

Chapter 8: Ladder Logic Language Reference

I. Ladder Logic Fundamentals: Contacts, Coils, Timers and Counters

1. Contacts

Ladder logic programs mimic the electrical circuit diagrams used for wiring control systems in the electrical industry. The basic purpose of an electrical control system is to determine whether a load should be turned ON or turned OFF, under what circumstances and when it should happen. To understand a ladder program, just remember the concept of current flow - a load is turned ON when the current can flow to it and is turned OFF when the current could not flow to it.

The fundamental element of a ladder diagram is a "Contact". A contact has only two states: open or closed. An open contact breaks the current flow whereas a closed contact allows current to flow through it to the next element. The simplest contact is an On/OFF switch, which requires external force (e.g. the human hand) to activate it. Limit switches are those small switches that are placed at certain location so that when a mechanical device moves towards it, the contact will be closed and when the device moves away from it, the contact will be open.

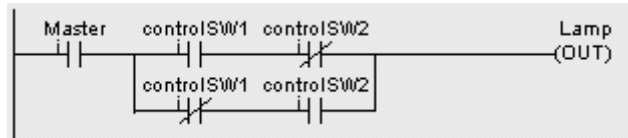
If a contact is connected to a load and the contact is closed, the load will be turned ON. This simple concept can be illustrated by the most basic ladder diagram as follow:



The vertical line on the left is the "Power" line; current must flow through the "Switch" contact in order to turn ON the load "Lamp". (In fact, there should be a second vertical line on the right end of the ladder diagram to provide a return path for the current flow, but this is omitted to simplify the circuit diagram). Now, if instead of wiring the switch to the lamp directly as suggested in the above diagram, you could connect the switch to the PLC's **input** and connect the lamp to the PLC's **output**, and then write the above ladder program to perform the same job. Of course it makes little sense to use a PLC if that is all you want to do. We will see how a PLC can simplify wiring shortly.

Note: The contact "Switch" shown in the above diagram is termed a **Normally-open (N.O.) contact**.

Now, let's say if there are 3 switches that must work together to control the lamp. A Master switch must be ON, and one of the two control switches "controlsw1" and "controlsw2" must be ON while the other must be OFF in order to turn ON the lamp (think of three-way switches in your house and you will get the idea). We can wire all 3 switches to 3 inputs of the PLC and the lamp to the output of the PLC. We can write the following ladder program to perform this task:



A contact with a "/" across its body is a **Normally-Closed (N.C.) contact**. What it means is that the ladder program is using the "inverse" of the logic state of the input to interpret the diagram.

Hence in the above ladder diagram, if "Master" and "controlSW1" are turned ON but "controlSW2" is turned OFF, the lamp will be turned ON since the inverse logic state of an OFF state "controlSW2" is true. Think of an imaginary current flowing through the "Master" contact, then through the "controlSW1" and finally through the normally-closed "controlSW2" contact to turn ON the lamp.

On the other hand, if "controlSW1" is OFF but "controlSW2" is ON, the Lamp is also turned ON because the current could flow via "Master" and then through the lower parallel branch via N.C. "controlSW1" and the N.O. "controlSW2".

Note: As you can see, although the switch "controlSW1" is connected to only 1 physical input to the PLC, but it appears twice in the ladder diagram. If you actually try to connect physical wires to implement the above circuits, both "controlSW1" and "controlSW2" will have to be of multiple poles type. But if you use a PLC, then these two switches only need to be of single-pole type since there is only one physical connection, which is to the input terminal of the PLC. But in the ladder diagram the same contact may appear as many times as you wish as if it has unlimited number of poles.

The above example may be simple but it illustrates the basic concept of logical "AND" and "OR" very clearly. "controlSW1" and "controlSW2" are connected in series and both must be TRUE for the outcome to be TRUE. Hence, this is a logical AND connection. On the other hand, either one of the two parallel branches may be used to conduct current, hence this is a logical OR connection.

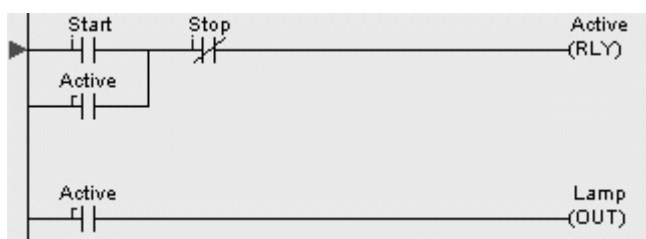
Once you understand this fundamental principle of interpreting a ladder diagram, everything should become clearer and simpler. Ladder diagram programming can be used to create a rather sophisticated control system. However, In TRiLOGI we augment its power further by allowing a ladder program to activate customized functions created in TBASIC.

2. Relay Coils

A contact can also be activated by the presence of an electrical current. This makes it possible for a control system to control the turning ON or OFF of a large load by using electrical current to activate a switch that can conduct high current. The simplest form of this type of contact is a relay.

In traditional electromagnetic relay, a coil of wire is wound around an iron core that turns it into an electromagnet. When current passes through the "coil" the magnet is "energized" and the force is used to either close a contact (that makes it a normally-open contact, closed only when energized) or open it (that will be a normally-closed contact since it is closed when not energized).

Ladder Logic programming language borrows some of those terms used to describe the electromagnetic relay for its own use. You connect a relay coil to the right end of the ladder diagram just like an output, as follow:



In a PLC, there are hundreds of internal "relays" which are supposed to behave like the typical electromagnetic relay. Unlike an output (e.g. the "Lamp" output) which has a physical connection out of the PLC, when an internal relay is turned ON, it is said to be "energized" but you will not see any changes in the PLC's physical I/Os. The logic state is kept internally in the PLC. The **contact** of the relay can then be used in the ladder diagram for turning ON or OFF of other relays or outputs. A relay contact in the ladder diagram can be Normally-Open (NO) or Normally Closed (NC) and there is no limit to the number of contacts a relay can have.

3. Out Coils

A PLC **output** is really just an internal relay with a physical connection that can supply electrical power to control an external load. Thus, like a relay, an output can also have unlimited number of contacts that can be used in the ladder program.

4. Timer Coils

A timer is a special kind of relay that, when its coil is energized, must **wait for a fixed length of time** before closing its contact. The waiting time is dependent on the "Set Value" (SV) of the timer. Once the delay time is up, the timer's N.O. contacts will be closed for as long as its coil remains energized. When the coil is de-energized (i.e. turned OFF), all the timer's N.O. contacts will be opened immediately.

However, if the coil is de-energized before the delay time is up, the timer will be reset and its contact will never be closed. When a last aborted timer is re-energized, the delay timing will restart afresh using the SV of the timer and **not** continue from the last aborted timing operation.

5. Counter Coils

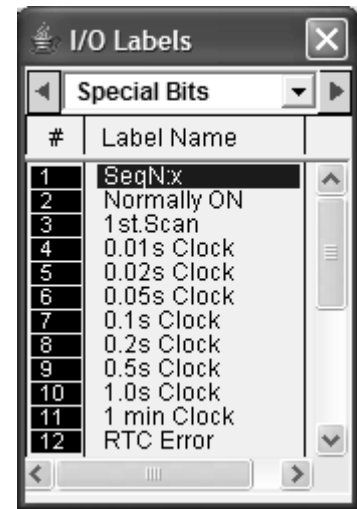
A counter is also a special kind of relay that has a programmable Set Value (SV). When a counter coil is energized for the first time after a reset, it will load the value of SV-1 into its count register. From there on, every time the counter coil is energized from OFF to ON, the counter decrements its count register value by 1. Note that the coil must go through OFF to ON cycle in order to decrement the counter. If the coil remains energized all the time, the counter will not decrement. Hence counter is suitable for counting the number of cycles an operation has gone through.

When the count register hits zero, all the counter's N.O. contacts will be turned ON. These counter contacts will remain ON regardless of whether the counter's coil is energized or not. To turn OFF these contacts, you have to reset the counter using a special counter reset function [RSctr].

II. Special Bits

TRiLOGI contains a number of special purpose bits that are useful for certain applications. These include 8 clock pulses ranging from periods of 0.01 second to 1 minute, a "Normally-ON" flag and a "First Scan Pulse", etc.

To use any of these bits, enter the ladder editor and create a "contact"; when the I/O table pops up, scroll the windows until a "Special Bits" menu pops up. *This menu is located after the "Counter Table" and before the "Input" table.* as shown below:

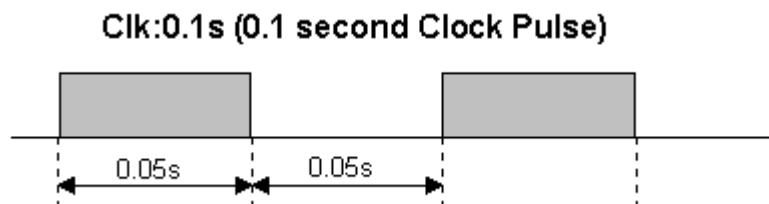


1. Clock pulse bits

The 8 clock pulses supported by TRiLOGI are:

Clock Pulse Period	Ladder Symbol
0.01 second	Clk:.01s
0.02 second	Clk:.02s
0.05 second	Clk:.05s
0.1 second	Clk:0.1s
0.2 second	Clk:0.2s
0.5 second	Clk:0.5s
1.0 second	Clk:1.0s
1 minute	Clk:1min

A clock pulse bit is ON for the first half of the rated period, then OFF for the second half. Duty cycles for these clock pulse bits are therefore 50%, as follow:



The clock pulse bits are often used with counter instructions to create timers. Additionally, they can be used as timing source for "Flasher" circuit. A reversible counter can also work with a clock pulse bit to

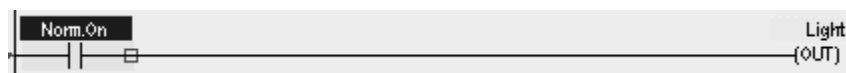
create secondary clock pulses of periods that are multiples of the basic clock pulse rate.

2. SeqN:X

These are special "Sequencer" contacts which are activated only when the step counter of a Sequencer N reaches step #X. E.g. a Normally Open contact Seq2:6 is closed only when Sequencer #2 reaches Step #6. At any other step, this contact is opened.

3. Normally ON Flag - Norm.ON

You can make use of this flag if you need to keep something permanently ON regardless of any input conditions. This is because with the exception of Interlock Off function `———[Iloff]`, a coil or a special function is not allowed to connect directly to the power line (the vertical line on the left end of the ladder diagram). If you need to permanently enable a coil, consider using the "Normally-ON" bit from the "Special Bits" menu, as follow:





4. First Scan Pulse - 1st.Scan

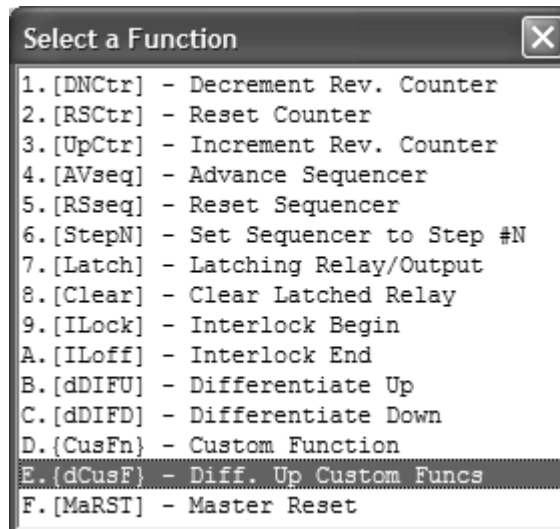
This special bit will only be turned ON in the very first scan time of the ladder program. After that it will be permanently turned OFF. This is useful if you need to initialize certain conditions at the beginning. When the program is transferred to the PLC, this bit will only be ON when the PLC is first powered up or after it has been reset.

5. Real Time Clock Error - RTC.Err

This bit is turned ON if the M-series PLC does not have battery-backed MX-RTC option and the clock has been reset due to power failure or watchdog timer reset. This gives warning to applications that require a correct real world time (such as scheduled ON/OFF operation) that the clock data is incorrect, hence enabling corrective action to be taken.


III. Special Functions

During ladder circuit editing, when you click on the  or  icon to create a special function coil, a special function menu will pop up as shown below:



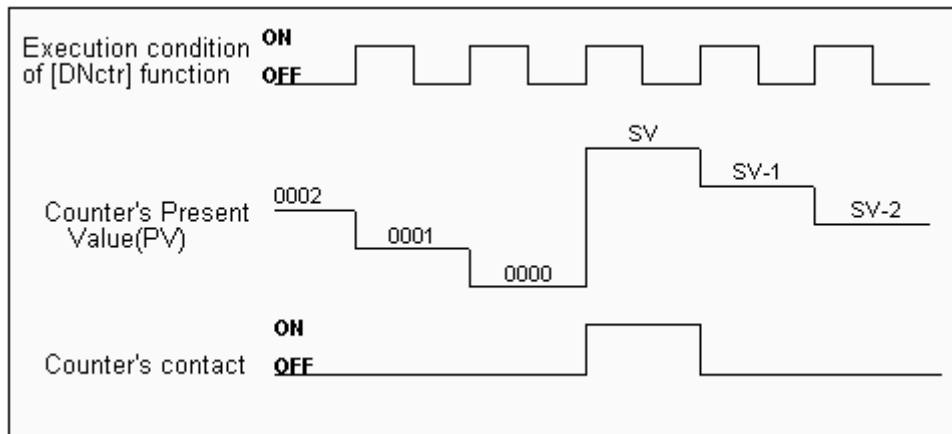
1. Reversible Counter Functions: [DNctr], [Upctr] and [RSctr]

The [DNctr], [UPctr] and [RSctr] functions work together to implement reversible counter functions on any of the 128 counters supported by TRILOGI.

The ordinary down counter (created by clicking on the  icon) essentially decrements the counter value by 1 from the "Set Value" (SV) and will stop when its count becomes zero. Unlike the ordinary down counter, a reversible counter is a circular counter that changes the counter present value (PV) between 0 and the SV. When you try to increment the counter past the "Set Value", it will **overflow** to become '0'. Likewise if you try to decrement the counter beyond '0', it will **underflow** to become the "Set Value".

All three counter functions [DNctr], [UPctr] and [RSctr] can operate on the same counter (i.e. assigned to the same counter label) on different circuits. Although these circuits may be located anywhere within the ladder program, it is recommended that the two or three functions which operate on the same counter be grouped together in the following order: [DNctr], [UPctr] and [RSctr]. Note that **NOT** all three functions need to be used to implement the reversible counter.

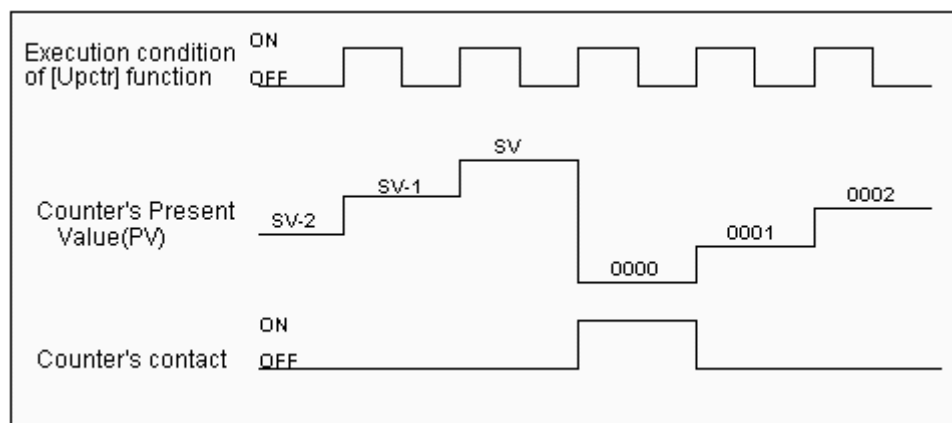
Decrement Counter [DNctr]



Each time when the execution condition of a [DNctr] function changes from OFF to ON, the present value of the designated counter is changed as follow:

- If the counter's present value (PV) is inactive, load the counter register with the "Set Value" (SV, defined in the Counter table) **minus 1**.
- If the counter's present value (PV) is already '0', then load the counter's PV with the SV defined in the counter table and turn on the counter's contact (also known as the completion flag).
- Otherwise, decrement the counter PV register by 1.

Increment Counter [Upctr]



Each time when the execution condition of an [Upctr] function changes from OFF to ON, the present value of the designated counter is affected as follow:

- If the counter is inactive, load the counter register with the number '0001'.

- b. If the counter's present value (PV) is equal to the Set Value (SV, defined in the Counter table), load the counter register with number '0000' and turn on the counter's contact (also known as the completion flag).
- c. Otherwise, increment the counter PV register by 1.

Reset Counter [RSctr]

When the execution condition of this function changes from OFF to ON, the counter will reset to inactive state. This function is used to reset both a reversible counter and an ordinary down-counter coil.

2. Sequencer Functions: [AVseq], [RSseq] and [StepN]

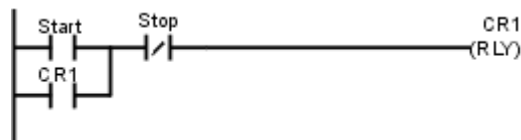
Please refer to the later section in the chapter on "Using TRILOGI Sequencers"

3. Latch Relay Function [Latch]

Latching relay is convenient for keeping the status of an execution condition even if the condition is subsequently removed. The program elements that are assigned as Latching Relays will remain ON once they are energized. Only Relays and Outputs may be assigned as Latching Relays.

On selecting [Latch] function, you can use the left/right cursor keys or click on the left/right arrow keys to move between the Relay and Output tables. The selected relay or output will now be assigned as a Latching Relay. You will be able to see the label name of the program element above the [Latch] symbol in the ladder diagram.

Although latch-relay can be used in place of self-latching (Seal) circuits, a latch-relay in an interlock section will not be cleared when the interlock occurs. Only a self-latching circuit as shown in the following will be cleared in an interlock section:



4. Clear Relay Function [Clear]

To de-energize a program element that has been latched by the [Latch] function, it is necessary to use [Clear] function. On selecting [Clear], choose the output or relay to be de-energized. When the execution condition for that circuit is ON, the designated output or

relay will be reset. In the ladder diagram, the program element label name will be shown above the [Clear] symbol.

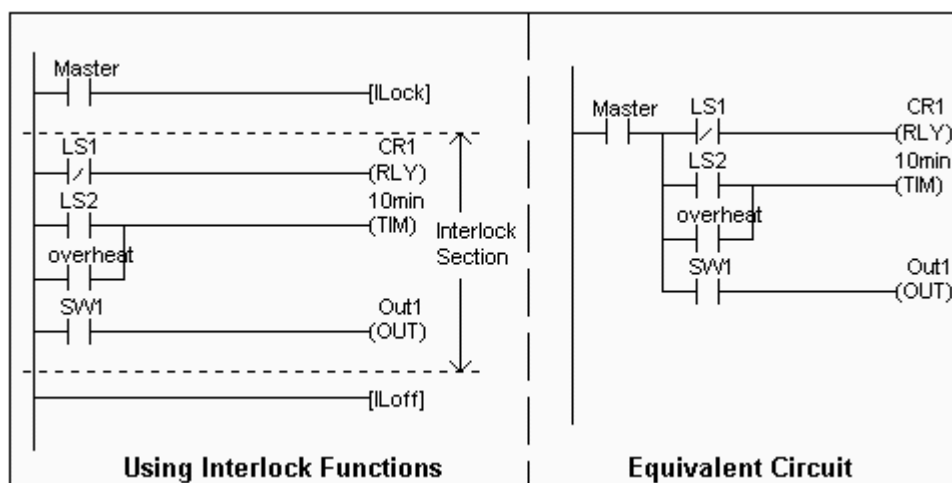
If the execution condition for [Latch] and [Clear] functions are both ON at the same time, then the effect of the designated bit depends on the relative locations of these two functions. Remember that an output or relay bit energized by [Latch] will remain ON until it is turned OFF by [Clear]. It is recommended that [Clear] circuit be placed just after the [Latch] circuit for the same output or relay controlled by these two functions. This ensures that [Clear] function has higher priority over [Latch] function, which is normally so in hardware latch-relay or other industrial PLCs.

5. Interlock [ILock]

The "Interlock" [ILock] and "Interlock Off" [LOff] functions work together to control an entire section of ladder circuits. If the execution condition of a [ILock] function is ON, the program will be executed as normal. If the execution condition of [ILock] is OFF, the program elements between the [ILock] and [LOff] will behave as follow:

- a. All output coils are turned OFF.
- b. All timers are reset to inactive.
- c. All counters retain their present values.
- d. Latched relays by [Latch] function are not affected.
- e. [dDIFU] and [dDIFD] functions are not executed.
- f. All other functions are not executed.

An Interlock section is equivalent to a master control relay controlling a number of sub-branches as follow:



Note that [LOff] is the only function that does not need to be energized by other program elements. When you use one or more [ILock] functions, there must be at least one [LOff] function before the

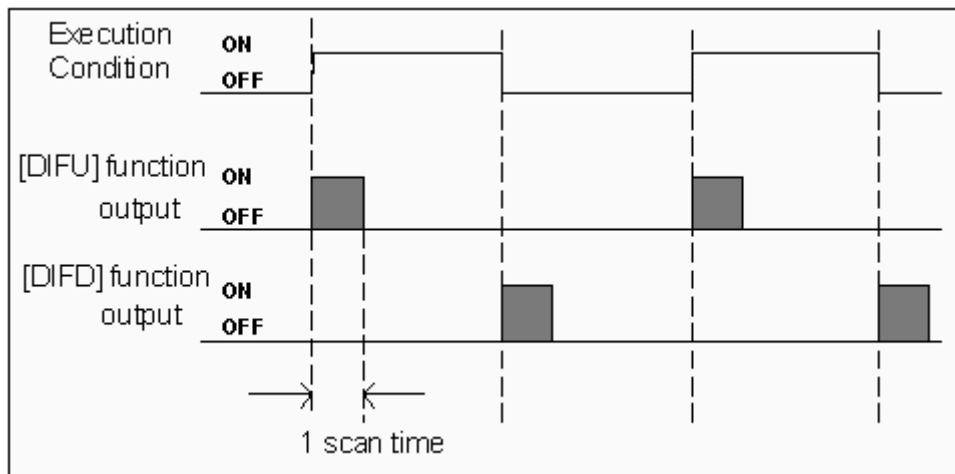
end of the program. Otherwise the compiler will warn you for the missing [LOff]. The logic simulator always clears the Interlock at the end of the scan if you omit the [LOff] function.

You can program a second or third level Interlock within an Interlock section using a few [Llock] functions. However, you only need to program one [LOff] function for the outermost Interlock section, i.e. [LOff] need not be a matching pair for an [Llock] function.

6. Differentiate Up and Down [d DIFU] and [d DIFD]

When the execution condition for [dDIFU] goes from OFF to ON, the designated output or relay will be turned ON for one scan time only. After that it will be turned OFF. This means that the function generates a single pulse for one scan time in response to the rising-edge of its execution condition. When its execution condition goes from ON to OFF nothing happens to the output or relay that it controls.

On the other hand, when the execution condition for [dDIFD] goes from ON to OFF, the designated output or relay will be turned ON for one scan time only. After that it will be turned OFF. This means that the function generates a single pulse for one scan time in response to the trailing edge of its execution condition. When its execution condition goes from ON to OFF, nothing happens to the output or relay that it controls.



7. Custom Functions: [CusFn] and [dCusF]

These two functions allow you to connect a user-defined custom function (CusFn) to the ladder logic as if it is a relay coil. Custom functions are created using the integrated text editor provided by TRILOGI Version 6.x.

8. Master Reset

An ON condition to this function clears all mailbox inputs, outputs, relays, timers and counter bits to OFF, resets all timers counters/sequencers to inactive state, and clears all latched relay bits. All integer variables will be cleared to zeros and all string variables will be assigned to empty string.

IV. Using TRILOGI Sequencers

A sequencer is a highly convenient feature for programming machines or processes that operate in fixed sequences. These machines operate in fixed, clearly distinguishable step-by-step order, starting from an initial step and progressing to the final step and then restart from the initial step again. At any moment, there must be a "step counter" to keep track of the current step number. Every step of the sequence must be accessible and can be used to trigger some action, such as turning on a motor or solenoid valve, etc.

As an example, a simple Pick-and-Place machine that can pick up a component from point 'A' to point 'B' may operate as follow:

Step #	Action
0	Wait for "Start" signal
1	Forward arm at point A
2	Close gripper
3	Retract arm at point A
4	Move arm to point B
5	Forward arm at point B
6	Open gripper
7	Retract arm at point B
8	Move arm to point A

TRILOGI Version 6 supports **eight** sequencers of 32 steps each. Each sequencer uses one of the first eight counters (Counter #1 to Counter #8) as its step counter. Any one or all of the first eight counters can be used as sequencers "Seq1" to "Seq8".

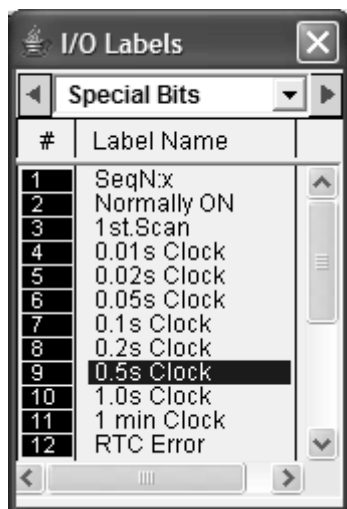
To use a sequencer, first define the sequencer name in the Counter table by pressing the <F2> key and scroll to the Counter Table. Any counter to be used as sequencer can only assume label names "Seq1" to "Seq8" corresponding to the counter numbers. For e.g. if Sequencer #5 is to be

used, Counter #5 must be defined as "Seq5". Next, enter the last step number for the program sequence in the "Value" column of the table.

Construct a circuit that uses the special function "Advance Sequencer" [AVSeq]. The first time the execution condition for the [AVseq] function goes from OFF to ON, the designated sequencer will go from inactive to step 1. Subsequent change of the sequencer's execution condition from OFF to ON will advance (increment) the sequencer by one step. This operation is actually identical to the [UPctr] instruction.

The upper limit of the step counter is determined by the "Set Value" (SV) defined in the Counter table. When the SV is reached, the next advancement of sequencer will cause it to overflow to step 0. At this time, the sequencer's contact will turn ON until the next increment of the sequencer. This contact can be used to indicate that a program has completed one cycle and is ready for a new cycle.

Accessing individual steps of the sequencer is extremely simple when programming with TRiLOGI. Simply create a "contact" (NC or NO) in ladder edit mode. When the I/O window pops up for you to pick a label, scroll to the "Special Bits" table as follow:



The "Special Bits" table is located after the "Counters" table and before the "Inputs" table.

Then click on the "SeqN:x" item to insert a sequencer bit. You will be prompted to select a sequencer from a pop-up menu. Choose the desired sequencer (1 to 8) and another dialog box will open up for you to enter the specific step number for this sequencer.

Each step of the sequencer can be programmed as a contact on the ladder diagram as "SeqN:X" where N = Sequencers # 1 to 8. X = Steps # 0 - 31.

e.g. Seq2:4 = Step #4 of Sequencer 2.

Seq5:25 = Step #25 of Sequencer 5.

Although a sequencer may go beyond Step 31 if you define a larger SV for it, only the first 32 steps can be used as contacts to the ladder logic. Hence it is necessary to limit the maximum step number to not more than 31.

1. Special Sequencer Functions

Quite a few of the ladder logic special functions are related to the use of the sequencer. These are described below:

Advance Sequencer - [AVseq]

Increment the sequencer's step counter by one until it overflows. This function is the identical to (and hence interchangeable with) the [UpCtr] function.

Resetting Sequencer - [RSseq]

The sequencer can also be reset to become inactive by the [RSseq] function at any time. Note that a sequencer that is inactive is not the same as sequencer at Step 0, as the former does not activate the SeqN:0 contact. To set the sequencer to step 0, use the [StepN] function described next.

Setting Sequencer to Step N - [StepN]

In certain applications it may be more convenient to be able to set the sequencer to a known step asynchronously. This function will set the selected sequencer to step #N, regardless of its current step number or logic state. The ability to jump steps is a very powerful feature of the sequencers.

Reversing a Sequencer

Although not available as a unique special function, a Sequencer may be stepped backward (by decrementing its step-counter) using the [DNctr] command on the counter that has been defined as a sequencer. This is useful for creating a reversible sequencer or for replacing a reversible "drum" controller.

2. Other Applications

a. Driving Stepper Motor

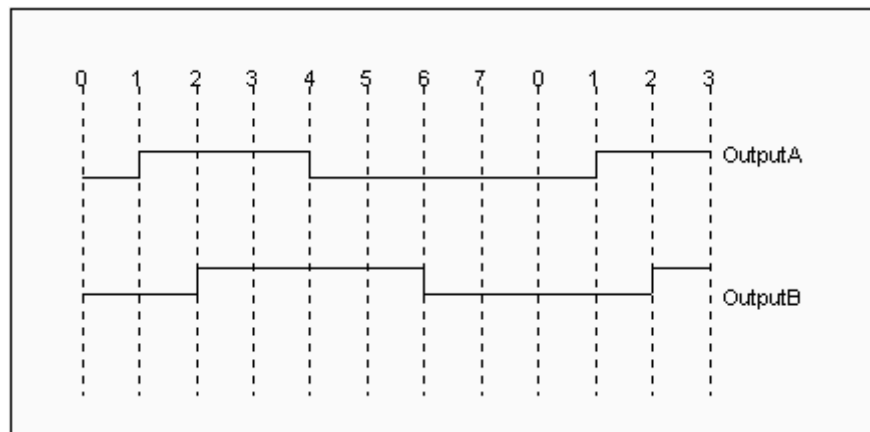
A sequencer may be used to drive a stepper motor directly. A two-phase stepper motor can be driven by four transistor outputs of the controller directly (for small motors with phase current < 0.5A) or via solid-state relays. The stepper motor can be driven using a sequencer that cycles through Step#0 to Step#3 (full-step mode) or Step#0 through Step#7 (half-step mode). Each step of the sequencer is used to energize different phases of the stepper motor. A clock source is needed to drive the stepper motor through

its stepping sequence. The stepping rate is determined by the frequency (which is equal to $1/\text{period}$) of the clock source.

Clock pulses with periods in multiples of 0.01 second can be generated easily using the "Clk:.01s" bit and an [Upctr] function. For e.g., to generate a clock source of period = 0.05s, use "Clk.01s" to feed to an [Upctr] counter with Set Value = 4. The counter's contact (completion flag) will be turned ON once every 5 counts (0,1,2,3,4), which is equivalent to a 0.05 sec. clock source.

b. Replacing a Drum Controller

A drum controller can be replaced easily by a sequencer if the timing of the drum's outputs can be divided into discrete steps. Assuming a drum controls two outputs with the timing diagram shown in the following figure:



This can be replaced by an 8-step sequencer. Step 1 (e.g "Seq1:1") turns ON and latch Output A using [Latch] function, Step 2 turns ON and latch Output B, Step 4 turns OFF Output A using the [Clear] function, and Step 6 turns OFF Output B. All other steps (3,5,7,0) have no connection.

3. Program Example

Assume that we wish to create a running light pattern which turns on the LED of Outputs 1 to 4 one at a time every second in the following order: LED1, LED2, LED3, LED4, LED4, LED3, LED2, LED1, all LED OFF and then restart the cycle again. This can be easily accomplished with the program shown in Figure 6.9.

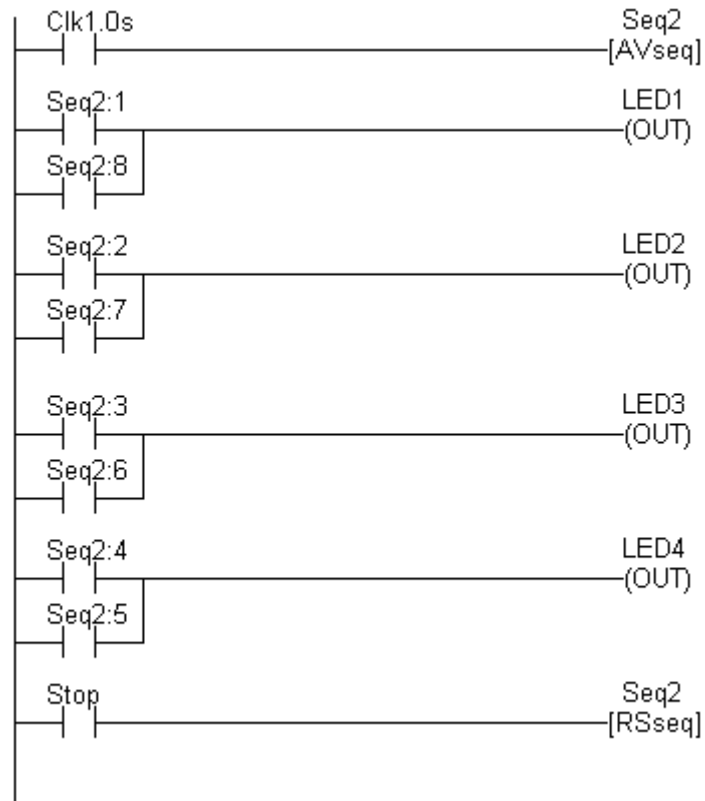


Figure 6.9

The 1.0s clock pulse bit will advance (increment) Sequencer #2 by one step every second. Sequencer 2 should be defined with Set Value = 8. Each step of the sequencer is used as a normally open contact to turn on the desired LED for the step. A "Stop" input resets the sequencer asynchronously. When the sequencer counts to eight, it will become Step 0. Since none of the LED is turned ON by Step 0, all LEDs will be OFF.