in the date. After making the necessary changes using the arrow keys, press the ENT key to register the changes.



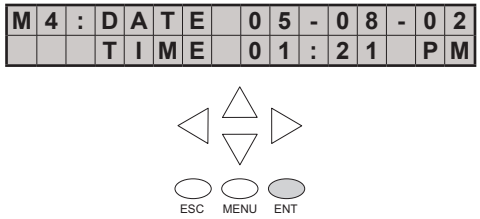
Step 4.5

You will be asked if you want to set the date to the chosen value. Press ENT again if the date is correct. You will automatically return to Step 4.2, and the new date will be displayed.

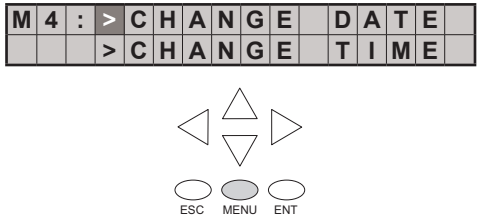


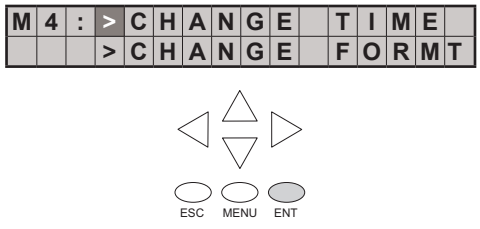
Returns to Step 4.2

In order to change the time or date/time format, press ENT again at Step 4.2.

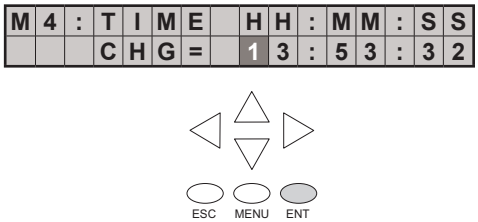


Use the up or down arrow keys or the MENU key to scroll through the sub-menu choices. At this point in our example, we will change the time setting.

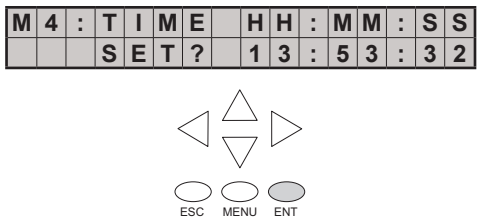




At Step 4.4, use the up and down arrows to change the value for hour, minute, or second. Use the left and right arrow keys to move between the different digits in the time. After making the necessary changes using the arrow keys, press the ENT key to register the changes.



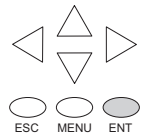
You will be asked if you want to set the date to the chosen value. Press ENT again if the date is correct. You will automatically return to Step 4.2, and the new date will be displayed.



Returns to Step 4.2

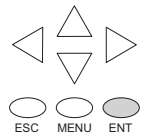
If you want to change the format for the date or time, return to Step 4.2 and press ENT.

M	4	:	D	A	T	E		0	5	-	0	8	-	0	2
			T	I	M	E		0	1	:	2	1		P	M



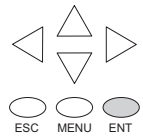
Press ENT, MENU, MENU to arrive at the menu selection for changing the date or time formats. Press ENT again to enter the format changing location.

M	4	:	>	C	H	A	N	G	E		F	O	R	M	T
			>	C	H	A	N	G	E		D	A	T	E	



Press ENT again to enter the date format changing location, or press MENU, ENT to change the time format.

M	4	:	>	D	A	T	E		F	O	R	M	A	T	
			>	T	I	M	E		F	O	R	M	A	T	



At Step 4.4, use the up and down arrow keys to scroll through the date formats. The choices are as follows:

MM-DD-YY (US format)

DD-MM-YY (European format)

YY-MM-DD (Asian format)

M	4	:	D	A	T	E		F	O	R	M	A	T		
			C	H	G	=		M	M	-	D	D	-	Y	Y

Press the ENT key to save the format changes.

M	4	:	T	I	M	E		F	O	R	M	A	T		
			C	H	G	=		H	H	:	M	M	:	S	S

If you have chosen to make a time format change, your choices are:

HH:MM US (12 hour 12:00 - 11:59AM/PM US format)

HH:MM AS (12 hour 00:00 - 11:59AM/PM Asian format)

HH:MM:SS (24 hour format)

Press the ENT key to save the format changes. Press ESC until the default screen reappears.

Date and Time Variables and Formats		
_date:us	US format	MM/DD/YY
_date:e	European format	DD/MM/YY
_date:a	Asian format	YY/MM/DD
_time:12	12 hour format	HH:MMAM/PM
_time:24	24 hour format	HH:MM:SS

Setting Password and Locking

Menu 5, M5 : PASSWORD R/W

The LCD Display Panel has its own password protection separate from the *ladder password* protection of the PLC. An LCD Display password can be used to prevent unauthorized changes to clock and calendar setup and V-memory data values. Individuals with password authorization can change clock, calendar, V-memory values, force bits on or off, etc.

The LCD password inhibits unauthorized personnel from modifying the data in the DL06 with the LCD keypad. Even though the LCD password is locked, the user can still modify the data in the DL06 with ***Direct***SOFT or the D2-HPP. The LCD Display Panel does not support the multi-level password.

Only menu 5 on the LCD Display can modify the LCD password.

WARNING: The password protection available in *Direct*SOFT or the HPP does not prevent changes from the LCD Display Panel. To prevent changes from the LCD Display Panel, it is necessary to use the LCD password locking feature.

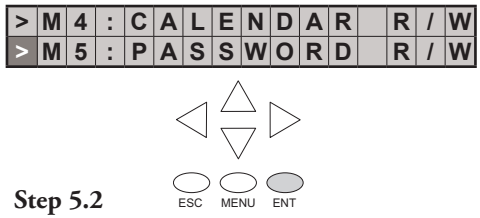
WARNING: The LCD password is apparently entered into the PLC, with no possibility to clear it. If you forget your LCD password, a new PLC must be purchased.

Use the MENU key to navigate to the M5 menu option. Press ENT to arrive at the display shown as Step 5.2.

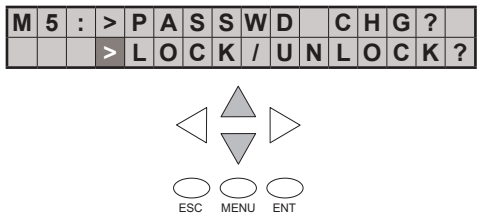
Assigning a password without locking the display allows access to all features and capabilities of the LCD.

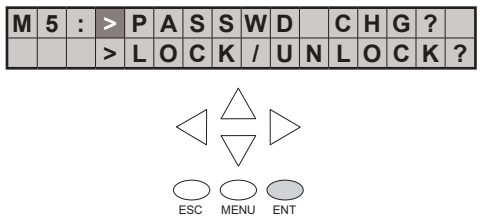
Use the up arrow or down arrow keys to toggle between PASSWD CHG? and LOCK/UNLOCK? Eight zeroes removes the password. If the password is eight zeroes, the display will not LOCK.

Step 5.1



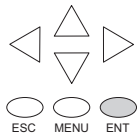
Step 5.2

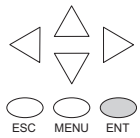




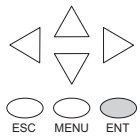
Use the up arrow or down arrow keys to scroll through number choices, and use the right arrow and left arrow keys to move from one digit position to another.

***NOTE**: It is important to record the password where it will not be forgotten and to issue the password only to qualified personnel. Full access to the LCD Display Panel gives access to change data values within the PLC.*



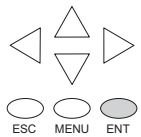


Return to Step 5.2



It is not possible to lock the display without assigning a password. It is possible to assign a password without locking the display, but doing so will not protect sensitive data.

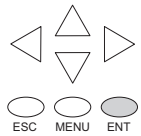
Press the ENT key at Step 5.2, and the display is now locked. If you do not wish to lock the display at this point, press ESC.



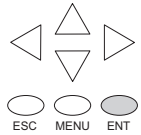
Return to Step 5.2

Before assigning a password, you can select LOCK/UNLOCK by pressing ENT at Step 5.2.

M	5	:	>	P	A	S	S	W	D		C	H	G	?	
			>	L	O	C	K	/	U	N	L	O	C	K	?



M	5	:	S	T	A	T	:	U	N	L	O	C	K	E	D
			E	N	T		T	O		L	O	C	K		



Here, the display prompts you to enter a password.

M	5	:	P	S	W	D		*	*	*	*	*	*	*	*
			L	O	C	K		0	0	0	0	0	0	0	0

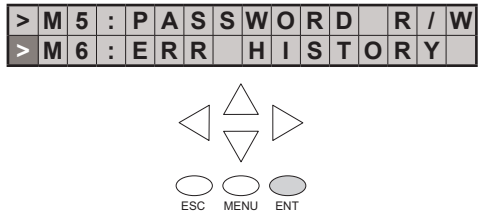
Reviewing Error History

Menu 6, M6 : ERR HISTORY

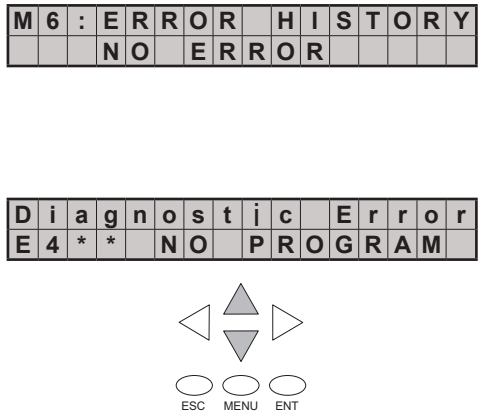
From the default screen, press the MENU key six times to arrive at Step 6.1.

Default screen

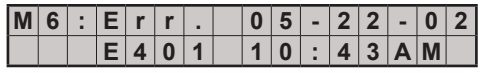
Step 6.1



The Error History screen will display **NO ERROR** if there is no record of errors. If errors have occurred, they can be identified by their Error Code. The Error Code table (see appendix B) will explain the source of the error message. The last 16 messages are displayed.Error messages are displaced when a new error message arrives



To review past error messages use the down arrow key to scroll through the historical record of error messages.



Toggle Light and Beeper, Test Keys

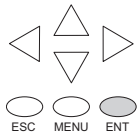
Menu 7, M7 : LCD TEST&SET

This menu selection gives you an opportunity to:

- Test each LCD key to assure that the PLC is receiving its input appropriately
- Turn the beep sound off or on
- Turn the LCD back light off or on

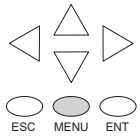
Make a menu selection by pressing the ENT key.

```
>M6:ERR  HISTORY
>M7:LCD  TEST&SET
```



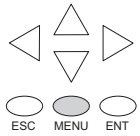
Press ENT to enter the LCD KEY TEST. All keys can be tested for proper function in this menu.
To return to the menu, press the ESC key twice or hold the ESC key down until the menu layer reappears.

```
M7:LCD  TEST&SET
   >LCD  KEY  TEST
```



Press ENT to enter the Back light test menu.

```
M7:LCD  TEST&SET
   >BACK  LIGHT
```



The piezo electric buzzer can be configured to provide pushbutton feedback.

```
M7:LCD  TEST&SET
   >BEEP
```

PLC Memory Information for the LCD Display Panel

The valid memory ranges for storing text messages in the DL06 are:

V400 - V677 V1200 - V7577 V10000 - V17777

Data Format Suffixes for Embedded V-memory Data

Several data formats are available for displaying V-memory data on the LCD. The choices are shown in the table below. A colon is used to separate the embedded V-memory location from the data format suffix and modifier.

Data Format	Modifier		Example	Character Position/Content of the Output												
none (16 bit binary in HEX format)			V2000 = 0012	**1**	**2**	**3**	**4**									
			V2000	s	s	1	8									
	S	[:S]	V2000:S	1	8											
	C0	[:C0]	V2000:C0	0	0	1	8									
	0	[:0]	V2000:0	s	s	1	8									
:B (4 digit BCD)			V2000 = 0012	**1**	**2**	**3**	**4**									
		[:B]	V2000:B	0	0	1	2									
	S	[:BS]	V2000:BS	1	2											
	C0	[:BC0]	V2000:BC0	0	0	1	2									
	0	[:B0]	V2000:B0	s	s	1	2									
:D (32 bit binary)			V2000 = 0000													
			V2001 = 0001	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**		
		[:D]	V2000:D	s	s	s	s	s	s	6	5	5	3	6		
	S	[:DS]	V2000:DS	6	5	5	3	6								
	C0	[:DC0]	V2000:DC0	0	0	0	0	0	0	6	5	5	3	6		
	0	[:D0]	V2000:D0	s	s	s	s	s	s	6	5	5	3	6		
:DB (8 digit BCD)			V2000 = 0000													
			V2001 = 0001	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**					
		[:DB]	V2000:DB	0	0	0	1	0	0	0	0					
	S	[:DBS]	V2000:DBS	1	0	0	0	0								
	C0	[:DBC0]	V2000:DBC0	0	0	0	1	0	0	0	0					
	0	[:DB0]	V2000:DB0	s	s	s	1	0	0	0	0					
:R (Floating point number)			Value = 222.11111													
			V2000 = 1C72													
			V2001 = 435E	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
		[:R]	V2000:R	s	s	s	f	2	2	2	.	1	1	1	1	1
	S	[:RS]	V2000:RS	f	2	2	2	.	1	1	1	1	1			
	C0	[:RC0]	V2000:RC0	f	0	0	0	2	2	2	.	1	1	1	1	1
	0	[:R0]	V2000:R0	s	s	s	f	2	2	2	.	1	1	1	1	1
:E (Floating point number with exponent)			Value = 222.1													
			V2000 = 199A													
			V2001 = 435E	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
		[:E]	V2000:E	s	f	2	.	2	2	1	0	0	E	+	0	2
	S	[:ES]	V2000:ES	f	2	.	2	2	1	0	0	E	+	0	2	
	C0	[:EC0]	V2000:EC0	f	2	.	2	2	1	0	0	E	+	0	2	
	0	[:E0]	V2000:E0	f	2	.	2	2	1	0	0	E	+	0	2	

s = space f = plus/minus flag (plus = no symbol, minus = -)

The S, C0 and 0 modifiers alter the presentation of leading zeros and spaces. S removes leading spaces and left justifies the result. C0 replaces leading spaces with leading zeros. 0 is a modification of C0. 0 eliminates any leading zeros in the C0 format version and converts them to spaces.

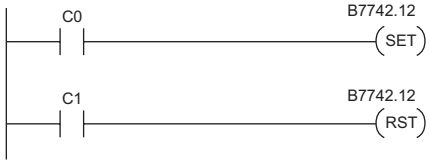
Reserved memory registers for the LCD Display Panel

Two V-memory registers are reserved for making changes to LCD functions via ladder logic. V7742 allows for bit flags to be set in the ladder program. The bit flags control such things as data formats, the back light, and the beeper. All V7742 bit flags are defined in the table on the next page.

The other reserved register is V7743. This register is used to write a customized default screen message to the LCD. A sample program for this purpose is illustrated later in this chapter.

V-memory address	Contents
V7742	**Various LCD flags** • Calendar date and time format • Default operation menu • Data format of data monitor • LCD password status flag • Key press acknowledgement buzzer on/off setting • Back light on/off setting
V7743	Default message location (writing 0 to this address returns the default message to the factory setting)

The following program segment uses the SET and RST coils to turn on and off bit 12 of V7742. When C0 is on, bit 12 is turned on. Bit 12 turns on the beeper in the LCD Display Panel. The C1 contact resets bit 12 to the off state.



V7742 bit definitions

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V7742	*	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit 1, 0	**Date display format (default = 00)**		
	00, 11	=	Month/Day/Year (US format)
	01	=	Day/Month/Year (EU format)
	10	=	Year/Month/Day (Asian format)
Bit 3, 2	**Time display format (default = 00)**		
	00, 11	=	HH:MM:SS (24 hour format)
	01	=	HH:MM PM/AM (12 hour US format - 12:00 - 11:59)
	10	=	HH:MM PM/AM (12 hour Asian format - 00:00 - 11:59)
Bit 6 - 4	**Default menu setting (default = 000)**		
	000	=	Default menu sequence, begins menu sequence with Menu 1
	001	=	Begin menu sequence with Menu 1
	010	=	Begin menu sequence with Menu 2
	011	=	Begin menu sequence with Menu 3
	100	=	Begin menu sequence with Menu 4
	101	=	Begin menu sequence with Menu 5
	110	=	Begin menu sequence with Menu 6
	111	=	Begin menu sequence with Menu 7
Bit 8	**Data monitor format (default = 0)**		
	0	=	BCD/HEX format (0000 - FFFF)
	1	=	Decimal format (00000 - 65535)
Bit 9	**New message overwrite (default = 0)**		
	0	=	New LCD message clears both lines of previous message
	1	=	New LCD message leaves previous message, overwrites specified char. only
Bit 11	**LCD password status flag (Read only)**		
	0	=	Password unlock
	1	=	Password lock
Bit 12	**Status flag beep on/off control (default = 0)**		
	0	=	Beep OFF
	1	=	Beep ON (LCD beeps continuously during ON status of this flag)
Bit 13	**Keypad beep on/off control (default = 0)**		
	0	=	Beep OFF
	1	=	Beep ON (LCD beeps when keys are pressed)
Bit 14	**LCD back light setting flag (default = 1)**		
	0	=	Light OFF
	1	=	Light ON
Bit 15	**LCD installed status flag (Read only)**		
	0	=	LCD is not installed
	1	=	LCD is installed

Changing the Default Screen

At power-up the default screen is displayed. The default screen message is set at the factory but can be customized by the user. One method of customizing the default message uses the VPRINT instruction. The VPRINT instruction is described in Chapter 5.

Factory default message

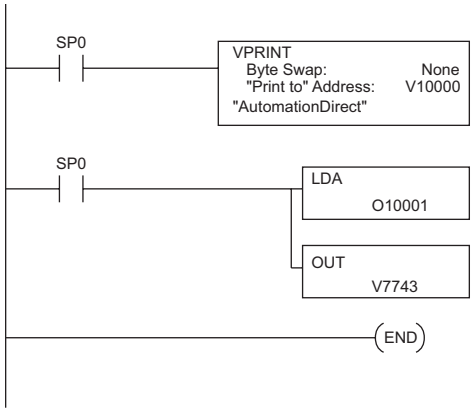
D	L	0	6		P	L	C			M	a	y		0	8
								1	4	:	2	0	:	4	9

Example program for setting the default screen message

The following program can be used to set up the default screen message. This program uses the VPRINT instruction to load ASCII text to a designated V-memory location and to embed a pointer to the current date.

The LDA and OUT instructions are used to point to the V-memory location (+1) where the text is located. The memory location V7743 is reserved for the pointer to the default message.

***NOTE**: The VPRINT instruction adds a one word (2 bytes) non-printing header to the text. For this reason, the LDA instruction points to the V-memory location V10001 rather than V10000.*



V10000	00h	16h
V10001	u	A
V10002	o	t
V10003	a	m
V10004	i	t
V10005	n	o
V10006	i	D
V10007	e	r
V10010	t	c
V10011		
V10012		
V10013		
V10014		
V10015		
V10016		
V10017		
V10020		

After running this program, press MENU, then ESC or cycle power. The new default message should look as indicated. See Menu 4 instructions for changing date and time information.

A	u	t	o	m	a	t	i	o	n	D	i	r	e	c	t

***NOTE**: It is possible to return to the factory default screen by writing 0 to V7743 and cycling power.*

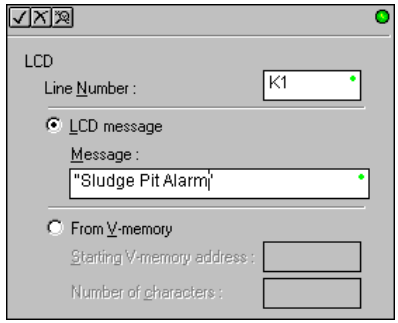
DL06 LCD Display Panel Instruction (LCD)

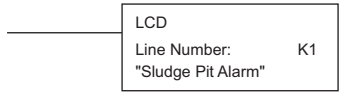
From the DirectSOFT project folder, use the Instruction Browser to locate the LCD instruction. When you select the LCD instruction and click OK, the LCD dialog will appear.

The LCD Display Panel instruction is inserted into the ladder program via the set-up dialog box shown to the right. The dialog is used to specify a message to be displayed on line 1 or line 2 LCD Display Panel.

S	l	u	d	g	e		P	i	t		A	l	a	r	m
E	f	f	l	u	e	n	t		O	v	e	r	f	l	o

figure A



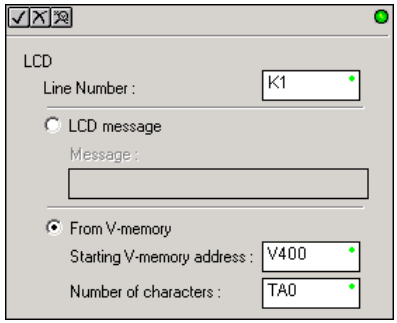


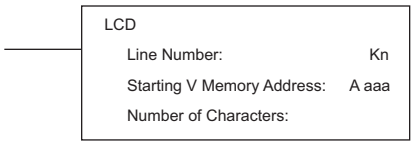
Source of message

The text of the message can originate from one of two places. It can be input directly from the instruction as a literal text string (see figure A), or it can originate as ASCII text stored in a V-memory location (figure B). In the latter case, it is necessary to specify its beginning V-memory location and length within the dialog box.

Display text strings can include embedded data. Any V-memory value or date and time settings can be embedded in the displayed text.

figure B



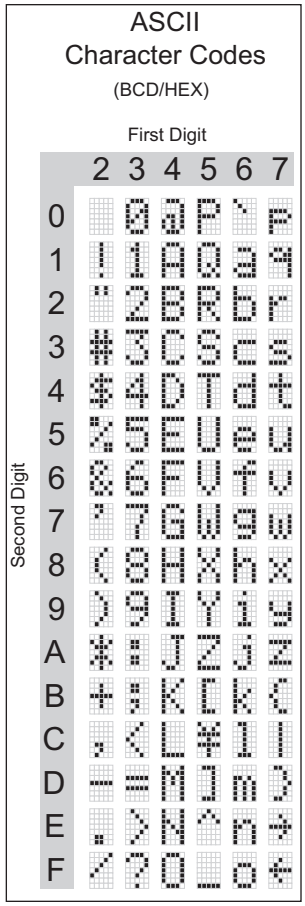


***NOTE**: The LCD Display Panel instruction is supported by DirectSOFT, Ver. 5 or later. It is not supported by the D2-HPP handheld programmer.*

ASCII Character Codes

ASCII characters can be written directly to V-memory locations and then displayed using the LCD instruction. The table to the right shows the two-digit BCD/HEX code for each character available for display.

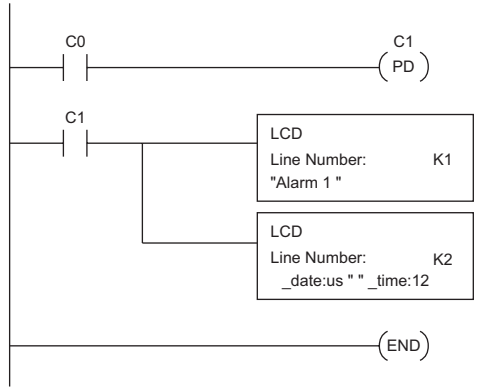
Example:
To display an upper case A, write 41 HEX to the memory location identified by the LCD instruction.



Example program: alarm with embedded date/time stamp

The following program will display the message “Alarm ” and the time on line K1 of the display screen with the date on line K2.

The one-shot, or positive differential (PDd), is used so that the message displays but does not block other messages or menu options. Pressing MENU or ESC will cause the alarm message text to disappear.



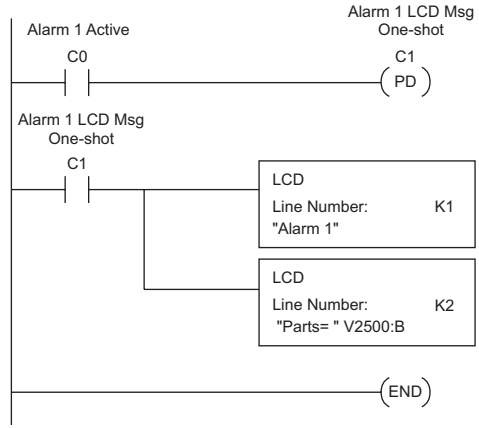
A	l	a	r	m		1								
0	5	/	0	8	/	0	4			5	:	2	3	P	M

Example program: alarm with embedded V-memory data

In this program example, the alarm notification text is displayed along with the contents of V2500. The suffix "B" is added to the memory location (V2500:B) to cause the data to be displayed as a BCD number.

In the first example, the alarm text is loaded directly via the LCD instruction. In the second example, the alarm text is loaded into V-memory and the LCD instruction is used to point to that text.

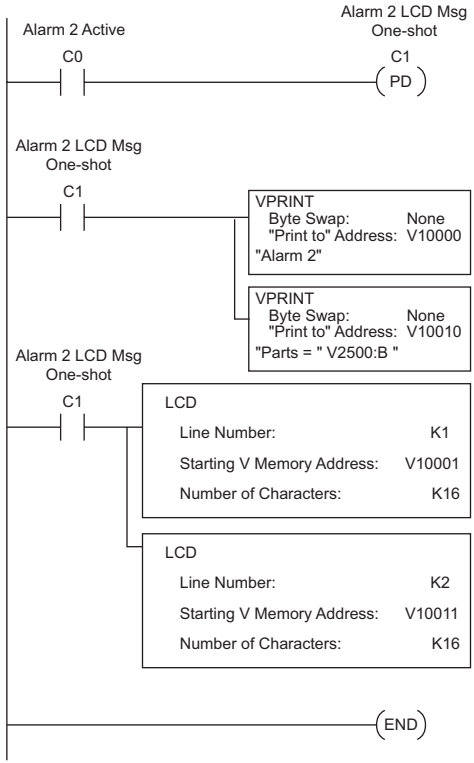
***NOTE**: When using the LCD instruction to display V2000:R, there is a limit of three characters of text because V2000:R uses 13 characters.*



A	l	a	r	m		1								
P	a	r	t	s		=		2	4	3	7			

Example program: alarm text from V-memory with embedded V-memory data

This program example uses the VPRINT instruction to write ASCII text (in the appropriate character sequence) to V10000 and V10010. The LCD instruction is used as a pointer to the V-memory location where the text for each line of the display resides.



A	l	a	r	m		2								
P	a	r	t	s		=		3	5	8	9			

Auxilary Functions

In This Appendix...

Introduction

A

Purpose of Auxiliary Functions

Many CPU setup tasks involve the use of Auxiliary (AUX) Functions. The AUX Functions perform many different operations, including clearing ladder memory, displaying the scan time, and copying programs to EEPROM in the handheld programmer. They are divided into categories that affect different system resources. You can access the AUX Functions from DirectSOFT or from the D2–HPP Handheld Programmer. The manuals for those products provide step-by-step procedures for accessing the AUX Functions. Some of these AUX Functions are designed specifically for the Handheld Programmer setup, so they will not be needed (or available) with the DirectSOFT package. Even though this Appendix provides many examples of how the AUX functions operate, you should supplement this information with the documentation for your choice of programming device.

***NOTE**: The Handheld Programmer may have additional AUX functions that are not supported with the DL06 PLCs.*

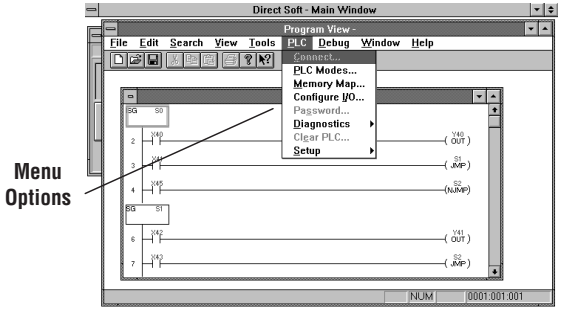
AUX Function and Description		DL06
AUX 2* — RLL Operations		
21	Check Program	*
22	Change Reference	*
23	Clear Ladder Range	*
24	Clear All Ladders	*
AUX 3* — V-Memory Operations		
31	Clear V Memory	*
AUX 4* — I/O Configuration		
41	Show I/O Configuration	*
AUX 5* — CPU Configuration		
51	Modify Program Name	*
53	Display Scan Time	*
54	Initialize Scratchpad	*
55	Set Watchdog Timer	*
56	Set Communication Port 2	*
57	Set Retentive Ranges	*
58	Test Operations	*
59	Override Setup	*
5B	HSIO Interface Configuration	*
5D	Scan Control Setup	*

AUX Function and Description		DL06
AUX 6* — Handheld Programmer Configuration		
61	Show Revision Numbers	*
62	Beeper On / Off	HP
65	Run Self Diagnostics	HP
AUX 7* — EEPROM Operations		
71	Copy CPU memory to HPP EEPROM	HP
72	Write HPP EEPROM to CPU	HP
73	Compare CPU to HPP EEPROM	HP
74	Blank Check (HPP EEPROM)	HP
75	Erase HPP EEPROM	HP
76	Show EEPROM Type (CPU and HPP)	HP
AUX 8* — Password Operations		
81	Modify Password	*
82	Unlock CPU	*
83	Lock CPU	*

*NOTE: * - Supported HP - Handheld Programmer function*

Accessing AUX Functions via DirectSOFT

DirectSOFT provides various menu options during both online and offline programming. Some of the AUX functions are only available during online programming, some only during offline programming, and some during both online and offline programming. The following diagram shows an example of the PLC operations menu available within DirectSOFT.



Accessing AUX Functions via the Handheld Programmer

You can also access the AUX functions by using a Handheld Programmer. Plus, remember some of the AUX functions are only available from the Handheld. Sometimes the AUX name or description cannot fit on one display. If you want to see the complete description, just press the arrow keys to scroll left and right. Also, depending on the current display, you may have to press CLR more than once.

CLR AUX

```
AUX FUNCTION SELECTION
AUX 2* RLL OPERATIONS
```

Use NXT or PREV to cycle through the menus

NEXT

```
AUX FUNCTION SELECTION
AUX 3* V OPERATIONS
```

Press ENT to select sub-menus

ENT

```
AUX 3* V OPERATIONS
AUX 31 CLR V MEMORY
```

You can also enter the exact AUX number to go straight to the sub-menu.

Enter the AUX number directly

CLR D 3 B 1 AUX

```
AUX 3* V OPERATIONS
AUX 31 CLR V MEMORY
```

AUX 2* — RLL Operations

RLL Operations auxiliary functions allow you to perform various operations on the ladder program.

AUX 21 Check Program

Both the Handheld and *Direct*SOFT automatically check for errors during program entry. However, there may be occasions when you want to check a program that has already been in the CPU. Two types of checks are available:

- Syntax
- Duplicate References

The Syntax check will find a wide variety of programming errors, such as missing END statements. If you perform this check and get an error, see Appendix B for a complete listing of programming error codes. Correct the problem and then continue running the Syntax check until the message **NO SYNTAX ERROR** appears.

Use the Duplicate Reference check to verify you have not used the same output coil reference more than once. Note, this AUX function will also find the same outputs even if they have been used with the OROUT instruction, which is perfectly acceptable.

This AUX function is available from the PLC Diagnostics sub-menu in *Direct*SOFT.

AUX 22 Change Reference

There will probably be times when you need to change an I/O address reference or control relay reference. AUX 22 allows you to quickly and easily change all occurrences, (within an address range), of a specific instruction. For example, you can replace every instance of X5 with X10.

AUX 23 Clear Ladder Range

There have been many times when we've taken existing programs and added or removed certain portions to solve new application problems. By using AUX 23, you can select and delete a portion of the program. *Direct*SOFT does not have a menu option for this AUX function, but you can just select the appropriate portion of the program and cut it with the editing tools.

AUX 24 Clear Ladders

AUX 24 clears the entire program from CPU memory. Before you enter a new program, you should always clear ladder memory. This AUX function is available on the PLC/Clear PLC sub-menu within *Direct*SOFT.

AUX 3* — V-memory Operations

AUX 31 Clear V-memory

AUX 31 clears all the information from the V-memory locations available for general use. This AUX function is available on the PLC/Clear PLC sub-menu within *Direct*SOFT.

AUX 4* — I/O Configuration

AUX 41 Show I/O Configuration

This AUX function allows you to display the current I/O configuration on the DL06. Both the Handheld Programmer and *Direct*SOFT will show the I/O configuration.

AUX 5* — CPU Configuration

The following auxiliary AUX functions allow you to setup, view, or change the CPU configuration.

AUX 51 Modify Program Name

DL06 PLCs can use a program name for the CPU program or a program stored on EEPROM in the Handheld Programmer. (Note, you cannot have multiple programs stored on the EEPROM.) The program name can be up to eight characters in length and can use any of the available characters (A–Z, 0–9). AUX 51 allows you to enter a program name. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu. Once you've entered a program name, you can only clear the name by using AUX 54 to reset the system memory. Make sure you understand the possible effects of AUX 54 before you use it!

AUX 53 Display Scan Time

AUX 53 displays the current, minimum, and maximum scan times. The minimum and maximum times are the ones that have occurred since the last Program Mode to Run Mode transition. You can also perform this operation from within DirectSOFT by using the PLC/Diagnostics sub-menu.

AUX 54 Initialize Scratchpad

The CPU maintains system parameters in a memory area often referred to as the ***scratchpad***. In some cases, you may make changes to the system setup that will be stored in system memory. For example, if you specify a range of Control Relays (CRs) as retentive, these changes are stored.

NOTE: *You may never have to use this feature unless you have made changes that affect system memory. Usually, you'll only need to initialize the system memory if you are changing programs and the old program required a special system setup. You can usually change from program to program without ever initializing system memory.*

AUX 54 resets the system memory to the default values. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

AUX 55 Set Watchdog Timer

DL06 PLCs have a watchdog timer that is used to monitor the scan time. The default value set from the factory is 200 ms. If the scan time exceeds the watchdog time limit, the CPU automatically leaves RUN mode and enters PGM mode. The Handheld displays the message, **E003 S/W TIMEOUT,** when the scan overrun occurs.

Use AUX 55 to increase or decrease the watchdog timer value. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

AUX 56 CPU Network Address

Since the DL06 CPU has an additional communication port, you can use the Handheld to set the network address for port 2 and the port communication parameters. The default settings are:

- Station address 1
- HEX mode
- Odd parity

You can use this port with either the Handheld Programmer, DirectSOFT, or, as a communication port for *Direct*NET and MODBUS. Refer to DirectNET and MODBUS manuals for additional information about communication settings required for network operation.

NOTE: *You will only need to use this procedure if you have port 2 connected to a network. Otherwise, the default settings will work fine.*

Use AUX 56 to set the network address and communication parameters. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

AUX 57 Set Retentive Ranges

DL06 CPUs provide certain ranges of retentive memory by default. Some of the retentive memory locations are backed up by a super-capacitor, and others are in non-volatile FLASH memory. The FLASH memory locations are V7400 to V7577 (may be non-volatile if MOV instruction is used). The default ranges are suitable for many applications, but you can change them if your application requires additional retentive ranges or no retentive ranges at all. The default settings are:

Memory Area	DL06	
	Default Range	Available Range
Control Relays	C1000 – C1777	C0 – C1777
V-memory	V400 – V37777	V0 – V37777
Timers	None by default	T0 – T377
Counters	CT0 – CT177	CT0 – CT177
Stages	None by default	S0 – S1777

Use AUX 57 to change the retentive ranges. You can also perform this operation from within *Direct*SOFT by using the PLC/Setup sub-menu.

WARNING: The DL06 CPUs have optional battery-backed RAM which is set as retentive. The super-capacitor will retain the values in the event of a power loss, but only up to 3 weeks. The retention time may be as short as 4 1/2 days in 60° C operating temperature.

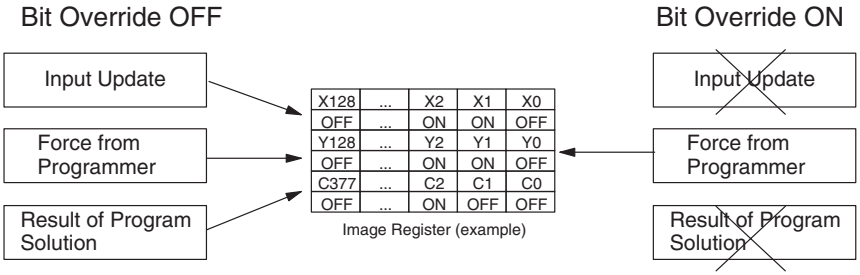
AUX 58 Test Operations

AUX 58 is used to override the output disable function of the Pause instruction. Use AUX 58 to program a single output or a range of outputs which will operate normally even when those points are within the scope of the pause instruction.

AUX 59 Bit Override

Bit override can be enabled on a point-by-point basis by using AUX 59 from the Handheld Programmer or by a menu option from within *Direct*SOFT. Bit override basically disables any changes to the discrete point by the CPU. For example, if you enable bit override for X1, and X1 is off at the time, then the CPU will not change the state of X1. This means that even if X1 comes on, the CPU will not acknowledge the change. So, if you used X1 in the program, it would always be evaluated as **OFF** in this case. Of course, if X1 was on when the bit override was enabled, then X1 would always be evaluated as **ON**.

There is an advantage available when you use the bit override feature. The regular forcing is not disabled because the bit override is enabled. For example, if you enabled the Bit Override for Y0 and it was off at the time, then the CPU would not change the state of Y0. However, you can still use a programming device to change the status. Now, if you use the programming device to force Y0 on, it will remain on and the CPU will not change the state of Y0. If you then force Y0 off, the CPU will maintain Y0 as off. The CPU will never update the point with the results from the application program or from the I/O update until the bit override is removed from the point. The following diagram shows a brief overview of the bit override feature. Notice the CPU does not update the Image Register when bit override is enabled.



AUX 5B Counter Interface Configuration

AUX 5B is used with the High-Speed I/O (HSIO) function to select the configuration. You can choose the type of counter, set the counter parameters, etc. See Chapter 3 for a complete description of how to select the various counter features.

AUX 5D Select PLC Scan Mode

The DL06 CPU has two program scan modes, fixed and variable. In fixed mode, the scan time is lengthened to the time you specify (in milliseconds). If the actual scan time is longer than the fixed scan time, then the error code **E504 BAD REF/VAL** is displayed. In variable scan mode, the CPU begins each scan as soon as the previous scan's activities complete.

AUX 6* — Handheld Programmer Configuration

The following auxiliary functions allow you to setup, view, or change the Handheld Programmer configuration.

AUX 61 Show Revision Numbers

As with most industrial control products, there are cases when additional features and enhancements are made. Sometimes these new features only work with certain releases of firmware. By using AUX 61 you can quickly view the CPU and Handheld Programmer firmware revision numbers. This information (for the CPU) is also available from within *Direct*SOFT from the PLC/Diagnostics sub-menu.

AUX 62 Beeper On/Off

The Handheld has a beeper that provides confirmation of keystrokes. You can use Auxiliary (AUX) Function 62 to turn off the beeper.

AUX 65 Run Self Diagnostics

If you think the Handheld Programmer is not operating correctly, you can use AUX 65 to run a self diagnostics program. You can check the following items.

- Keypad
- Display
- LEDs and Backlight
- Handheld Programmer EEPROM check

AUX 7* — EEPROM Operations

The following auxiliary functions allow you to move the ladder program from one area to another and perform other program maintenance tasks.

Transferrable Memory Areas

Option and Memory Type	DL06 Default Range
1:PGM — Program	$00000 – $02047
2:V — V-memory	$00000 – $07777
3:SYS — System	Non-selectable copies system parameters
4:etc (All)— Program, System and non-volatile V — V-memory only	Non-selectable

Many of these AUX functions allow you to copy different areas of memory to and from the CPU and handheld programmer. The following table shows the areas that may be mentioned.

AUX 71 CPU to HPP EEPROM

AUX 71 copies information from the CPU memory to an EEPROM installed in the Handheld Programmer.You can copy different portions of EEPROM (HP) memory to the CPU memory as shown in the previous table.

AUX 72 HPP EEPROM to CPU

AUX 72 copies information from the EEPROM installed in the Handheld Programmer to CPU memory in the DL06. You can copy different portions of EEPROM (HP) memory to the CPU memory as shown in the previous table.

AUX 73 Compare HPP EEPROM to CPU

AUX 73 compares the program in the Handheld programmer (EEPROM) with the CPU program. You can compare different types of information as shown previously.

AUX 74 HPP EEPROM Blank Check

AUX 74 allows you to check the EEPROM in the handheld programmer to make sure it is blank. It's a good idea to use this function anytime you start to copy an entire program to an EEPROM in the handheld programmer.

AUX 75 Erase HPP EEPROM

AUX 75 allows you to clear all data in the EEPROM stored in the handheld programmer. You should use this AUX function before you copy a program from the CPU.

AUX 76 Show EEPROM Type

You can use AUX 76 to quickly determine what size EEPROM is installed in the Handheld Programmer.

AUX 8* — Password Operations

There are several AUX functions available that you can use to modify or enable the CPU password. You can use these features during on-line communications with the CPU. You can also use them with an EEPROM installed in the Handheld Programmer during off-line operation. This will allow you to develop a program in the Handheld Programmer and include password protection.

- AUX 81 — Modify Password
- AUX 82 — Unlock CPU
- AUX 83 — Lock CPU

AUX 81 Modify Password

You can use AUX 81 to provide an extra measure of protection by entering a password that prevents unauthorized machine operations. The password must be an eight-character numeric (0–9) code. Once you've entered a password, you can remove it by entering all zeros (00000000). (This is the default from the factory.)

Once you've entered a password, you can lock the CPU against access. There are two ways to lock the CPU with the Handheld Programmer.

- The CPU is always locked after a power cycle (if a password is present).
- You can use AUX 82 and AUX 83 to lock and unlock the CPU.

You can also enter or modify a password from within *Direct*SOFT by using the PLC/ Password sub-menu. This feature works slightly differently in *Direct*SOFT. Once you've entered a password, the CPU is automatically locked when you exit the software package. It will also be locked if the CPU is power cycled.

WARNING: Make sure you remember the password before you lock the CPU. Once the CPU is locked you cannot view, change, or erase the password. If you do not remember the password, you have to return the CPU to the factory for password removal. It is the policy of Automationdirect to clear the PLC memory along with the password.

NOTE: *The DL06 CPUs support multi-level password protection of the ladder program. This allows password protection while not locking the communication port to an operator interface. The multi-level password can be invoked by creating a password with an upper case "A" followed by seven numeric characters (e.g. A1234567).*

AUX 82 Unlock CPU

AUX 82 can be used to unlock a CPU that has been password protected. *Direct*SOFT will automatically ask you to enter the password if you attempt to communicate with a CPU that contains a password.

AUX 83 Lock CPU

AUX 83 can be used to lock a CPU that contains a password. Once the CPU is locked, you will have to enter a password to gain access. Remember, this is not necessary with *Direct*SOFT since the CPU is automatically locked whenever you exit the software package.

DL05 Error Codes

In This Appendix...

DL06 Error Codes

DL06 Error Code	Description
E001 CPU FATAL ERROR	You may possibly clear the error by power cycling the CPU. If the error returns, replace the DL06.
E003 SOFTWARE TIME-OUT	If the program scan time exceeds the time allotted to the watchdog timer, this error will occur. SP51 will be on and the error code will be stored in V7755. To correct this problem use AUX 55 to extend the time allotted to the watchdog timer.
E041 CPU BATTERY LOW	The DL06 battery is low and should be replaced. SP43 will be on and the error code will be stored in V7757. The CPU indicator will blink if the battery is less than 2.5 VDC (refer to the table on page 3-6).
E104 WRITE FAILED	A write to the DL06 was not successful. Power cycle the DL06. If the error returns, replace the DL06.
E151 BAD COMMAND	A parity error has occurred in the application program. SP44 will be on and the error code will be stored in V7755. This problem may possibly be due to electrical noise. Clear the memory and download the program again. Correct any grounding problems. If the error returns replace the Micro DL06.
E155 RAM FAILURE	A checksum error has occurred in the system RAM. SP44 will be on and the error code will be stored in V7755. This problem may possibly be due to a low battery, electrical noise or a CPU RAM failure. Clear the memory and download the program again. Correct any grounding problems. If the error returns replace the DL06.
E2** I/O MODULE FAILURE	An I/O module has failed. Run AUX42 to determine the actual error.
E202 MISSING I/O MODULE	An I/O module has failed to communicate with the DL06 or is missing from the slot. SP45 will be on and the error code will be stored in V7756. Run AUX42 to determine the slot and base location of the module reporting the error.
E210 POWER FAULT	A short duration power drop-out occurred on the main power line supplying power to the DL06.
E252 NEW I/O CFG	This error occurs when the auto configuration check is turned on in the DL06 and the actual I/O configuration has changed, either by moving modules in a base, or changing types of modules in a base. You can return the modules to the original position/types or run AUX45 to accept the new configuration. SP45 will be on and the error code will be stored in V7755.
E262 I/O OUT OF RANGE	An out of range I/O address has been encountered in the application program. Correct the invalid address in the program. SP45 will be on and the error code will be stored in V7755.
E263 CONFIGURED I/O ADDRESS OUT OF RANGE	Out of range addresses have been assigned while manually configuring the I/O. Correct the address assignments using AUX46.
E311 HP COMM ERROR 1	A request from the handheld programmer could not be processed by the DL06. Clear the error and retry the request. If the error continues replace the DL06 SP46 will be on and the error code will be stored in V7756.
E312 HP COMM ERROR 2	A data error was encountered during communications with the DL06. Clear between the two devices, replace the handheld programmer, then if necessary replace the DL06. The error code will be stored in V7756.
E313 HP COMM ERROR 3	An address error was encountered during communications with the DL06. Clear the error and retry the request. If the error continues, check the cabling between the two devices, replace the handheld programmer; then, if necessary, replace the DL06. The error code will be stored in V7756.
E316 HP COMM ERROR 6	A mode error was encountered during communications with the DL06. Clear the error and retry the request. If the error continues, replace the handheld programmer; then, if necessary, replace the DL06. The error code will be stored in V7756.
E320 HP COMM TIME-OUT	The DL06 did not respond to the handheld programmer communication request. Check to ensure cabling is correct and not defective. Power cycle the system. If the error continues, replace the DL06 first and then the handheld programmer, 00 if necessary.

DL06 Error Code	Description
E321 COMM ERROR	A data error was encountered during communication with the DL06. Check to ensure cabling is correct and not defective. Power cycle the system and, if the error continues, replace the DL06 first and then the handheld programmer, if necessary.
E4** NO PROGRAM	A syntax error exists in the application program. The most common is a missing END statement. Run AUX21 to determine which one of the E4** series of errors is being flagged. SP52 will be on and the error code will be stored in V7755.
E401 MISSING END STATEMENT	All application programs must terminate with an END statement. Enter the END statement in appropriate location in your program. SP52 will be on and the error code will be stored in V7755.
E402 MISSING LBL	A MOVMC or LDLBL instruction was used without the appropriate label. Refer to Chapter 5 for details on these instructions. SP52 will be on and the error code will be stored in V7755.
E403 MISSING RET	A subroutine in the program does not end with the RET instruction. SP52 will be on and the error code will be stored in V7755.
E404 MISSING FOR	A NEXT instruction does not have the corresponding FOR instruction. SP52 will be on and the error code will be stored in V7755.
E405 MISSING NEXT	A FOR instruction does not have the corresponding NEXT instruction. SP52 will be on and the error code will be stored in V7755.
E406 MISSING IRT	An interrupt routine in the program does not end with the IRT instruction. SP52 will be on and the error code will be stored in V7755.
E412 SBR/LBL>256	There is greater than 256 SBR or DLBL instructions in the program. This error is also returned if there is greater than 4 INT instructions used in the program. SP52 will be on and the error code will be stored in V7755.
E421 DUPLICATE STAGE REFERENCE	Two or more SG or ISG labels exist in the application program with the same number. A unique number must be allowed for each Stage and Initial Stage. SP52 will be on and the error code will be stored in V7755.
E422 DUPLICATE LBL REFERENCE	Two or more LBL instructions exist in the application program with the same number. A unique number must be allowed for each label. SP52 will be on and the error code will be stored in V7755.
E423 NESTED LOOPS	Nested loops (programming one FOR/NEXT loop inside of another) are not allowed. SP52 will be on and the error code will be stored in V7755.
E431 INVALID ISG/SG ADDRESS	An ISG or SG instruction must not be placed after the end statement (such as inside a subroutine). SP52 will be on and the error code will be stored in V7755.
E432 INVALID JUMP (GOTO) ADDRESS	A LBL that corresponds to a GOTO instruction must not be programmed after the end statement such as in a subroutine. SP52 will be on and the error code will be stored in V7755.
E433 INVALID SBR ADDRESS	An SBR must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.
E434 INVALID RTC ADDRESS	An RTC must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.
E435 INVALID RT ADDRESS	An RT must be programmed after the end statement, not in the main body of the program or in an interrupt routine. SP52 will be on and the error code will be stored in V7755.
E436 INVALID INT ADDRESS	An INT must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.

DL06 Error Code	Description
E437 INVALID IRTC ADDRESS	An IRTC must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E438 INVALID IRT ADDRESS	An IRT must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E440 INVALID DATA ADDRESS	Either the DLBL instruction has been programmed in the main program area (not after the END statement), or the DLBL instruction is on a rung containing input contact(s).
E441 ACON/NCON	An ACON or NCON must be programmed after the end statement, not in the main body of the program. SP52 will be on and the error code will be stored in V7755.
E451 BAD MLS/MLR	MLS instructions must be numbered in ascending order from top to bottom.
E452 X AS COIL	An X data type is being used as a coil output.
E453 MISSING T/C	A timer or counter contact is being used where the associated timer or counter does not exist.
E454 BAD TMRA	One of the contacts is missing from a TMRA instruction.
E455 BAD CNT/UDC	One of the contacts is missing from a CNT or UDC instruction.
E456 BAD SR	One of the contacts is missing from the SR instruction.
E461 STACK OVERFLOW	More than nine levels of logic have been stored on the stack. Check the use of OR STR and AND STR instructions.
E462 STACK UNDERFLOW	An unmatched number of logic levels have been stored on the stack. Ensure the number of AND STR and OR STR instructions match the number of STR instructions.
E463 LOGIC ERROR	An STR/STRN instruction was not used to begin a rung of ladder logic.
E464 MISSING CKT	A rung of ladder logic is not terminated properly.
E471 DUPLICATE COIL REFERENCE	Two or more OUT instructions reference the same I/O point.
E472 DUPLICATE TMR REFERENCE	Two or more TMR instructions reference the same number.
E473 DUPLICATE CNT REFERENCE	Two or more CNT instructions reference the same number.
E480 INVALID CV ADDRESS	The CV instruction is used in a subroutine or program interrupt routine. The CV instruction may only be used in the main program area (before the END statement).
E481 CONFLICTING INSTRUCTION	An instruction exists between convergence stages.
E482 MAX. CV INSTRUCTIONS EXCEEDED	Number of CV instructions exceeds 17.
E483 INVALID CV JUMP ADDRESS	CVJMP has been used in a subroutine or a program interrupt routine.
E484 MISSING CV INSTRUCTION	CVJMP is not preceded by the CV instruction. A CVJMP must immediately follow the CV instruction.

DL06 Error Code	Description
E485 MISSING REQUIRED INSTRUCTION	A CV JMP instruction is not placed between the CV and the [SG, ISG, ST BLK, END BLK, END] instruction.
E486 INVALID CALL BLK ADDRESS	CALL BLK is used in a subroutine or a program interrupt routine. The CALL BLK instruction may only be used in the main program area (before the END statement).
E487 MISSING ST BLK INSTRUCTION	The CALL BLK instruction is not followed by a ST BLK instruction.
E488 INVALID ST BLK ADDRESS	The ST BLK instruction is used in a subroutine or a program interrupt. Another ST BLK instruction is used between the CALL BLK and the END BLK instructions.
E489 DUPLICATE CR REFERENCE	The control relay used for the BLK instruction is being used as an output elsewhere.
E490 MISSING SG INSTRUCTION	The BLK instruction is not immediately followed by the SG instruction.
E491 INVALID ISG INSTRUCTION ADDRESS	There is an ISG instruction between the ST BLK and END BLK instructions.
E492 INVALID END BLK ADDRESS	The END BLK instruction is used in a subroutine or a program interrupt routine. The END BLK instruction is not followed by a ST BLK instruction.
E493 MISSING END REQUIRED INSTRUCTION	A [CV, SG, ISG, ST BLK, END] instruction must immediately follow the END BLK instruction.
E494 MISSING END BLK INSTRUCTION	The ST BLK instruction is not followed by a END BLK instruction.
E499 PRINT INSTRUCTION	Invalid PRINT instruction usage. Quotations and/or spaces were not entered or entered incorrectly.
E501 BAD ENTRY	An invalid keystroke or series of keystrokes was entered into the handheld programmer.
E502 BAD ADDRESS	An invalid or out of range address was entered into the handheld programmer.
E503 BAD COMMAND	An invalid command was entered into the handheld programmer.
E504 BAD REF/VAL	An invalid value or reference number was entered with an instruction.
E505 INVALID INSTRUCTION	An invalid instruction was entered into the handheld programmer.
E506 INVALID OPERATION	An invalid operation was attempted by the handheld programmer.
E520 BAD OP–RUN	An operation which is invalid in the RUN mode was attempted by the handheld programmer.
E521 BAD OP–TRUN	An operation which is invalid in the TEST RUN mode was attempted by the handheld programmer.
E523 BAD OP–TPGM	An operation which is invalid in the TEST PROGRAM mode was attempted by the handheld programmer.
E524 BAD OP–PGM	An operation which is invalid in the PROGRAM mode was attempted by the handheld programmer.
E525 MODE SWITCH	An operation was attempted by the handheld programmer while the DL06 mode switch was in a position other than the TERM position.

DL06 Error Code	Description
E526 OFF LINE	The handheld programmer is in the OFFLINE mode. To change to the ONLINE mode use the MODE key.
E527 ON LINE	The handheld programmer is in the ON LINE mode. To change to the OFF LINE mode use the MODE key.
E528 CPU MODE	The operation attempted is not allowed during a Run Time Edit.
E540 CPU LOCKED	The DL06 has been password locked. To unlock the DL06 use AUX82 with the password.
E541 WRONG PASSWORD	The password used to unlock the DL06 with AUX82 was incorrect.
E542 PASSWORD RESET	The DL06 powered up with an invalid password and reset the password to 00000000. A password may be re-entered using AUX81.
E601 MEMORY FULL	Attempted to enter an instruction which required more memory than is available in the DL06.
E602 INSTRUCTION MISSING	A search function was performed and the instruction was not found.
E603 DATA MISSING	A search function was performed and the data was not found.
E604 REFERENCE MISSING	A search function was performed and the reference was not found.
E610 BAD I/O TYPE	The application program has referenced an I/O module as the incorrect type of module.
E620 OUT OF MEMORY	An attempt to transfer more data between the DL06 and handheld programmer than the receiving device can hold.
E621 EEPROM NOT BLANK	An attempt to write to a non-blank EEPROM in the handheld programmer was made. Erase the EEPROM and then retry the write.
E622 NO HPP EEPROM	A data transfer was attempted with no EEPROM (or possibly a faulty EEPROM) installed in the handheld programmer.
E623 SYSTEM EEPROM	A function was requested with an EEPROM in the handheld programmer which contains system information only.
E624 V-MEMORY ONLY	A function was requested with an EEPROM in the handheld programmer which contains V-memory data only.
E625 PROGRAM ONLY	A function was requested with an EEPROM in the handheld programmer which contains program data only.
E627 BAD WRITE	An attempt to write to a faulty EEPROM in the handheld programmer was made. Replace the EEPROM if necessary.
E628 EEPROM TYPE ERROR	The wrong size EEPROM is being used.
E640 COMPARE ERROR	A compare between the EEPROM handheld programmer and the DL06 was found to be in error.
E642 CHECKSUM ERROR	An error was detected while data was being transferred to the handheld programmer's EEPROM. Check cabling and retry the operation.
E650 HPP SYSTEM ERROR	A system error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns, replace the handheld programmer.
E651 HPP ROM ERROR	A ROM error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns, replace the handheld programmer.
E652 HPP RAM ERROR	A RAM error has occurred in the handheld programmer. Power cycle the handheld programmer. If the error returns, replace the handheld programmer.

Instruction Execution Times

In This Appendix...

Introduction

This appendix contains several tables that provide the instruction execution times for DL06 Micro PLCs. Many of the execution times depend on the type of data used with the instruction. Registers may be classified into the following types:

- Data (word) Registers
- Bit Registers

V-Memory Data Registers

Some V-memory locations are considered data registers, such as timer or counter current values. Standard user V-memory is classified as a V-memory data register. Note that you can load a bit pattern into these types of registers, even though their primary use is for data registers. The following locations are data registers:

Data Registers	DL06
Timer Current Values	V0 - V377
Counter Current Values	V1000 - V1177
User Data Words	V400 - V677 V1200 - V7377 V10000 - V17777

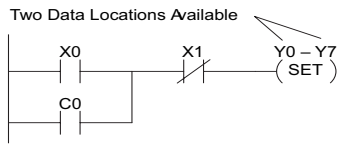
V-Memory Bit Registers

You may recall that some of the discrete points such as X, Y, C, etc., are automatically mapped into V-memory. The following bit registers contain this data:

Bit Registers	DL06
Input Points (X)	V40400 - V40437
Output Points (Y)	V40500 - V40537
Control Relays (C)	V40600 - V40677
Stages (S)	V41000 - V41077
Timer status Bits	V41100 - V41177
Counter status Bits	V41140 - V41147
Special Relays (SP)	V41200 - V41237

How to Read the Tables

Some instructions can have more than one parameter. For example, the SET instruction shown in the ladder program to the right can set a single point or a range of points.



In these cases, execution times depend on the amount and type of parameters. The execution time tables list execution times for both situations, as shown below:

SET	1st #: X, Y, C, 2nd #: X, Y, C, S (N pt)	9.2 µs 9.6 µs + 0.9 V x N
RST	1st #: X, Y, C, 2nd #: X, Y, C, S (N pt)	9.2 µs 9.6 µs + 0.9 V x N

Execution depends on numbers of locations and types of data used

Instruction Execution Times

Boolean Instructions

Boolean Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
STR	X, Y, C, T, CT, S, SP, GX, GY	0.67 µs	0.00 µs
STRN	X, Y, C, T, CT, S, SP, GX, GY	0.67 µs	0.0 µs
OR	X, Y, C, T, CT, S, SP, GX, GY	0.51 µs	0.51 µs
ORN	X, Y, C, T, CT, S, SP, GX, GY	0.55 µs	0.55 µs
AND	X, Y, C, T, CT, S, SP, GX, GY	0.42 µs	0.42 µs
ANDN	X, Y, C, T, CT, S, SP, GX, GY	0.51 µs	0.51 µs
ANDSTR	None	0.37 µs	0.37 µs
ORSTR	None	0.37 µs	0.37 µs
OUT	X, Y, C, GX, GY	1.82 µs	1.82 µs
OROUT	X, Y, C, GX, GY	2.09 µs	2.09 µs
NOT	None	1.04 µs	1.04 µs
SET	1st #: X, Y, C, S, 2nd #: X, Y, C, S (N pt)	9.2 µs 9.6 µs+0.9 µs x N	1.0 µs 1.1 µs
RST	1st #: X, Y, C, S, GX, GY 2nd #: X, Y, C, S (N pt), GX, GY	9.2 µs 9.6 µs+0.9 µs x N	1.0 µs 1.1 µs
	1st #: T, CT, GX, GY 2nd #: T, CT (N pt), GX, GY	25.7 µs 16.8 µs + 2.7 µs x N	1.1 µs 1.4 µs
PAUSE	1wd: Y 2wd: Y (N points)	5.6 µs 9.2 µs + 0.3 µs x N	5.4 µs 4.8 µs

C

Comparative Boolean Instructions

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
STRE	*1st*	*2nd*		
	V Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.7 µs	27.7 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.7 µs	27.7 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
STRNE	*1st*	*2nd*		
	V: Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	30.3 µs	30.3 µs
		V:Bit Reg.	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	30.3 µs	30.3 µs
		V:Bit Reg.	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ORE	***1st***	***2nd***		
	V Data Reg	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	30.3 µs	30.3 µs
		V:Bit Reg	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	50.4 µs	50.4 µs
		P:Indir. (Bit)	50.4 µs	50.4 µs
	P:Indir. (Bit)	V:Data Reg	30.3 µs	30.3 µs
		V:Bit Reg	30.3 µs	30.3 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	50.4 µs	50.4 µs
		P:Indir. (Bit)	50.4 µs	50.4 µs
ORNE	***1st***	***2nd***		
	Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

C

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ANDE	***1st*** V Data Reg.	***2nd*** V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
ANDNE	***1st*** V: Data Reg.	***2nd*** V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
STR	***1st***	***2nd***		
	T, CT	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V Data Reg	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
STRN	***1st***	***2nd***		
	T, CT	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit	30.2 µs	30.2 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
STRN (cont.)	***1st*** V: Bit Reg	***2nd*** V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg.	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg.	29.9 µs	29.9 µs
		V:Bit Reg.	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
OR	***1st*** T, CT	***2nd*** V Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V Data Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ORN	*1st* T, CT	*2nd* V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Data Reg	V:Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
AND	*1st*	*2nd*		
	T, CT	V Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Data Reg.	V:Data Reg	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	V: Bit Reg.	V:Data Reg.	7.6 µs	7.6 µs
		V:Bit Reg	7.6 µs	7.6 µs
		K:Constant	4.8 µs	4.8 µs
		P:Indir. (Data)	30.2 µs	30.2 µs
		P:Indir. (Bit)	30.2 µs	30.2 µs
	P:Indir. (Data)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs
	P:Indir. (Bit)	V:Data Reg	29.9 µs	29.9 µs
		V:Bit Reg	29.9 µs	29.9 µs
		K:Constant	27.4 µs	27.4 µs
		P:Indir. (Data)	51.0 µs	51.0 µs
		P:Indir. (Bit)	51.0 µs	51.0 µs

Comparative Boolean Instructions (cont'd)

Comparative Boolean Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
ANDN	*1st* T, CT	*2nd* V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	 7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Data Reg.	V:Data Reg V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	V: Bit Reg.	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs	7.6 µs 7.6 µs 4.8 µs 30.2 µs 30.2 µs
	P:Indir. (Data)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs
	P:Indir. (Bit)	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs	29.9 µs 29.9 µs 27.4 µs 51.0 µs 51.0 µs

Immediate Instructions

Immediate Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
LDI	V	20.6 µs	1.1 µs
LDIF	1st #: Y 2nd #: K Constant	26.6 µs+0.9µs x N	1.4 µs
STRI	X	19.3 µs	19.3 µs
STRNI	X	19.4 µs	19.4 µs
ORI	X	19.1 µs	18.7 µs
ORNI	X	19.2 µs	18.9 µs
ANDI	X	18.7 µs	18.7 µs
ANDNI	X	18.8 µs	18.8 µs
OUTI	Y	25.5 µs	25.5 µs
OROUTI	Y	25.7 µs	25.7 µs
OUTIF	1st #: Y 2nd #: Y (N pt)	66.1 µs+0.9µs x N	1.4 µs
SETI	1st #: Y 2nd #: K Constant	23.1 µs, 22.8 µs+1.4µsxN	0.9 µs, 0.9 µs
RSTI	1st #: Y 2nd #: Y (N pt)	23.2 µs, 22.8 µs+1.4µsxN	0.9 µs, 0.9 µs

Bit of Word Boolean Instructions

Bit of Word Boolean Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
STRB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.1 µs 3.1 µs 30.0 µs 30.0 µs	3.1 µs 3.1 µs 30.0 µs 30.0 µs
STRNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.0 µs 3.0 µs 29.8 µs 29.8 µs	3.0 µs 3.0 µs 29.8 µs 29.8 µs
ORB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.9 µs 2.9 µs 29.9 µs 29.9 µs	2.9 µs 2.9 µs 29.9 µs 29.9 µs
ORNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.8 µs 2.8 µs 29.6 µs 29.6 µs	2.8 µs 2.8 µs 29.6 µs 29.6 µs
ANDB	V:Data Reg V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.8 µs 2.8 µs 29.6 µs 29.6 µs	2.8 µs 2.8 µs 29.6 µs 29.6 µs
ANDNB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	2.7 µs 2.7 µs 29.6 µs 29.6 µs	2.7 µs 2.7 µs 29.6 µs 29.6 µs
OUTB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	3.1 µs 3.1 µs 30.3 µs 30.3 µs	3.4 µs 3.4 µs 30.7 µs 30.7 µs
SETB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.4 µs 13.4 µs 41.1 µs 41.1 µs	3.4 µs 3.4 µs 29.1 µs 29.1 µs
RSTB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.5 µs 13.5 µs 41.3 µs 41.3 µs	1.4 µs 1.4 µs 29.1 µs 29.1 µs

Timer, Counter and Shift Register

Timer, Counter and Shift Register			DL06	
Instruction	Legal Data Types		Execute	Not Execute
	1st	***2nd***		
TMR	T	V Data Reg.	26.8 µs	7.3 µs
		V:Bit Reg	26.8 µs	7.3 µs
		K:Constant	20.0 µs	4.8 µs
		P:Indir. (Data)	45.6 µs	30.2 µs
		P:Indir. (Bit)	45.6 µs	30.2 µs
TMRF	T	V:Data Reg.	51.4 µs	7.3 µs
		V:Bit Reg	51.4 µs	7.3 µs
		K:Constant	48.4 µs	4.6 µs
		P:Indir. (Data)	75.9 µs	30.2 µs
		P:Indir. (Bit)	75.9 µs	30.2 µs
TMRA	T	V:Data Reg.	48.9 µs	7.3 µs
		V:Bit Reg	48.9 µs	7.3 µs
		K:Constant	45.0 µs	4.6 µs
		P:Indir. (Data)	75.9 µs	30.2 µs
		P:Indir. (Bit)	75.9 µs	30.2 µs
	1st	***2nd***		
TMRAF	T	V Data Reg.	54.2 µs	7.3 µs
		V:Bit Reg	54.2 µs	7.3 µs
		K:Constant	50.3 µs	4.6 µs
		P:Indir. (Data)	81.2 µs	30.2 µs
		P:Indir. (Bit)	81.2 µs	30.2 µs
CNT	CT	V:Data Reg.	25.8 µs	7.3 µs
		V:Bit Reg	25.8 µs	7.3 µs
		K:Constant	22.2 µs	4.6 µs
		P:Indir. (Data)	53.5 µs	30.2 µs
		P:Indir. (Bit)	53.5 µs	30.2 µs
SGCNT	CT	V:Data Reg.	27.3 µs	7.3 µs
		V:Bit Reg	27.3 µs	7.3 µs
		K:Constant	23.5 µs	4.6 µs
		P:Indir. (Data)	54.9 µs	30.2 µs
		P:Indir. (Bit)	54.9 µs	30.2 µs
UDC	CT	V:Data Reg	39.8 µs	7.3 µs
		V:Bit Reg	39.8 µs	7.3 µs
		K:Constant	35.4 µs	4.6 µs
		P:Indir. (Data)	67.8 µs	30.2 µs
		P:Indir. (Bit)	67.8 µs	30.2 µs
SR	C (N points to shift)		17.8 µs + 0.9 µs x N	9.8 µs

Accumulator Data Instructions

Accumulator / Stack Load and Output Data Instructions			DL06	
Instruction	Legal Data Types		Execute	Not Execute
LD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		11.8 µs 11.8µs 9.0 µs 33.9 µs 33.9 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
LDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		12.2 µs 12.2 µs 9.0 µs 37.8 µs 37.8 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
LDF	***1st*** X, Y, C, S T, CT, SP	***2nd*** K:Constant	20.5 µs+0.9 µsxN	0.9 µs
LDA	O: (Octal constant for address)		10.4 µs	1.0 µs
LDR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)		29.5 µs 29.5 µs 25.5 µs 54.9 µs 54.9 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
LDSX	K: Constant		14.6 µs	1.0 µs
LDX	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		10.8 µs 10.8 µs 45.2 µs 45.2 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
OUT	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		9.3 µs 9.3 µs 35.2 µs 35.2 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
OUTD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		10.2 µs 10.2 µs 35.8 µs 35.8 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
OUTF	***1st*** X, Y, C	***2nd*** K:Constant	54 µs+1.0 µsxN	0.9 µs
OUTL	V:Data Reg. V:Bit Reg.		13.5 µs 13.5 µs	1.0 µs 1.0 µs
OUTM	V:Data Reg. V:Bit Reg.		13.7 µs 13.7 µs	1.0 µs 1.0 µs
OUTX	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)		17.2 µs 17.2 µs 43.4 µs 43.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
POP	NONE		8.4 µs	1.0 µs

Logical Instructions

Logical (Accumulator) nstructions		DL06	
Instruction	**Legal Data Types**	**Execute**	**Not Execute**
AND	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	7.9 µs 7.9 µs 33.4 µs 33.4 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ANDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	8.9 µs 8.9 µs 5.7 µs 34.4 µs 34.4 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
ANDF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	21.6 µs + 0.9 µs x N	1.0 µs
ANDS	None	10.0 µs	1.0 µs
OR	V:Data Reg V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	8.1 µs 8.1 µs 33.8 µs 33.8 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.0 µs 9.0 µs 5.8 µs 34.5 µs 34.5 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
ORF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 0.9 µs x N	1.0 µs
ORS	None	10.2 µs	1.0 µs
XOR	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	8.0 µs 8.0 µs 33.6 µs 33.6 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
XORD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.0 µs 9.0 µs 5.4 µs 34.4 µs 34.4 µs	1.0 µs 1.0 µs 1.0 µs 0.9 µs 0.9 µs
XORF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 0.9 µs x N	1.0 µs
XORS	None	10.1 µs	1.0 µs
CMP	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	9.4 µs 9.4 µs 34.9 µs 34.9 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
CMPD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	9.9 µs 9.9 µs 6.7 µs 35.4 µs 35.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs

Logical Instructions (cont'd)

Logical (Accumulator) Instructions		DL06	
Instruction	**Legal Data Types**	**Execute**	**Not Execute**
CMPF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	20.9 µs + 1.0 µs x N	1.0 µs
CMPR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	42.8 µs 42.8 µs 38.4 µs 69.0 µs 69.0 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
CMPS	None	11.2 µs	1.0 µs

Math Instructions

Math Instructions (Accumulator)		DL06	
Instruction	**Legal Data Types**	**Execute**	**Not Execute**
ADD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	78.4 µs 78.4 µs 101.2 µs 101.2 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
ADDD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Daa) P:Indir. (Bit)	83.3 µs 83.3 µs 67.7 µs 101.2 µs 101.2 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
SUB	V:Data Reg. V:Bit Reg P:Indir. (Data) P:Indir. (Bit)	77.4 µs 77.4 µs 95.1 µs 95.1 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
SUBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	82.5 µs 82.5 µs 66.0 µs 99.7 µs 99.7 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
MUL	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	266.1 µs 266.1 µs 286.9 µs 290.0 µs 290.0 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
MULD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	839.1 µs 839.1 µs 863.1 µs 863.1 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs
DIV	V:Data Reg. V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (Bit)	363.9 µs 363.9 µs 384.4 µs 419.8 µs 419.8 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs 0.9 µs
DIVD	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	398.3 µs 398.3 µs 390.9 µs 390.9 µs	0.9 µs 0.9 µs 0.9 µs 0.9 µs

Math Instructions (cont'd)

Math Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
INC	V:Data Reg V:Bit Reg P:Indir. (Data) P:Indir. (Bit)	48.5 µs 48.5 µs 74.7 µs 74.7 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
DEC	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	47.5 µs 47.5 µs 71.5 µs 71.5 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
INCB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.2 µs 13.2 µs 38.6 µs 38.6 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
DECB	V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	13.2 µs 13.2 µs 38.0 µs 38.0 µs	1.0 µs 1.0 µs 0.9 µs 0.9 µs
ADDB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.9 µs 24.9 µs 23.5 µs 51.1 µs 51.1 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.4 µs 24.4 µs 20.7 µs 50.7 µs 50.7 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
SUBB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.7 µs 24.7 µs 23.3 µs 50.6 µs 50.6 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
SUBBD	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	24.2 µs 24.2 µs 20.2 µs 50.2 µs 50.2 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
MULB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	10.8 µs 10.8 µs 8.2 µs 37.1 µs 37.1 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
DIVB	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	28.7 µs 28.7 µs 26.1 µs 54.9 µs 54.9 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	48.1 µs 48.1 µs 41.7 µs 74.3 µs 74.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs

Math Instructions (cont'd)

Math Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
SUBR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	50.1 µs 50.1 µs 58.7 µs 76.3 µs 76.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
MULR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	54.2 µs 54.2 µs 42.7 µs 80.4 µs 80.4 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
DIVR	V:Data Reg. V:Bit Reg. K:Constant P:Indir. (Data) P:Indir. (Bit)	50.1 µs 50.1 µs 58.7 µs 76.3 µs 76.3 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs 1.0 µs
ADDF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	109.3 µs + 0.9 µs x N	1.0 µs
SUBF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	107.3 µs + 0.9 µs x N	1.0 µs
MULF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	352.5 µs + 0.9 µs x N	1.0 µs
DIVF	***1st:*** X, Y, C, S T, CT, SP, GX, GY ***2nd:*** K:Constant	477.3 µs + 0.8 µs x N	1.0 µs
ADDS	None	99.5 µs	1.0 µs
SUBS	None	97.5 µs	1.0 µs
MULS	None	342.5 µs	1.0 µs
DIVS	None	467.3 µs	1.0 µs
ADDBS	None	24.3 µs	1.0 µs
SUBBS	None	23.7 µs	1.0 µs
MULBS	None	11.7 µs	1.0 µs
DIVBS	None	29.7 µs	1.0 µs
SQRTR	None	87.9 µs	1.0 µs
SINR	None	226.8 µs	1.0 µs
COSR	None	213.1 µs	1.0 µs
TANR	None	285.5 µs	1.0 µs
ASINR	None	489.8 µs	1.0 µs
ACOSR	None	508.3 µs	1.0 µs
ATANR	None	317.1 µs	1.0 µs

Differential Instructions

Differential Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
PD	X, Y, C	14.4 µs	14.4 µs
STRPD	X, Y, C, S, T, CT	5.4 µs	5.4 µs
STRND	X, Y, C, S, T, CT	7.3 µs	7.3 µs
ORPD	X, Y, C, S, T, CT	6.8 µs	5.2 µs
ORND	X, Y, C, S, T, CT	7.1 µs	4.9 µs
ANDPD	X, Y, C, S, T, CT	6.8 µs	5.2 µs
ANDND	X, Y, C, S, T, CT	7.1 µs	4.9 µs

Bit Instructions

Bit Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
SUM	None	6.7 µs	1.0 µs
SHFR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	12.1 µs + 0.1 x N 8.4 µs + 0.1 x N	0.9 µs
SHFL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	12.1 µs + 0.1 x N 8.4 µs + 0.1 x N	0.9 µs
ROTR	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	16.4 µs 16.4 µs 12.9 µs	1.0 µs 1.0 µs 1.0 µs
ROTL	V:Data Reg. (N bits) V:Bit Reg. (N bits) K:Constant (N bits)	16.4 µs 16.4 µs 12.7 µs	1.0 µs 1.0 µs 1.0 µs
ENCO	None	33.9 µs	0.9 µs
DECO	None	5.7 µs	1.0 µs

Number Conversion Instructions

Number Conversion Instructions (Accumulator)		DL06	
Instruction	Legal Data Types	Execute	Not Execute
BIN	None	100.2 µs	0.9 µs
BCD	None	95.2 µs	0.9 µs
INV	None	2.5 µs	1.0 µs
BCDPL	None	75.6 µs	1.0 µs
ATH	V	25.4 µs	1.0 µs
HTA	V	25.4 µs	1.0 µs
GRAY	None	110.8 µs	1.0 µs
SFLDGT	None	23.1 µs	1.0 µs
BTOR	None	18.6 µs	1.0 µs
RTOB	None	8.6 µs	1.0 µs
RADR	None	51.4 µs	1.0 µs
DEGR	None	81.5 µs	1.0 µs

Table Instructions

Table Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
MOV	Move V:data reg. to V:data reg Move V:bit reg. to V:data reg Move V:data reg. to V:bit reg Move V:bit reg. to V:bit reg. N=#of words	60.2 µs+9.5 x N	0.9 µs
MOVMC	Move V:data Reg to EEPROM Move V:Bit Reg to EEPROM Move from Date Label to V: Data Reg Move from Data Label to V: Bit Reg N= #of words	35 µs + 10.4 µs x N	0.9 µs
LDLBL	K	6.4 µs	1.3 µs
FILL	V: Data Reg V:Bit Reg K:Constant P:Indir. (Data) P:Indir. (bit)	29.4 µs + 8.0 µs x N 26.2 µs + 8.0 µs x N 55.1 µs + 8.0 µs x N	1.0 µs 1.0 µs 1.0 µs
FIND	V: Data Reg (N bits) V:Bit Reg. (N bits) K:Constant(N bits)	66.8 µs 66.8 µs 64.0 µs	1.0 µs 1.0 µs 1.0 µs

Table Instructions (cont'd)

Table Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
FDGT	V: Data Reg (N bits) V:Bit Reg. (N bits) K:Constant(N bits)	66.1 µs 66.1 µs 55.2 µs	1.0 µs 1.0 µs 1.0 µs
FINDB	V: Data Reg (N bits) V:Bit Reg. (N bits) P:Indir. (Data) P:Indir. (Bit)	210.8 µs 210.8 µs 237.0 µs 237.0 µs	1.0 µs 1.0 µs 1.0 µs 1.0 µs
TTD	V: Data Reg V:Bit Reg	66.9 µs 66.9 µs	1.0 µs 1.0 µs
RFB	V: Data Reg V:Bit Reg	66.8 µs 66.8 µs	1.0 µs 1.0 µs
STT	V: Data Reg V:Bit Reg K:Constant	67.8 µs 67.8 µs 65.0 µs	1.0 µs 1.0 µs 1.0 µs
RFT	V: Data Reg V:Bit Reg	51.1 µs 51.1 µs	1.0 µs 1.0 µs
ATT	V: Data Reg V:Bit Reg K:Constant	53.5 µs 53.5 µs 50.8 µs	1.0 µs 1.0 µs 1.0 µs
TSHFL	V: Data Reg V:Bit Reg	134.0 µs 134.0 µs	1.0 µs 1.0 µs
TSHFR	V: Data Reg V:Bit Reg	133.9 µs 133.9 µs	1.0 µs 1.0 µs
ANDMOV	V: Data Reg V:Bit Reg	80.2 µs 80.2 µs	1.0 µs 1.0 µs
ORMOV	V: Data Reg V:Bit Reg	80.4 µs 80.4 µs	1.0 µs 1.0 µs
XORMOV	V: Data Reg V:Bit Reg	80.4 µs 80.4 µs	1.0 µs 1.0 µs
SWAP	V: Data Reg V:Bit Reg	84.1 µs 84.1 µs	1.0 µs 1.0 µs
SETBIT	V: Data Reg (N bits) V:Bit Reg. (N bits)	59.5 µs 59.5 µs	1.0 µs 1.0 µs
RSTBIT	V: Data Reg (N bits) V:Bit Reg. (N bits)	59.5 µs 59.5 µs	1.0 µs 1.0 µs

CPU Control Instructions

CPU Control Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
NOP	None	1.1 µs	1.1 µs
END	None	24.0 µs	24.0 µs
STOP	None	10.0 µs	1.1 µs
RSTWT	None	5.9 µs	2.2 µs

Program Control Instructions

Program Control Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
GOTO	K	5.1 µs	4.8 µs
LBL	K	5.7 µs	0.0 µs
FOR	V, K	125.9 µs	14.5 µs
NEXT	None	64.4 µs	64.4 µs
GTS	K	27.5 µs	14.8 µs
SBR	K	1.5 µs	1.5 µs
RTC	None	25.7 µs	12.1 µs
RT	None	21.2 µs	21.2 µs
MLS	K	(1–7) 35.2 µs	35.2 µs
MLR	K	(0–7) 30.9 µs	30.9 µs

Interrupt Instructions

Interrupt Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ENI	None	24.2 µs	2.7 µs
DISI	None	9.4 µs	2.3 µs
INT	O(0,1)	7.5 µs	–
IRTC	None	0.9 µs	1.3 µs
IRT	None	6.6 µs	–

Network Instructions

Network Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
RX	X, Y, C, T, CT, SP, S, $ V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	852.0 µs 852.0 µs 852.0 µs 868.2 µs 868.2 µs	4.4 µs 4.4 µs 4.4 µs 4.2 µs 4.2 µs
WX	X, Y, C, T, CT, SP, S, $ V:Data Reg. V:Bit Reg. P:Indir. (Data) P:Indir. (Bit)	1614.0 µs 1614.0 µs 1614.0 µs 1630.0 µs 1630.0 µs	4.4 µs 4.4 µs 4.4 µs 4.4 µs 4.4 µs

Intelligent I/O Instructions

Network Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
RD	V:Data Reg. V:Bit Reg.	385.7 µs 385.7 µs	1.2 µs 1.2 µs
WT	V:Data Reg. V:Bit Reg.	385.6 µs 385.6 µs	1.2 µs 1.2 µs

Message Instructions

Message Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
FAULT	V:Data Reg. V:Bit Reg. K:Constant	65.0 µs 65.0 µs 204.7 µs	4.4 µs 4.4 µs 4.4 µs
DLBL	K	–	–
NCON	K	–	–
ACON	A	–	–
PRINT	ASCII	631.0 µs	3.6 µs

RLL^{PLUS} Instructions

RLL^{PLUS} Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
ISG	S	44.0 µs	41.1 µs
SG	S	44.0 µs	41.1 µs
JMP	S	76.0 µs	9.3 µs
NJMP	S	77.4 µs	9.3 µs
CV	S	42.1 µs	27.5 µs
CVJMP	S	89.5 µs	17.6 µs
BCALL	C	22.1 µs	22.6 µs
BLK	C	17.1 µs	14.6 µs
BEND	None	8.7 µs	0.0 µs

Drum Instructions

Drum Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
DRUM	CT	840.0 µs	339.6 µs
EDRUM	CT	753.2 µs	357.0 µs
MDRMD	CT	411.3 µs	216.4 µs
MDRMW	CT	378.6 µs	147.0 µs

Clock/Calendar Instructions

Clock/Calendar Instructions		DL06	
Instruction		Execute	Not Execute
DATE	V:Data Reg. V:Bit Reg.	24.0 µs	1.2 µs
TIME	V:Data Reg. V:Bit Reg.	50.8 µs	1.2 µs

MODBUS Instructions

Clock/Calendar Instructions		DL06	
Instruction		Execute	Not Execute
MRX	Input, Input Register Coil, Holding Register	120.2 µs	1.3 µs
MWX	Input, Input Register Coil, Holding Register	21.3 µs	1.3 µs

ASCII Instructions

ASCII Instructions		DL06	
Instruction	Legal Data Types	Execute	Not Execute
AIN	V	13.9 µs	12.0 µs
AFIND	V	111.5 µs	1.3 µs
AEX	V	111.7 µs	1.3 µs
CMPV	V	12.2 µs	1.3 µs
SWAPB	V	109.8 µs	1.3 µs
VPRINT	Text Data	161.6 µs	1.3 µs
PRINTV	V	163.3 µs	1.3 µs
ACRB	V	3.9 µs	1.1 µs

Special Relays

In This Appendix...

DL06 PLC Special Relays

Special Relays are just contacts that are set by the CPU operating system to indicate a particular system event has occurred. These contacts are available for use in your ladder program. Knowing just the right special relay contact to use for a particular situation can save a lot of programming time. Since the CPU operating system sets and clears special relay contacts, the ladder program only has to use them as inputs in ladder logic.

Startup and Real-Time Relays

SP0	First scan	On for the first scan after a power cycle or program to run transition only. The relay is reset to off on the second scan. It is useful where a function needs to be performed only on program startup.
SP1	Always ON	Provides a contact to ensure an instruction is executed every scan.
SP2	Always OFF	Provides a contact that is always off.
SP3	1 minute clock	On for 30 seconds and off for 30 seconds.
SP4	1 second clock	On for 0.5 second and off for 0.5 second.
SP5	100 ms clock	On for 50 ms. and off for 50 ms.
SP6	50 ms clock	On for 25 ms. and off for 25 ms.
SP7	Alternate scan	On every other scan.

CPU Status Relays

SP11	Forced run mode	On when the mode switch is in the run position and the CPU is running.
SP12	Terminal run mode	On when the mode switch is in the TERM position and the CPU is in the run mode.
SP13	Test run mode	On when the CPU is in the test run mode.
SP15	Test stop mode	On when the CPU is in the test stop mode.
SP16	Terminal PGM mode	On when the mode switch is in the TERM position and the CPU is in program mode.
SP17	Forced stop	On when the mode switch is in the STOP position.
SP20	Forced stop mode	On when the STOP instruction is executed.
SP22	Interrupt enabled	On when interrupts have been enabled using the ENI instruction.

System Monitoring

SP36	Override setup relay	On when the override function is used.
SP37	Scan controller	On when the actual scan time runs over the prescribed scan time.
SP40	Critical error	On when a critical error such as I/O communication loss has occurred.
SP41	Warning	On when a non critical error has occurred.
SP42	Diagnostics error	On when a diagnostics error or a system error occurs.
SP43	Low battery error	On when the CPU battery voltage is low (only if bit 12 of V7633 is set).
SP44	Program memory error	On when a memory error such as a memory parity error has occurred.
SP45	I/O error	On when an I/O error such as a blown fuse occurs.
SP46	Communications error	On when a communication error occurs on any of the CPU ports.
SP50	Fault instruction	On when a Fault Instruction is executed.
SP51	Watch Dog timeout	On if the CPU Watch Dog timer times out.
SP52	Grammatical error	On if a grammatical error has occurred either while the CPU is running or if the syntax check is run. V7755 will hold the exact error code.
SP53	Solve logic error	On if CPU cannot solve the logic.
SP54	Communication error	On when RX, WX, instructions are executed with the wrong parameters.
SP56	Table instruction overrun	On if a table instruction with a pointer is executed and the pointer value is outside the table boundary.

Accumulator Status

SP60	Value less than	On when the accumulator value is less than the instruction value.
SP61	Value equal to	On when the accumulator value is equal to the instruction value.
SP62	Greater than	On when the accumulator value is greater than the instruction value.
SP63	Zero	On when the result of the instruction is zero (in the accumulator).
SP64	Half borrow	On when the 16 bit subtraction instruction results in a borrow.
SP65	Borrow	On when the 32 bit subtraction instruction results in a borrow.
SP66	Half carry	On when the 16 bit addition instruction results in a carry.
SP67	Carry	On when the 32 bit addition instruction results in a carry.
SP70	Sign	On anytime the value in the accumulator is negative.
SP71	Pointer reference error	On when the V-memory specified by a pointer (P) is not valid.
SP72	Floating point number	On anytime the value in the accumulator is a valid floating point number.
SP73	Overflow	On if overflow occurs in the accumulator when a signed addition or subtraction results in an incorrect sign bit.
SP74	Underflow	On anytime a floating point math operation results in an underflow error.
SP75	Data error	On if a BCD number is expected and a non–BCD number is encountered.
SP76	Load zero	On when any instruction loads a value of zero into the accumulator.

HSIO Input Status

SP100	X0 status	On when X0 is on
SP101	X1 status	On when X1 is on
SP102	X2 status	On when X2 is on
SP103	X3 status	On when X3 is on

HSIO Pulse Output Relay

SP104	Profile Complete	On when the pulse output profile is completed. (Mode 30)

Communication Monitoring Relay

SP116	CPU port busy Port 2	On when port 2 is the master and sending data.
SP117	Communications error Port 2	On when port 2 is the master and has a communication error.

Option Slot Communication Monitoring Relay

SP120	Slot 1 busy	H0-ECOM/D0-DCM Port2
SP121	Slot 1 error	H0-ECOM/D0-DCM Port2
SP122	Slot 2 busy	H0-ECOM/D0-DCM Port2
SP123	Slot 2 error	H0-ECOM/D0-DCM Port2
SP124	Slot 3 busy	H0-ECOM/D0-DCM Port2
SP125	Slot 3 error	H0-ECOM/D0-DCM Port2
SP126	Slot 4 busy	H0-ECOM/D0-DCM Port2
SP127	Slot 4 error	H0-ECOM/D0-DCM Port2

Option Slot Special Relay

SP140-237	Slot 1	SP relay for option card
SP240-337	Slot 2	SP relay for option card
SP340-437	Slot 3	SP relay for option card
SP430-537	Slot 4	SP relay for option card

Counter 1 Mode 10 Equal Relays

SP540	Current = target value	On when the counter current value equals the value in V3631/3630
SP541	Current = target value	On when the counter current value equals the value in V3633/3632
SP542	Current = target value	On when the counter current value equals the value in V3635/3634
SP543	Current = target value	On when the counter current value equals the value in V3637/3636
SP544	Current = target value	On when the counter current value equals the value in V3641/3640
SP545	Current = target value	On when the counter current value equals the value in V3643/3642
SP546	Current = target value	On when the counter current value equals the value in V3645/3644
SP547	Current = target value	On when the counter current value equals the value in V3647/3646
SP550	Current = target value	On when the counter current value equals the value in V3651/3650
SP551	Current = target value	On when the counter current value equals the value in V3653/3652
SP552	Current = target value	On when the counter current value equals the value in V3655/3654
SP553	Current = target value	On when the counter current value equals the value in V3657/3656
SP554	Current = target value	On when the counter current value equals the value in V3661/3660
SP555	Current = target value	On when the counter current value equals the value in V3663/3662
SP556	Current = target value	On when the counter current value equals the value in V3665/3664
SP557	Current = target value	On when the counter current value equals the value in V3667/3666
SP560	Current = target value	On when the counter current value equals the value in V3671/3670
SP561	Current = target value	On when the counter current value equals the value in V3673/3672
SP562	Current = target value	On when the counter current value equals the value in V3675/3674
SP563	Current = target value	On when the counter current value equals the value in V3677/3676
SP564	Current = target value	On when the counter current value equals the value in V3771/3770
SP565	Current = target value	On when the counter current value equals the value in V3703/3702
SP566	Current = target value	On when the counter current value equals the value in V3705/3704
SP567	Current = target value	On when the counter current value equals the value in V3707/3706

Counter 2 Mode 10 Equal Relays

SP570	Current = target value	On when the counter current value equals the value in V3711/3710
SP571	Current = target value	On when the counter current value equals the value in V3713/3712
SP572	Current = target value	On when the counter current value equals the value in V3715/3714
SP573	Current = target value	On when the counter current value equals the value in V3717/3716
SP574	Current = target value	On when the counter current value equals the value in V3721/3720
SP575	Current = target value	On when the counter current value equals the value in V3723/3722
SP576	Current = target value	On when the counter current value equals the value in V3725/3724
SP577	Current = target value	On when the counter current value equals the value in V3727/3726
SP600	Current = target value	On when the counter current value equals the value in V3731/3730
SP601	Current = target value	On when the counter current value equals the value in V3733/3732
SP602	Current = target value	On when the counter current value equals the value in V3735/3734
SP603	Current = target value	On when the counter current value equals the value in V3737/3736
SP604	Current = target value	On when the counter current value equals the value in V3741/3740
SP605	Current = target value	On when the counter current value equals the value in V3743/3742
SP606	Current = target value	On when the counter current value equals the value in V3745/3744
SP607	Current = target value	On when the counter current value equals the value in V3747/3746
SP610	Current = target value	On when the counter current value equals the value in V3751/3750
SP611	Current = target value	On when the counter current value equals the value in V3753/3752
SP612	Current = target value	On when the counter current value equals the value in V3755/3754
SP613	Current = target value	On when the counter current value equals the value in V3757/3756
SP614	Current = target value	On when the counter current value equals the value in V3761/3760
SP615	Current = target value	On when the counter current value equals the value in V3763/3762
SP616	Current = target value	On when the counter current value equals the value in V3765/3764
SP617	Current = target value	On when the counter current value equals the value in V3767/3766

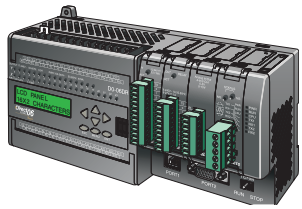
High-speed Input and Pulse Output Features

Appendix E

In This Appendix...

Introduction

Built-in Motion Control Solution



Many machine control applications require various types of simple high-speed monitoring and control. These applications usually involve some type of motion control, or high-speed interrupts for time-critical events. The DL06 Micro PLC solves this traditionally expensive problem with built-in CPU enhancements. Let's take a closer look at the available high-speed I/O features.

The available **high-speed input features** are:

- High Speed Counter (7 kHz max.) with up to 24 counter presets and built-in interrupt subroutine, counts up only, with reset
- Quadrature encoder inputs to measure counts and clockwise or counter-clockwise direction (7 kHz max.), counts up or down, with reset
- High-speed interrupt inputs for immediate response to critical or time-sensitive tasks
- Pulse catch feature to monitor one input point, having a pulse width as small as 100µs (0.1ms)
- Programmable discrete filtering (both on and off delay up to 99ms) to ensure input signal integrity (this is the default mode for inputs X0–X3)

The available **pulse output features** are:

- Single-axis programmable pulse output (10 kHz max.) with three profile types, including trapezoidal moves, registration, and velocity control

Availability of HSIO Features

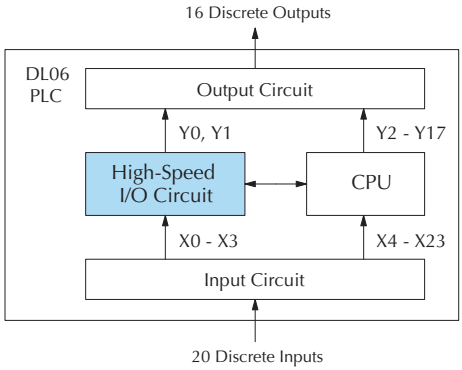
IMPORTANT: Please note the following restrictions on availability of features:

- High-speed input options are available only on DL06s with DC inputs.
- Pulse output options are available only on DL06s with DC outputs.
- Only one HSIO feature may be in use at one time. You cannot use a high–speed input feature and the pulse output at the same time.

Specifications				
DL06 Part Number	**Discrete Input Type**	**Discrete Output Type**	**High-Speed Input**	**Pulse Output**
D0–06AA	AC	AC	No	No
D0–06AR	AC	Relay	No	No
D0–06DA	DC	AC	Yes	No
D0–06DD1	DC	DC	Yes	Yes
D0–06DD2	DC	DC	Yes	Yes
D0–06DR	DC	Relay	Yes	No
D0–06DD1–D	DC	DC	Yes	Yes
D0–06DD2–D	DC	DC	Yes	Yes
D0–06DR–D	DC	Relay	Yes	No

Dedicated High- Speed I/O Circuit

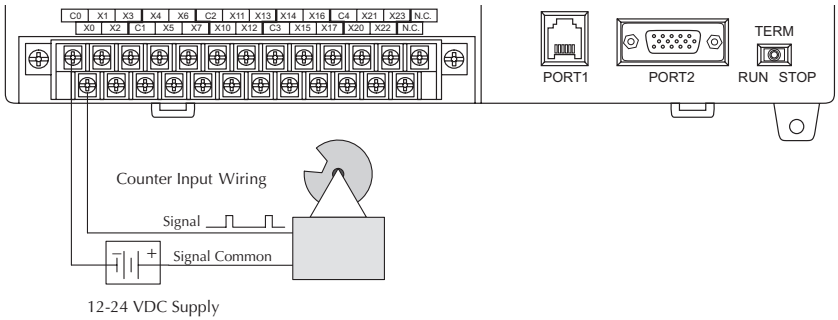
The internal CPU's main task is to execute the ladder program and read/write all I/O points during each scan. In order to service high-speed I/O events, the DL06 includes a special circuit which is dedicated to a portion of the I/O points. Refer to the DL06 block diagram in the figure below.



The high-speed I/O circuit (HSIO) is dedicated to the first four inputs (X0 – X3) and the first two outputs (Y0 – Y1). We might think of this as a *CPU helper*. In the default operation (called "Mode 60") the HSIO circuit just passes through the I/O signals to or from the CPU, so that all twenty inputs behave equally and all sixteen outputs behave equally. When the CPU is configured in any other HSIO Mode, the HSIO circuit imposes a specialized function on the portion of inputs and outputs shown. The HSIO circuit *operates independently of the CPU program scan*. This provides accurate measurement and capturing of high-speed I/O activity while the CPU is busy with ladder program execution.

Wiring Diagrams for Each HSIO Mode

After choosing the appropriate HSIO mode for your application, you'll need to refer to the section in this appendix for that specific mode. Each section includes wiring diagrams to help you connect the High-Speed I/O points correctly to field devices. An example of a High Speed Counter mode diagram is shown below.



Choosing the HSIO Operating Mode

Understanding the Six Modes

The High-Speed I/O circuit operates in one of 6 basic modes as listed in the table below. The number in the left column is the mode number (later, we'll use these numbers to configure the PLC). Choose one of the following modes according to the primary function you want from the dedicated High-Speed I/O circuit. You can simply use all twenty inputs and sixteen outputs as regular I/O points with Mode 60.

High Speed I/O Basic Modes		
Mode		Mode Features
10	High-Speed Counter	Two 7 kHz counters with 24 presets and reset input, counts up only, cause interrupt on preset
20	Up/Down Counter	7 kHz up/down counter with 24 presets and reset, causes interrupt on preset
		Channel A / Channel B 7 kHz quadrature input, counts up and down
30	Pulse Output	Stepper control – pulse and direction signals, programmable motion profile (10 kHz max.)
40	High-Speed Interrupt	Generates an interrupt based on input transition or time
50	Pulse Catch	Captures narrow pulses on a selected input
60	Filtered Input	Rejects narrow pulses on selected inputs

In choosing one of the six high-speed I/O modes, the I/O points listed in the table below operate only as the function listed. If an input point is not specifically used to support a particular mode, it usually operates as a filtered input by default. Similarly, output points operate normally unless Pulse Output mode is selected.

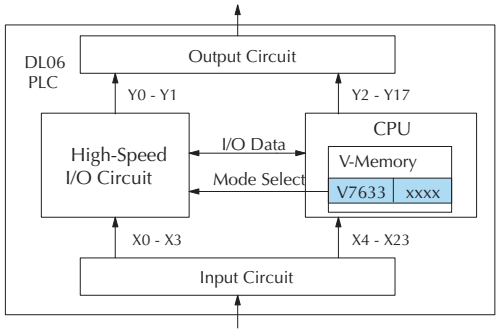
Physical I/O Point Usage							
Mode		DC Input Points				DC Output Points	
		X0	X1	X2	X3	Y0	Y1
10	High-Speed Counter	Counter #1	Counter #2, Interrupt, Pulse Input or Filtered Input	Reset #1, Interrupt, Pulse Input or Filtered Input	Reset #2, Interrupt, Pulse Input or Filtered Input	Regular Output	Regular Output
20	Up/Down counter (Standard counting)	Up Counting	Down Counting	Reset, Pulse Input or Filtered Input	Pulse Input or Filtered Input	Regular Output	Regular Output
	Up/Down counter (Quadrature counting)	Phase A Input	Phase B Input				
30	Pulse Output	Pulse Input or Filtered Input	Pulse Input or Filtered Input	Pulse Input or Filtered Input	Pulse Input or Filtered Input	Pulse or CW Pulse	Direction or CCW Pulse
40	High-Speed Interrupt	Interrupt	Interrupt, Pulse Input or Filtered Input	Interrupt, Pulse Input or Filtered Input	Interrupt, Pulse Input or Filtered Input	Regular Output	Regular Output
50	Pulse Catch	Pulse Input	Pulse Input, Interrupt or Filtered Input	Pulse Input, Interrupt or Filtered Input	Pulse Input, Interrupt or Filtered Input	Regular Output	Regular Output
60	Filtered Input	Filtered Input	Filtered Input	Filtered Input	Filtered Input	Regular Output	Regular Output

Default Mode

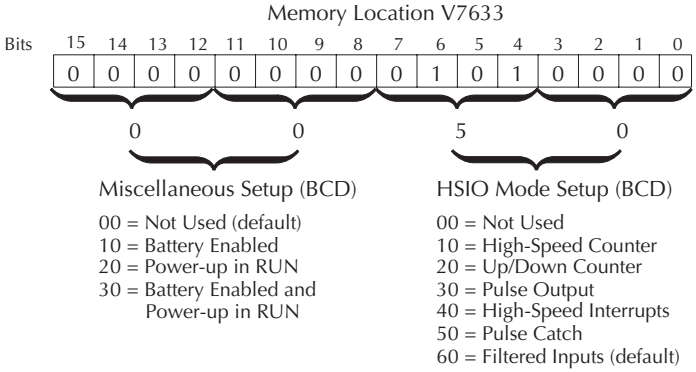
Mode 60 (Filtered Inputs) is the default mode. The DL06 is initialized to this mode at the factory, and any time you initialize the scratchpad memory. In the default condition, X0–X3 are filtered inputs (10 ms delay) and Y0–Y1 are standard outputs.

Configuring the HSIO Mode

If you have chosen a mode suited to the high-speed I/O needs of your application, we're ready to proceed to configure the PLC to operate accordingly. In the block diagram below, notice the V-memory detail in the expanded CPU block. V-memory location V7633 determines the functional mode of the high-speed I/O circuit. *This is the most important V-memory configuration value for HSIO functions!*



The contents of V7633 is a 16-bit word, to be entered in binary–coded decimal. The figure below defines what each 4-bit BCD digit of the word represents.



Bits 0 – 7 define the mode number 00, 10.. 60 previously referenced in this appendix. The example data "0050" shown selects Mode 50 – Pulse Catch (BCD = 50).

Configuring Inputs X0 – X3

In addition to configuring V7633 for the HSIO mode, you'll need to program the next four locations in certain modes according to the desired function of input points X0 – X3. Other memory locations may require configuring, depending on the HSIO mode (see the corresponding section for particular HSIO modes).

	V-Memory	
Mode	V7633	xxxx
X0	V7634	xxxx
X1	V7635	xxxx
X2	V7636	xxxx
X3	V7637	xxxx

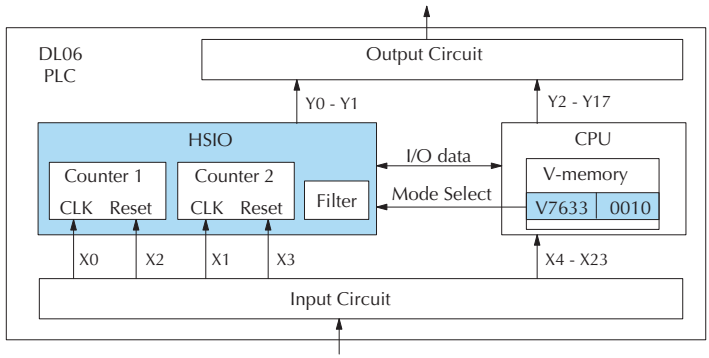
Mode 10: High-Speed Counter

Purpose

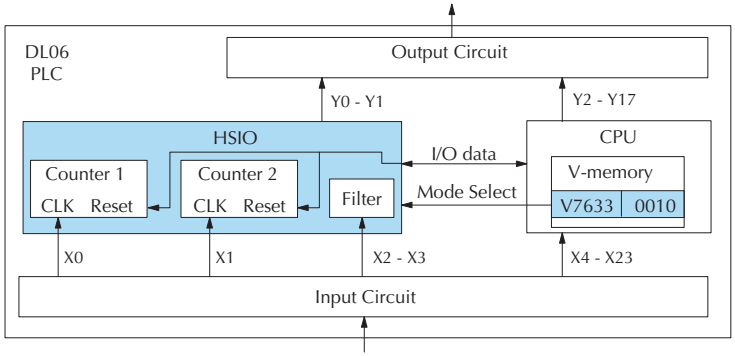
The HSIO circuit contains two high-speed counters. A single pulse train from an external source (X0) clocks the counter on each signal leading edge. The counter counts only upwards, from 0 to 99999999. The counter compares the current count with up to 24 preset values, which you define. The purpose of the presets is to quickly cause an action upon arrival at specific counts, making it ideal for such applications as cut-to-length. It uses counter registers CT174 to CT177 in the CPU.

Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD "10", the high-speed up counter in the HSIO circuit is enabled. X0 and X1 automatically become the "clock" inputs for the high-speed counters, incrementing them upon each off-to-on transition. The external reset input on X2 and X3 are the default configuration for Mode 10.

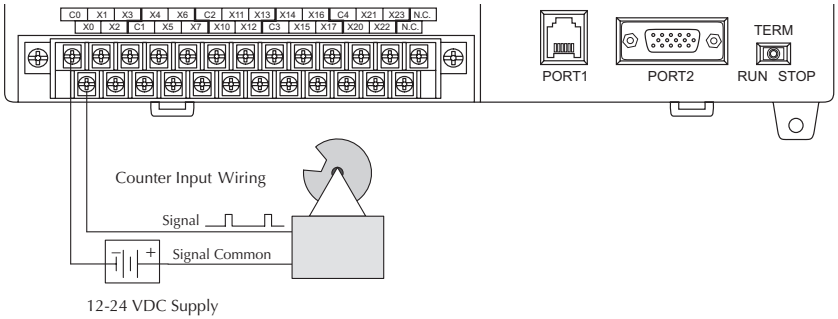


Instead of using X2 and X3 as dedicated reset inputs, you can configure X2 and X3 as normal filtered inputs. In this way, the counter reset must be generated in ladder logic.



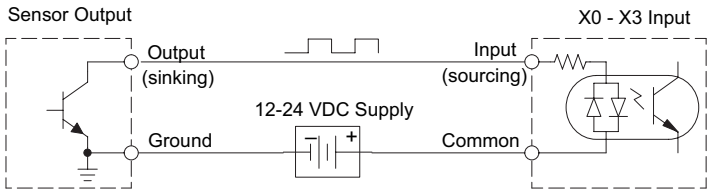
Wiring Diagram

A general wiring diagram for counters/encoders to the DL06 in HSIO Mode 10 is shown below. Many types of pulse-generating devices may be used, such as proximity switches, single-channel encoders, magnetic or optical sensors, etc. Devices with sinking outputs (NPN open collector) are probably the best choice for interfacing. If the counter sources to the inputs, it must output 12 to 24 VDC. Note that devices with 5V sourcing outputs will not work with DL06 inputs.

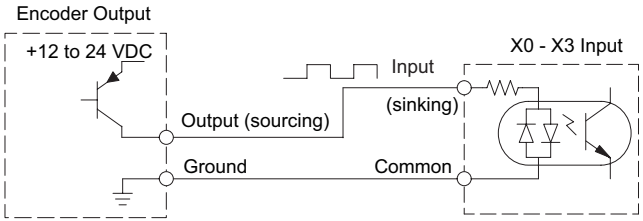


Interfacing to Counter Inputs

The DL06's DC inputs are flexible in that they detect current flow in either direction, so they can be wired to a sensor with either sourcing or sinking outputs. In the following circuit, a sensor has an open-collector NPN transistor output. It sinks current from the PLC input point, which sources current. The power supply can be the FA–24PS or another supply (+12VDC or +24VDC), as long as the input specifications are met.

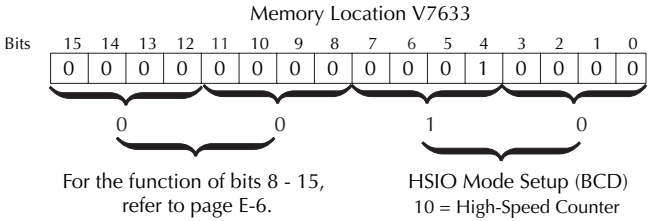


In the circuit diagram below, an encoder has open-emitter PNP transistor outputs. It sources current to the PLC input point, which sinks the current back to ground. Since the encoder sources current, no additional power supply is required. However, note that the encoder output must be 12 to 24 volts (5V encoder outputs will not work).



Setup for Mode 10

V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 10 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

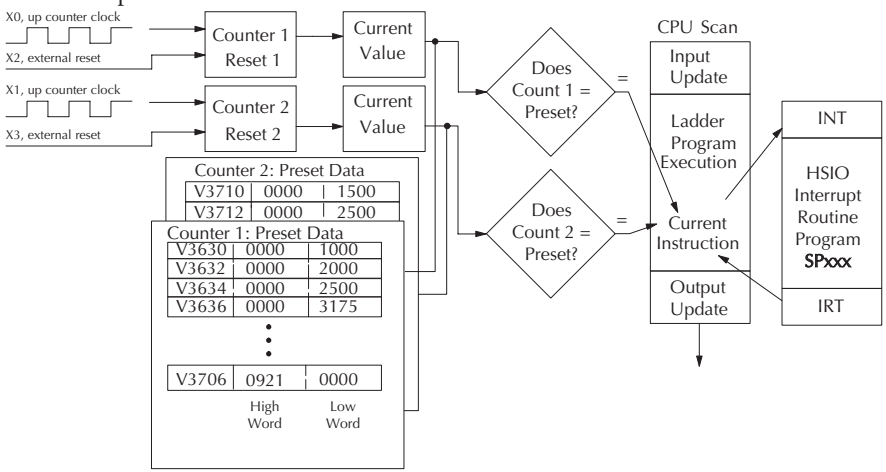
- Include load and out instructions in your ladder program
- *Direct*SOFT's memory editor or Data View
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this

Presets and Special Relays

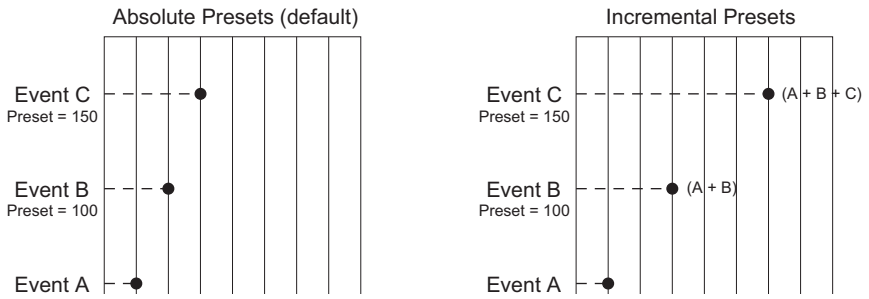
Presets are used to cause a particular action to occur when the count reaches the preset value. Refer to the figure below. Each counter features 24 presets, which you can program. Presets are double word numbers so they occupy two V-memory registers. The user selects the preset values, and the counter continuously compares the current count with the preset. When the two are equal, a special relay contact is energized and program execution jumps to the interrupt routine.

We recommend using the special relay(s) in the interrupt service routine to cause any immediate action you desire. After the interrupt service routine is complete, the CPU returns to the ladder program, resuming program execution from the point of interruption. The compare function is ready for the next preset event



Absolute and Incremental Presets

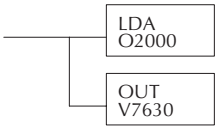
Two preset modes are available, absolute and incremental. Presets are entered into a contiguous block of V-memory registers. In the absolute mode, each preset is treated as the total count. In the incremental mode, the presets are cumulative. Incremental presets represent the number of counts between events.



In the example above, presets are established at 50, 100, and 150. The difference between absolute and incremental modes is shown. Absolute presets trigger events at the preset values, 50, 100, and 150. Incremental presets trigger events at the cumulative totals: 50, 150, and 300.

Preset Data Starting Location

V7630 is the pointer to the V-memory location which contains the beginning of the Preset Data Tables. The default starting location for the Preset Data Tables is V3630 (default after initializing scratchpad). However, you may change this by programming a different value in V7630. Use the LDA and OUT instructions as shown:



Preset Table Pointer

V7630	2000

Preset Table

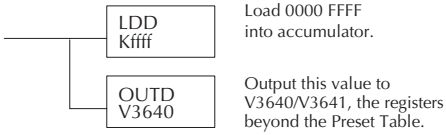
V2001	V2000	0000	1000
V2003	V2002	0000	2000
V2005	V2004	0000	2500
V2007	V2006	0000	3175
V2077	V2076	0000	0000

Using Fewer than 24 Presets

When all 24 available presets are used, the CPU knows automatically when it reaches the end of the preset table. When using fewer than 24 presets, however, it is necessary to signal the CPU that it has reached the last preset. The way to signal the end of the block of presets is to insert one of the following **table-end** codes into the next available register pair:

Table-end Code	Applicable Mode	Meaning
0000 FFFF	Absolute and Incremental	Signals end of presets
0000 00FF	Incremental	Signals end of presets and restarts presets. Does not reset accumulated pulse counts of CT174 or CT176.
0000 FF00	Incremental	Signals end of presets, restarts presets and resets accumulated pulse counts of CT174 or CT176.

As shown in the table above, each of the **table-end** signals has a different meaning. Use the LDD Kffff instruction to insert the table-end code into the next register pair beyond the preset table. In the example, four presets are used. The 0000 FFFF in V3641-V3640 indicates the previous preset was the last preset.



Default Preset Table Example

V3631	V3630	0000	1000
V3633	V3632	0000	2000
V3635	V3634	0000	2500
V3637	V3636	0000	3175
V3641	V3640	0000	FFFF

In incremental mode, you can choose not to reset the counter or the cumulative total, or you can choose to reset only the counter, or you can choose to reset the counter and the cumulative total when the table-end code is read. In the example, FFFF has been placed in V3640 since the last preset was in V3636, and we are using fewer than 24 presets.

NOTE: *In absolute mode each successive preset must be greater than the previous preset value. If a preset value is less than a lower-numbered preset value, the CPU cannot compare to that value, since the counter can only count upwards.*

Equal Relay Numbers

The following table lists all 24 preset register default locations for each high-speed counter. Each occupies two 16-bit V-memory registers. The corresponding special relay contact number is in the next column. We might also call these *equal* relay contacts, because they are true (closed) when the present high-speed counter value is equal to the preset value. Each contact remains closed until the counter value equals the next preset value.

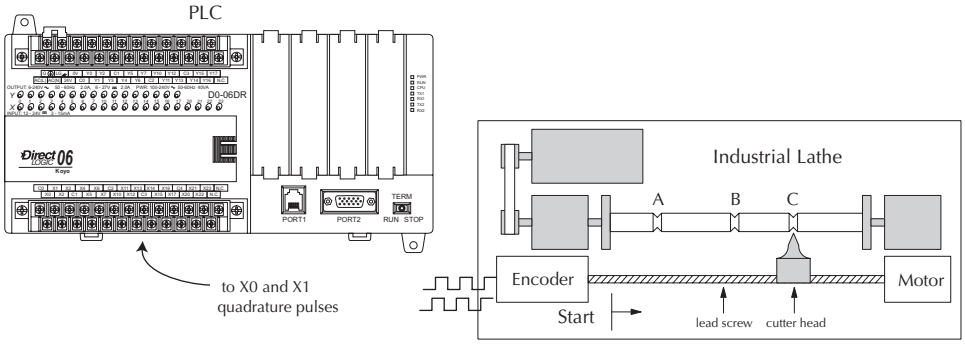
Preset Register Table					
Counter 1 Preset	Preset V-memory Register	Special Relay Number	Counter 2 Preset	Preset V-memory Register	Special Relay Number
1	V3631 / V3630	SP540	1	V3711 / V3710	SP570
2	V3633 / V3632	SP541	2	V3713 / V3712	SP571
3	V3635 / V3634	SP542	3	V3715 / V3714	SP572
4	V3637 / V3636	SP543	4	V3717 / V3716	SP573
5	V3641 / V3640	SP544	5	V3721 / V3720	SP574
6	V3643 / V3642	SP545	6	V3723 / V3722	SP575
7	V3645 / V3644	SP546	7	V3725 / V3724	SP576
8	V3647 / V3646	SP547	8	V3727 / V3726	SP577
9	V3651 / V3650	SP550	9	V3731 / V3730	SP600
10	V3653 / V3652	SP551	10	V3733 / V3732	SP601
11	V3655 / V3654	SP552	11	V3735 / V3734	SP602
12	V3657 / V3656	SP553	12	V3737 / V3736	SP603
13	V3661 / V3660	SP554	13	V3741 / V3740	SP604
14	V3663 / V3662	SP555	14	V3743 / V3742	SP605
15	V3665 / V3664	SP556	15	V3745 / V3744	SP606
16	V3667 / V3666	SP557	16	V3747 / V3746	SP607
17	V3671 / V3670	SP560	17	V3751 / V3750	SP610
18	V3673 / V3672	SP561	18	V3753 / V3752	SP611
19	V3675 / V3674	SP562	19	V3755 / V3754	SP612
20	V3677 / V3676	SP563	20	V3757 / V3756	SP613
21	V3701 / V3700	SP564	21	V3761 / V3760	SP614
22	V3703 / V3702	SP565	22	V3763 / V3762	SP615
23	V3705 / V3704	SP566	23	V3765 / V3764	SP616
24	V3707 / V3706	SP567	24	V3767 / V3766	SP617

The consecutive addresses shown above for each relay are those assigned by the CPU as default addresses. The Pointer for the start of these addresses is stored by the CPU at V7630. If you have a conflict of addresses because of pre-existing code written to these addresses, you can change the default block of addresses merely by having your ladder logic place a different pointer value in V7630. To change the table location. use the LDA and OUT instructions as shown on the previous page.

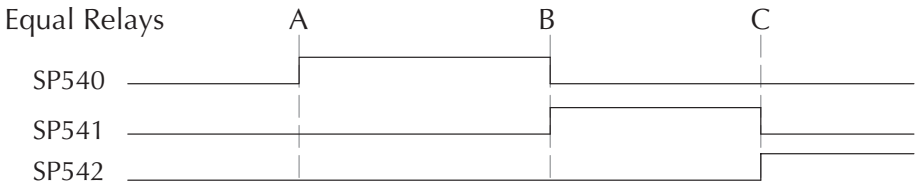
Calculating Your Preset Values

The preset values occupy two data words each. They can range in value from -8388608 to 8388607, just like the high-speed counter value. All 24 values are absolute values, meaning that each one is an offset from the counter zero value.

The preset values must be individually derived for each application. In the industrial lathe diagram below, the PLC monitors the position of the lead screw by counting pulses. At points A, B, and C along the linear travel, the cutter head pushes into the work material and cuts a groove.



The timing diagram below shows the duration of each equal relay contact closure. Each contact remains on until the next one closes. All go off when the counter resets.



NOTE: *Each successive preset must be two numbers greater than the previous preset value. In the industrial lathe example, B>A+2 and C>B+2.*

X Input Configuration

The configurable discrete input options for High-Speed Counter Mode are listed in the table below. Input X0 is dedicated for the first counter clock input. Input X1 can be the clock for the second counter or a filtered input. The section on Mode 60 operation at the end of this appendix describes programming the filter time constants. Inputs X2 and X3 can be configured as the counter resets, with or without the interrupt option. The interrupt option allows the reset input (X2 and X3) to cause an interrupt like presets do, but there is no SP relay contact closure (instead, X2 and X3 will be on during the interrupt routine, for 1 scan). Or finally, X2 and X3 may be left simply as a filtered input.

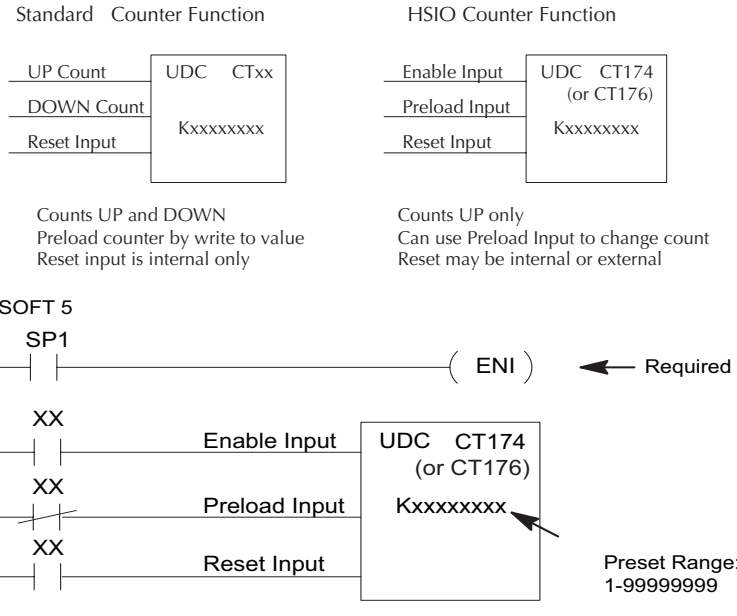
Input Options			
Input	**Configuration Register**	**Function**	**Hex Code Required**
X0	V7634	Counter #1 Clock	0001 (absolute) (default) 0101 (incremental)
X1	V7635	Counter #2 Clock	0001 (absolute) (default) 0101 (incremental)
		Interrupt	0004
		Pulse Input	0005
		Filtered Input	xx06, xx = filter time 0 - 99 ms (BCD)
X2	V7636	Counter #1 Reset (no interrupt)	0007* (default) 0207*
		Counter #1 Reset (with interrupt)	0107* 0307*
		Interrupt	0004
		Pulse Input	0005
		Filtered Input	xx06, xx= filter time 0 - 99 ms (BCD)
X3	V7637	Counter #2 Reset (no interrupt)	0007* (default) 0207*
		Counter #2 Reset (with interrupt)	0107* 0307*
		Interrupt	0004
		Pulse Input	0005
		Filtered Input	xx06, xx= filter time 0 - 99 ms (BCD)

*With the counter reset, you have the option of a normal reset or a faster reset. However, the fast reset does not recognize changed preset values during program execution. When '0007' or '0107' are set in V7636 or V7637 and preset values are changed during program execution, the DL06 recognizes the changed preset values at the time of the reset. When '0207' or '0307' are set in V7636 or V7637, the CPU does not check for changed preset values, so the DL06 has a faster reset time.

Writing Your Control Program

The mnemonic for the counter instruction is UDC (up-down counter).The DL06 can have up to 128 counters, labeled CT0 through CT177. The high speed counter in the HSIO circuit is accessed in ladder logic by using UDC CT174 and CT176. It uses counter registers CT174 through CT177 exclusively when the HSIO mode 10 is active (otherwise, CT174 through CT177 are available for standard counter use). The HSIO counter needs two registers because it is a double-word counter. It has three inputs as shown. The first input (Enable) allows counting when active. The middle input is used to preload the counter value. The bottom signal is the reset. The Preload Input must be off while the counter is counting.

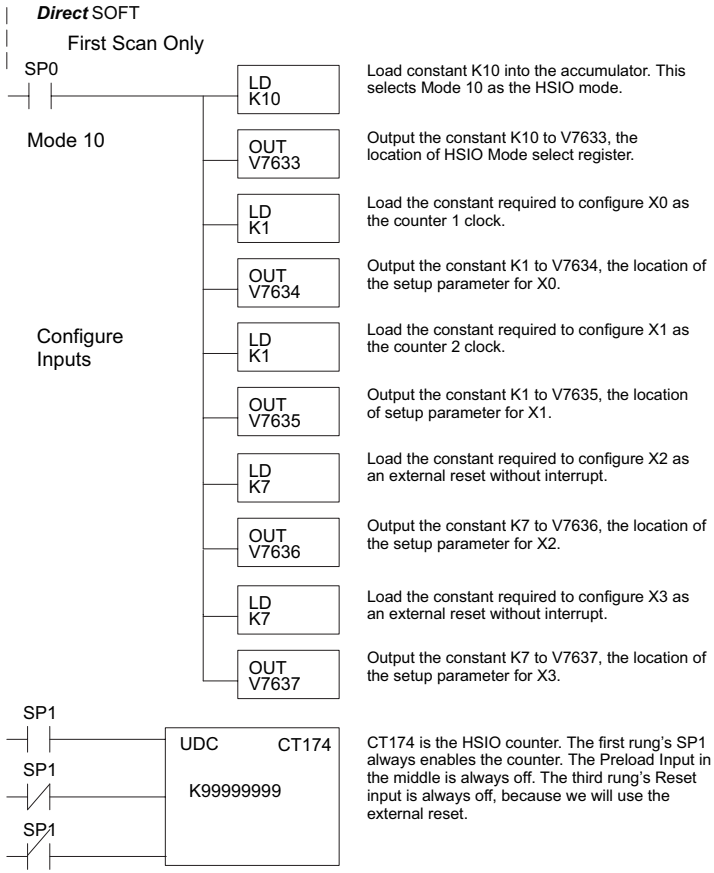
The next figure shows how the HSIO counter will appear in a ladder program. Note that the Enable Interrupt (ENI) command must execute before the counter value reaches the first preset value. We do this at powerup by using the first scan relay. When using the counter but not the presets and interrupt, we can omit the ENI.



When the enable input is energized, the up/down counter CT174 will respond to pulses on X0 and increment. The up/down counter CT176 will respond to pulses on X1 and increment. The reset input contact behaves in a logical OR fashion with the physical reset input. X2 (when selected) resets counter 1. X3 (when selected) resets counter 2. So, the high speed counter can receive a reset from either the contact(s) on the reset rung in the ladder, OR the external reset X2 or X3, if you have configured X2 or X3 as an external reset.

Program Example 1: Counter Without Presets

The following example is the simplest way to use the high-speed counters, which does not use the presets and special relays in the interrupt routine. The program configures the HSIO circuit for Mode 10 operation, so X0 is automatically the counter clock input for the first counter, and X1 is the counter clock input for the second counter. It uses the Compare-double (CMPD) instruction to cause action at certain count values. Note that this allows you to have more than 24 presets. Then, it configures X2 and X3 to be the external reset of the counter.

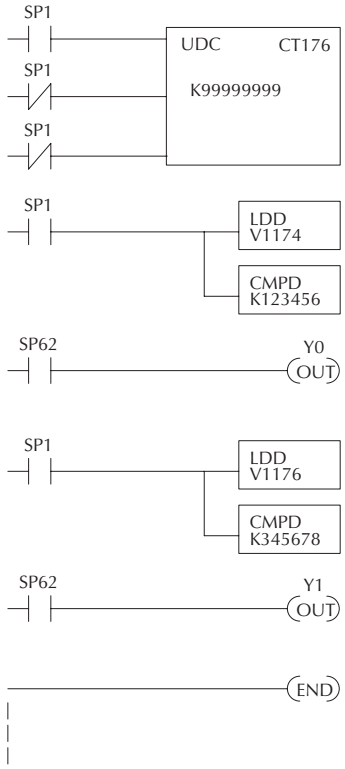


continued on next page

Program Example: (cont'd)

The compare double instructions below use the current count of the HSIO counter to turn on Y0 and Y1. This technique can make more than 24 comparisons, but it is scan-time dependent. However, use the 24 built-in presets with the interrupt routine if your application needs a very fast response time, as shown in the next example.

continued from previous page



CT176 always enables the counter. The Preload Input in the middle is always of f. The third rung's Reset input is always off, because we will use the external reset.

Load the current count of the HSIO counter in V1174 and V1175 into the accumulator

Use the Compare-double instruction to compare the double word in the accumulator to the constant K123456.

The execution of the above CMPD instruction turns on special relay contact SP62 if the current count is greater than the comparison number (K123456).

Load the current count of the HSIO counter in V1176 and V1177 into the accumulator

Use the Compare-double instruction to compare the double word in the accumulator to the constant K345678

The execution of the above CMPD instruction turns on special relay contact SP62 if the current count is greater than the comparison number (K345678).

END coil marks the end of the main program.

Program Example 2: Counter With Presets

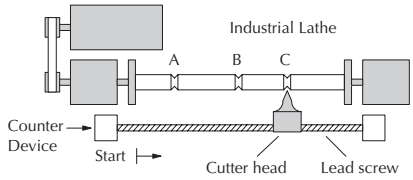
The following example shows how to program the HSIO circuit to trigger on three preset values. You may recall the industrial lathe example from the beginning of this appendix. This example program shows how to control the lathe cutter head to make three grooves in the work-piece at precise positions. When the lead screw turns, the counter device generates pulses which the DL06 can count. The three preset variables A, B, and C represent the positions (number of pulses) corresponding to each of the three grooves. In this example, only one high-speed counter is used. The second counter could be used in the same manner.

Preset Data

A	V3630	0000	1500
B	V3632	0000	3780
C	V3634	0000	4850
	V3636	0000	FFFF

I/O Assignments

X3 - Cutter head extended
X4 - Cutter head retracted
Y0 - Lead screw motor
Y1 - Cutter head solenoid

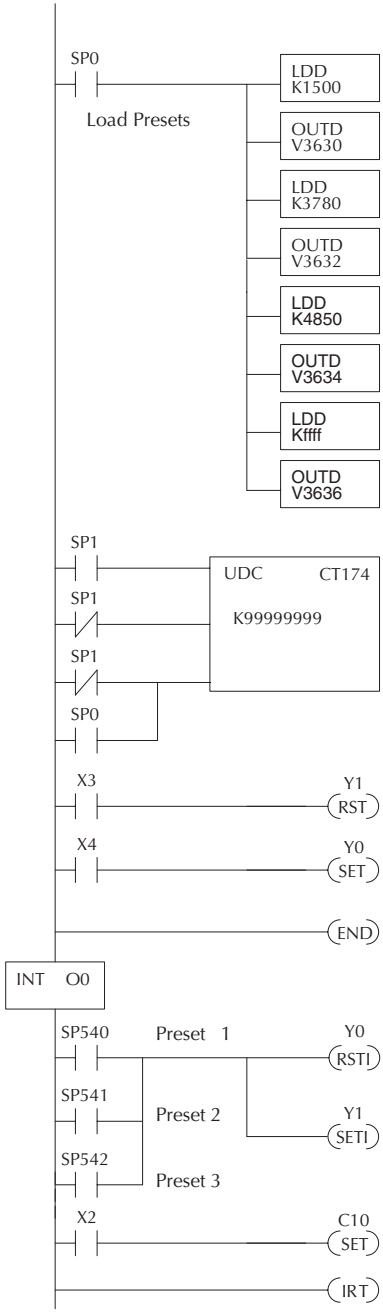


***Direct*SOFT**

SP0	ENI	Enable Interrupts before reaching a preset generates an interrupt. Special Relay SP0 is on during the first CPU scan.
SP0	LD K10	Load constant K10 into the accumulator. This selects Mode 10 as the HSIO mode.
	OUT V7633	Output the constant to V7633, the location of HSIO Mode select register.
	LD K1	Load the constant required to configure X0 as the counter clock.
	OUT V7634	Output the constant K1 to V7634, the location of the setup parameter for X0.
Select Mode 10	LD K1006	Load the constant required to configure X1 as a filtered input.
	OUT V7635	Output the constant to V7635, the location of the setup parameter for X1.
	LD K107	Load the constant required to configure X2 as an external reset with interrupt.
	OUT V7636	Output the constant to V7636, the location of the setup parameter for X2.
	LD K1006	Load the constant required to configure X3 as a filtered input.
	OUT V7637	Output the constant to V7637, the location of the setup parameter for X3.
	LDA O3630	Load the octal address O3630 into the accumulator. This instruction automatically converts the address into hex.
	OUT V7630	Output this address to V7630, the location of the pointer to the Preset Table.

continued on next page

continued from previous page



Load the preset A value into the accumulator .

Output the accumulator contents to the memory location for preset 1.

Load the preset B value into the accumulator .

Output the accumulator contents to the memory location for preset 2.

Load the preset C value into the accumulator .

Output the accumulator contents to the memory location for preset 3.

Load the constant Kffff into the accumulator. This value represents the end of the preset list.

Output the accumulator contents to the memory location for preset 4 (end of preset marker).

CT174 is the HSIO counter. The first rung's SP1 always enables the counter. The preload input in the middle is off (unused in this example).

The third rung's Reset input is normally off, because we will use the external reset. You can optionally reset the counter value on each powerup using the SP0 contact.

Input X3 energizes when the groove has finished cutting. So, we retract the cutter head.

Turn lead screw on again, after cutter head has retracted.

END coil marks the end of the main program.

The INT label marks the beginning of the interrupt service routine program.

Inside the interrupt service routine, we turn OFF the lead screw motor immediately.

These special "equal" relays turn on individually as the corresponding preset is reached. In this application, each results in the cutting of a groove (Y1), so they are logically ORed together .

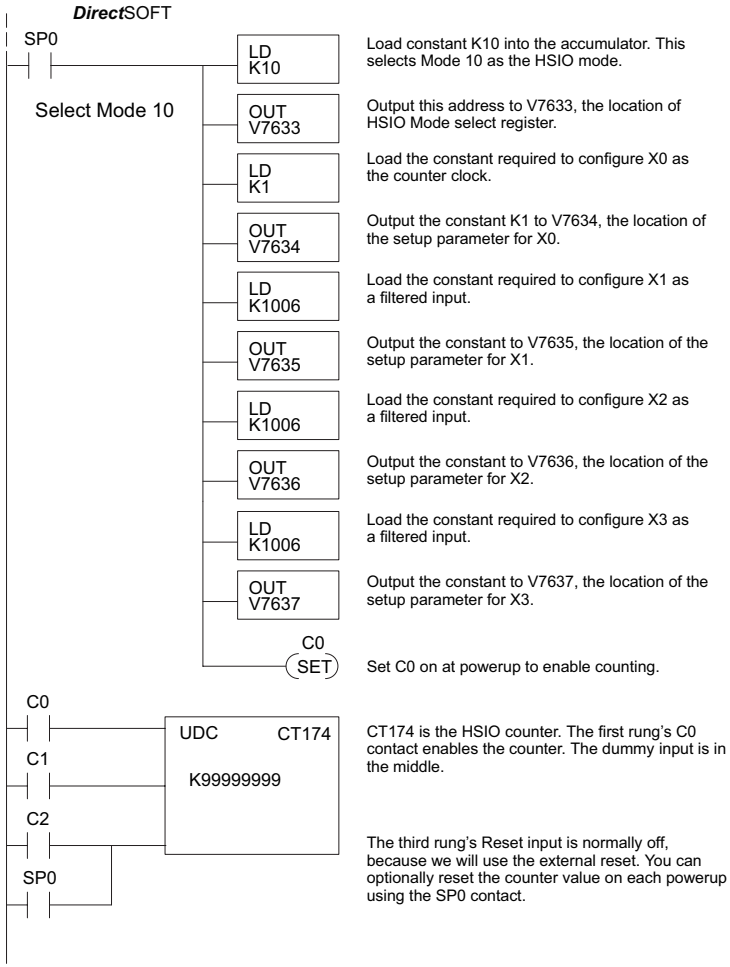
Input X2 will be energized inside the interrupt routine if X2 external interrupt was the source.

Return from the interrupt service routine.

Some applications will require a different type of action at each preset. It is possible for the interrupt routine to distinguish one preset event from another, by turning on a unique output for each equal relay contact SPxxx. We can determine the source of the interrupt by examining the equal relay contacts individually, as well as X2. The X2 contact will be on (inside the interrupt routine only) if the interrupt was caused by the external reset, X2 input.

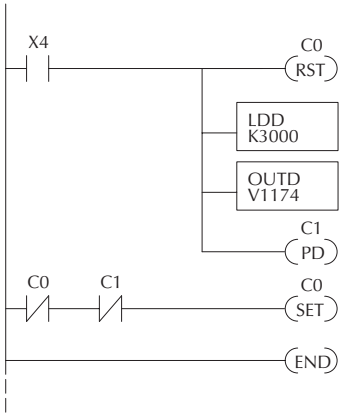
Program Example 3: Counter With Preload

The following example shows how you can preload the current count with another value. When the preload command input (X4 in this example) is energized, we disable the counter from counting with C0. Then, we write the value K3000 to the count register (V1076-V1077). We preload the current count of the counter with K3000. When the preload command (X4) is turned off, the counter resumes counting any pulses, but now starting from K3000. In this example, only one high-speed counter is used. The second counter could be used in the same manner.



continued on next page

continued from previous page



When the preload request is made, the user turns on X4. First we disable counting by resetting C0, the counter's enable input.

Load the BCD value K3000 into the accumulator.

Output the constant to V1174/V1175, the location of the accumulated count for CT76.

Generate a preload counter input pulse, which causes the counter to preload from V1174-V1175.

Enable the counter by setting C0, when the preolad pulse on C1 has occurred (C1 is off).

END coil marks the end of the main program.

Troubleshooting Guide for Mode 10

If you're having trouble with Mode 10 operation, please study the following symptoms and possible causes. The most common problems are listed below.

Symptom: The counter does not count.

Possible causes:

1. **Field sensor and wiring** – Verify that the encoder, proximity switch, or sensor actually turns on and illuminates the status LED for X0 (counter 1) and X1 (counter 2). The problem could be due to sinking-sourcing wiring problem, etc. Remember to check the signal ground connection. Also verify that the pulse on-time is long enough for the PLC to recognize it.
2. **Configuration** – use the Data View window to check the configuration parameters. V7633 must be set to 10, and V7634 must be set to 1 or 101 to enable the first high-speed counter. V7635 must be set to 1 or 101 to enable the second high-speed counter.
3. **Stuck in reset** – check the input status of the reset input, X2 and X3. If X2 is on, the counter will not count because it is being held in reset.
4. **Ladder program** – make sure you are using counter CT174 and CT176 in your program. The top input is the enable signal for the counter. It must be on before the counter will count. The middle input is the dummy input. The bottom input is the counter reset, and must be off during counting.

Symptom: The counter counts but the presets do not function.

Possible causes:

1. **Configuration** – Ensure the preset values are correct. The presets are 32-bit BCD values having a range of 0 to 99999999. Make sure you write all 32 bits to the reserved locations by using the LDD and OUTD instructions. Use only even–numbered addresses, from V3630 to V3767. If using less than 24 presets, be sure to place "0000FFFF," "0000FF00," or "000000FF" in the location after the last preset used.
2. **Interrupt routine** – Only use Interrupt #0. Make sure the interrupt has been enabled by executing an ENI instruction prior to needing the interrupt. The interrupt routine must be placed after the main program, using the INT label and ending with an interrupt return IRT.
3. **Special relays** – Check the special relay numbers in your program. Use SP540 for Preset 1, SP541 for Preset 2, etc. Remember that only one special equal relay contact is on at a time. When the counter value reaches the next preset, the SP contact which is on now goes off and the next one turns on.

Symptom: The counter counts up but will not reset.

Possible causes:

1. Check the LED status indicator for X2 (counter 1) and X3 (counter 2) to make sure it is active when you want a reset. Or, if you are using an internal reset, use the status mode of DirectSOFT to monitor the reset input to the counter.

Mode 20: Up/Down Counter

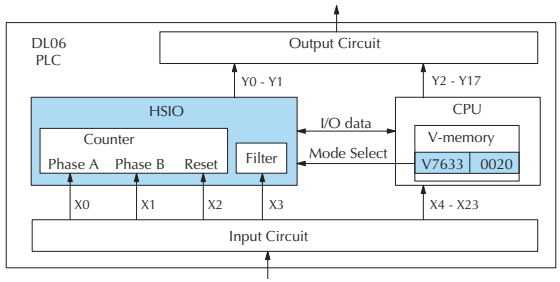
Purpose

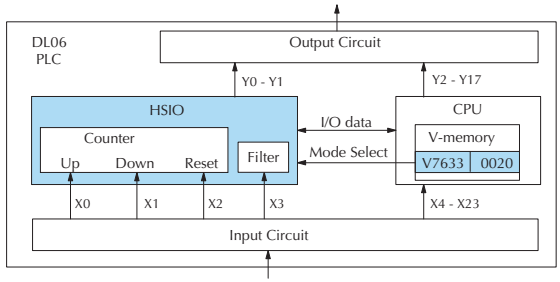
The counter in the HSIO circuit can count up/down signals from two separate sources (i.e., two single channel encoders) or two quadrature signal pulses. Quadrature signals are commonly generated from incremental encoders, which may be rotary or linear. The up/down counter has a range from -8388608 to 8388607. Using CT174 and CT175, the quadrature counter can count at up to a 7 kHz rate.

NOTE: *The count is 32 bit BCD sign + magnitude format.*

Functional Block Diagram

The diagram below shows HSIO functionality in Mode 20. When the lower byte of HSIO Mode register V7633 contains a BCD "20", the up/down counter in the HSIO circuit is enabled. For quadrature counting, input X0 is dedicated to the Phase A quadrature signal, and input X1 receives Phase B signal. X2 is dedicated to reset the counter to zero value when energized.





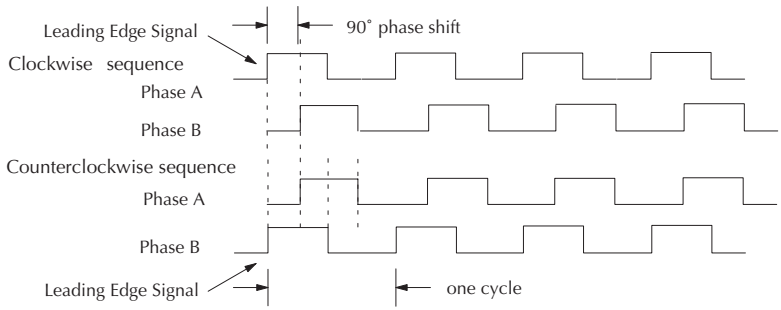
For standard up/down counting, input X0 is dedicated to the up counting signal, and input X1 is dedicated to the down counting signal. The X2 input resets the counter to zero when energized.

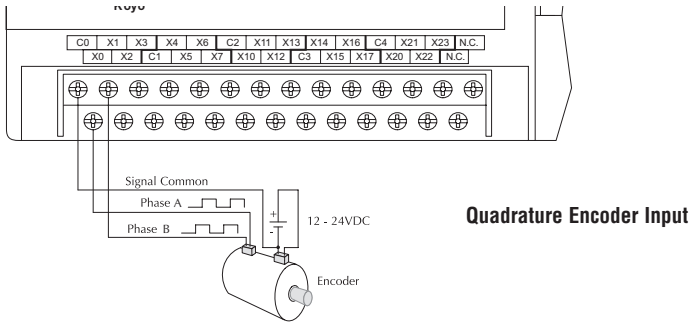
Quadrature Encoder Signals

Quadrature encoder signals contain position and direction information, while their frequency represents speed of motion. Phase A and B signals shown below are phase-shifted 90 degrees, thus the quadrature name. When the rising edge of Phase A precedes Phase B's leading edge (indicates clockwise motion by convention), the HSIO counter counts UP. If Phase B's rising edge precedes Phase A's rising edge (indicates counter-clockwise motion), the counter counts DOWN.

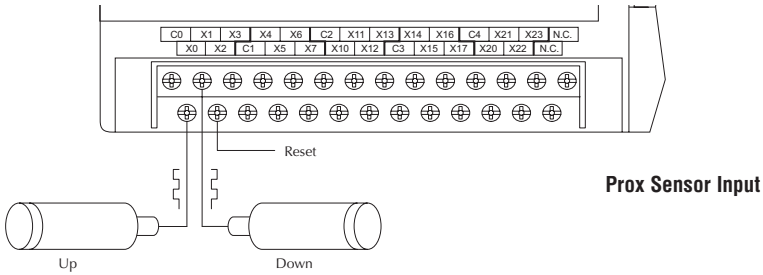
Wiring Diagram

A general wiring diagram for encoders to the DL06 in HSIO Mode 20 is shown below. Encoders with sinking outputs (NPN open collector) are probably the best choice for interfacing. If the encoder sources to the inputs, it must output 12 to 24 VDC. Note that encoders with 5V sourcing outputs will not work with DL06 inputs.





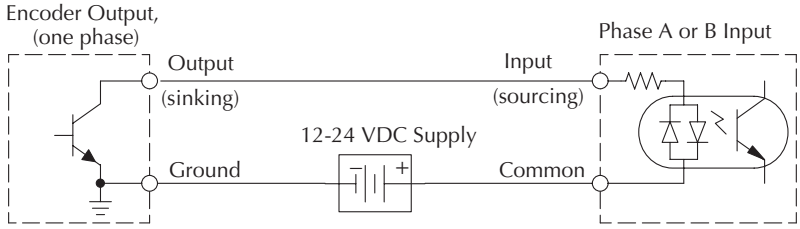
Quadrature Encoder Input



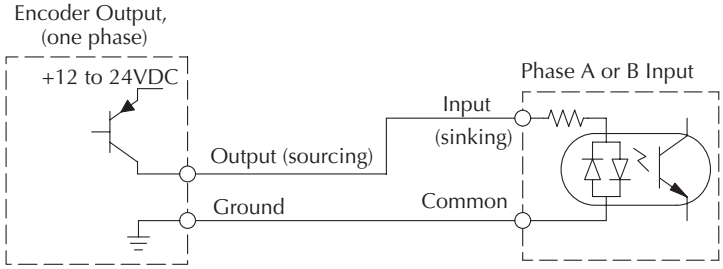
Prox Sensor Input

Interfacing to Encoder Outputs

The DL06's DC inputs are flexible in that they detect current flow in either direction, so they can be wired to an encoder with either sourcing or sinking outputs. In the following circuit, an encoder has open-collector NPN transistor outputs. It sinks current from the PLC input point, which sources current. The power supply can be the +24VDC auxiliary supply or another supply (+12VDC or +24VDC), as long as the input specifications are met.

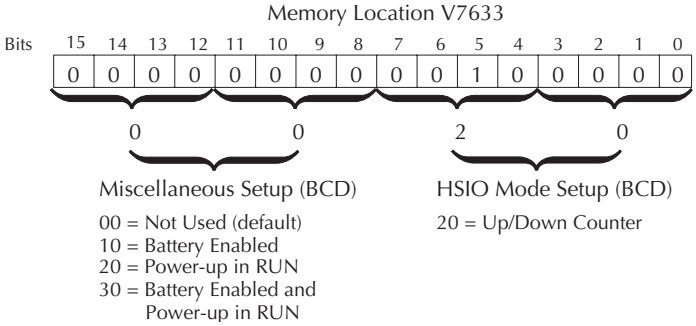


In the next circuit, an encoder has open-emitter PNP transistor outputs. It sources current to the PLC input point, which sinks the current back to ground. Since the encoder sources current, no additional power supply is required. However, note that the encoder output must be 12 to 24 volts (5V encoder outputs will not work).



Setup for Mode 20

Remember that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 20 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

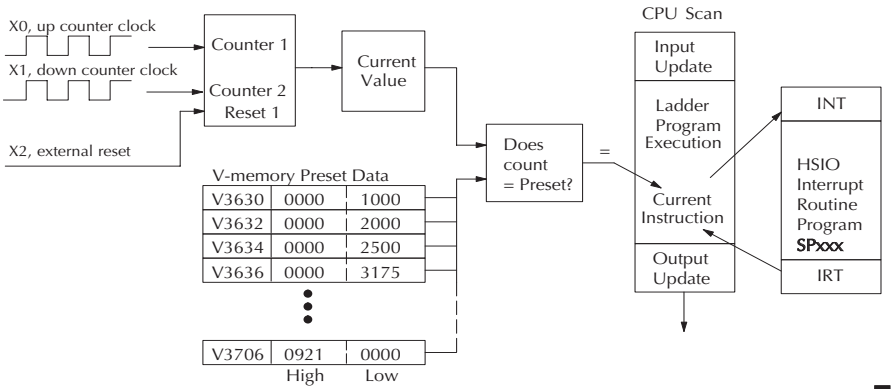
- Include load and out instructions in your ladder program
- ***Direct***SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

Presets and Special Relays

The goal of counting is to cause a particular action to occur when the count reaches a preset value. Refer to the figure below. Each counter features 24 presets, which you can program. A preset is a number you select and store so that the counter will continuously compare the current count with the preset. When the two are equal, a special relay contact is energized and program execution jumps to the interrupt routine.

We recommend using the special relay(s) in the interrupt service routine to cause any immediate action you desire. After the interrupt service routine is complete, the CPU returns to the ladder program, resuming program execution from the point of interruption. The compare function is ready for the next preset event.



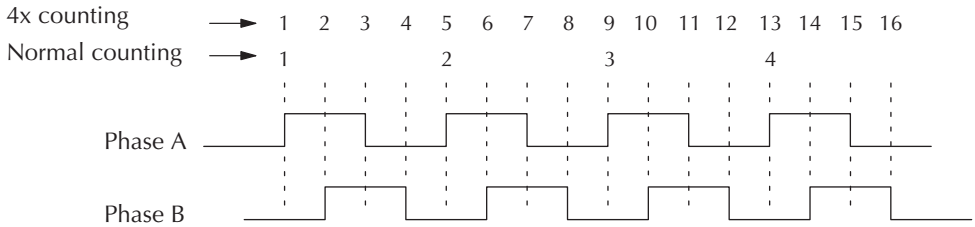
X Input Configuration

The configurable discrete input options for High-Speed Counter Mode are listed in the table below. The section on Mode 60 operation at the end of this appendix describes programming the filter time constants.

Mode 20 Up/Down Counter

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Up counting	0202 (standard, absolute)
			0302 (standard, incremental)
		Phase A	0002 (quadrature, absolute) (default)
			0102 (quadrature, incremental)
			1002 (quadrature, absolute) 4x counting*
			1102 (quadrature, incremental) 4x counting*
X1	V7635	Down counting or Phase B	0000
X2	V7636	Counter Reset (no interrupt)	0007** (default) 0207**
		Counter Reset (with interrupt)	0107** 0307**
		Pulse input	0005
		Filtered input	xx06 (xx = filter time, 0 - 99ms (BCD)
X3	V7637	Pulse input	0005
		Filtered input	xx06 (xx = filter time, 0 - 99ms (BCD) (default)

* With this feature, you can count 4 times more with the same encoder.

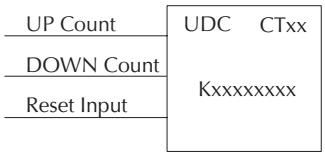


** With the counter reset you have the option of a normal reset or a faster reset. However, the fast reset does not recognize changed preset values during program execution. When '0007' or '0107' are set in V7636 and preset values are changed during program execution, the DL06 recognizes the changed preset values at the time of the reset. When '0207' or '0307' are set in V7636 the CPU does not check for changed preset values, so the DL06 has a faster reset time.

Writing Your Control Program

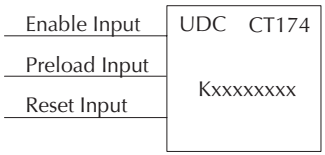
The mnemonic for the counter is UDC (up-down counter). The DL06 can have up to 128 counters, labeled CT0 through CT177. The quadrature counter in the HSIO circuit is accessed in ladder logic by using UDC CT174. It uses counter registers CT174 and CT175 exclusively when the HSIO mode 20 is active (otherwise, CT174 and CT175 are available for standard counter use). The HSIO counter needs two registers because it is a double-word counter. It has three inputs as shown. The first input is the enable signal, the middle is a preload (write), and the bottom is the reset. The enable input must be on before the counter will count. The enable input must be off during a preload.

Standard Counter Function



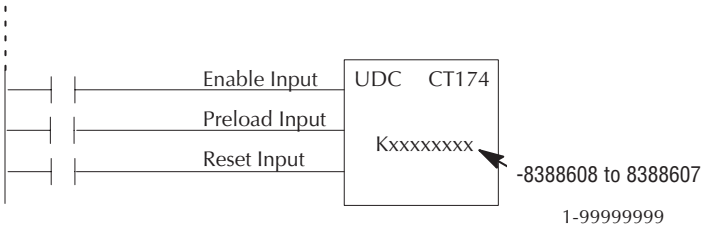
Counts UP and DOWN
Preload counter by write to value
Reset input is internal only

HSIO Counter Function



Counts UP and DOWN (from X0, X1)
Can use Preload Input to change count
Reset may be internal or external

The next figure shows the how the HSIO quadrature counter will appear in a ladder program.

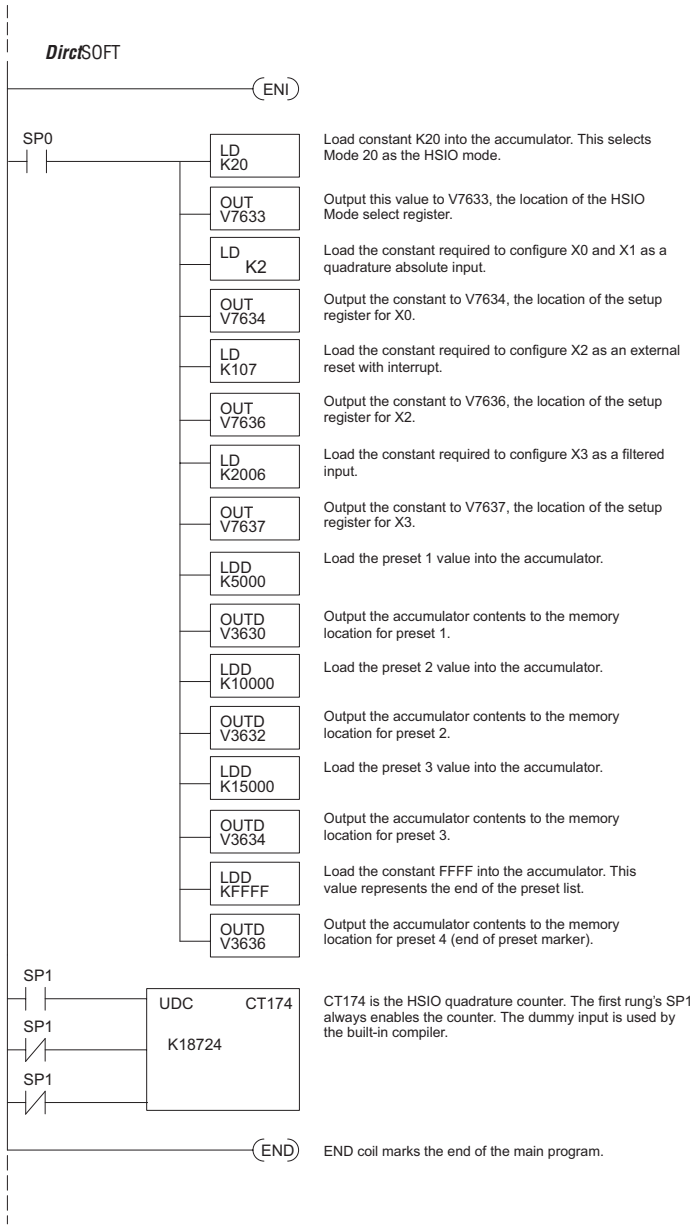


When the enable input is energized, the counter will respond to quadrature pulses on X0 and X1, incrementing or decrementing the counter at CT174 – CT175. The reset input contact behaves in a logical OR fashion with the physical reset input X2. This means the quadrature counter can receive a reset from either the contact(s) on the reset rung in the ladder, OR the external reset X2.

NOTE: *The count is 32 bit BCD sign + magnitude format.*

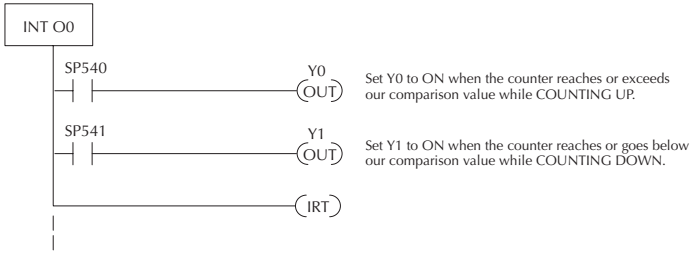
Program Example 1: Quadrature Counting with an Interrupt

Below is a simple example of how quadrature counting with an interrupt can be programmed.



continued on next page

continued from previous page



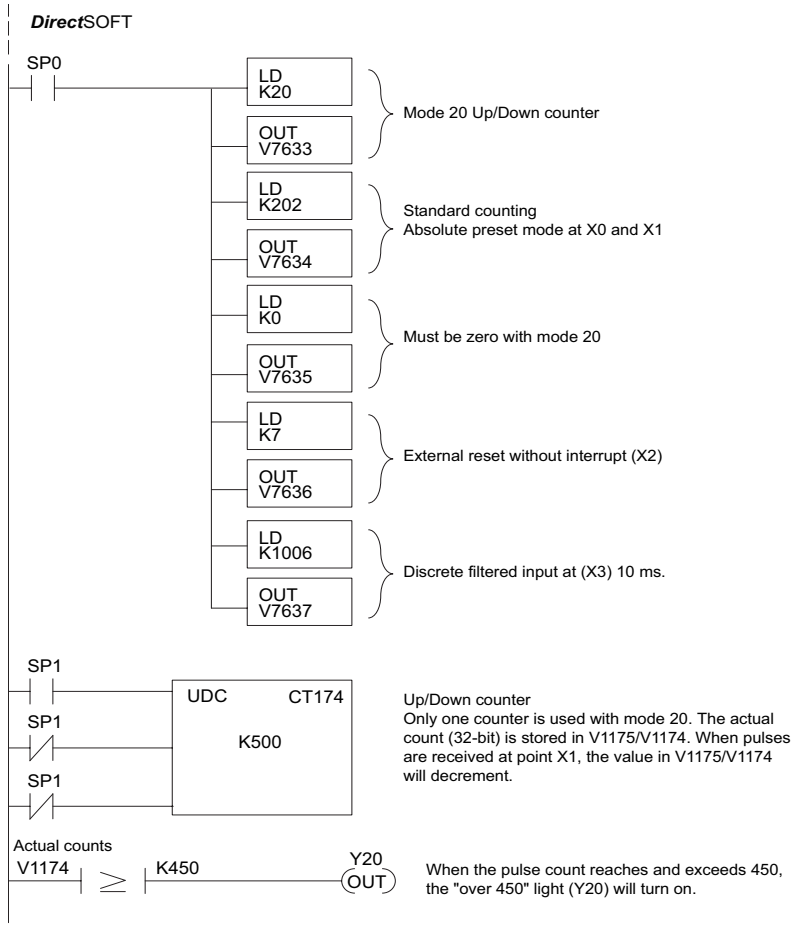
The Load Accumulator instructions have set up the V-memory as required, i.e., 20 in V7633 for the mode and 0202 in V7634 to designate the standard up/down with the absolute preset mode. By placing 0107 in V7636, an external reset for counter CT174 is selected and it will execute interrupt 0 on the rising edge of the reset. Presets for up/down counting have been stored in memory locations V3630 through V3635. The next even numbered location following this has FFFF to indicate we have no more presets.

Program Example 2: Up/Down Counting with Standard Inputs

In this example, there is a conveyor belt "A" that transports bottles to be inspected. During the course of the process, one sensor is keeping track of the bottles that are going onto belt "A" for inspection, and another sensor is keeping track of how many bottles are being removed to the finished product line.

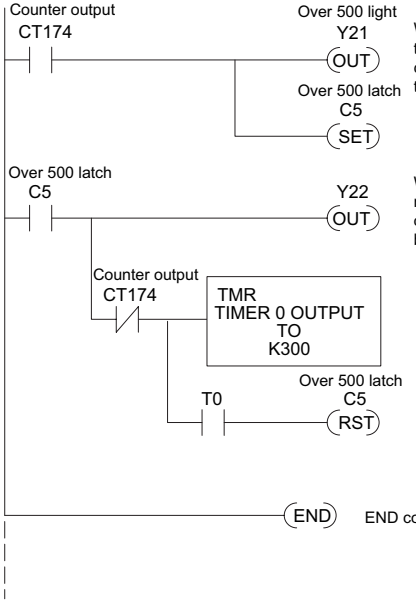
When we have reached 500 bottles in the process, an "over 500" light turns on and a rerouting gate is activated to channel the incoming bottles to conveyor belt "B". The rerouting gate will stay activated for 30 seconds after the conveyor belt "A" contains less than 500 bottles.

The program below shows how ladder logic might be written to handle the job. Note the use of V1174. This memory location stores the current count for CT174 which is used with the DL06.



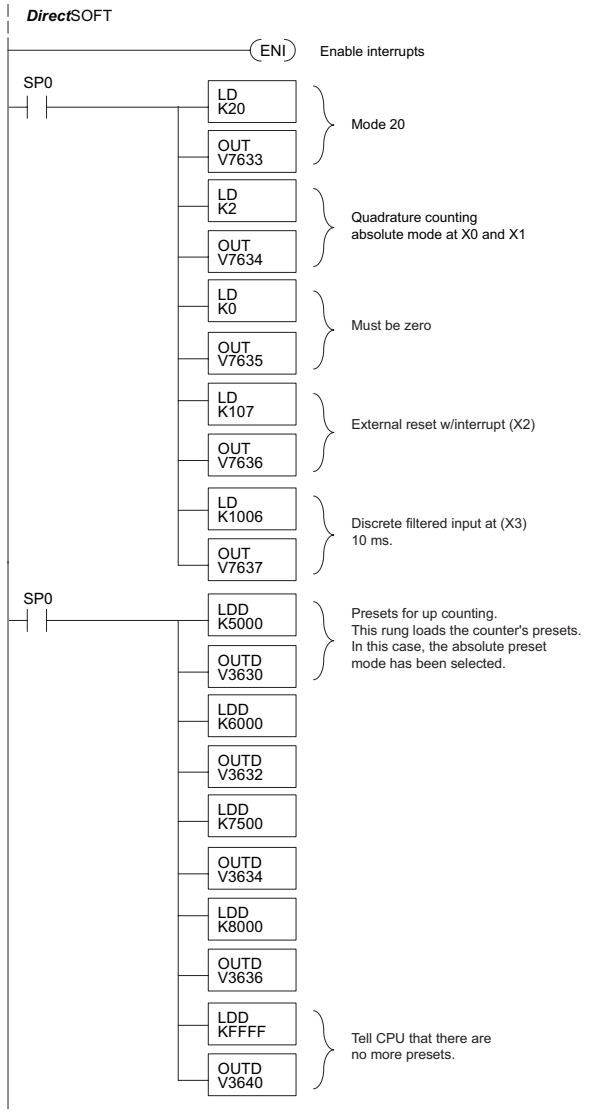
Continued on next page.

continued from previous page



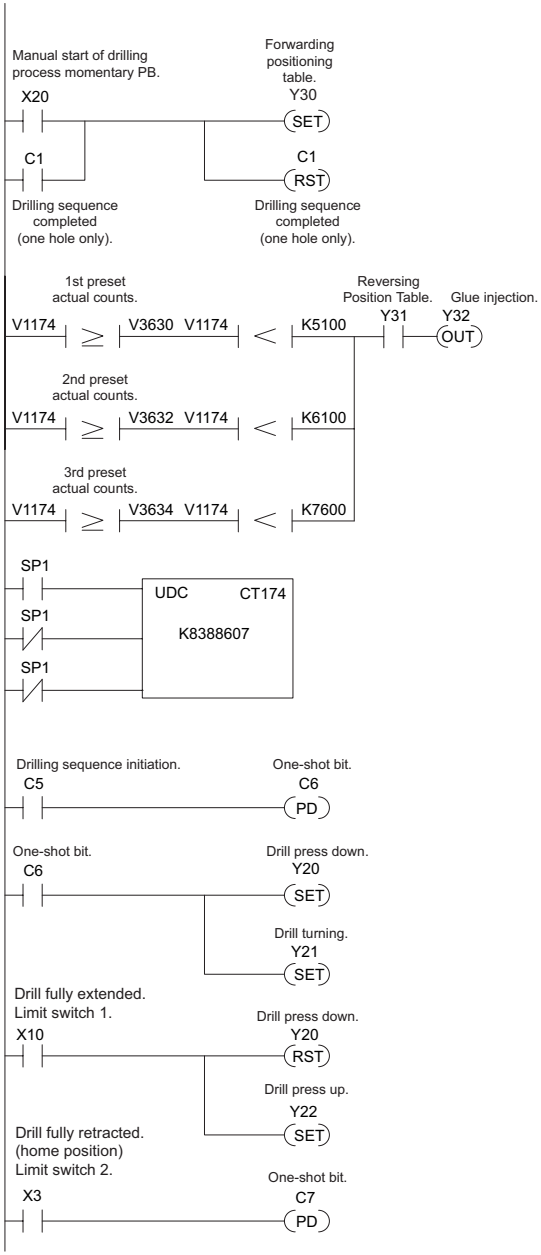
Program Example 3: Quadrature Counting

In this example, a wooden workpiece is being drilled with 3 holes and then the holes are injected with glue for dowels to be inserted at another workstation. A quadrature encoder is connected to a positioning table which is moving a drill press horizontally over the workpiece. The positioning table will stop and the drill press will lower to drill a hole in an exact location. After the three holes are drilled in the workpiece, the positioning table reverses direction and injects glue into the same holes.



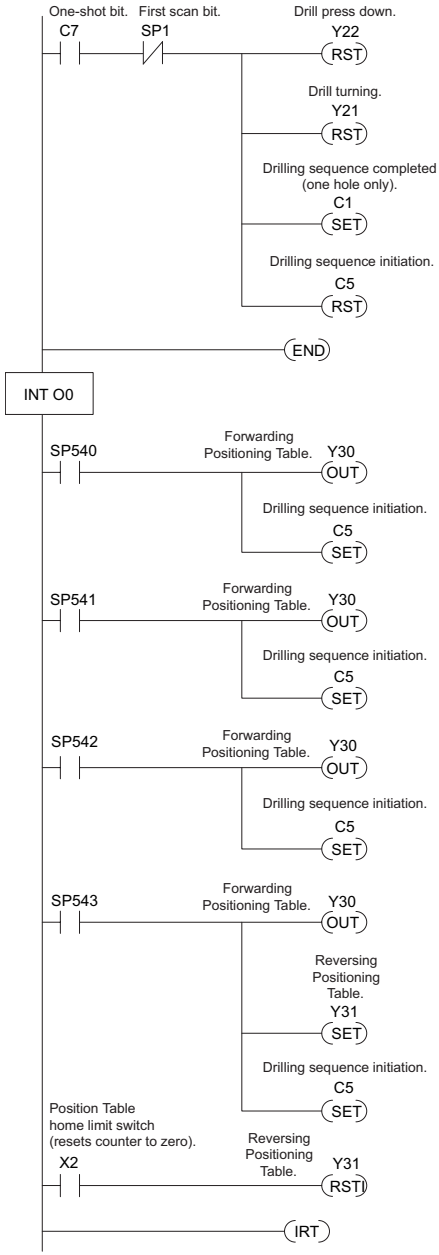
Continued on next page.

Continued from previous page.



Continued on next page.

Continued from previous page.



Troubleshooting Guide for Mode 20

If you're having trouble with Mode 20 operation, please study the following symptoms and possible causes. The most common problems are listed below.

Symptom: The counter does not count.

Possible causes:

1. **Field sensor and wiring** – Verify that the encoder or other field device inputs actually turn on and illuminate the status LEDs for X0 and X1. A standard incremental encoder will visibly, alternately turn on the LEDs for X0 and X1 when rotating slowly (1 RPM). Or, the problem could be due to a sinking-sourcing wiring problem, etc. Remember to check the signal ground connection. Also verify that the pulse on-time, duty cycle, voltage level, and frequency are within the input specifications.
2. **Configuration** – make sure all of the configuration parameters are correct. V7633 must be set to 20, and V7634 must be set to "0002" to enable the Phase A input, and V7635 must be set to "0000" to enable the Phase B input.
3. Stuck in reset – check the input status of the reset input, X2. If X2 is on, the counter will not count because it is being held in reset.
4. **Ladder program** – make sure you are using counter CT174 in your program. The top input is the enable signal for the counter. It must be on before the counter will count. The middle input is the dummy input and must be off for the counter to count. The bottom input is the counter reset, and must be off during counting.

Symptom: The counter counts in the wrong direction (up instead of down, and visa-versa).

Possible causes:

1. **Channel A and B assignment** – It's possible that Channel A and B assignments of the encoder wires is backwards from the desired rotation/counting orientation. Just swap the X0 and X1 inputs, and the counting direction will be reversed.

Symptom: The counter counts up and down but will not reset.

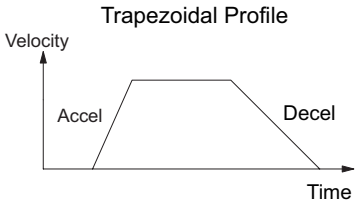
Possible causes:

1. Check the LED status indicator for X2 to make sure it is active when you want a reset. Also verify the configuration register V7636 for X2 is set to 7. Or, if you are using an internal reset, use the status mode of ***Direct***SOFT to monitor the reset input to the counter.

Mode 30: Pulse Output

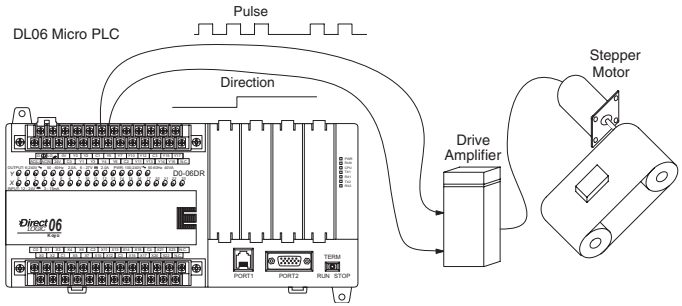
Purpose

The HSIO circuit in Mode 30 generates output pulse trains suitable for open-loop control of a single-axis motion positioning system. It generates pulse (stepper increment) and direction signals which you can connect to motor drive systems and perform various types of motion control. Using Mode 30 Pulse Output, you can select from three profile types detailed later in this appendix:

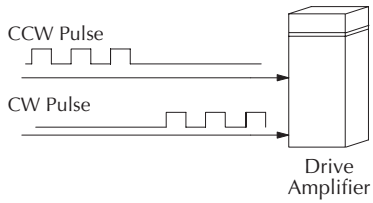


- **Automatic Trapezoidal** – Accel Slope to Target Velocity to Decel Slope
- **Step Trapezoidal** – User defined step acceleration/deceleration and target velocity
- **Velocity Control** – Speed and Direction only

The HSIO circuit becomes a high-speed pulse generator (up to 10 kHz) in Mode 30. By programming acceleration and deceleration values, position and velocity target values, the HSIO function automatically calculates the entire motion profile. The figure below shows the DL06 generating pulse and direction signals to the drive amplifier of a stepper positioning system. The pulses accomplish the profile independently and without interruption to ladder program execution in the CPU.



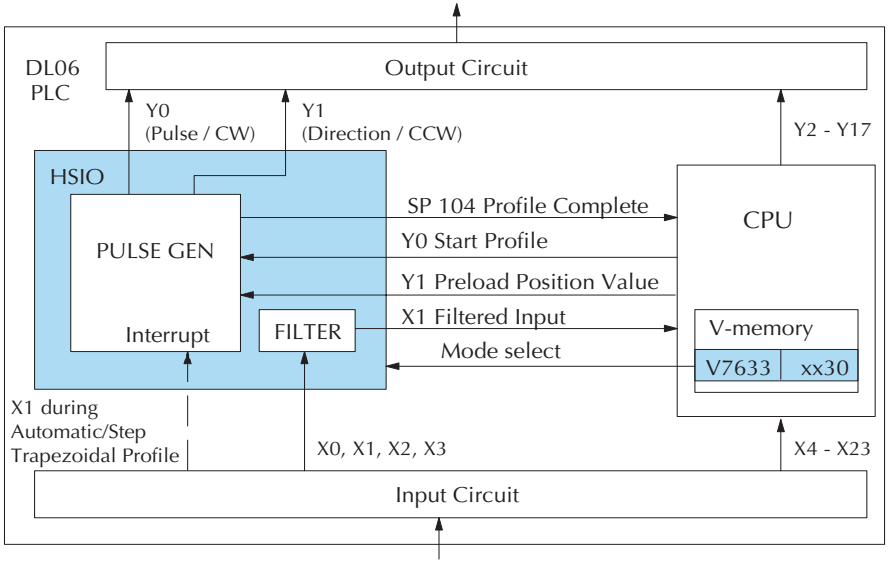
In the figure above, the DL06 generates pulse and direction signals. Each pulse represents the smallest increment of motion to the positioning system (such as one step or micro-step to a stepper system). Alternatively, the HSIO Pulse Output Mode may be configured to deliver counter clock-wise (CCW) and clock-wise (CW) pulse signals as shown to the right.



NOTE: *The pulse output is designed for open loop stepper motor systems. This, plus its minimum velocity of 40 pps, make it unsuitable for servo motor control.*

Functional Block Diagram

The diagram below shows HSIO functionality in Mode 30. When the lower byte of HSIO Mode register V7633 contains a BCD "30", the pulse output capability in the HSIO circuit is enabled. The pulse outputs use Y0 and Y1 terminals on the output connector. Remember that the outputs can only be DC type to operate.



IMPORTANT NOTE: *In Pulse Output Mode, Y0 and Y1 references are redefined or are used differently in two ways. Physical references refer to terminal screws, while logical references refer to I/O references in the ladder program. Please read the items below to understand this very crucial point.*

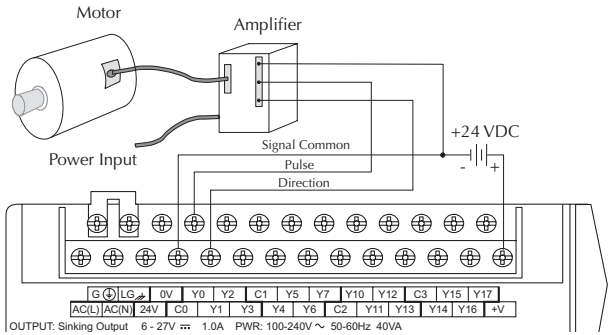
Notice the I/O point assignment and usage in the above diagram:

- X0, X1, X2 and X3 can be filtered inputs or pulse inputs in Pulse Output Mode, and they are available as input contacts to the ladder program.
- X1 behaves as an external interrupt to the pulse generator for automatic/step trapezoidal profiles. In other profile modes, it can be used as a filtered input or pulse input just like X0 (registration mode configuration shown above).
- References "Y0" and "Y1" are used in two different ways. At the discrete output connector, Y0 and Y1 terminals deliver the pulses to the motion system. The ladder program uses logical references Y0 and Y1 to initiate "Start Profile" and "Load Position Value" HSIO functions in Mode 30.

Hopefully, the above discussion will explain why some I/O reference names have dual meanings in Pulse Output Mode. **Please read the remainder of this section with care,** to avoid confusion about which actual I/O function is being discussed.

Wiring Diagram

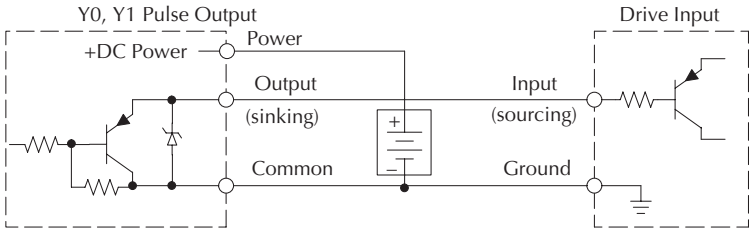
The generalized wiring diagram below shows pulse outputs Y0 and Y1 connected to the drive amplifier inputs of a motion control system.



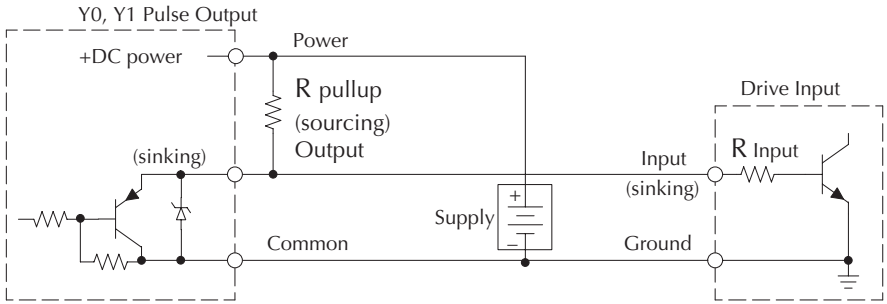
NOTE: *Do not use this drawing to wire your device. This is a general example and is not specific to any PLC model, stepper or encoder. Always refer to the device documentation for proper wiring connections.*

Interfacing to Drive Inputs

The pulse signals from Y0 and Y1 outputs will typically go to drive input circuits as shown above. It will be helpful to locate equivalent circuit schematics of the drive amplifier. The following diagram shows how to interface to a sourcing drive input circuit.



The following circuit shows how to interface to a sinking drive input using a pullup resistor. Please refer to Appendix 2 to learn how to calculate and install R pullup.



Motion Profile Specifications

The motion control profiles generated in Pulse Output Mode have the following specifications:

Motion Control Profile Specifications	
Parameter	**Specification**
Profiles	Automatic Trapezoidal – Accel Slope / Target Velocity / Decel Slope
	Step Trapezoidal - Step Acceleration / Deceleration
	Velocity Control – Speed and Direction only
Position Range	–8388608 to 8388607
Positioning	Absolute / relative command
Velocity Range	40 Hz to 10 kHz
V-memory registers	V3630 to V3652 (Profile Parameter Table)
Current Position	CT174 and CT175 (V1174 and V1175)

Physical I/O Configuration

The configurable discrete I/O options for Pulse Output Mode are listed in the table below. The CPU uses SP 104 contact to sense "profile complete". V7632 is used to select pulse/direction or CW/CCW modes for the pulse outputs. Input X1 is dedicated as the external interrupt for use in registration mode.

Physical I/O Configuration			
Input	**Configuration Register**	**Function**	**Hex Code Required**
–	V7632	Y0 = Pulse Y1 = Direction	0103
		Y0 = CW Pulse Y1 = CCW Pulse	0003 (default)
X0	V7634	pulse input	0005
		filtered input	xx06, xx = filter time, 0-9 (BCD) (default)
X1	V7635	pulse input	0005
		filtered input	xx06, xx = filter time, 0-99 (BCD) (default)
X2	V7636	pulse input	0005
		filtered input	xx06, xx = filter time, 0-99 (BCD) (default)
X3	V7637	pulse input	0005
		filtered input	xx06, xx = filter time, 0-99 (BCD) (default)

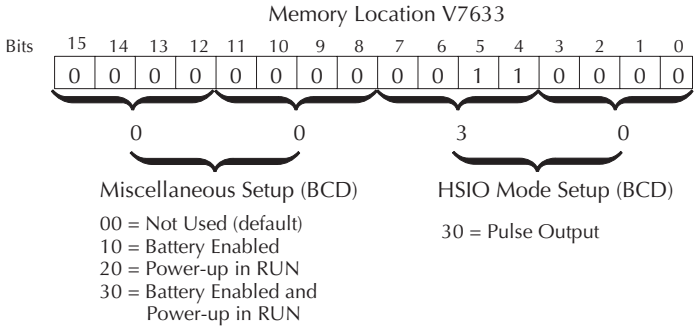
Logical I/O Functions

The following logical I/O references define functions that allow the HSI to communicate with the ladder program.

Logical I/O/ Functions	
Logical I/O	**Function**
SP104	Profile Complete – the HSIO turns on SP104 to the CPU when the profile completes, and it goes back off when Start Profile (Y0) turns on.
X1	External Interrupt - If the interrupt feature is selected for the Automatic Trapezoidal profile or the Step Trapezoidal Profile, the DL06 keeps outputting pulses until X1 turns on. After it is on, the unit outputs the pulses that are defined as the Target position.
Y0	Start Profile – the ladder program turns on Y0 to start motion. If turned off before the move completes, the pulses ramp down and motion stops. Turning it on again will start another profile, unless the current position equals the target position.
Y1	Preload Position Value – if motion is stopped and Start Profile is off, you can load a new value in CT174/CT175, and turn on Y1. At that transition, the value in CT174/ CT175 becomes the current position.

Setup for Mode 30

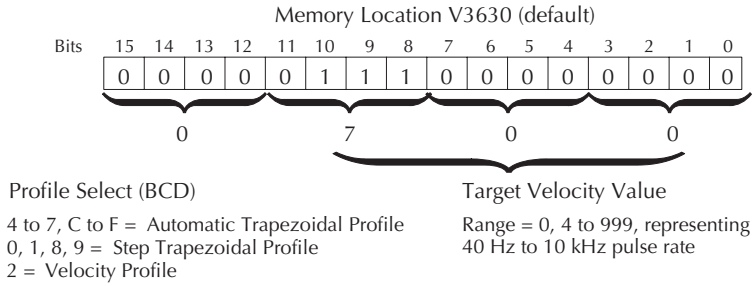
Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 30 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- ***Direct***SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.



Profile / Velocity Select Register

The first location in the Profile Parameter Table stores two key pieces of information. The upper four bits (12–15) select the type of profile required. The lower 12 bits (0-11) select the Target Velocity.

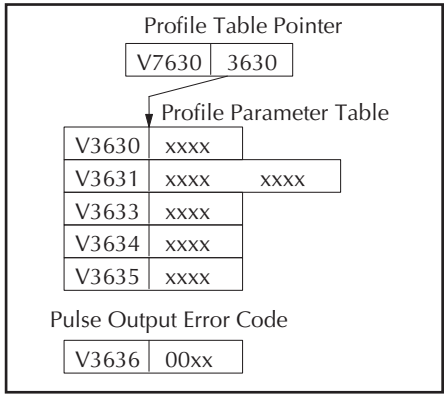
The ladder program must program this location before initiating any of the three profiles. The LD and OUT instruction will write all 16 bits, so be sure to fully specify the full four-digit BCD value for the Profile / Velocity Select Register each time.

The absolute and relative selection determines how the HSIO circuit will interpret your specified target position. Absolute position targets are referenced to zero. Relative position targets are referenced to the current position (previous target position). You may choose whichever reference method is most convenient for your application.

Profile Parameter Table

V7630 is a pointer location which points to the beginning of the Profile Parameter Table. The default starting location for the profile parameter table is V3630. However, you may change this by programming a different value in V7630. Remember to use the LDA (load address) instruction, converting octal into hex.

The HSIO uses the next V-memory register past the bottom of the profile parameter table to indicate profile errors. See the error table at the end of this section for error code definitions.



Automatic Trapezoidal Profile

V-Memory	Function	Range	Units
V3630, bits 12–15	Automatic Trapezoidal Profile without Ending Velocity (Ending Velocity is fixed to 0.)	4=absolute w/o interrupt 5=absolute with interrupt* C=relative w/o interrupt D=relative with interrupt*	–
	Automatic Trapezoidal Profile with Ending Velocity (Use V3637 to set up Ending Velocity.)	6=absolute w/o interrupt 7=absolute with interrupt* E=relative w/o interrupt F=relative with interrupt*	–
V3630, bits 0–11	Target Velocity	0, 4-999 where 0=1000	x 10 pps
V3631 / V3632	Target Position**	–8388608 to 8388607	Pulses
V3633	Starting Velocity	4 to 100	x 10 pps
V3634	Acceleration Time	1 to 100	x 100 mS
V3635	Deceleration Time	1 to 100	x 100 mS
V3636	Error Code	(see end of section)	–
V3637	Ending Velocity	4 to 100	x 10 pps

* If you select to use interrupt, the DL06 will not start looking for your target count until the interrupt X1 is on.

**To set a negative number, put 8 in the most significant digit. For example: -8388608 is 88388608 in V3631 and V3632.

Step Trapezoidal Profile

V-Memory	Function	Range	Units
V3630, bits 12–15	Step Trapezoidal Profile	0=absolute w/o interrupt 7=absolute with interrupt* 8=relative w/o interrupt 9=relative with interrupt*	–
V3630, bits 0–11	Target Velocity	0, 4-999 where 0=1000	x 10 pps
V3631 / V3632	Target Position**	–8388608 to 8388607	Pulses
V3633	Step 1 Acceleration	4 to 1000	x 10 pps
V3634	Step 1 Distance	1 to 9999	Pulses
V3635	Step 2 Acceleration	4 to 1000	x 10 pps
V3636	Step 2 Distance	1 to 9999	Pulses
V3637	Step 3 Acceleration	4 to 1000	x 10 pps
V3640	Step 3 Distance	1 to 9999	Pulses
V3641	Step 4 Acceleration	4 to 1000	x 10 pps
V3642	Step 4 Distance	1 to 9999	Pulses
V3643	Step 5 Deceleration	4 to 1000	x 10 pps
V3644	Step 5 Distance	1 to 9999	Pulses
V3645	Step 6 Deceleration	4 to 1000	x 10 pps
V3646	Step 6 Distance	1 to 9999	Pulses
V3647	Step 7 Deceleration	4 to 1000	x 10 pps
V3650	Step 7 Distance	1 to 9999	Pulses
V3651	Step 8 Deceleration	4 to 1000	x 10 pps
V3652	Step 8 Distance	1 to 9999	Pulses

* If you select to use interrupt, the DL06 will not start looking for your target count until the interrupt X1 is on.

**To set a negative number, put 8 in the most significant digit. For example: -8388608 is 88388608 in V3631 and V3632.

Velocity Control

V-Memory	Function	Range	Units
V3630	Velocity Profile	2000 only	–
V3631 / 3632	Direction Select	0=CW, 80000000=CCW,	Pulses
V3633	Velocity	4 to 1000	x 10 pps
V3636	Error Code	(see end of section)	–

Choosing the Profile Type

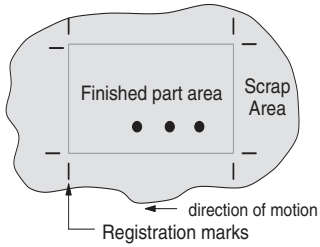
Pulse Output Mode generates three types of motion profiles. Most applications use one type for most moves. However, each move can be different if required.

- Automatic Trapezoidal – Accel Slope to Target Velocity to Decel Slope
- Step Trapezoidal – Velocity to Position Control on Interrupt
- Velocity Control – Speed and Direction only

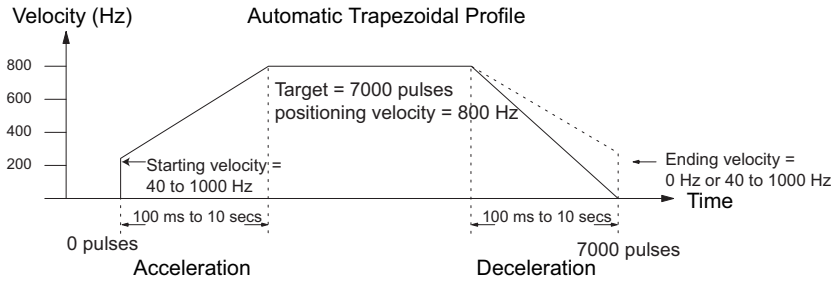
Automatic Trapezoidal Profile Defined

The automatic trapezoidal profile is the most common positioning profile. It moves the load to a pre-defined target position by creating a move profile. The acceleration slope is applied at the starting position. The deceleration slope is applied backwards from the target position. The remainder of the move in the middle is spent traveling at a defined velocity.

Registration profiles solve a class of motion control problems. In some applications, product material in work moves past a work tool such as a drill station. Shown to the right, registration marks on the scrap area of the work-piece allow a machine tool to register its position relative to the rectangle, to drill properly.



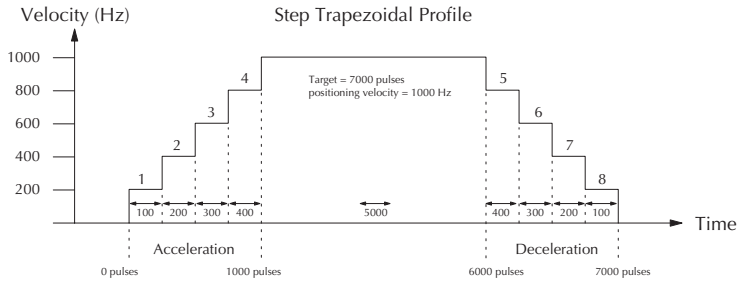
Home search moves allow open-loop motion systems to re-calibrate (preload) the current position value at powerup.



The user determines the starting velocity, the acceleration/deceleration times, and the total number of pulses. The CPU computes the profile from these inputs.

Step Trapezoidal Profiles Defined

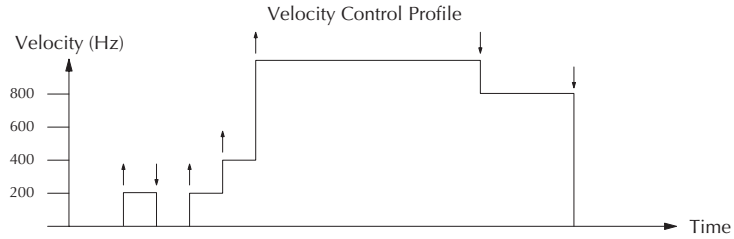
The step trapezoidal profile is a combination of velocity and position control modes. The move begins by accelerating to a programmed velocity. The velocity is sustained and the move is of indefinite duration. When an external interrupt signal occurs (due to registration sensing), the profile switches from velocity to position control. The move ends by continuing motion a pre-defined distance past the interrupt point (such as a drill hole location). The deceleration ramp is applied in advance of the target position.



Define steps 1 through 4 for gradual acceleration to the target velocity and define steps 5 through 8 for gradual deceleration from the target velocity. This type of profile is appropriate for applications involving large stepper motors and/or large inertia loads. It can, however, be used to provide gradual ramping in applications involving smaller motors and loads.

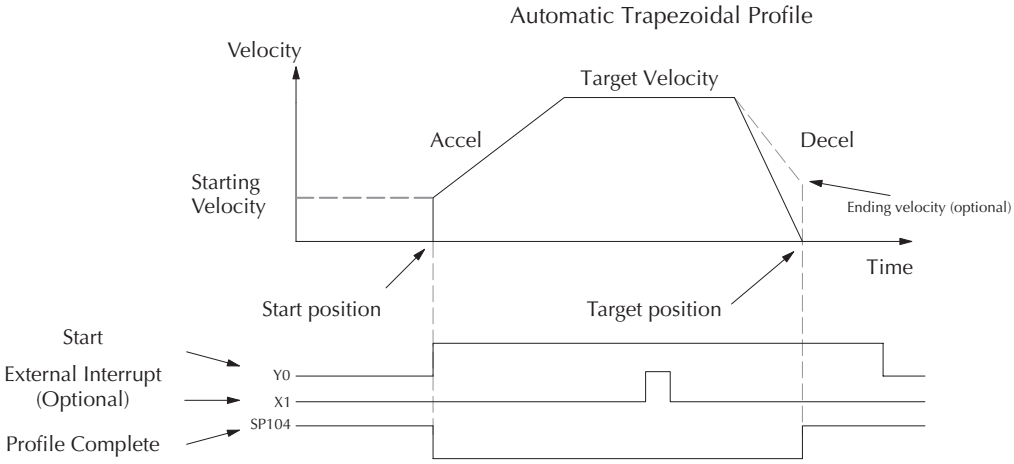
Velocity Control Defined

The Velocity Control defines only the direction and speed of motion. There is no target position specified, so the move can be of indefinite length. Only the first velocity value needs to be defined. The remaining velocity values can be created while motion is in progress. Arrows in the profile shown indicate velocity changes.



Automatic Trapezoidal Profile Operation

Starting velocities must be within the range of 40 pps to 1k pps. The remainder of the profile parameters are in the profile parameter table.



The time line of signal traces below the profile indicates the order of events. The HSIO uses logical output Y0 as the Start input to the HSIO, which starts the profile. Immediately the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the progress of the move. Typically, a ladder program will monitor this bit so it knows when to initiate the next profile move.

You can also use the external interrupt (X1). Once the external interrupt feature is selected for the profile, the DL06 keeps outputting the pulses until X1 turns on. Then, the DL06 outputs the pulses defined as the target position.

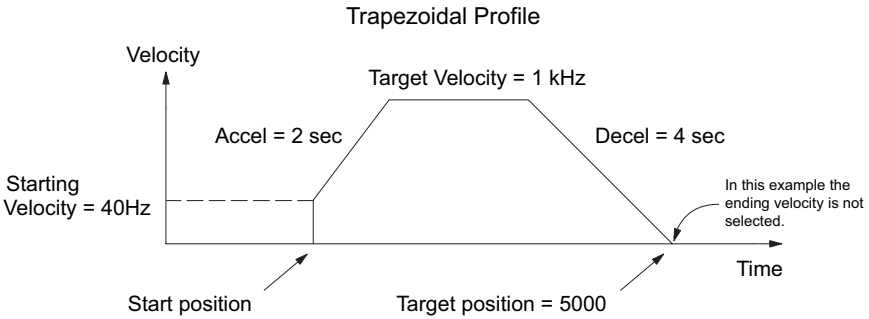
If you are familiar with motion control, you'll notice that we do not have to specify the direction of the move. The HSIO function examines the target position relative to the current position, and automatically outputs the correct direction information to the motor drive.

Notice that the motion accelerates immediately to the starting velocity. This segment is useful in stepper systems so we can jump past low speed areas when low-torque problems or a resonant point in the motor might cause a stall. (When a stepper motor stalls, we have lost the position of the load in open-loop positioning systems). However, it is preferable not to make the starting velocity too large, because the stepper motor will also *slip* some pulses due to the inertia of the system. You can also set up the ending velocity for the same reason.

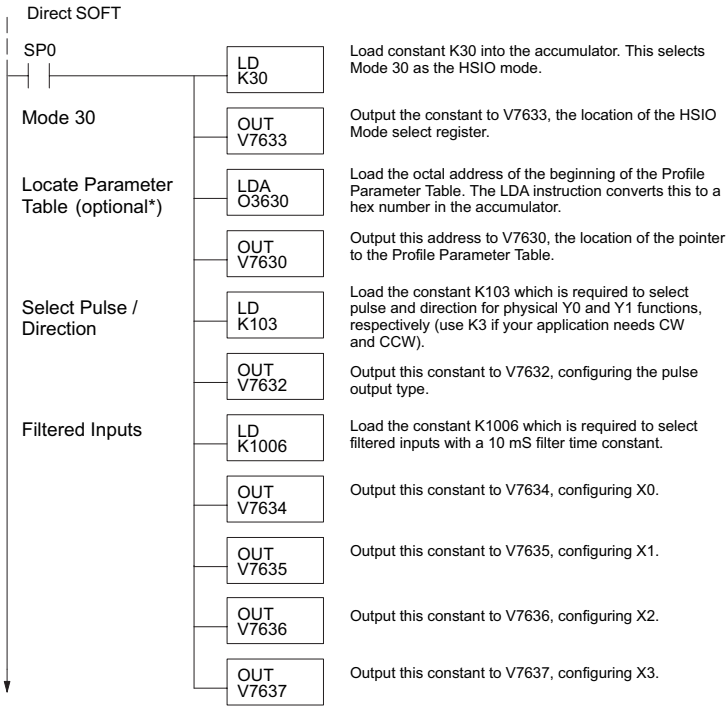
When you need to change the current position value, use logical Y1 output coil to load a new value into the HSIO counter. If the ladder program loads a new value in CT174/CT175 (V1174/V1175), then energizing Y1 will copy that value into the HSIO circuit counter. This must occur before the profile begins, because the HSIO ignores Y1 during motion.

Program Example 1: Automatic Trapezoidal Profile without External Interrupt

The Automatic Trapezoidal Profile we want to perform is drawn and labeled in the following figure. It consists of a non-zero starting velocity, and moderate target velocity.



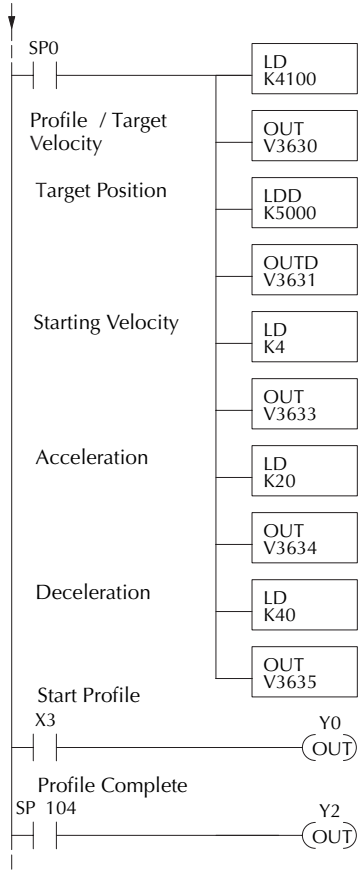
The following program will realize the profile drawn above, when executed. The beginning of the program contains all the necessary setup parameters for Pulse Output Mode 30. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.



* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

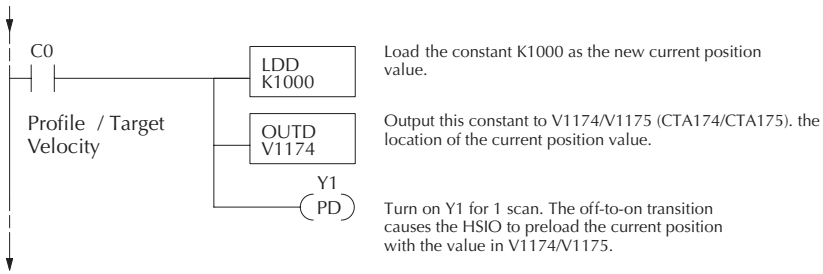
Continued on next page.

Continued from previous page.



Preload Position Value

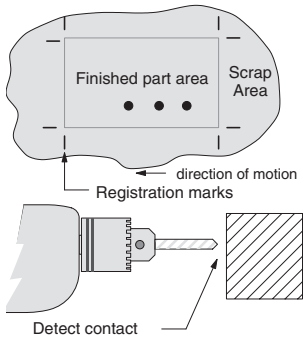
At any time you can write (preload) a new position into the current position value. This is often done after a home search (see the registration example programs).



Program Example 2: Automatic Trapezoidal Profile with External Interrupt

Registration Applications:

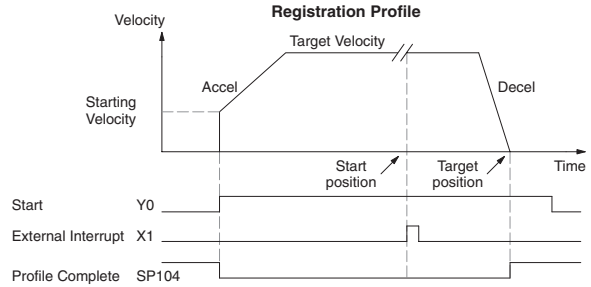
1. In a typical application shown to the right, product material in work moves past a work tool such as a drill. Registration marks on the scrap area of the work-piece allow a machine tool to register its position relative to the rectangle, to drill properly.



2. In other examples of registration, the work piece is stationary and the tool moves. A drill bit may approach the surface of a part in work, preparing to drill a hole of precise depth. However, the drill bit length gradually decreases due to tool wear. A method to overcome this is to detect the moment the drill makes contact with the surface of the part each time a part is drilled. The bit can then drill a constant depth after making contact with the part's surface.

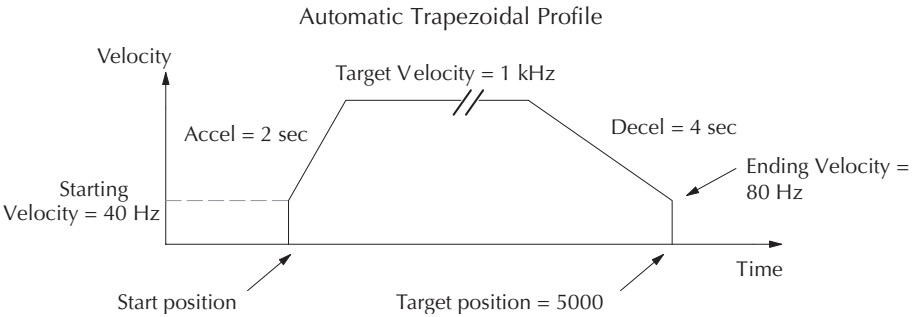
3. The home search move allows a motion system to calibrate its position on startup. In this case, the positioning system makes an indefinite move and waits for the load to pass by a home limit switch. This creates an interrupt at the moment when the load is in a known position. We then stop motion and preload the position value with a number which equates to the physical "home position".

When an interrupt pulse occurs on physical input X1, the starting position is declared to be the present count (current load position). The velocity control switches to position control, moving the load to the target position. Note that the minimum starting velocity is 40 pps. This instantaneous velocity accommodates stepper motors that can stall at low speeds.

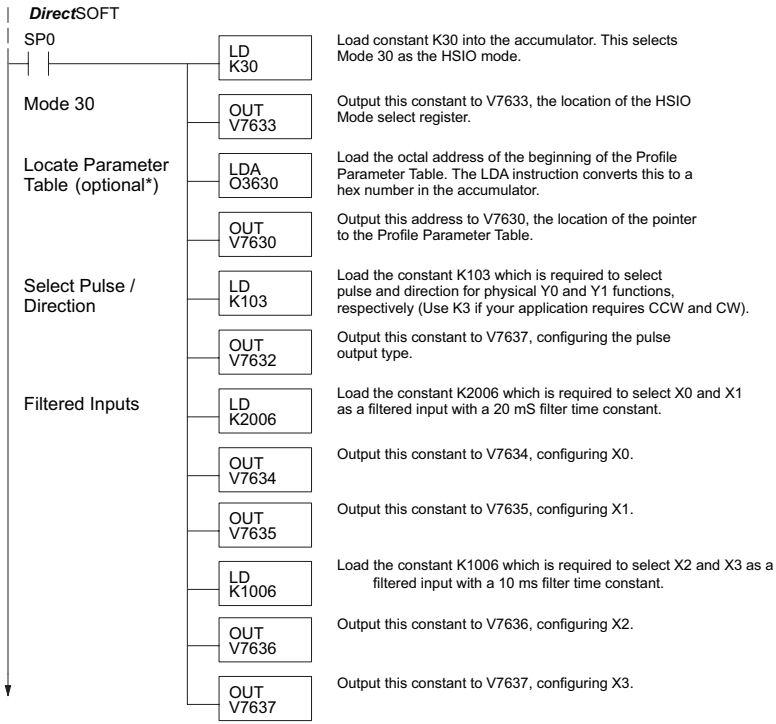


The time line of signal traces below the profile indicates the order of events. The CPU uses logical output Y0 to start the profile. Immediately the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the move's completion by sensing the signal's on state.

The Automatic Trapezoidal profile we want to perform is drawn and labeled in the following figure. It consists of a non-zero starting velocity, and moderate target velocity.

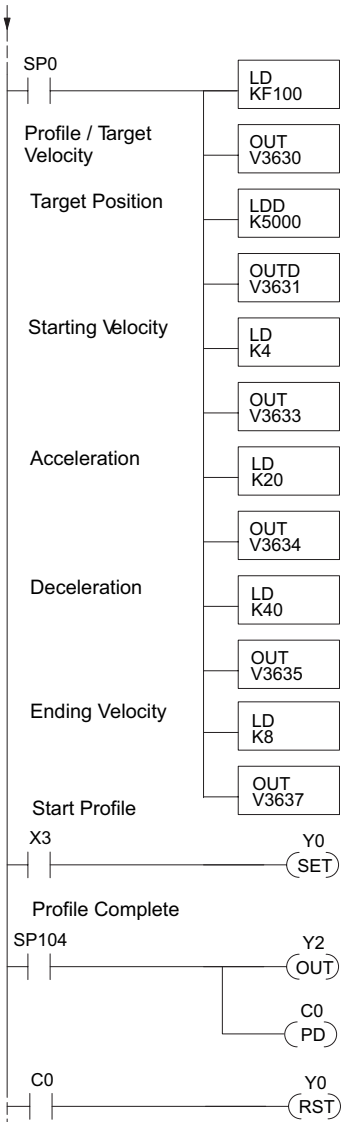


The following program will realize the profile drawn above, when executed. The first program rung contains all the necessary setup parameters. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.



* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

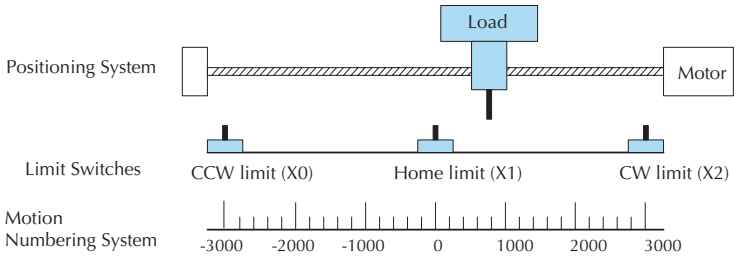
Continued from previous page



The profile will begin when the start input (X3) is given. Then the motion begins an indefinite move, which lasts until an external interrupt on X1 occurs. Then the motion continues on for 5000 more pulses before stopping.

Program Example 3: Automatic Trapezoidal Profile with Home Search

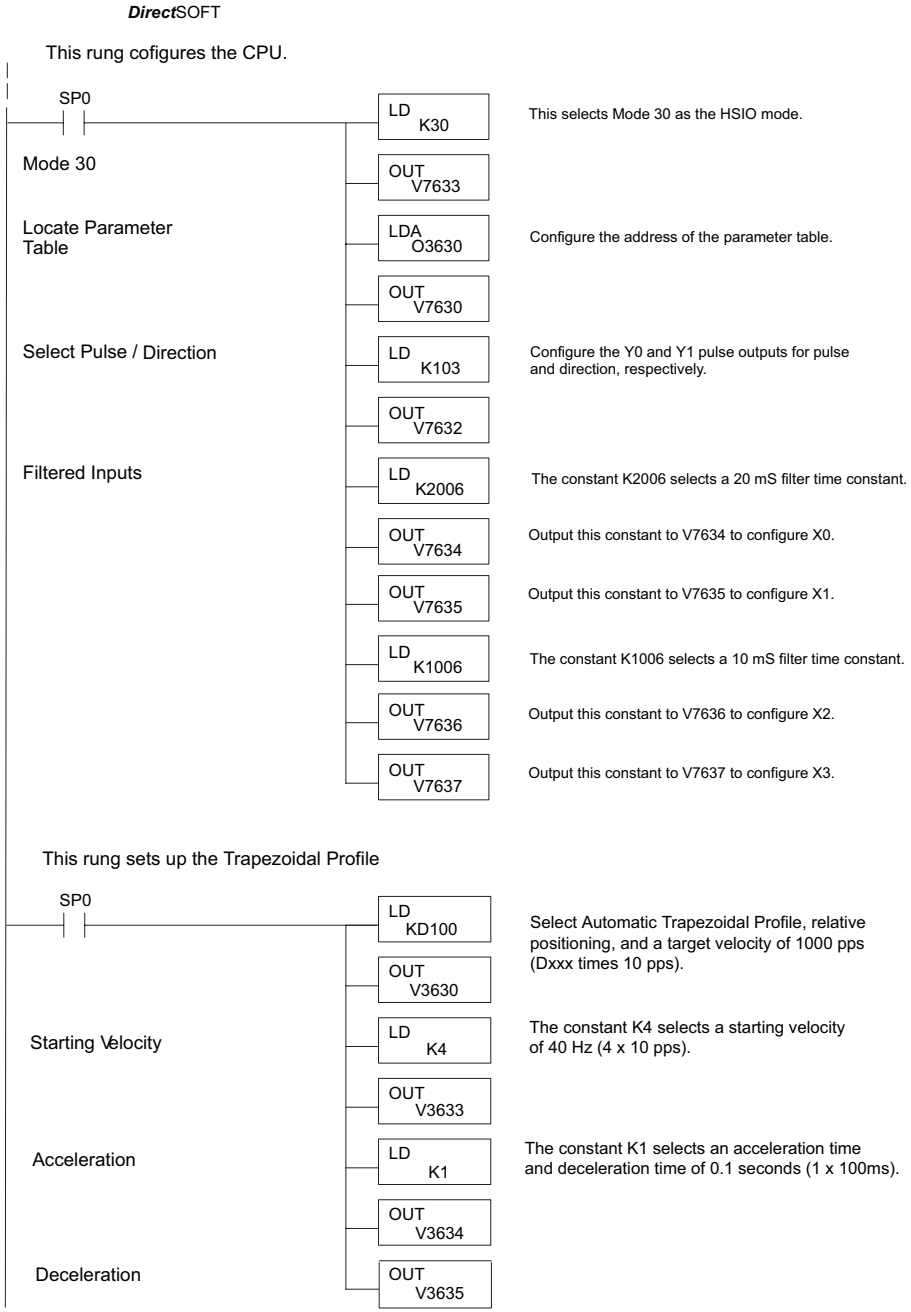
One of the more challenging aspects of motion control is the establishment of actual position at powerup. This is especially true for open-loop systems which do not have a position feedback device. However, a simple limit switch located at an exact location on the positioning mechanism can provide "position feedback" at one point. For most stepper control systems, this method is a good and economical solution.



In the drawing above, the load moves left or right depending on the CW/CCW direction of motor rotation. The PLC ladder program senses the CW and CCW limit switches to stop the motor, before the load moves out-of-bounds and damages the machine. The home limit switch is used at powerup to establish the actual position. The numbering system is arbitrary, depending on a machine's engineering units.

At powerup, we do not know whether the load is located to the left or to the right of the home limit switch. Therefore, we will initiate a home search profile, using the registration mode. The home limit switch is wired to X1, causing the interrupt. The example, beginning on the next page, preferentially starts in one direction only (CW), and pulses until it reaches the first Overtravel Limit (CW Limit). It will ignore the Home Switch if it passes it. This means that Homing is always accomplished from the same direction for better consistency.

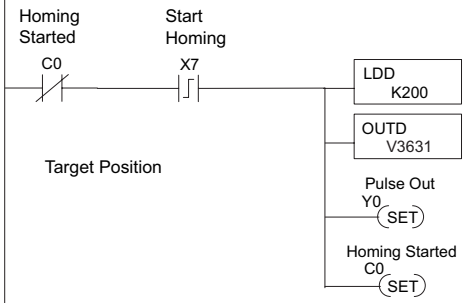
The CPU will then reverse the direction of pulses (loads 80000200 into V3631) and travel away from the CW Limit until it hits the Home Switch, then travels past it and reverse slowly back to the Home Switch. A value of 0 will be loaded to V1174 (CTA174) to represent the Home position.



Continued on next page

Continued from previous page

"Start Homing" is assigned as X7. This will set Y0, which starts the Pulse Output. Pulses will continue until the CCW Limit is reached.

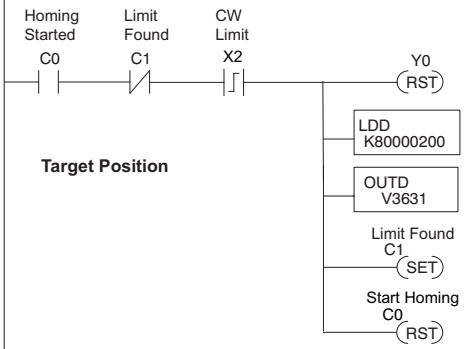


This constant K200 selects the target position 200 pulses (CW direction).

Turn on Pulse Out, searching for the CW Limit.

Set C0 to indicate the CW Limit search has begun.

Once the CW Limit is found, the DL06 will stop the Pulse Output, load a negative value, thus reversing direction. It does this by LDD K80000200 into V3631. The "8" in the left-most position of the value loaded (8xxxxxxx) will cause Y1 to turn on. This is how the PLC reverses direction.



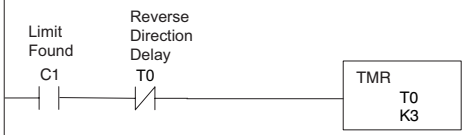
Turn off Start Profile.

The constant -K200 selects a target position of -200 pulses, which is in the CCW direction.

The CW limit has been reached.

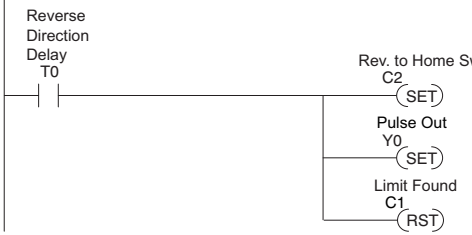
CW Limit was found, reset C0.

This rung will activate timer, T0, for a short 0.3 second delay. When T0 times out, the next rung is activated.



Add a timer to create a short delay before reversing the motor.

The Pulse Output is activated again, but in the reverse direction.



Set C2 to indicate the CCW home search has begun.

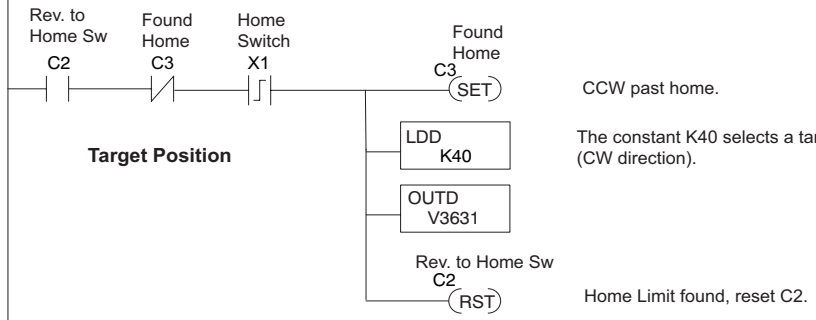
Turn on Pulse Out again.

Reset C1.

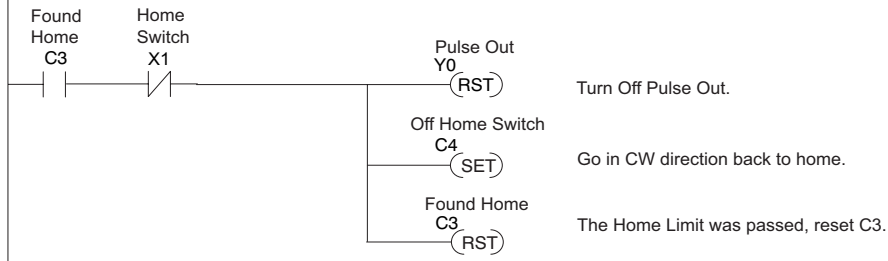
Continued from previous page

Continue
o
page

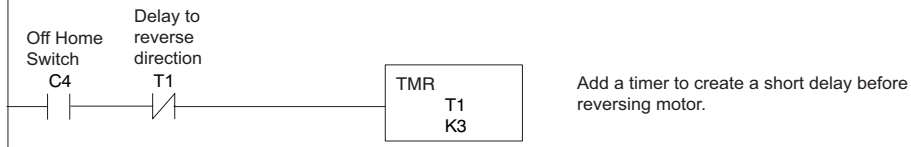
This rung waits until the Home Switch, X1, is triggered, then it prepares the next step by loading a new, slower, speed into V3631.
Notice that the value loaded is not in the form of 8xxxxxxx.



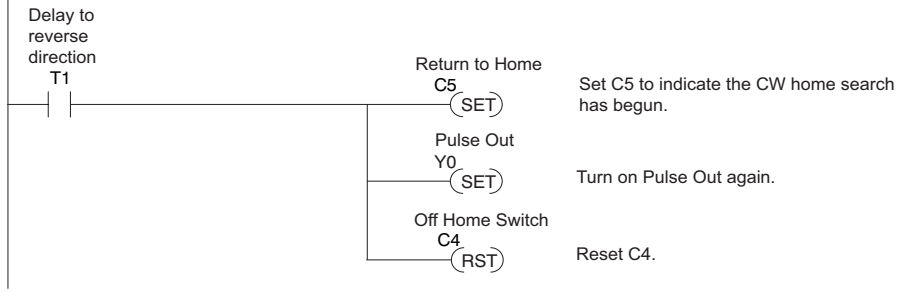
Once the Home Switch, X1, is deactivated, the Pulse Output is stopped.



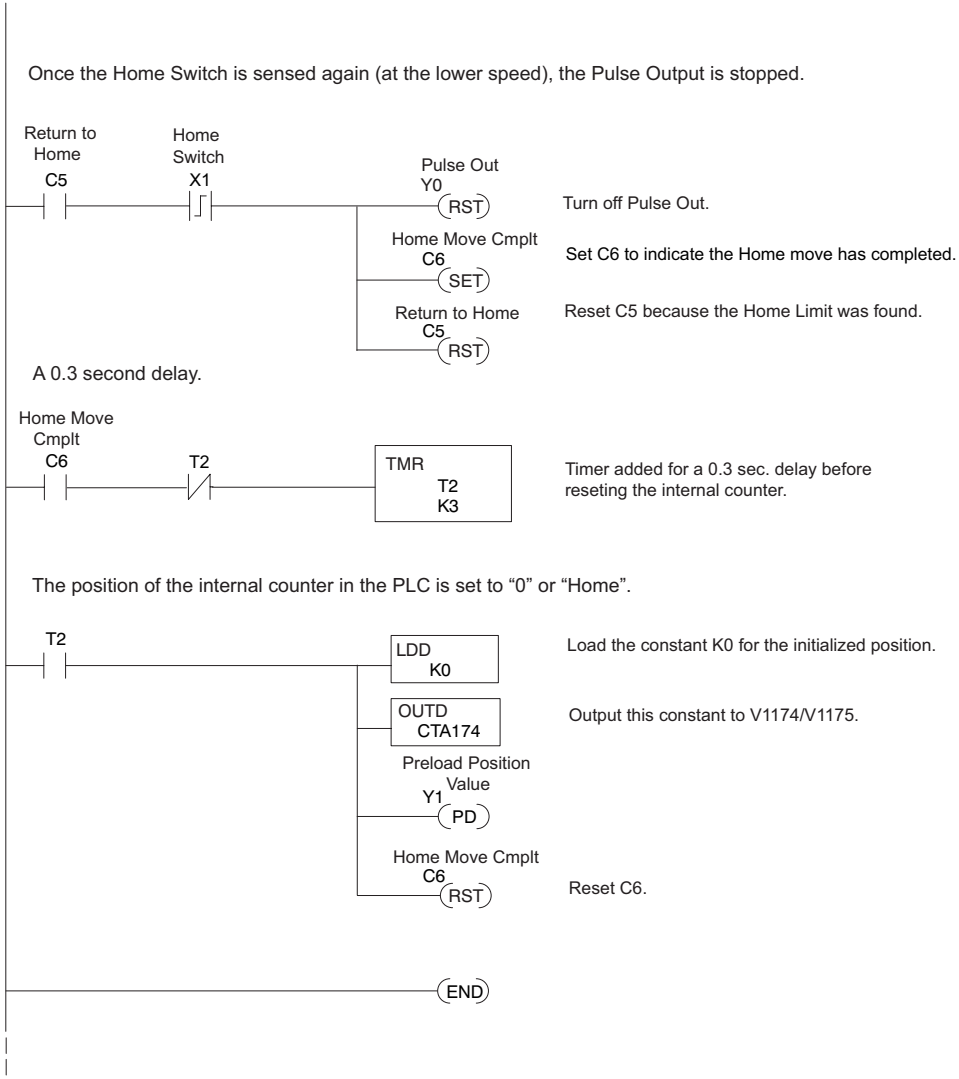
This rung adds a short delay of 0.3 seconds to delay the reversing of the motor.



The Pulse Output is now activated again, but in the direction the motor was initially started.



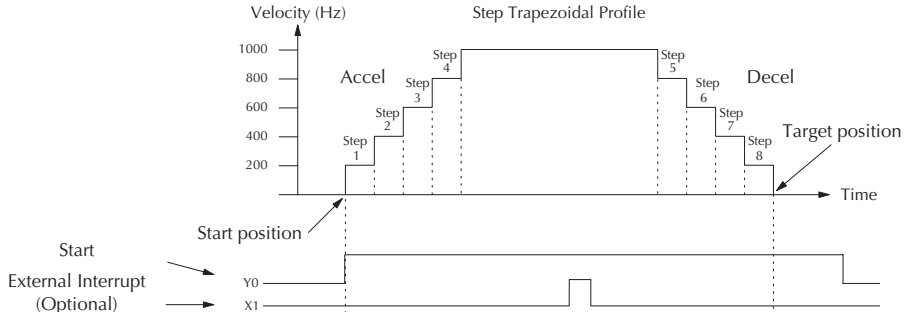
Continued from previous page



The home search profile will execute specific parts of the program, based on the order of detection of the limit switches. Ladder logic sets C0 to initiate a home search in the CW direction. If the CW limit is encountered, the program searches for home in the CCW direction, passes it slightly, and does the final CW search for home. After reaching home, the last ladder rung preloads the current position to “0”.

Step Trapezoidal Profile Operation

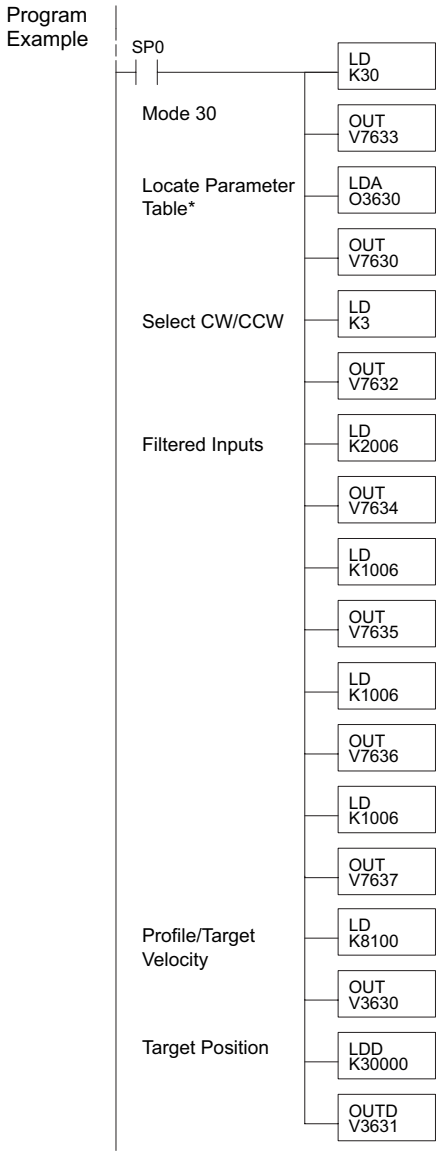
With this step trapezoidal profile, you can control the acceleration and deceleration slopes as you want.



The time line of signal traces below the profile indicates the order of events. The HSIO uses logical output Y0 as the start input to the HSIO, which starts the profile. Immediately, the HSIO turns off the Profile Complete signal (SP104), so the ladder program can monitor the progress of the move. Typically, a ladder program will monitor this bit so it knows when to initiate the next profile move. You can also use the external interrupt (X1). Once the external interrupt feature selected for the profile, the DL06 keeps outputting the pulses until X1 turns on. Then the DL06 outputs the pulses defined as the target position.

Each acceleration and deceleration slope consists of 4 steps. You can set up the velocity and the distance (number of pulses) of each step. You don't need to use all 4 steps of each slope. For instance, if you want to use only 2 steps, just set zero to the velocity and the distance of the 3rd and 4th step. If the acceleration slope and the deceleration slope are identical, you can just put zero into all the velocity and the distance parameters for the deceleration slope.

Program Example 4: Step Trapezoidal Profile

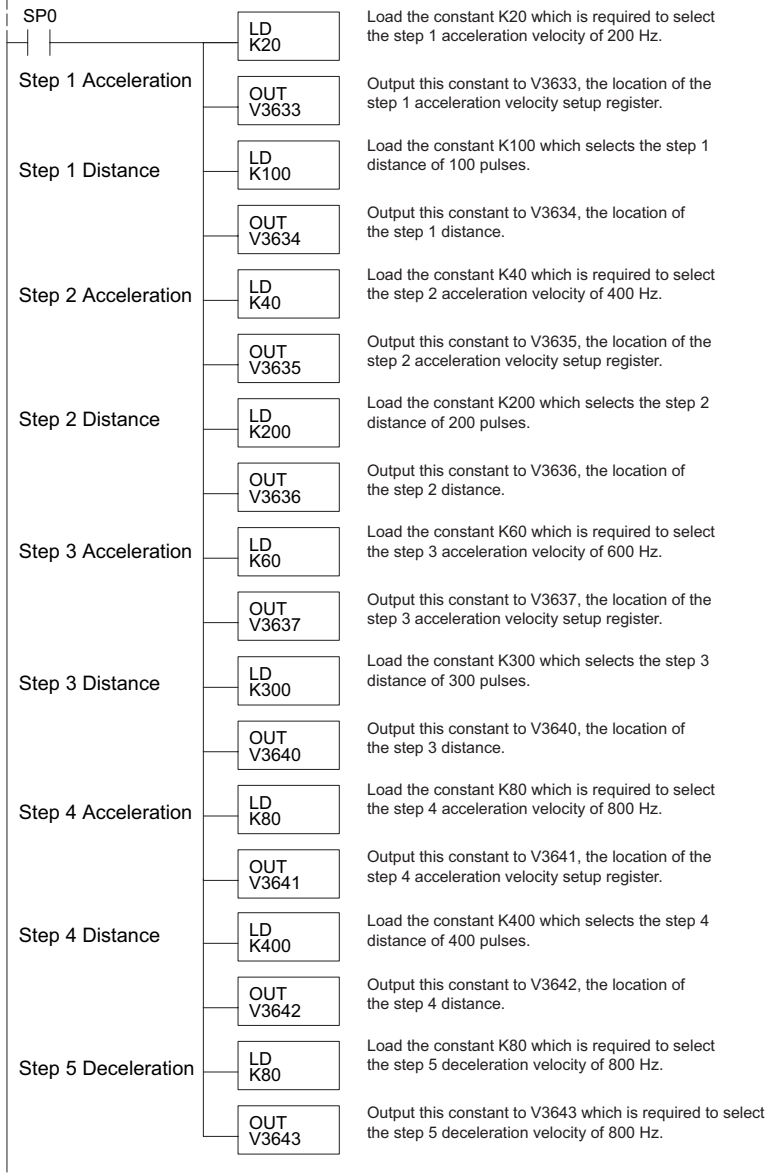


* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

Continued on next page

Continued from previous page

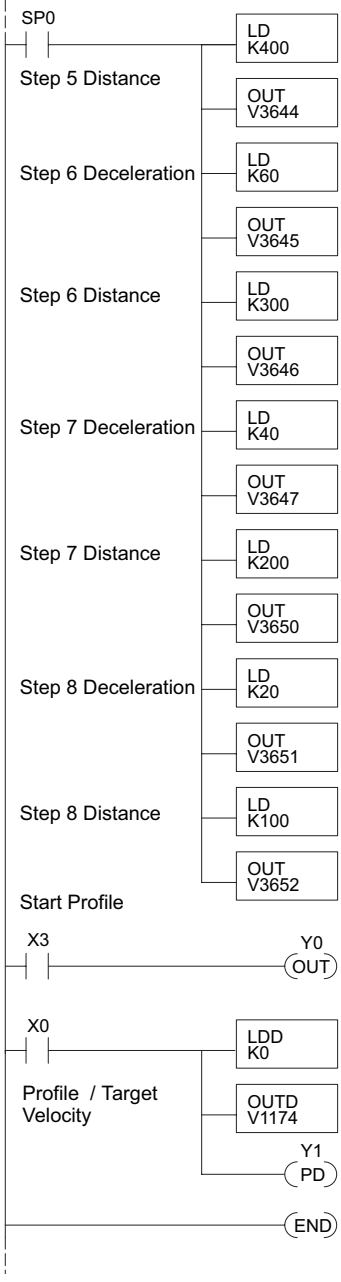
Program Example



Continued on next page

Continued from previous page

Program Example



Load the constant K400 which selects the step 5 distance of 400 pulses.

Output this constant to V3644, the location of the step 5 distance.

Load the constant K60 which is required to select the step 6 deceleration velocity of 600 Hz.

Output this constant to V3645, which is required to select the step 6 deceleration velocity of 600 Hz.

Load the constant K300 which selects the step 6 distance of 300 pulses.

Output this constant to V3646, the location of the step 6 distance.

Load the constant K40 which is required to select the step 6 deceleration velocity of 400 Hz.

Output this constant to V3647, which is required to select the step 6 deceleration velocity of 400 Hz.

Load the constant K200 which selects the step 7 distance of 200 pulses.

Output this constant to V3650, the location of the step 7 distance.

Load the constant K20 which is required to select the step 8 deceleration velocity of 200 Hz.

Output this constant to V3651, which is required to select the step 8 deceleration velocity of 200 Hz.

Load the constant K100 which selects the step 8 distance of 100 pulses.

Output this constant to V3652, the location of the step 8 distance.

We use a spare filtered input to allow the operator to start the profile. When the operator turns X3 ON, logical output Y0 starts the profile. The operator must maintain X3 on until the profile is complete.

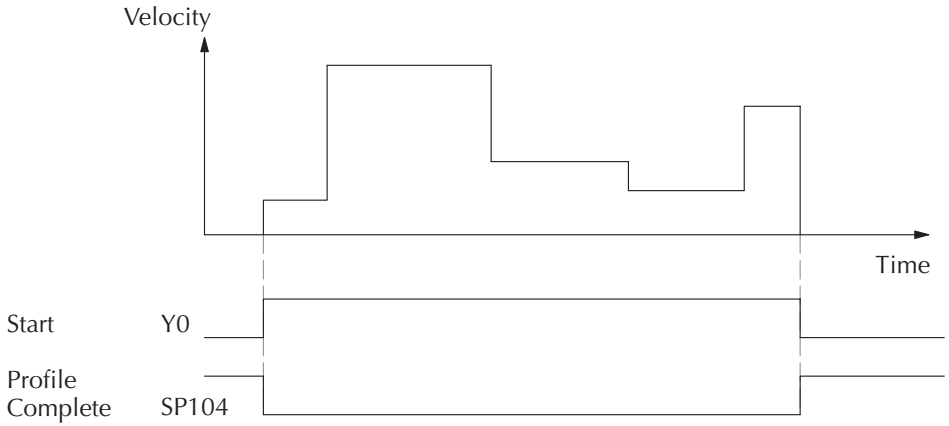
Load value.

Output this constant to V1174/V1175 (CT174/CT175), the location of the current position value.

Turn on Y1 for 1 scan. The of-to-on transition causes the HSIO to preload the current position with the value in V1174/V1175.

Velocity Profile Operation

The velocity profile is best suited for applications which involve motion but do not require moves to specific points. Conveyor speed control is a typical example.



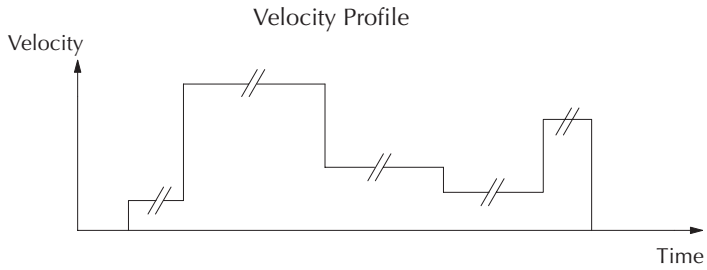
The time line of signal traces below the profile indicates the order of events. Assuming the velocity is set greater than zero, motion begins when the Start input (Y0) energizes. Since there is no end position target, the profile is considered in progress as long as the Start input remains active. The profile complete logical input to ladder logic (X0) correlates directly to the Start input status when velocity profiles are in use.

While the Start input is active, the ladder program can command a velocity change by writing a new value to the velocity register (V3633 by default). The full speed range of 40 Hz to 10 kHz is available. Notice from the drawing that there are no acceleration or deceleration ramps between velocity updates. This is how velocity profiling works with the HSIO. However, the ladder program can command more gradual velocity changes by incrementing or decrementing the velocity value more slowly. A counter or timer can be useful in creating your own acceleration/deceleration ramps. Unless the load must do a very complex move, it is easier to let the HSIO function generate the accel/decel ramps by selecting the trapezoidal or registration profiles, instead.

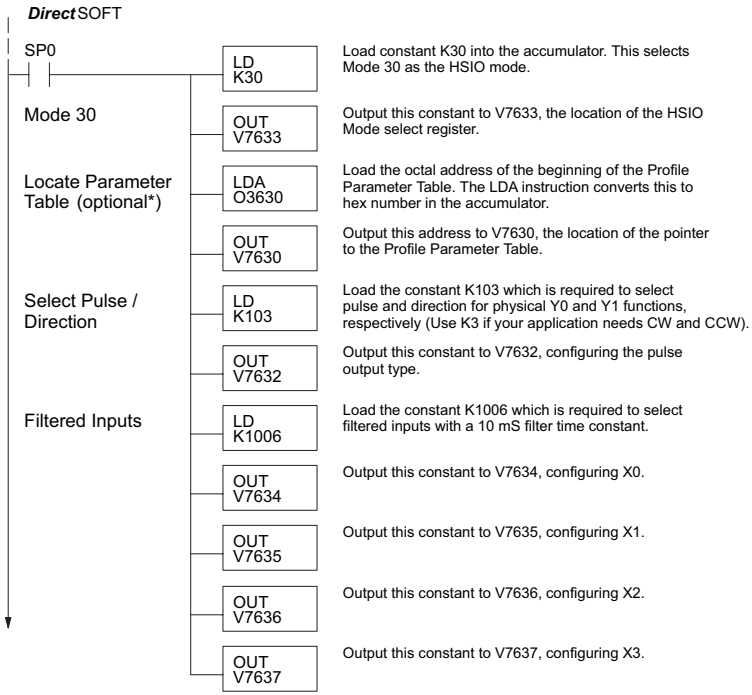
Unlike the trapezoidal and registration profiles, you must specify the desired direction of travel with velocity profiles. Load the direction select register (V3631/V3632 by default) with 8000 0000 hex for CCW direction, or 0 for CW direction.

Program Example 5: Velocity Profile

The velocity profile we want to perform is drawn and labeled in the following figure. Each velocity segment is of indefinite length. The velocity only changes when ladder logic (or other device writing to V-memory) updates the velocity parameter.

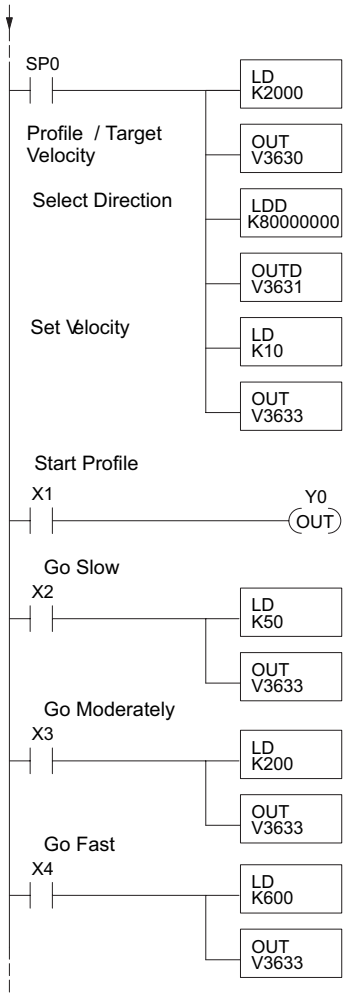


The following program uses dedicated discrete inputs to load in new velocity values. This program is fun to try, because you can create an infinite variety of profiles with just two or three input switches. The intent is to turn on only one of X2, X3, or X4 at a time. The beginning of the program contains all the necessary setup parameters for Pulse Output Mode 30. We only have to do this once in the program, so we use first-scan contact SP0 to trigger the setup.



* If the Locate Parameter Table is not used you must remove both the LDA function and the OUT function below it.

Program Example Cont'd



Load the constant K2000 which is required to select Velocity Profile. This data word contains no velocity information in the case of velocity mode.

Output this constant to V3630, the location of the Profile Select setup register.

Load the constant K80000000 which selects CCW direction for Velocity Profiles. Don't forget to use double word size (8-digit BCD position value).

Output this constant to V3631 and V3632, the location of the Target Position double-word register.

Load the constant K10 which is required to select an initial velocity of 100 pps (uses x10 multiplier).

Output this constant to V3633, the location of the velocity parameter register. After the program is running, we can write here again, using discrete input switches.

We use a spare filtered input to allow the operator to start the profile. When the operator turns X1 ON, logical output Y0 starts the profile. The operator must maintain X1 on until the profile is complete.

Load the constant K50 which is required to select a velocity of 500 pps when the operator closes X2.

Output this constant to V3633, the location of the velocity parameter register. The speed will change immediately.

Load the constant K200 which is required to select a velocity of 2000 pps when the operator closes X3.

Output this constant to V3633, the location of the velocity parameter register. The speed will change immediately.

Load the constant K600 which is required to select a velocity of 6000 pps when the operator closes X4.

Output this constant to V3633, the location of the velocity parameter register. The speed will change immediately.

Automatic Trapezoidal Profile Error Codes

The Profile Parameter Table starting at V3630 (default location) defines the profile. Certain numbers will result in an error when the HSIO attempts to use the parameters to execute a move profile. When an error occurs, the HSIO writes an error code in V3636.

Most errors can be corrected by rechecking the Profile Parameter Table values. The error is automatically cleared at powerup and at Program-to-Run Mode transitions.

Error Code	Error Description
0000	No error
0010	Requested profile type code is invalid (must use 4 to 6 or C to F)
0011	Interrupt is selected for absolute mode
0020	Target Velocity is not in BCD
0021	Target Velocity is specified to be less than 40 pps
0022	Target Velocity is specified to be greater than 10,000 pps
0030	Target Position value is not in BCD
0031	Target Position value is zero
0032	Direction Select is not 0 or 80000000.
0040	Starting Velocity is not in BCD
0041	Starting Velocity is specified to be less than 40 pps
0042	Starting Velocity is specified to be greater than 10,000 pps
0050	Acceleration Time is not in BCD
0051	Acceleration Time is zero
0052	Acceleration Time is greater than 10 seconds
0060	Deceleration Time is not in BCD
0061	Deceleration Time is zero
0062	Deceleration Time is greater than 10 seconds
0070	Ending Velocity is not BCD
0071	Ending Velocity is specified to be less than 40 pps
0072	Ending Velocity is specified to be greater than 1,000 pps
0073	Ending Velocity is specified to be greater than Target Velocity

Troubleshooting Guide for Mode 30

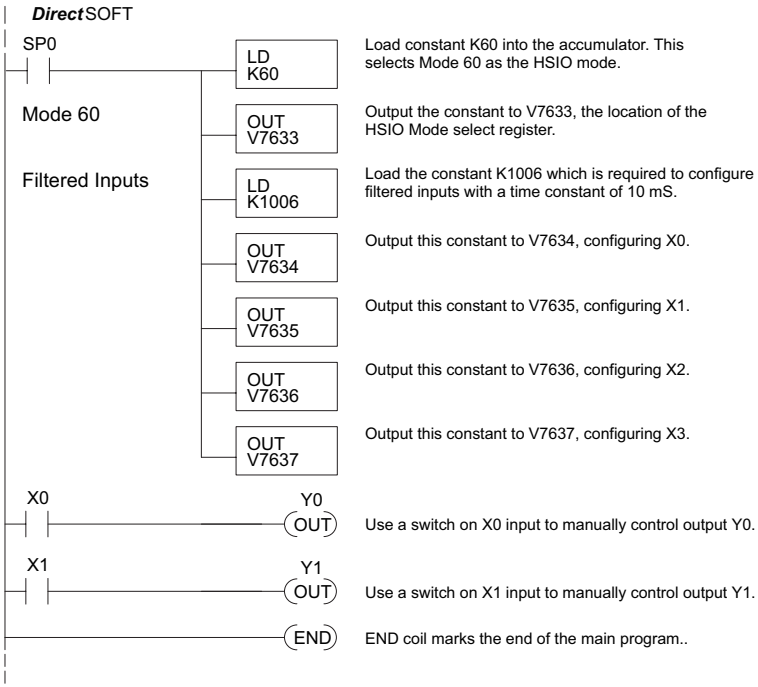
If you're having trouble with Mode 30 operation, please study the following symptoms and possible causes. The most common problems are listed below.

Symptom: The stepper motor does not rotate.

Possible causes:

1. **Configuration –** Verify that the HSIO actually generates pulses on outputs Y0 and Y1. Watch the status LEDs for Y0 and Y1 when you start a motion profile. If the LEDs flicker on and off or are steadily on, the configuration is probably correct.
2. **Programming error –** If there are no pulses on Y0 or Y1 you may have a programming error. Check the contents of V3636 for an error code that may be generated when the PLC attempts to do the move profile. Error code descriptions are given above.
3. **Check target value –** The profile will not pulse if the count value is equal to the target value (ex. count =0, target=0).

4. **Wiring** – Verify the wiring to the stepper motor is correct. Remember the signal ground connection from the PLC to the motion system is required.
5. **Motion system** – Verify that the drive is powered and enabled. To verify the motion system is working, you can use Mode 60 operation (normal PLC inputs/outputs) as shown in the test program below. With it, you can manually control Y0 and Y1 with X0 and X1, respectively. Using an input simulator is ideal for this type of manual debugging. With the switches you can single-step the motor in either direction. If the motor will not move with this simple control, Mode 30 operation will not be possible until the problem with the motor drive system or wiring is corrected.



6. **Memory Error** – HSIO configuration parameters are stored in the CPU system memory. Corrupted data in this memory area can sometimes interfere with proper HSIO operation. If all other corrective actions fail, initializing the scratchpad memory may solve the problem. With *Direct*SOFT, select **PLC > Setup > Initialize Scratch Pad** from the Menu bar.

Symptom: The motor turns in the wrong direction.

Possible causes:

1. **Wiring** – If you have selected CW and CCW type operation, just swap the wires on Y0 and Y1 outputs.
2. **Direction control** – If you have selected Pulse and Direction type operation, just change the direction bit to the opposite state.

Mode 40: High-Speed Interrupts

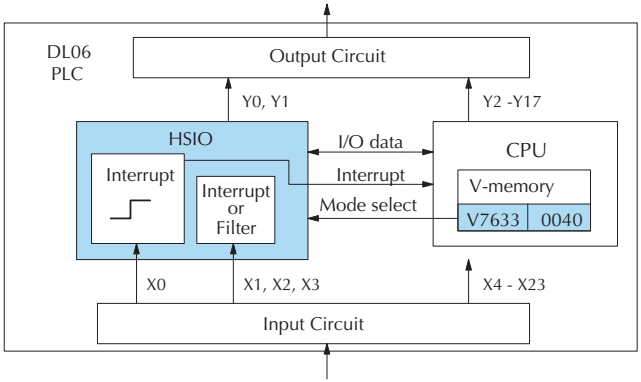
Purpose

The HSIO Mode 40 provides a high-speed interrupt to the ladder program. This capability is provided for your choice of the following application scenarios:

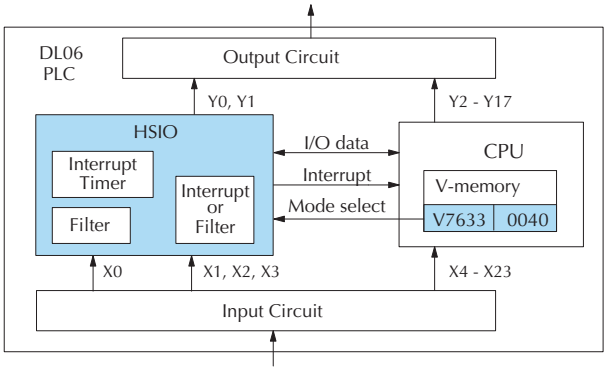
- External events need to trigger an interrupt subroutine in the CPU. Using immediate I/O instructions in the subroutine is typical.
- An interrupt routine needs to occur on a timed basis which is different from the CPU scan time (either faster or slower). The timed interrupt is programmable, from 5 to 999 mS.

Functional Block Diagram

The HSIO circuit creates the high-speed interrupt to the CPU. The following diagram shows the external interrupt option, which uses X0. In this configuration X1, X2 and X3 are external interrupts or normal filtered inputs.

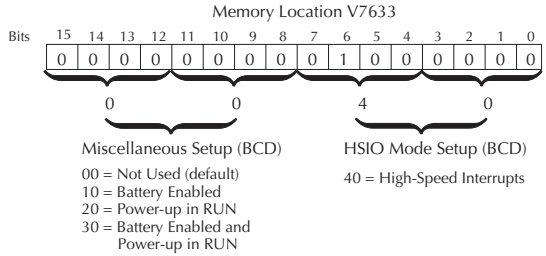


Alternately, you may configure the HSIO circuit to generate interrupts based on a timer, as shown below. In this configuration, inputs X0 is a filtered input.



Setup for Mode 40

Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 40 in the lower byte of V7633 to select high-speed interrupts.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- *Direct*SOFT's memory editor
- Use the Handheld Programmer D2–HPP

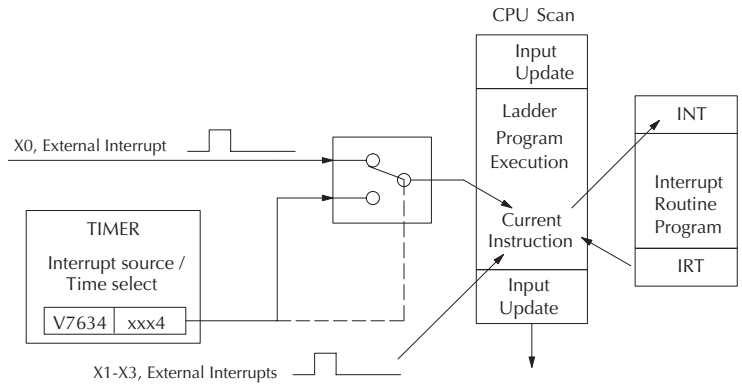
We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

Interrupts and the Ladder Program

Refer to the drawing below. The source of the interrupt may be external (X0 - X3). An internal timer can be used instead of X0 as the interrupt source. The setup parameter in V7634 serves a dual purpose:

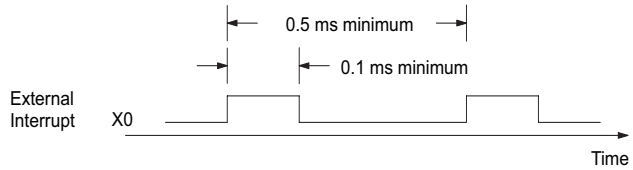
- It selects between the two interrupt sources (external or internal timer). The timed interrupt can only be used with X0.
- In the case of the timer interrupt, it programs the interrupt timebase between 5 and 999 mS.

The resulting interrupt uses label INT 0, 1, 2 or 3 in the ladder program. Be sure to include the Enable Interrupt (ENI) instruction at the beginning of your program. Otherwise, the interrupt routine will not be executed.



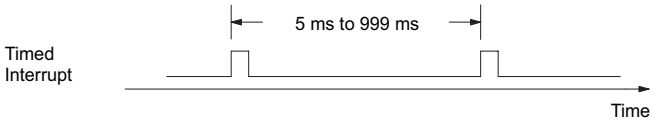
External Interrupt Timing Parameters

External interrupt signals must meet certain timing criteria to guarantee an interrupt will result. Refer to the timing diagram below. The minimum pulse width is 0.1 ms. There must be some delay before the next interrupt pulse arrives, such that the interrupt period cannot be smaller than 0.5 ms.



Timed Interrupt Parameters

When the timed interrupt is selected, the HSIO generates the interrupt to ladder logic. There is no interrupt pulse width in this case, but the interrupt period can be adjusted from 5 to 999 ms.



X Input / Timed INT Configuration

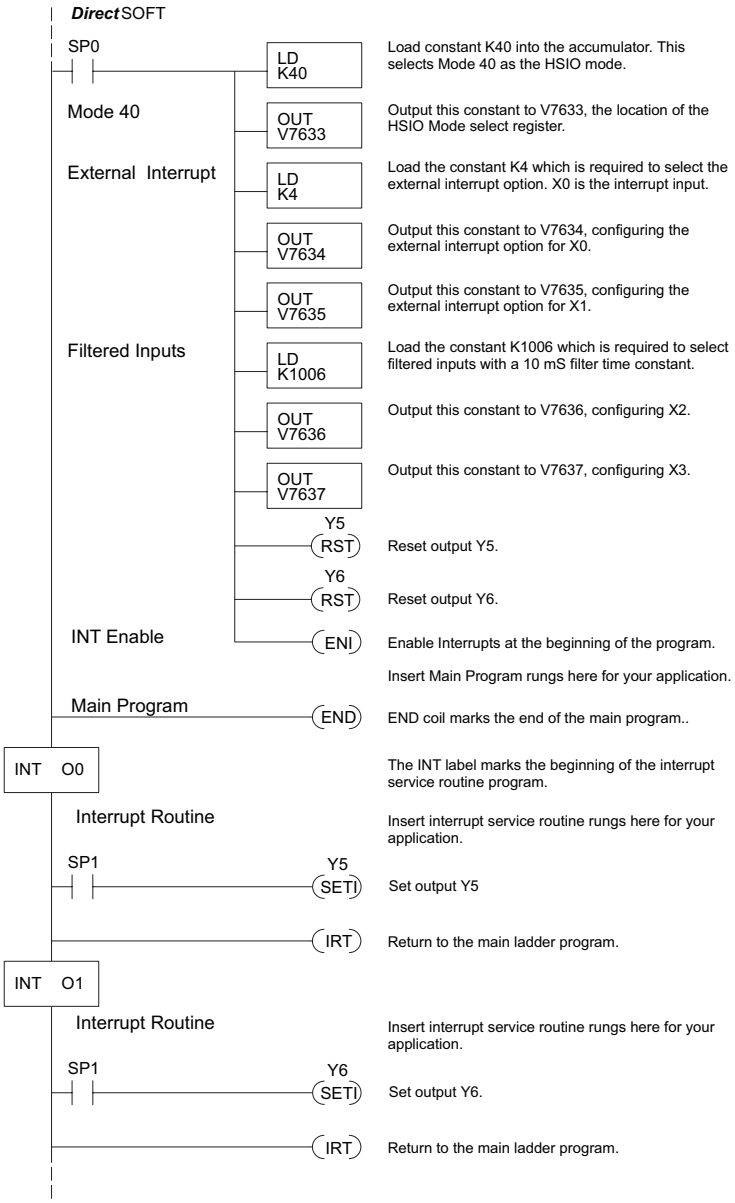
The configurable discrete input options for High-Speed Interrupt Mode are listed in the table below. Input X0 is the external interrupt when "0004" is in V7634. If you need a timed interrupt instead, then V7634 contains the interrupt time period, and input X0 becomes a filtered input (uses X1's filter time constant by default). Inputs X0, X1, X2, and X3, can be filtered inputs, having individual configuration registers and filter time constants, interrupt inputs or counter inputs.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	External Interrupt	0004 (default)
		Timed Interrupt	xxx4, xxx = INT timebase 5 - 999 ms (BCD)
X1	V7635	Interrupt	0004 (default)
		Pulse Input	0005
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X2	V7636	Interrupt	0004 (default)
		Pulse Input	0005
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X3	V7637	Interrupt	0004 (default)
		Pulse Input	0005
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)

If you are only using *one* of the points for an interrupt, you may want to select a different *main* mode (i.e. 10, 20, 30, 50, or 60); and then, just configure one of the terminals not taken as an interrupt. For example, you might want to configure your CPU for the UP counter mode (Mode 10) and use point 03 for a high speed interrupt. You should read the individual sections for any alternate mode you might choose. There you will find instructions on how to select a high speed interrupt as a secondary feature.

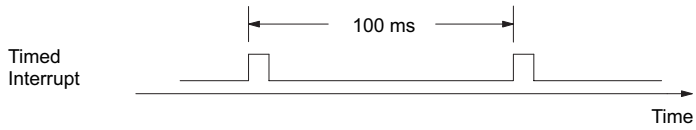
Program Example 1: External Interrupt

The following program selects Mode 40, then selects the external interrupt option for inputs X0 and X1. Inputs X2 and X3 are configured as filtered inputs with a 10 ms time constant. The program is otherwise generic, and may be adapted to your application.

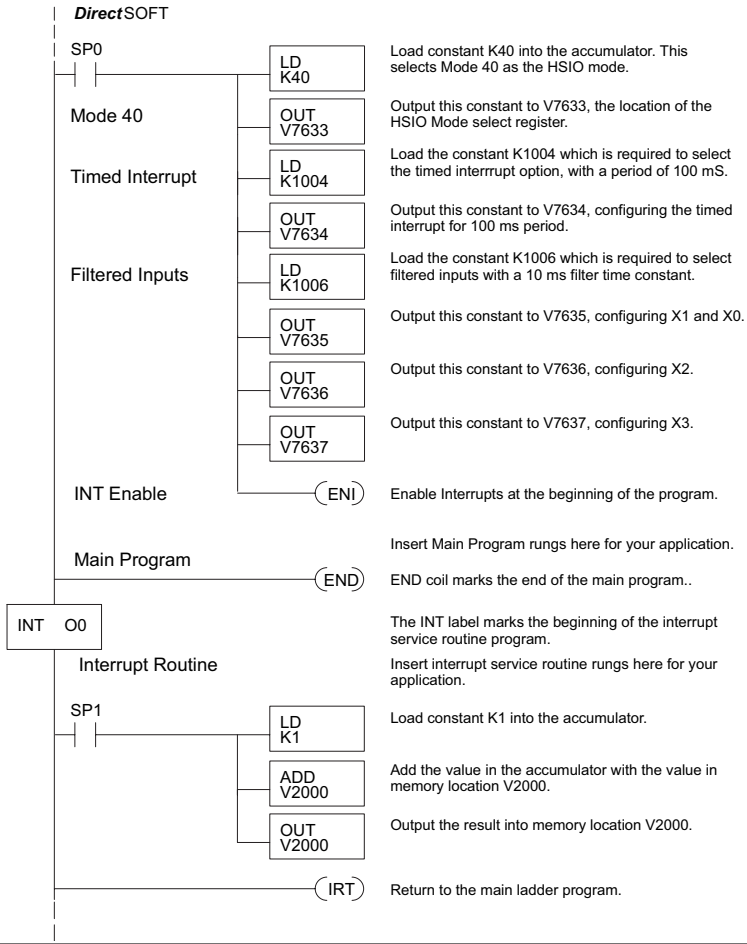


Program Example 2: Timed Interrupt

The following program selects Mode 40, then selects the timed interrupt option, with an interrupt period of 100 ms.



Inputs X0, X1, X2, and X3, are configured as filtered inputs with a 10 ms time constant. Note that X0 uses the time constant from X1. The program is otherwise generic, and may be adapted to your application.



NOTE: *X0 Cannot be used in the main program logic; however, using X0 to set C10, for instance, will allow the use of C10 in the main program logic. Do not forget to reset C10.*

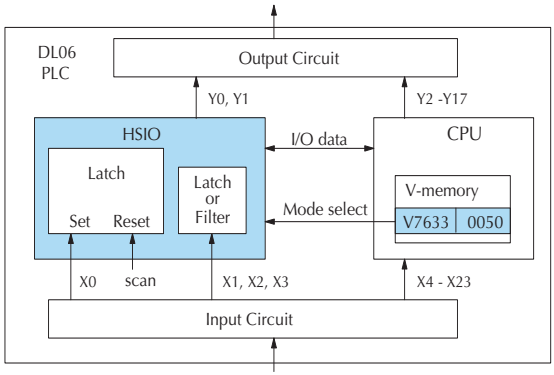
Mode 50: Pulse Catch Input

Purpose

The HSIO circuit has a pulse-catch mode of operation. It monitors the signal on inputs X0 - X3, preserving the occurrence of a narrow pulse. The purpose of the pulse catch mode is to enable the ladder program to *see* an input pulse which is shorter in duration than the current scan time. The HSIO circuit latches the input event on input X0 - X3 for one scan. This contact automatically goes off after one scan.

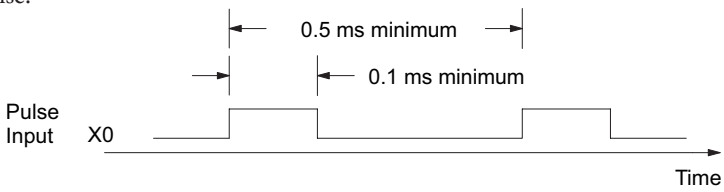
Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD "50", the pulse catch mode in the HSIO circuit is enabled. X0 - X3 automatically become the pulse catch inputs, which set the latch on each rising edge. The HSIO resets the latch at the end of the next CPU scan. Inputs X1 X2, and X3 can be filtered discrete inputs, also.



Pulse Catch Timing Parameters

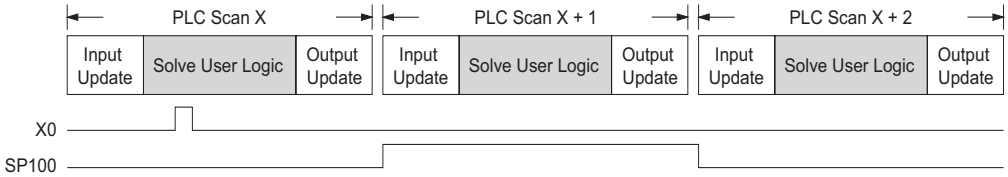
Signal pulses at X0 - X3 must meet certain timing criteria to guarantee a pulse capture will result. Refer to the timing diagram below. The input characteristics of X0 are fixed (it is not a programmable filtered input). The minimum pulse width is 0.1 ms. There must be some delay before the next pulse arrives, such that the pulse period cannot be smaller than 0.5 ms. If the pulse period is smaller than 0.5 ms, the next pulse will be considered part of the current pulse.



NOTE: *The pulse catch and filtered input functions are opposite in nature. The pulse catch feature seeks to capture narrow pulses, while the filter input feature seeks to reject narrow pulses.*

When to use Pulse Catch Mode

Use the pulse catch mode for applications where the input (e.g. X0) can not be used in the user program because the pulse width is very narrow. Use SP100 instead of X0. The SP100 contact stays on through the next scan, as shown above. If X0 is on for more than scan, SP100 will remain on until X0 transitions off.

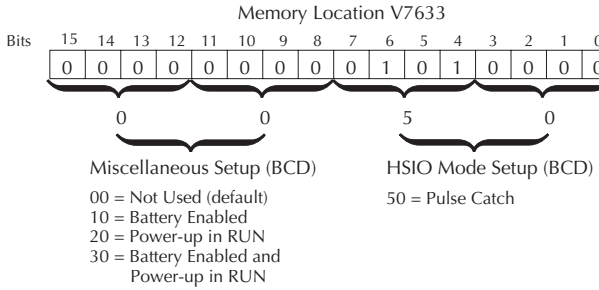


The status relay for X0 is SP100. The other status relays are shown in the table below.

Input	Status Relay
X0	SP100
X1	SP101
X2	SP102
X3	SP103

Setup for Mode 50

Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 50 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- ***Direct***SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

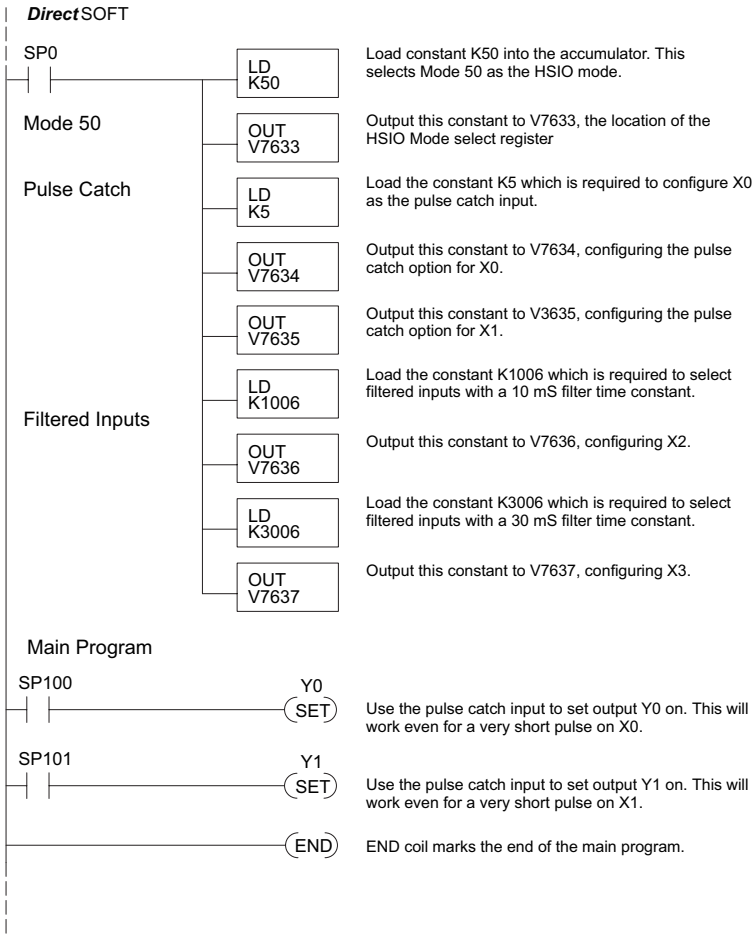
X Input Configuration

The configurable discrete input options for Pulse Catch Mode are listed in the table below. Each input has its own configuration register and filter time constant.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Pulse Catch Input	0005 (default)
X1	V7635	Interrupt	0004
		Pulse Catch Input	0005 (default)
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X2	V7636	Interrupt	0004
		Pulse Catch Input	0005 (default)
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)
X3	V7637	Interrupt	0004
		Pulse Catch Input	0005 (default)
		Filtered Input	xx06 (xx = filter time) 0 - 99 ms (BCD)

Program Example 1: Pulse Catch

The following program selects Mode 50, then programs the pulse catch code for X0 and X1. Inputs X2, and X3 are configured as filtered inputs with 10 and 30 mS time constants respectively. The program is otherwise generic, and may be adapted to your application.



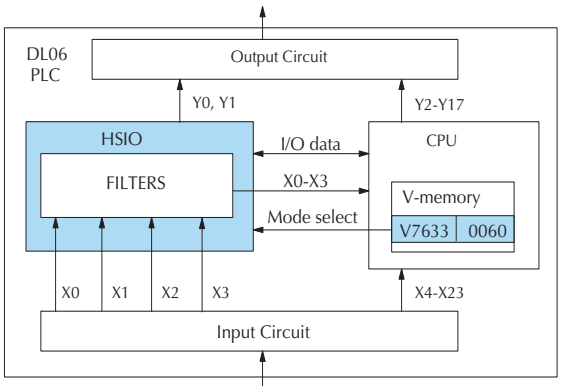
Mode 60: Discrete Inputs with Filter

Purpose

The last mode we will discuss for the HSIO circuit is Mode 60, Discrete Inputs with Filter. The purpose of this mode is to allow the input circuit to reject narrow pulses and accept wide ones, as viewed from the ladder program. This is useful in especially noisy environments or other applications where pulse width is important. In all other modes in this appendix, X0 to X3 usually support the mode functions as special inputs. Only spare inputs operate as filtered inputs by default. Now in Mode 60, all four inputs X0 through X3 function only as discrete filtered inputs.

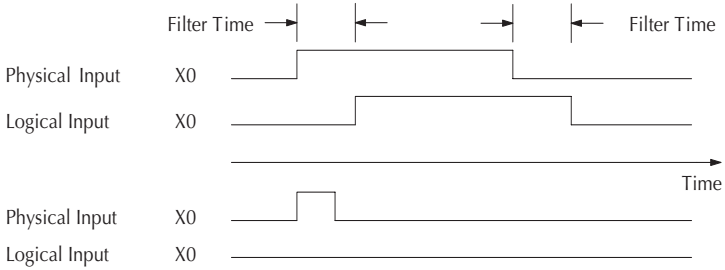
Functional Block Diagram

Refer to the block diagram below. When the lower byte of HSIO Mode register V7633 contains a BCD "60", the input filter in the HSIO circuit is enabled. Each input X0 through X3 has its own filter time constant. The filter circuit assigns the outputs of the filters as logical references X0 through X3.



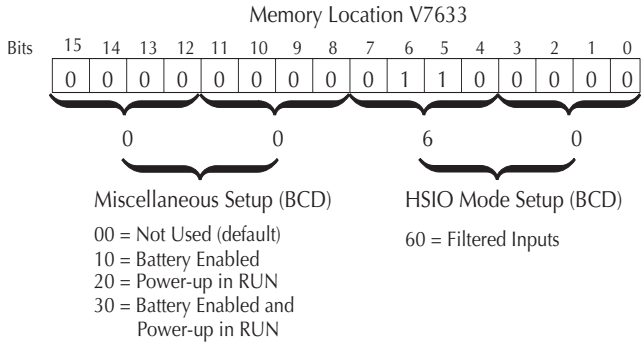
Input Filter Timing Parameters

Signal pulses at inputs X0 – X3 are filtered by using a delay time. In the figure below, the input pulse on the top line is longer than the filter time. The resultant logical input to ladder is phase-shifted (delayed) by the filter time on both rising and falling edges. In the bottom waveforms, the physical input pulse width is smaller than the filter time. In this case, the logical input to the ladder program remains in the OFF state (input pulse was filtered out).



Setup for Mode 60

Recall that V7633 is the HSIO Mode Select register. Refer to the diagram below. Use BCD 60 in the lower byte of V7633 to select the High-Speed Counter Mode.



Choose the most convenient method of programming V7633 from the following:

- Include load and out instructions in your ladder program
- *Direct*SOFT's memory editor
- Use the Handheld Programmer D2–HPP

We recommend using the first method above so that the HSIO setup becomes an integral part of your application program. An example program later in this section shows how to do this.

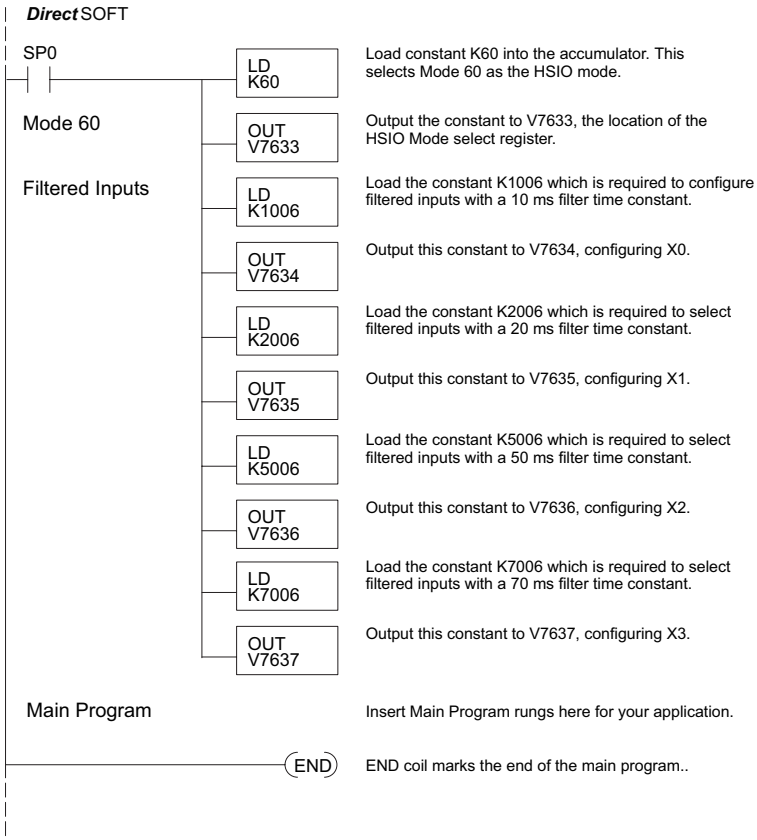
X Input Configuration

The configurable discrete input options for Discrete Filtered Inputs Mode are listed in the table below. The filter time constant (delay) is programmable from 0 to 99 mS (the input acts as a normal discrete input when the time constant is set to 0). The code for this selection occupies the upper byte of the configuration register in BCD. We combine this number with the required "06" in the lower byte to get "xx06", where xx = 0 to 99. Input X0, X1, X2, and X3 can only be filtered inputs. Each input has its own configuration register and filter time constant.

Input	Configuration Register	Function	Hex Code Required
X0	V7634	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)
X1	V7635	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)
X2	V7636	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)
X3	V7637	Filtered Input	xx06 (xx = filter delay time) 0 - 99 ms (BCD) (default)

Program Example: Filtered Inputs

The following program selects Mode 60, then programs the filter delay time constants for inputs X0, X1, X2, and X3. Each filter time constant is different, for illustration purposes. The program is otherwise generic, and may be adapted to your application.



PLC Memory

In this Appendix...

DL06 PLC Memory

When designing a PLC application, it is important for the PLC user to understand the different types of memory in the PLC. The DL06 CPU uses two types of memory: RAM and EEPROM. RAM is Random Access Memory and EEPROM is Electrically Erasable Programmable Read Only Memory. The PLC program is stored in EEPROM, and the PLC V-memory data is stored in RAM. There is also a small range of V-memory that can be copied to EEPROM which will be explained later.

The V-memory in RAM can be configured as either retentive or non-retentive.

Retentive memory is memory that is configured by the user to maintain values through a power cycle or a PROGRAM to RUN transition. Non-retentive memory is memory that is configured by the PLC user to clear data after a power cycle or a PROGRAM to RUN transition. The retentive ranges can be configured with either the handheld programmer using AUX57 or ***Direct***SOFT (PLC Setup).

The contents of RAM memory can be written to and read from an infinite number of times, but RAM requires a power source to maintain the contents of memory. The contents of RAM are maintained by the internal power supply (5VDC) only while the PLC is powered by an external source, normally 120VAC. When power to the PLC is turned off, the contents of RAM are maintained by a Super-Capacitor. If the Super-Capacitor ever discharges, the contents of RAM will be lost. The data retention time of the Super-Capacitor backed RAM is 3 weeks maximum, and 4 1/2 days minimum (at 60° C). An optional battery, D2-BAT-1, can be added to maintain RAM retentive memory if the DL06 is ever without external power (see Volume I, page 3-8 for a detailed explanation).

The contents of EEPROM memory can be read from an infinite number of times, but there is a limit to the number of times it can be written to (typical specification is 100,000 writes). EEPROM does not require a power source to maintain the memory contents. It will retain the contents of memory indefinitely.

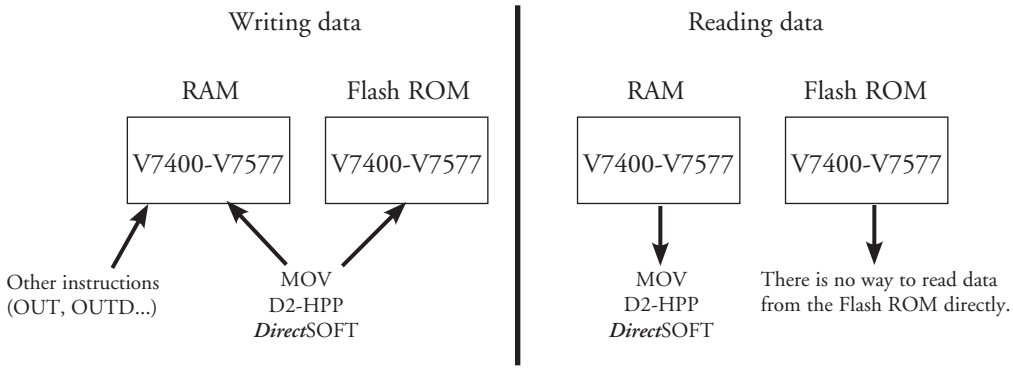
PLC user V-memory is stored in both volatile RAM and non-volatile EEPROM memory. Data being stored in RAM uses V400-V677, V1200-V7377 and V10000-V17777 which is volatile. Data stored in EEPROM uses V7400-V7577 and V700-V777, V7600-V7777 and V36000-V37777 are non-volatile.

Data values that must be retained for long periods of time, when the PLC is powered off, should be stored in EEPROM based V-memory. Since EEPROM is limited to the number of times it can be written to, it is suggested that transitional logic, such as a one-shot, be used to write the data one time instead on each CPU scan. The MOV instruction must be used to move the data into EEPROM.

Data values that are continually changing or which can be initialized with program logic should be stored in RAM based V-memory.

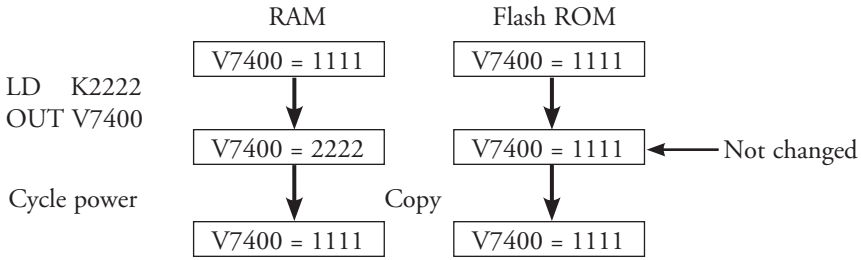
Non-volatile V-memory in the DL06

There are 2 types of memory assigned for the non-volatile V-memory area. They are RAM and flash ROM (EEPROM). They are sharing the same V-memory addresses; however, **you can only use the MOV instruction, D2-HPP and *Direct*SOFT to write data to the flash ROM.** When you write data to the flash ROM, the same data is also written to RAM. If you use other instructions, you can only write data to RAM. When you read data from the non-volatile V-memory area, the data is always read from RAM.



After a power cycle, the PLC always copies the data in the flash ROM to the RAM.

If you use the instructions except for the MOV instruction to write data into the non-volatile V-memory area, you only update the data in RAM. After a power cycle, the PLC copies the previous data from the flash memory to the RAM, so you may think the data you changed has disappeared. To avoid trouble such as this, we recommend that you use the MOV instruction.



This appears to be previous data returning.

Notes

ASCII Table

In this Appendix...

ASCII Conversion Table

DECIMAL TO HEX TO ASCII CONVERTER											
DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII
0	00	NUL	32	20	space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	“	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	‘	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	TAB	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Product Weights

In this Appendix...

Product Weight Table

PLC	Weight
D0–06AR	1.78 lb./807g.
D0–06DR	1.76 lb./798 g.
D0–06DR-D	1.72 lb./780 g.
D0–06AA	1.78 lb./807 g.
D0–06DA	1.76 lb./798 g.
D0–06DD1	1.68 lb./762 g.
D0–06DD1-D	1.64 lb./743 g.
D0–06DD2-D	1.64 lb./743 g.
D0–06DD2	1.68 lb./798 g.
D0–06LCD	0.12 lb./54.4g.

NUMBERING SYSTEMS

In this Appendix...

Introduction

As almost anyone who uses a computer is somewhat aware, the actual operations of a computer are done with a binary number system. Traditionally, the two possible states for a binary system are represented by the digits for "zero" (0) and "one" (1) although "off" and "on" or sometimes "no" and yes" are closer to what is actually involved. Most of the time a typical PC user has no need to think about this aspect of computers, but every now and then one gets confronted with the underlying nature of the binary system.

A PLC user, specifically the PLC programmer, should be familiar with the binary system. This appendix will provide an explanation of the numbering systems most commonly used by a PLC.

Binary Numbering System

Computers, including PLCs, use the Base 2 numbering system, which is called Binary and often called Decimal. As with a computer, there are only two valid digits a PLC relies on, zero and one, or off and on respectively. You would think that it would be hard to have a numbering system built on Base 2 with only two possible values, but the secret is by encoding using several digits.

Each digit in the base 2 system, when referenced by a computer, is called a bit. When four bits are grouped together, they form what is known as a nibble. Eight bits or two nibbles would be a byte. Sixteen bits or two bytes would be a word (Table 1). Thirty-two bits or two words is a double word.

Word															
Byte								Byte							
Nibble				Nibble				Nibble				Nibble			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1

Binary is not natural for us to use since we grow up using the base 10 system. Base 10 uses the numbers 0-9, as we are all well aware. From now on, the different bases will be shown as a subscripted number following the number. Example; 10 decimal would be 10_{10}.

Table 2 shows how base 2 numbers relate to their decimal equivalents.

A nibble of 1001_2 would be equal to a decimal number 9 ($1*2^3 + 1*2^0$ or $8_{10} + 1_{10}$). A byte of 11010101_2 would be equal to 213 ($1*2^7 + 1*2^6 + 1*2^4 + 1*2^2 + 1*2^0$ or $128_{10} + 64_{10} + 16_{10} + 4_{10} + 1_{10}$).

Binary/Decimal Bit Pattern																
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Power	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Bit Value	32678	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Max Value	65535_{10}															

Table 2

Hexadecimal Numbering System

The binary numbering system can be difficult and cumbersome to interpret for some users. Therefore, the hexadecimal numbering system was developed as a convenience for humans since the PLC (computer) only uderstands pure binary. The hexadecimal system is useful because it can represent every byte (8 bits) as two consecutive hexadecimal digits. It is easier for us to read hexadecimal numbers than binary numbers.

The hexadecimal numbering system uses 16 characters (base 16) to represent values. The first ten characters are the same as our decimal system (0-9), and the first six letters of the alphabet (A-F). Table 3 lists the first eighteen decimal numbers; 0-17 in the left column and the equivalent hexadecimal numbers are shown in the right column.

Decimal	Hex	Decimal	Hex
0	0	9	9
1	1	10	A
2	2	11	B
3	3	12	C
4	4	13	D
5	5	14	E
6	6	15	F
7	7	16	10
8	8	17	11

Table 3

Note that “10” and “11” in hex are not the same as “10” and “11” in decimal. Only the first ten numbers 0-9 are the same in the two representations. For example, consider the hex number “D8AF”. To evaluate this hex number use the same method used to write decimal numbers. Each digit in a decimal number represents a multiple of a power of ten (base 10). Powers of ten increase from right to left. For example, the decimal number 365 means $3x10^2 + 6x10 + 5$. In hex each digit represents a multiple of a power of sixteen (base 16). Therefore, the hex number D8AF translated to decimal means $13x16^3 + 8x16^2 + 10x16 + 15 = 55471$. However, going through the arithmetic for hex numbers in order to evaluate them is not really necessary. The easier way is to use the calculator that comes as an accessory in Windows. It can convert between decimal and hex when in “Scientific” view.

Note that a hex number such as “365” is not the same as the decimal number “365”. Its actual value in decimal terms is $3x16^2 + 6x16 + 5 = 869$. To avoid confusion, hex numbers are often labeled or tagged so that their meaning is clear. One method of tagging hex numbers is to append a lower case “h” at the end. Another method of labeling is to precede the number with 0x. Thus, the hex number “D8AF” can also be written “D8AFh”, where the lower case “h” at the end is just a label to make sure we know that it is a hex number. Also, D8AF can be written with a labeling prefix as “0xD8AF”.

Octal Numbering System

Many of the early computers used the octal numbering system for compiled printouts. Today, the PLC is about the only device that uses the Octal numbering system. The octal numbering system uses 8 values to represent numbers. The values are 0-7 being Base 8. Table 4 shows the first 31 decimal digits in octal. Note that the octal values are 0-7, 10-17, 20-27, and 30-37.

Octal	Decimal	Octal	Decimal
0	0	20	16
1	1	21	17
2	2	22	18
3	3	23	19
4	4	24	20
5	5	25	21
6	6	26	22
7	7	27	23
10	8	30	24
11	9	31	25
12	10	32	26
13	11	33	27
14	12	34	28
15	13	35	29
16	14	36	30
17	15	37	31

Table 4

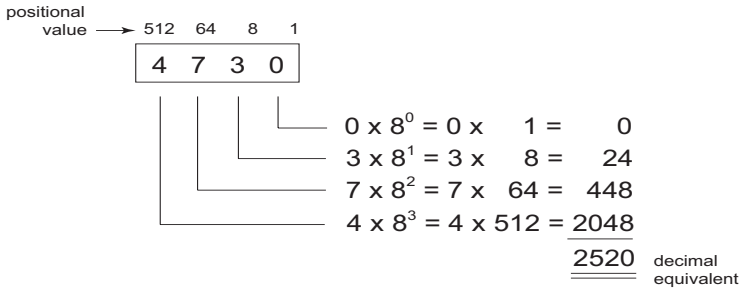
This follows the *Direct*LOGIC PLCs. Refer to Chapter 3 bit maps and notice that the memory addresses are numbered in octal, as well as each bit. The octal system is much like counting in the decimal system without the digits 8 and 9 being available.

The general format for four digits of the octal number is:

$$(d \times 8^0) + (d \times 8^1) + (d \times 8^2) + (d \times 8^3)$$

where "d" means digit. This is the same format used in the binary, decimal, or hexadecimal systems except that the base number for octal is 8.

Using the powers of expansion, the example below shows octal 4730 converted to decimal.



Binary Coded Decimal (BCD) Numbering System

BCD is a numbering system where four bits are used to represent each decimal digit. The binary codes corresponding to the hexadecimal digits A-F are not used in the BCD system. For this reason numbers cannot be coded as efficiently using the BCD system. For example, a byte can represent a maximum of 256 different numbers (i.e. 0-255) using normal binary, whereas only 100 distinct numbers (i.e. 0-99) could be coded using BCD. Also, note that BCD is a subset of hexadecimal and neither one does negative numbers.

	BCD Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Power	10^3				10^2				10^1				10^0			
Bit Value	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
Max Value	9				9				9				9			

Table 5

One plus for BCD is that it reads like a decimal number, for example, 867 in BCD would be expressed the same as 867 decimal. No conversion is needed; however, within the PLC, BCD calculations can be performed if numbers are adjusted to BCD after normal binary arithmetic.

Real (Floating Point) Numbering System

The terms Real and floating-point both describe IEEE-754 floating point arithmetic. This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented as well as how arithmetic should be carried out on them.

	Real (Floating Point 32) Bit Pattern															
Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Sign	Exponent								Mantissa						
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Mantissa (continues from above)															

Table 6

Most PLCs use the 32-bit format for floating point (or Real) numbers.

Floating point numbers which ***Direct***LOGIC PLCs use have three basic components: sign, exponent and mantissa. The 32 bit word required for the IEEE standard floating point numbers is shown in Table 6. It is represented as a number from 0 to 31, left to right. The first bit (31) is the sign bit, the next eight bits (30-23) are the exponent bits and the final 23 bits (22-0) are the fraction bits. In summary:

The sign bit is either "0" for positive or "1" for negative;

The exponent uses base 2;

The first bit of the mantissa is typically assumed to be "1.*fff*", where "*f*" is the field of fraction bits.

The Internet can provide a more in-depth explanation of the floating point numbering system. One website to look at is:

http://www.psc.edu/general/software/packages/ieee/ieee.html

BCD/Binary/Decimal/Hex/Octal - What is the Difference?

Sometimes there is confusion about the differences between the data types used in a PLC. The PLC's native data format is BCD, while the I/O numbering system is octal. Other numbering formats used are binary and Real. Although data is stored in the same manner (0's and 1's), there are differences in the way that the PLC interprets it.

While all of the formats rely on the base 2 numbering system and bit-coded data, the format of the data is dissimilar. Table 7 below shows the bit patterns and values for various formats.

Binary/Decimal Bit Pattern																
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decimal Bit Value	32678	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Max Value	65535															

	Hexadecimal Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decimal Bit Value	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
Max Value	F				F				F				F			

	BCD Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Decimal Bit Value	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
Max Value	9				9				9				9			

	Octal Bit Pattern															
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bit Value	1	4	2	1	4	2	1	4	2	1	4	2	1	4	2	1
Max Value	1	7			7			7			7			7		

	Real (Floating Point 32) Bit Pattern															
Bit #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Sign	Exponent								Mantissa						
Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Mantissa (continued from above)															

Table 7

As seen in Table 7, the BCD and hexadecimal formats are similar, although the maximum number for each grouping is different (9 for BCD and F for hexadecimal). This allows both formats to use the same display method. The unfortunate side effect is that unless the data type is documented, it's difficult to know what the data type is unless it contains the letters A-F.

Data Type Mismatch

Data type mismatching is a common problem when using an operator interface. Diagnosing it can be a challenge until you identify the symptoms. Since the PLC uses BCD as the native format, many people tend to think it is interchangeable with binary (unsigned integer) format. This is true to some extent, but not in this case. Table 8 shows how BCD and binary numbers differ.

Data Type Mismatch												
Decimal	0	1	2	3	4	5	6	7	8	9	10	11
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	0001 0000	0001 0001
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	0000 1010	0000 1011

Table 8

As the table shows, BCD and binary share the same bit pattern up until you get to the decimal number 10. Once you get past 10, the bit pattern changes. The BCD bit pattern for the decimal 10 is actually equal to a value of 16 in binary, causing the number to jump six digits by when viewing it as the BCD. With larger numbers, the error multiplies. Binary values from 10 to 15 Decimal are actually invalid for the BCD data type.

Looking at a larger number, such as the value shown in Table 9, both the BCD bit pattern and the decimal bit pattern correspond to a base 10 value of 4095_{10}. If bit patterns are read, or interpreted, in a different format than what is used to write them, the data will not be correct. For instance, if the BCD bit pattern is interpreted as a decimal (binary) bit pattern, the result is a base 10 value of 16533_{10}. Similarly, if you try to view the decimal (binary) bit pattern as a BCD value, it is not a valid BCD value at all, but could be represented in hexadecimal as 0xFFF.

Base 10 Value	BCD Bit Pattern	Binary Bit Pattern
4095	0100 0000 1001 0101	1111 1111 1111

Table 9

Look at the following example and note the same value represented by the different numbering systems.

0100	0011	Binary	0001	0010 0011 0100	Binary
6	7	Decimal	4	6 6 0	Decimal
4	3	Hex	1	2 3 4	Hex
0110	0111	BCD	0100	0110 0110 0000	BCD
1 0	3	Octal	1	1 0 6 4	Octal

Signed vs. Unsigned Integers

So far, we have dealt with unsigned data types only. Now we will deal with signed data types (negative numbers). The BCD and hexadecimal numbering systems do not use signed data types.

In order to signify that a number is negative or positive, we must assign a bit to it. Usually, this is the Most Significant Bit (MSB) as shown in Table 10. For a 16-bit number, this is bit 15. This means that for 16-bit numbers we have a range of -32767 to 32767.

Bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSB															LSB

Table 10

There are two ways to encode a negative number: two's complement and Magnitude Plus sign. The two methods are not compatible.

The simplest method to represent a negative number is to use one bit of the PLC word as the sign of a number while the remainder of the word gives its magnitude. It is general convention to use the most significant bit (MSD) as the sign bit: a 1 will indicate a negative, and a 0 a positive number. Thus, a 16 bit word allows numbers in the range ±32767. Table 12 shows a representations of 100 and a representation of -100 in this format.

Magnitude Plus Sign	
Decimal	**Binary**
100	0000 0000 0110 0100
-100	1000 0000 0110 0100

Table 11

Two's complement is a bit more complicated. Without getting involved with a full explanation, a simple formula for two's complement is to invert the binary and add one (see Table 12). Basically, 1's are being changed to 0's and all 0's are being changed to 1.

Two's Compliment	
Decimal	**Binary**
100	0000 0000 0110 0100
-100	1111 1111 1001 1100

Table 12

More information about 2's complement can be found on the Internet at the following websites:

http://www.evergreen.edu/biophysics/technotes/program/2s_comp.htm

AutomationDirect.com Products and Data Types

*Direct*LOGIC PLCs

The *Direct*LOGIC PLC family uses the octal numbering system for all addressing which includes: inputs, outputs, internal V-memory locations, timers, counters, internal control relays (bits), etc. Most data in the PLC, including timer and counter current values, is in BCD format by default. User data in V-memory locations may be stored in other data types if it is changed by the programmer, or comes from some external source, such as an operator interface. Any manipulation of data must use instructions appropriate for that data type which includes: Load instructions, Math instructions, Out box instructions, comparison instructions, etc. In many cases, the data can be changed from one data type to another, but be aware of the limitations of the various data types when doing so. For example, to change a value from BCD to decimal (binary), use a BIN instruction box. To change from BCD to a real number, use a BIN and a BTOR instruction box. When using Math instructions, ***the data types must match.*** For example, a BCD or decimal (binary) number cannot be added to a real number, and a BCD number cannot be added to a decimal (binary) number. If the data types are mismatched, the results of any math operation will be meaningless.

To simplify making, number conversions Intelligent Box (IBox) Instructions are available with *Direct*SOFT. These instruction descriptions are located in Volume 1, page 5-230, in the Math IBox group.

Most *Direct*LOGIC analog modules can be setup to give the raw data in decimal (binary) format or in BCD format, so it is necessary to know how the module is being used. *Direct*LOGIC PID is another area where not all values are in BCD. In fact, nearly all of the PID parameters are stored in the PLC memory as decimal (binary) numbers.

NOTE: *The PID algorithm uses magnitude plus sign for negative decimal (binary) numbers, whereas the standard math functions use two's complement. This can cause confusion while debugging a PID loop.*

When using the Data View in *Direct*SOFT, be certain that the proper format is selected for the element to be viewed. The data type and length is selected using the drop-down boxes at the top of the Data View window. Also notice that BCD is called BCD/Hex. Remember that BCD is a subset of hexadecimal so they share a display format even though the values may be different. This is where good documentation of the data type stored in memory is crucial.

C-more/C-more Micro-Graphic Panels

In the C-more and C-more Micro-Graphic HMI operator panels, the 16-bit BCD format is listed as "BCD int 16". Binary format is either "Unsigned int 16" or "Signed int 16" depending on whether or not the value can be negative. Real number format is "Floating PT 32".

More available formats are, "BCD int 32", "Unsigned int 32" and "Signed int 32".

Notes

European Union Directives (CE)

In This Appendix...

European Union (EU) Directives

NOTE: *The information contained in this section is intended as a guideline and is based on our interpretation of the various standards and requirements. Since the actual standards are issued by other parties, and in some cases governmental agencies, the requirements can change over time without advance warning or notice. Changes or additions to the standards can possibly invalidate any part of the information provided in this section.*

This area of certification and approval is absolutely vital to anyone who wants to do business in Europe. One of the key tasks that faced the EU member countries and the European Economic Area (EEA) was the requirement to bring several similar yet distinct standards together into one common standard for all members. The primary purpose of a single standard was to make it easier to sell and transport goods between the various countries and to maintain a safe working and living environment. The Directives that resulted from this merging of standards are now legal requirements for doing business in Europe. Products that meet these Directives are required to have a CE mark to signify compliance.

Member Countries

As of January 1, 2007, the members of the EU are Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Latvia, Lithonia, Luxembourg, Malta, Netherlands, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, and United Kingdom. Iceland, Liechtenstein, and Norway together with the EU members make up the European Economic Area (EEA) and all are covered by the Directives.

Applicable Directives

There are several Directives that apply to our products. Directives may be amended or added as required.

- **Electromagnetic Compatibility Directive (EMC)** — this Directive attempts to ensure that devices, equipment and systems have the ability to function satisfactorily in an electromagnetic environment without introducing intolerable electromagnetic disturbance to anything in that environment.
- **Machinery Safety Directive** — this Directive covers the safety aspects of the equipment, installation, etc. There are several areas involved, including testing standards covering both electrical noise immunity and noise generation.
- **Low Voltage Directive** — this Directive is also safety related and covers electrical equipment that has voltage ranges of 50–1000VAC and/or 75–1500VDC.
- **Battery Directive** — this Directive covers the production, recycling and disposal of batteries.

Compliance

Certain standards within each Directive already require mandatory compliance. The EMC Directive, which has gained the most attention, became mandatory as of January 1, 1996. The Low Voltage Directive became mandatory as of January 1, 1997.

Ultimately, we are all responsible for our various pieces of the puzzle. As manufacturers, we must test our products and document any test results and/or installation procedures that are necessary to comply with the Directives. As a machine builder, you are responsible for

installing the products in a manner which will ensure compliance. You are also responsible for testing any combination of products that may (or may not) comply with the Directives when used together. The end user of the products must comply with any Directives that may cover maintenance, disposal, etc., of equipment or various components. *Although we strive to provide the best assistance available, it is impossible for us to test all possible configurations of our products with respect to any specific Directive. Because of this, it is ultimately your responsibility to ensure that your machinery (as a whole) complies with these Directives and to keep up with applicable Directives and/or practices that are required for compliance.*

As of January 1, 1999, the DL05, DL06, DL205, DL305, and DL405 PLC systems manufactured by Koyo Electronics Industries or FACTS Engineering, when properly installed and used, conform to the Electromagnetic Compatibility (EMC), Low Voltage Directive, and Machinery Directive requirements of the following standards.

- **EMC Directive Standards Relevant to PLCs**
  EN50081–1 Generic emission standard for residential, commercial, and light industry
  EN50081–2 Generic emission standard for industrial environment.
  EN50082–1 Generic immunity standard for residential, commercial, and light industry
  EN50082–2 Generic immunity standard for industrial environment.
- **Low Voltage Directive Standards Applicable to PLCs**
  EN61010–1 Safety requirements for electrical equipment for measurement, control, and laboratory use.
- **Product Specific Standard for PLCs**
  EN61131–2 Programmable controllers, equipment requirements and tests. This standard replaces the above generic standards for immunity and safety. However, the generic emissions standards must still be used in conjunction with the following standards:
  -EN 61000-3-2 Harmonics
  -EN 61000-3-2 Fluctuations
- **Warning on Electrostatic Discharge (ESD)**
  We recommend that all personnel take necessary precautions to avoid the risk of transferring static charges to inside the control cabinet, and clear warnings and instructions should be provided on the cabinet exterior. Such precautions may include the use of earth straps, similar devices or the powering off of the equipment inside the enclosure before the door is opened.
- **Warning on Radio Interference (RFI)**
  This is a class A product. In a domestic environment, this product may cause radio interference in which case the user may be required to take adequate measures.

General Safety

- External switches, circuit breaker or external fusing, are required for these devices.
- The switch or circuit breaker should be mounted near the PLC equipment.

AutomationDirect is currently in the process of changing testing procedures from the generic standards to the product specific standards.

Special Installation Manual

The installation requirements to comply with the requirements of the Machinery Directive, EMC Directive and Low Voltage Directive are slightly more complex than the normal installation requirements found in the United States. To help with this, we have published a special manual which you can order:

- **DA–EU–M** – EU Installation Manual that covers special installation requirements to meet the EU Directive requirements. Order this manual to obtain the most up-to-date information.

Other Sources of Information

Although the EMC Directive gets the most attention, other basic Directives, such as the Machinery Directive and the Low Voltage Directive, also place restrictions on the control panel builder. Because of these additional requirements it is recommended that the following publications be purchased and used as guidelines:

- BSI publication TH 42073: February 1996 – covers the safety and electrical aspects of the Machinery Directive
- EN 60204–1:1992 – General electrical requirements for machinery, including Low Voltage and EMC considerations
- IEC 1000–5–2: EMC earthing and cabling requirements
- IEC 1000–5–1: EMC general considerations

It may be possible for you to obtain this information locally; however, the official source of applicable Directives and related standards is:

The Office for Official Publications of the European Communities L–2985 Luxembourg; quickest contact is via the World Wide Web at http://euro–op.eu.int/indexn.htm

Another source is:

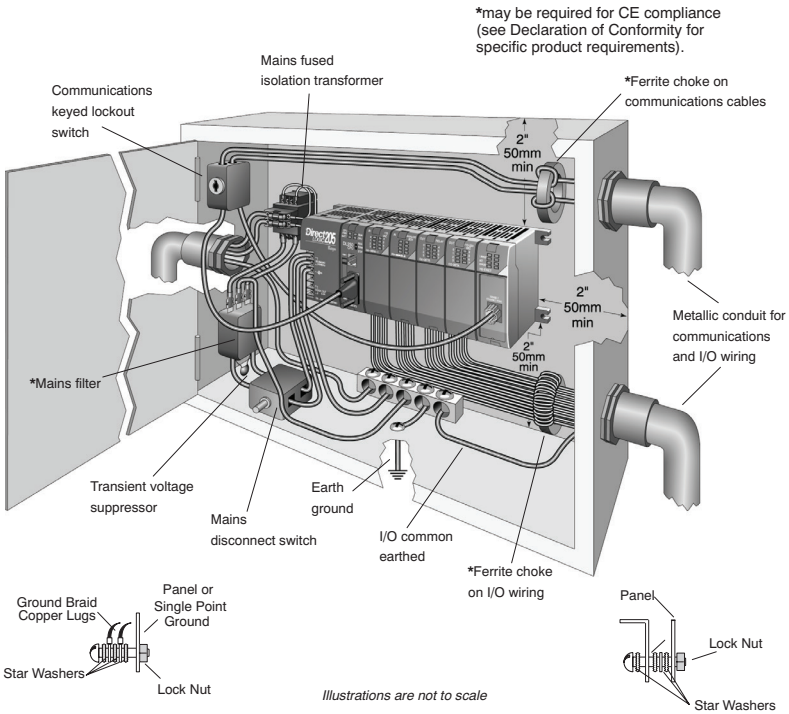
British Standards Institution – Sales Department
Linford Wood
Milton Keynes
MK14 6LE
United Kingdom

The quickest contact is via the World Wide Web at http://www.bsi.org.uk

Basic EMC Installation Guidelines

Enclosures

The following diagram illustrates good engineering practices supporting the requirements of the Machinery and Low Voltage Directives. House all control equipment in an industry standard lockable steel enclosure and use metallic conduit for wire runs and cables.



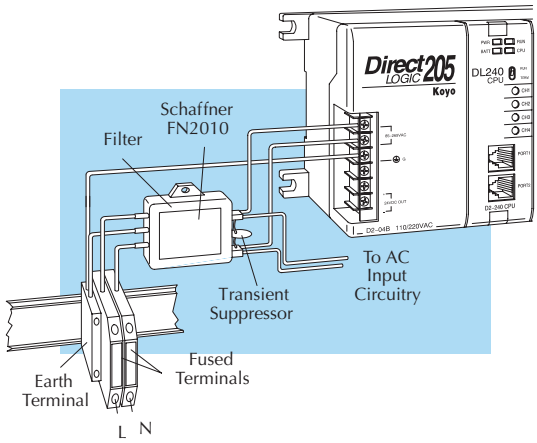
Electrostatic Discharge (ESD)

We specify in all declarations of conformity that our products are installed inside an industrial enclosure using metallic conduit for external wire runs; therefore, we test the products in a typical enclosure. However, we would like to point out that although our products operate normally in the presence of ESD, this is only the case when mounted within an enclosed industrial control cabinet. When the cabinet is open during installation or maintenance, the equipment and or programs may be at risk of damage from ESD carried by personnel.

We therefore recommend that all personnel take necessary precautions to avoid the risk of transferring static electricity to components inside the control cabinet. If necessary, clear warnings and instructions should be provided on the cabinet exterior, such as recommending the use of earth straps of similar devices, or the powering off of equipment inside the enclosure.

AC Mains Filters

The DL305 AC powered base power supplies require extra mains filtering to comply with the EMC Directive on conducted RF emissions. All PLC equipment has been tested with filters from Schaffner, which reduce emissions levels if the filters are properly grounded (earth ground). A filter with a current rating suitable to supply all PLC power supplies and AC input modules should be selected. We suggest the FN2080 for DL305 systems.



NOTE: *Very few mains filters can reduce problem emissions to negligible levels. In some cases, filters may increase conducted emissions if not properly matched to the problem emissions.*

Suppression and Fusing

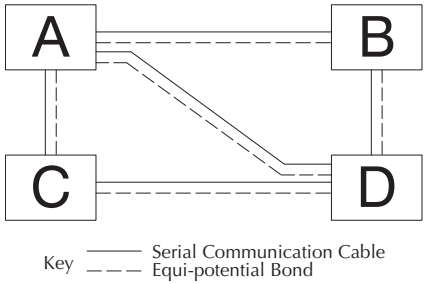
In order to comply with the fire risk requirements of the Low Voltage and Machinery Directive electrical standards EN 61010–1, and EN 60204–1, by limiting the power into "unlimited" mains circuits with power leads reversed, it is necessary to fuse both AC and DC supply inputs. You should also install a transient voltage suppressor across the power input connections of the PLC. Choose a suppressor such as a metal oxide varistor, with a rating of 275VAC working voltage for 230V nominal supplies (150VAC working voltage for 115V supplies) and high energy capacity (eg. 140 joules).

Transient suppressors must be protected by fuses and the capacity of the transient suppressor must be greater than the blow characteristics of the fuses or circuit breakers to avoid a fire risk. A recommended AC supply input arrangement for Koyo PLCs is to use twin 3 amp TT fused terminals with fuse blown indication, such as DINnectors DN–F10L terminals, or twin circuit breakers, wired to a Schaffner FN2010 filter or equivalent, with high energy transient suppressor soldered directly across the output terminals of the filter. PLC system inputs should also be protected from voltage impulses by deriving their power from the same fused, filtered, and surge-suppressed supply.

Internal Enclosure Grounding

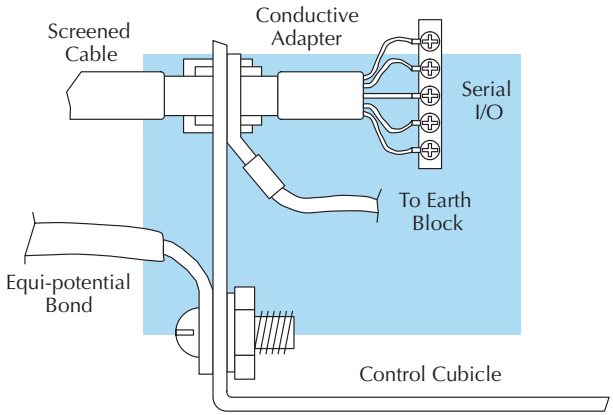
A heavy-duty star earth terminal block should be provided in every cubicle for the connection of all earth ground straps, protective earth ground connections, mains filter earth ground wires, and mechanical assembly earth ground connections. This should be installed to comply with safety and EMC requirements, local standards, and the requirements found in IEC 1000-5-2. The Machinery Directive also requires that the common terminals of PLC input modules and common supply side of loads driven from PLC output modules be connected to the protective earth ground terminal.

Equi–potential Grounding



Adequate site earth grounding must be provided for equipment containing modern electronic circuitry. The use of isolated earth electrodes for electronic systems is forbidden in some countries. Make sure you check any requirements for your particular destination. IEC 1000–5–2 covers equi-potential bonding of earth grids adequately, but special attention should be given to apparatus and control cubicles that contain I/O devices, remote I/O racks, or have inter-system communications with the primary PLC system enclosure. An equi-potential bond wire must be provided alongside all serial communications cables, and to any separate items of the plant which contain I/O devices connected to the PLC. The diagram shows an example of four physical locations connected by a communications cable.

Communications and Shielded Cables



Good quality 24 AWG minimum twisted-pair shielded cables, with overall foil and braid shields are recommended for analog cabling and communications cabling outside of the PLC enclosure. To date, it has been a common practice to provide an earth ground for one end of the cable shield only in order to minimize the risk of noise caused by earth ground loop currents between apparatus. The procedure of grounding only one end, which primarily originated as a result of trying to reduce hum in audio systems, is no longer applicable to the complex industrial environment. Shielded cables are also efficient emitters of RF noise from the PLC system, and can interact in a parasitic manner in networks and between multiple sources of interference.

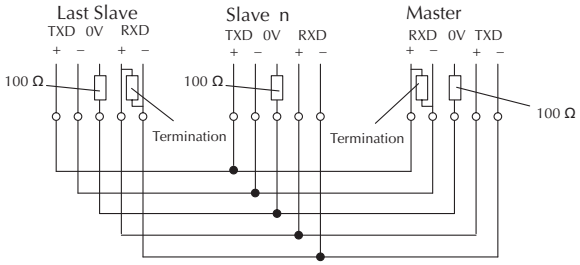
The recommendation is to use shielded cables as electrostatic "pipes" between apparatus and systems, and to run heavy gauge equi-potential bond wires alongside all shielded cables. When a shielded cable runs through the metallic wall of an enclosure or machine, it is recommended in IEC 1000–5–2 that the shield be connected over its full perimeter to the wall, preferably using a conducting adapter, and not via a pigtail wire connection to an earth ground bolt. Shields must be connected to every enclosure wall or machine cover that they pass through.

Analog and RS232 Cables

Providing an earth ground for both ends of the shield for analog circuits provides the perfect electrical environment for the twisted pair cable as the loop consists of signal and return, in a perfectly balanced circuit arrangement, with connection to the common of the input circuitry made at the module terminals. RS232 cables are handled in the same way.

Multidrop Cables

RS422 twin twisted pair, and RS485 single twisted pair cables also require a 0V link, which has often been provided in the past by the cable shield. It is now recommended that you use triple twisted pair cabling for RS422 links, and twin twisted pair cable for RS485 links. This is because the extra pair can be used as the 0V inter-system link. With loop DC power supplies earth grounded in both systems, earth loops are created in this manner via the inter-system 0v link. The installation guides encourage earth loops, which are maintained at a low impedance by using heavy equi-potential bond wires. **To account for non–European installations using single-end earth grounds, and sites with far from ideal earth ground characteristics, we recommend the addition of 100 ohm resistors at each 0V link es.**



Shielded Cables within Enclosures

When you run cables between PLC items within an enclosure which also contains susceptible electronic equipment from other manufacturers, remember that these cables may be a source of RF emissions. There are ways to minimize this risk. Standard data cables connecting PLCs and/or operator interfaces should be routed well away from other equipment and their associated cabling. You can make special serial cables where the cable shield is connected to the enclosure's earth ground at both ends, the same way as external cables are connected.

Analog Modules and RF Interference

All Automationdirect products are tested to withstand field strength levels up to 10V/m, which is the maximum required by the relevant EU standards. While all products pass this test, analog modules will typically exhibit deviations of their readings. This is quite normal; however, systems designers should be aware of this and plan accordingly.

When assembling a control system using analog modules, these standards must be followed and should be integrated into the system design. This is the responsibility of the system builder/commissioner.

Network Isolation

For safety reasons, it is a specific requirement of the Machinery Directive that **a keyswitch must be provided** that isolates any network input signal during maintenance, so that remote commands cannot be received that could result in the operation of the machinery. To avoid the introduction of noise into the system, any keyswitch assembly should be housed in its own earth grounded steel box and the integrity of the shielded cable must be maintained.

Again, for further information on EU directives we recommend that you get a copy of our EU Installation Manual (DA–EU–M). Also, if you are connected to the World Wide Web, you can check the EU commission's official site at: http://ec.europa.eu/index_en.htm.

DC Powered Versions

Due to slightly higher emissions radiated by the DC powered versions of the DL06, and the differing emissions performance for different DC supply voltages, the following stipulations must be met:

- The PLC must be housed within a metallic enclosure with a minimum number of orifices.
- I/O and communications cabling exiting the cabinet must be contained within metallic conduit/ trunking.

Items Specific to the DL06

- The rating between all circuits in this product is **as basic insulation only**, as appropriate for single fault conditions.
- There is no isolation offered between the PLC and the analog inputs of this product.
- It is the responsibility of the system designer to earth one side of all control and power circuits, and to earth the braid of screened cables.
- This equipment must be properly installed while adhering to the guidelines of the in-house PLC installation manual DA–EU–M, and the installation standards IEC 1000–5–1, IEC 1000–5–2 and IEC 1131–4.
- It is a requirement that all PLC equipment must be housed in a protective steel enclosure, which limits access to operators by a lock and power breaker. If access is required by operators or untrained personnel, the equipment must be installed inside an internal cover or secondary enclosure. A warning label must be used on the front door of the installation cabinet as follows: **Warning: Exposed terminals and hazardous voltages inside**
- It should be noted that the safety requirements of the machinery directive standard EN60204–1 state that all equipment power circuits must be wired through isolation transformers or isolating power supplies, and that one side of all AC or DC control circuits must be earthed.
- Both power input connections to the PLC must be separately fused using 3 amp T-type anti–surge fuses, and a transient suppressor fitted to limit supply overvoltages.
- **Warning: If the equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.**

INTRODUCTION TO SERIAL COMMUNICATIONS

In this Appendix...

Introduction to Serial Communications

*Direct*LOGIC® PLCs have two built-in serial communication ports which can be used to communicate to other PLCs or to other serial devices. To fully understand the capabilities and limitations of the serial ports, a brief introduction to serial communications is in order.

There are three major components to any serial communications setup:

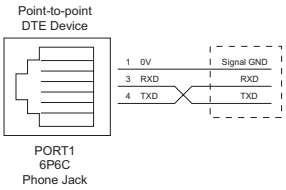
- Wiring standard
- Communications protocol
- Communications parameters

Each of these will be discussed in more detail as they apply to *Direct*LOGIC PLCs.

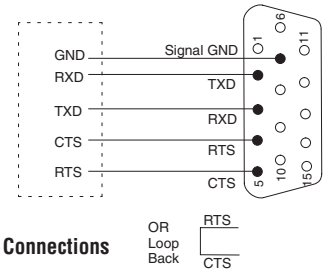
Wiring Standards

There are three different wiring standards that can be used with *Direct*LOGIC PLCs: RS-232C, RS-422 and RS-485.

RS-232C is a point-to-point wiring standard with a practical wiring distance of 15 meters (50 feet) maximum. This means that only two devices can communicate on an RS-232C network – a single master device and a single slave device. The total cable length cannot exceed 50 feet. AutomationDirect L19772 (Belden® 8102), or equivalent, is recommended for RS-232C networks.

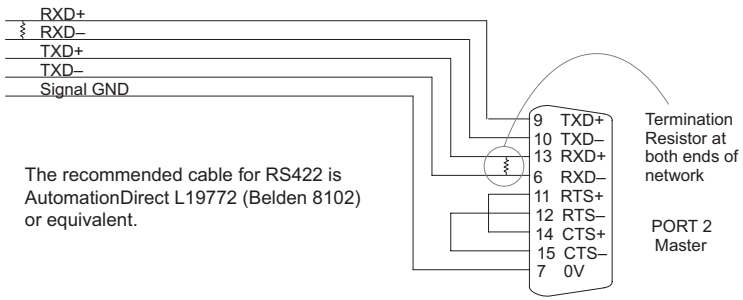


Port 1 Connections

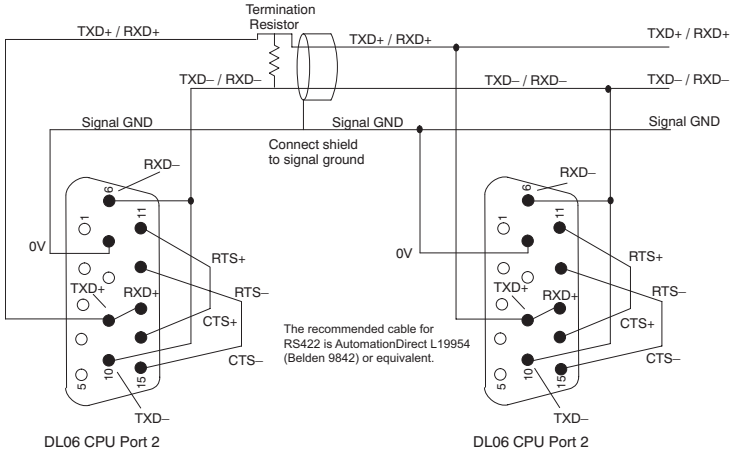


Port 2 Connections

The RS-422 wiring standard does not specify a network topology, but in practice, a daisy-chain topology with the master at one end is the only way to ensure network reliability. AutomationDirect L19772 (Belden® 8102), or equivalent, is recommended for RS-422 networks. Use a terminating resistor equal in value to the characteristic impedance of the cable being used (100 Ω for AutomationDirect L19772 [Belden® 8102]).



RS-485 is a multi-point wiring standard with a practical wiring distance of 4000 feet maximum. This wiring standard provides for the possibility of up to 32 masters communicating to up to 32 slaves, all within the maximum distance of 4000 feet. Note that while the RS-485 wiring standard provides for multiple masters on the same network, **the *Direct*LOGIC PLCs do not support multiple masters on a single network.** The RS-485 wiring standard does not specify a network topology, but in practice, a daisy-chain topology with the master at one end is the only way to ensure network reliability. AutomationDirect L19954 (Belden 9842), or equivalent is recommended for RS-485 networks. Use a terminating resistor equal in value to the characteristic impedance of the cable being used (120 Ω for AutomationDirect L19954 [Belden 9842]).



Communications Protocols

A communications protocol is the language the devices on a network use to communicate with each other. All the devices on the network must use the same communications protocol in order to be able to communicate with each other. The protocols available in the *Direct*LOGIC PLCs are listed in the following table.

Communications Protocols							
Protocol	**Master**	**Slave**	**Port 1***	**Port 2**	**RS-232C**	**RS-422**	**RS-485****
K-Sequence	No	Yes	Yes	Yes	Yes	Yes	No
***Direct*NET**	Yes	Yes	Yes	Yes	Yes	Yes	No
MODBUS RTU	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ASCII	Out	In	No	Yes	Yes	Yes	No

* Port 1 supports slave only and is only RS-232C with fixed communications parameters of 9600 baud, 8 data bits, 1 start bit, 1 stop bit, odd parity and station address 1. It is an asynchronous, half-duplex DTE port and auto-selects between K-Sequence, *Direct*NET and MODBUS RTU protocols.

** RS-485 is available on Port 2 for MODBUS RTU protocol only.

K-Sequence protocol is not available for use by a master DL06 PLC. Therefore, it cannot be used for networking between PLCs. Its primary use in the DL06 PLC is as a slave to *Direct*SOFT programming software and to an operator interface.

*Direct*NET protocol is available for use by a master or by a slave DL06 PLC. This, and the fact that it is *native* protocol, makes it ideal for PLC-to-PLC communication over a point-to-point to multipoint network using the RX and WX instructions.

MODBUS RTU protocol is a very common industry standard protocol, and can be used by a master or slave DL06 to communicate with a wide variety of industrial devices which support this protocol.

ASCII is another very common industry standard protocol, and is commonly used where alpha-numeric character data is to be transferred. Many input devices, such as, barcode readers and electronic scales use ASCII protocol, and many output devices accept ASCII commands, as well.

It doesn't matter which wiring standard or protocol is used, there are several communications parameters to select for each device before they will be able to communicate. These parameters include:

Baud Rate	Flow Control
Data Bits	Echo Suppression
Parity	Timeouts
Stop Bits	Delay Times
Station Address	Format

All of these parameters may not be necessary, or available, for your application. The parameters used will depend on the protocol being used and whether the device is a master or slave.

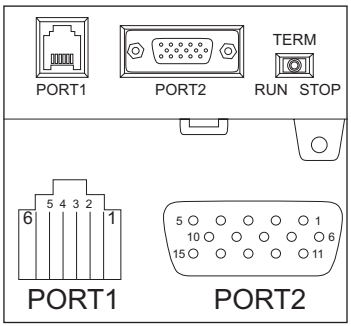
NOTE: *REMEMBER: When the same parameter is available in the master and in the slave (i.e. Baud Rate, Parity, Stop Bits, etc.), the settings must match.*

DL06 Port Specifications

Communications Port 1	
Port 1	Connects to HPP, *Direct*SOFT, operator interfaces, etc.
	6-pin, RS232C
	Communication speed (baud): 9600 (fixed)
	Parity: odd (fixed)
	Station Address: 1 (fixed)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocol (auto-select): K-sequence (slave only), *Direct*NET (slave only), MODBUS (slave only)

Communications Port 2	
Port 2	Connects to HPP, *Direct*SOFT, operator interfaces, etc.
	15-pin, multifunction port, RS232C, RS422, RS485 (RS485 with 2-wire is only available for MODBUS and Non-sequence)
	Communication speed (baud): 300, 600, 1200, 2400, 4800, 9600, 19200, 38400
	Parity: odd (default), even, none
	Station Address: 1 (default)
	8 data bits
	1 start, 1 stop bit
	Asynchronous, half-duplex, DTE
	Protocols can be pre-selected: K-sequence (slave only), *Direct*NET (master/slave), MODBUS (master/slave), non-sequence/print/ASCII in/out

DL06 Port Pinouts



Port 1 Pin Descriptions		
1	0V	Power (-) connection (GND)
2	5V	Power (+) connection
3	RXD	Receive data (RS-232C)
4	TXD	Transmit data (RS-232C)
5	5V	Power (+) connection
6	0V	Power (-) connection (GND)

Port 2 Pin Descriptions		
1	5V	Power (+) connection
2	TXD	Transmit data (RS-232C)
3	RXD	Receive data (RS-232C)
4	RTS	Ready to send (RS-232C)
5	CTS	Clear to send (RS232C)
6	RXD-	Receive data (-) (RS-422/485)
7	0V	Power (-) connection (GND)
8	0V	Power (-) connection (GND)
9	TXD+	Transmit data (+) (RS-422/485)
10	TXD-	Transmit data (-) (RS-422/485)
11	RTS+	Ready to send (+) (RS-422/485)
12	RTS-	Ready to send (-) (RS-422/485)
13	RXD+	Receive data (+) (RS-422/485)
14	CTS+	Clear to send (+) (RS-422/485)
15	CTS-	Clear to send (-) (RS-422/485)

Note that the default configuration for port 2 is:

Auto-detect among K-Sequence, *Direct*NET, and MODBUS RTU protocols

Timeout = Base Timeout x 1 (800 ms)

RTS on delay time = 0 ms

RTS off delay time = 0 ms

Station Number = 1

Baud rate = 19200

Stop bits = 1

Parity = odd

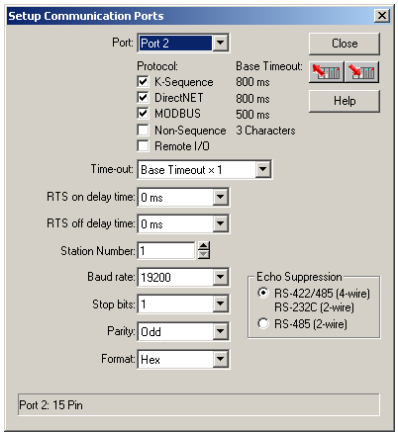
Format = Hex

Echo Suppression = RS-422/485 (4-wire) or RS-232C

Port Setup Using *Direct*SOFT or Ladder Logic Instructions

Port 2 on the DL06 can be configured for communications using the various protocols which have been previously mentioned. Also, the communications parameters can be configured to match the parameters in the other device(s) with which the PLC will be communicating. The port may be configured using the *Direct*SOFT PLC programming software, or by using ladder logic within the PLC program. It is important to note that the settings for port 2 are never saved to disk with *Direct*SOFT, so if you are using port 2 in other than its default configuration (see page K-6) it is a good idea to include the port setup in the ladder program, typically on a first scan bit, or in an initialization subroutine.

To setup port 2 using *Direct*SOFT, the PLC must be turned on and connected to *Direct*SOFT. If the PLC Setup toolbar is displayed, either select the **Port 2** button or select **PLC > Setup > Setup Sec. Comm Port...** from the menu bar located at the top of the programming window. A dialog box like the one below will appear. Make the appropriate settings and write them to the PLC.



In order to setup port 2 in your relay ladder logic, the appropriate values must be written to V7655 (Word 1), V7656 (Word 2) and V7650 (Word 3, for ASCII only) to specify the settings for the port. Then write the 'setup complete' flag (K0500) to V7657 (Word 3) to request the CPU to accept the port settings. Once the CPU sees the 'setup complete' flag in V7657 it will test the port settings which have selected for validity, and then change the value in V7657 according to the results of this test. If the port settings are valid, the CPU will change the value in V7657 to 0A00 (**A** for Accepted). If there was an error in the port settings, the CPU will change the value in V7657 to 0E00 (**E** for Error).

NOTE: Helpful Hint: *Rather than build the setup words manually from the tables, use* ***Direct****SOFT to setup the port as desired then use a Dataview to view the setup words as BCD/HEX. Then simply use these numbers in the setup code.*

The data that is written to the port setup words has two formats. The format that is used depends on whether K-Sequence, *Direct*NET, MODBUS RTU (method 1) or ASCII (method 2) is selected.

Port 2 Setup for RLL Using K-Sequence, *Direct*NET or MODBUS RTU

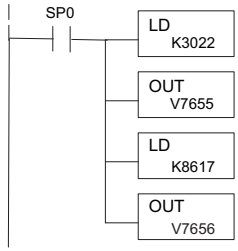
V7655 (Word 1)	RTS On-delay	Timeout (% of timeout)	Protocol	RTS Off-delay
0yyy 0ttt mmmm mxx	yyy	ttt	mmmmm	xxx
	000 = 0ms	000 = 100%	10000 = K-Sequence	000 = 0ms
	001 = 2ms	001 = 120%	01000 = ***Direct***NET	001 = 2ms
	010 = 5ms	010 = 150%	00100 = MODBUS RTU	010 = 5ms
	011 = 10ms	011 = 200%		011 = 10ms
	100 = 20ms	100 = 500%		100 = 20ms
	101 = 50ms	101 = 1000%		101 = 50ms
	110 = 100ms	110 = 2000%		110 = 100ms
	111 = 500ms	111 = 5000%		111 = 500ms

V7656 (Word 2)	Parity	Stop Bits	Echo Suppression	Baud Rate
K-Sequence, *Direct*NET & MODBUS RTU				
pps0 ebbb xaaa aaaa	pp	s	e	bbb
	00 = None	0 = 1 bit	0 = 232 or 422	000 = 300
	10 = Odd	1 = 2 bits	1 = 485, 2 wire	001 = 600
	11 = Even			010 = 1200
				011 = 2400
				100 = 4800
				101 = 9600
				110 = 19200
				111 = 38400

V7656 (Word 2) cont'd	Protocol	Secondary Address
K-Sequence, *Direct*NET & MODBUS RTU	(***Direct***NET)	xaaaaaaa (***Direct***NET)
pps0 ebbb xaaa aaaa	x	_aaaaaaa (K-Sequence & MODBUS RTU)
	0 = Hex	K-Sequence: 1-90
	1 = ASCII	***Direct***NET: 1-90
		MODBUS: 1-247

To setup port 2 for MODBUS protocol for the following: RTS On-delay of 10ms, Base timeout x1, RTS Off-delay of 5ms, Odd parity, 1 Stop bit, echo suppression for RS232-C/ RS422, 19,200 baud, Station Number 23 you would use the relay ladder logic shown below.

Port 2 Setup for RLL Using ASCII



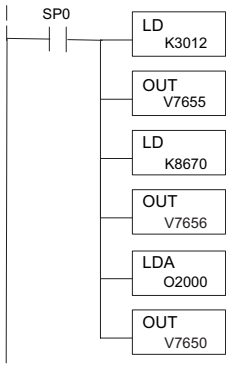
Word 1	RTS On-delay	Timeout (in% of std. timeout)	Protocol	RTS Off-delay
0yyy 0ttt mmmm mxx	yyy	ttt	mmmmm	xxx
DL05/06: V7655	000 = 0ms	000 = 100%	00010 = Non-Sequence	000 = 0ms
	001 = 2ms	001 = 120%		001 = 2ms
	010 = 5ms	010 = 150%		010 = 5ms
	011 = 10ms	011 = 200%		011 = 10ms
	100 = 20ms	100 = 500%		100 = 20ms
	101 = 50ms	101 = 1000%		101 = 50ms
	110 = 100ms	110 = 2000%		110 = 100ms
	111 = 500ms	111 = 5000%		111 = 500ms

Word 2	Parity	Stop Bits	Echo Suppression (valid for DL06 only)	Baud Rate
K-Sequence, *Direct*NET & MODBUS RTU				
pps0 ebbb xaaa aaaa	pp	s	e	bbb
DL05/06: V7656	00 = None	0 = 1 bit	0 = RS-232C, RS-422 or RS-485 (4 wire)	000 = 300
	10 = Odd	1 = 2 bits	1 = RS-485 (2 wire)	001 = 600
	11 = Even			010 = 1200
				011 = 2400
				100 = 4800
				101 = 9600
				110 = 19200
				111 = 38400

Word 2 cont'd	Protocol Mode
DL05/06: V7656	01110000 = No flow control
	01110001 = Xon/Xoff flow control
	01110010 = RTS flow control
	01110011 = Xon/Xoff and RTS flow control

Word 3	V-memory address for data
DL05/06: V7656	Hex value of the V-memory location to temporarily store the ASCII data coming into the PLC. Set this parameter to an unused V-memory location which has enough consecutive memory locations free to store the longest string that will come into the PLC. In order to accept the port 2 setup, a value of K0500 should be written to V7657. When the CPU has accepted it, it will generate on the same location a V0A00 is accepted or a K0E00 if there is an error.

To setup port 2 for Non-sequence (ASCII) communications with the following: RTS On-delay of 10ms; Base timeout x1; RTS Off-delay of 5ms; Odd parity; 1 Stop bit; echo suppression for RS232-C/RS422; 19,200 baud; 8 data bits; V-memory buffer starting at V2000; and no flow control, you would use the relay ladder logic shown below.



K-Sequence Communications

The K-Sequence protocol can be used for communication with *Direct*SOFT, an operator interface or any other device that can be a K-Sequence master. The DL06 PLC can be a K-Sequence **slave** on either port 1 or port 2. The DL06 PLC cannot be a K-Sequence **master.**

In order to use port 2 for K-Sequence communications you first need to set up the port using either *Direct*SOFT or ladder logic as described above.

*Direct*NET Communications

The *Direct*NET protocol can be used to communicate to another PLC or to other devices that can use the *Direct*NET protocol. The DL06 can be used as either a master, using port 2 or as a slave, using either port 1 or port 2.

In order to use port 2 for *Direct*NET communications you must first set up the port using either *Direct*SOFT or ladder logic as previously described.

For network slave operation, nothing more needs to be done. Port 2 will function as a slave unless network communications instructions are executed by the ladder logic program.

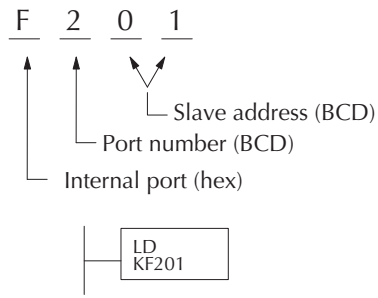
For a network master operation you will simply need to add some ladder rungs using the network communication instructions RX and/or WX. If more than one network communication instruction is used, the rungs need to be interlocked to ensure that only one communication instruction is executed at any given time. If you have just a few network communications instructions in your program, you can use discrete bits to interlock them. If you are using many network communications instructions, a counter or a shift register will be a more convenient way to interlock the instructions.

The following step-by-step procedure will provide the information necessary to set up your ladder program to receive data from a network slave.

K

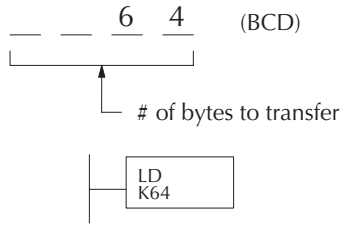
Step 1: Identify Master Port # and Slave

The first Load (LD) instruction identifies the communications port number on the network master (DL06) and the address of the slave station. This instruction can address up to 99 MODBUS slaves, or 90 *Direct*NET slaves. The format of the word is shown to the right. The **F2** in the upper byte indicates the use of the right port of the DL06 PLC, port number 2. The lower byte contains the slave address number in BCD (01 to 99).



Step 2: Load Number of Bytes to Transfer

The second Load (LD) instruction determines the number of bytes which will be transferred between the master and slave in the subsequent WX or RX instruction. The value to be loaded is in BCD format (decimal), from 1 to 128 bytes.



The number of bytes specified also depends on the type of data you want to obtain. For example, the DL06 Input points can be accessed by V-memory locations or as X input locations. However, if you only want X0 – X27, you'll have to use the X input data type because the V-memory locations can only be accessed in 2-byte increments. The following table shows the byte ranges for the various types of *Direct*LOGIC products.

DL05 / 06 / 205 / 350 / 405 Memory	Bits per unit	Bytes
V-memory	16	2
T / C current value	16	2
Inputs (X, SP)	8	1
Outputs (Y, C, Stage, T/C bits)	8	1
Scratch Pad Memory	8	1
Diagnostic Status	8	1

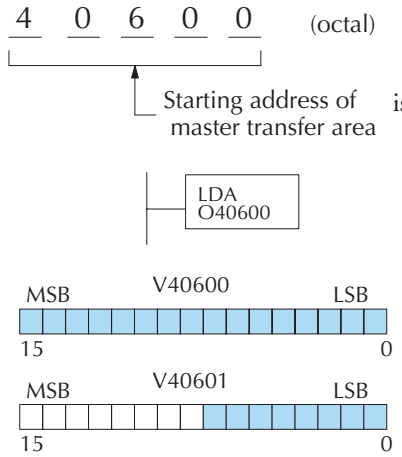
DL330 / 340 Memory	Bits per unit	Bytes
Data registers	8	1
T / C accumulator	16	2
I/O, internal relays, shift register bits, T/C bits, stage bits	1	1
Scratch Pad Memory	8	1
Diagnostic Status(5 word R/W)	16	10

Step 3: Specify Master Memory Area

The third instruction in the RX or WX sequence is a Load Address (LDA) instruction. Its purpose to load the starting address of the memory area to be transferred. Entered as an octal number, the LDA instruction converts it to hex and places the result in the accumulator.

For a WX instruction, the DL06 CPU sends the number of bytes previously specified from its memory area beginning at the LDA address specified.

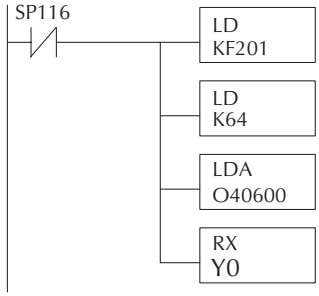
For an RX instruction, the DL06 CPU reads the number of bytes previously specified from the slave, placing the received data into its memory area beginning at the LDA address specified.



***NOTE**: Since V-memory words are always 16 bits, you may not always use the whole word. For example, if you only specify 3 bytes and you are reading Y outputs from the slave, you will only get 24 bits of data. In this case, only the 8 least significant bits of the last word location will be modified. The remaining 8 bits are not affected.*

Step 4: Specify Slave Memory Area

The last instruction in our sequence is the WX or RX instruction itself. Use WX to write to the slave, and RX to read from the slave. All four of our instructions are shown to the right. In the last instruction, you must specify the starting address and a valid data type for the slave.



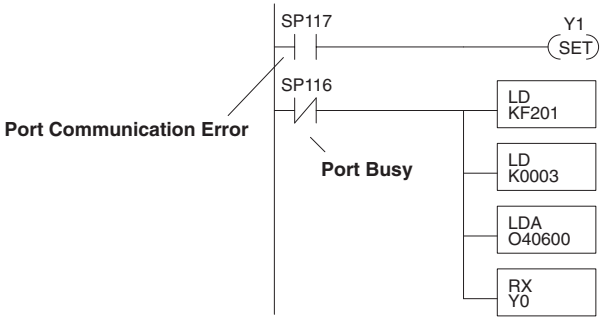
- ***Direct***NET slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS DL405, DL205, or DL06 slaves – specify the same address in the WX and RX instruction as the slave's native I/O address
- MODBUS 305 slaves – use the following table to convert DL305 addresses to MODBUS addresses

DL305 Series CPU Memory Type–to–MODBUS Cross Reference (excluding 350 CPU)					
PLC Memory Type	**PLC Base Address**	**MODBUS Base Address**	**PLC Memory Type**	**PLC Base Address**	**MODBUS Base Address**
TMR/CNT Current Values	R600	V0	TMR/CNT Status Bits	CT600	GY600
I/O Points	IO 000	GY0	Control Relays	CR160	GY160
Data Registers	R401,R400	V100	Shift Registers	SR400	GY400
Stage Status Bits (D3-330P only)	S0	GY200			

K

Communications from a Ladder Program

Typically, network communications will last longer than 1 scan. The program must wait for the communications to finish before starting the next transaction.



Port 2, which can be a master, has two Special Relay contacts associated with it (see Appendix D for comm port special relays). One indicates "Port busy"(SP116), and the other indicates "Port Communication Error"(SP117). The example above shows the use of these contacts for a network master that only reads a device (RX). The "Port Busy" bit is on while the PLC communicates with the slave. When the bit is off the program can initiate the next network request.

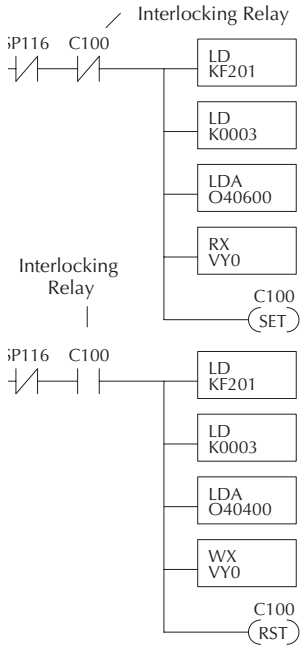
The "Port Communication Error" bit turns on when the PLC has detected an error. Use of this bit is optional. When used, it should be ahead of any network instruction boxes since the error bit is reset when an RX or WX instruction is executed.

Multiple Read and Write Interlocks

If you are using multiple reads and writes in the RLL program, you have to interlock the routines to make sure all the routines are executed. If you don't use the interlocks, then the CPU will only execute the first routine. This is because each port can only handle one transaction at a time.

In the example to the right, after the RX instruction is executed, C100 is set. When the port has finished the communication task, the second routine is executed and C100 is reset.

If you're using RLL*PLUS* Stage Programming, you can put each routine in a separate program stage to ensure proper execution and switch from stage to stage allowing only one of them to be active at a time.



K

MODBUS RTU Communications

The MODBUS RTU protocol can be used for communication with any device that uses the MODBUS RTU protocol. The protocol is very common and is probably the closest thing to an "industry standard" protocol in existence. The DL06 can be a MODBUS RTU slave on either port 1 or port 2, and it can be a MODBUS RTU master on port 2. The RS 485 wiring standard may be used on port 2 for the MODBUS RTU protocol only.

In order to use port 2 for MODBUS RTU communications you must first set up the port using either *Direct*SOFT or ladder logic as previously described.

For network slave operation, nothing more needs to be done. Port 2 will function as a slave unless network communications instructions are executed by the ladder logic program.

For network master operation the MODBUS RTU network communication instructions MRX and/or MWX must be added to some ladder rungs. If more than one network communication instruction is used, the rungs need to be interlocked to ensure that only one communication instruction is executed at any given time. If only a few network communications instructions are used in your program, discrete bits can be used to interlock them. If many network communications instructions are used, either a counter or a shift register will be a more convenient way to interlock the instructions.

INDEX

D

E

F

H

N

O

P

Q

R

S

T

U

V

W

X

Y