SHAPING THE FUTURE OF INDUSTRIAL AUTOMATION

OPTO 22

LADDER LOGIC/ FLOWCHART PROGRAMMING DIFFERENCES AND EXAMPLES

APPLICATION NOTE

Form 921-970228

overview

This document is designed as a quick-start primer to assist industrial automation programmers who are familiar with PLCs and Relay Ladder Logic programming to better understand the corresponding principles and techniques for constructing OptoControl programs. Also refer to document entitled *Solution Provider White Paper - A Comparison of PLCs and ladder logic vs. Opto 22's hardware and software.*

Control Basics

A control program in any language is simply a series of questions and commands. However, the way in which RLL and OptoControl allows the programmer to organize these questions and instructions differ radically.

RLL - The Scan and Rungs

There are fundamental differences in the methods in which PLCs and Opto controllers interact with their I/O systems. PLCs are scan-based processors: that is the cycle for solving logic is performed by reading all the inputs, processing user logic (solve the rungs), and then writing the outputs and updating registers.

In RLL a question/command operation is represented by the rung. The left-hand side of the rung looks for a specific condition and the right hand side performs the required action. Programs are assembled by creating many of these rungs and then getting them to execute in the proper sequence. Due to the PLC's scan-based logic solve method, RLL requires the programmer to think about the control process in a parallel fashion. That is, the programmer must remember that all the questions are being asked with every program scan.

Opto 22 - Immediate Mode and Flowcharts

All instructions in OptoControl operate in immediate mode and only the data required at that point in the code is accessed. When the processor is instructed to see if a switch is on, it immediately reads the input and when instructed to turn on an output, it immediately does so. This allows the programmer to focus on the sequence steps without regard to the rest of the program data.

OptoControl is a flowchart-based programming tool that allows the user to design the question/command control logic using a combination of four high-level constructs:

Action Blocks - Rectangles that contain a list of command instructions. Actions may have many entrances, but only one exit. (i.e. *Turn On*, *Turn Off, and Start Timer)*

Condition Blocks - Diamonds that contain a list of questions that are used to direct logic flow. Conditions may have multiple entrances and two exits: TRUE is all questions test true and FALSE if any of the questions test FALSE. (i.e. *On?, Greater Than?, and True?)*

Continue Blocks - Ovals provide the ability to jump to Action or Condition blocks. They contain the name of the destination block. They are used to improve chart readability.

Connections - Lines with arrows that connect the blocks together in the desired logic flow sequence.

English-based commands are then inserted into these constructs to provide the detailed program instructions. As many blocks as needed are assembled into the proper sequence to create a *Chart* that controls a specific aspect of the process. A real-time multitasking operating system provides the ability to run many charts simultaneously. An OptoControl program or “strategy” is constructed by breaking the process operation into logical segments, designing charts to control each segment, and then running the charts in the desired control sequence.

Control Options

Unlike the typical PLC where the majority of logic execution is performed by the processor, there are three basic ways that control can be executed in an Opto system:

1. Processor Control

The first is by the host processor solving the flowchart in an event-driven manner. There is no PLC-like scan on Opto controllers. The logic is solved sequentially as the code is executed. The inputs are polled and outputs are updated as needed in a command and response (processor to I/O) fashion. This arrangement is highly efficient because the only I/O being read or written is that which is necessary to solve a particular section of code. The processor does not waste time manipulating I/O that is not part of the current process sequence.

2. On-Board I/O Intelligence

The second way control is executed on an Opto system is by the intelligent I/O brain boards that act as local controllers on each I/O unit. Each brain board has its own microprocessor and custom ASIC to provide local control functionality. No programming is required to use most of these functions. It is simply a matter of configuring the input or output to enable the underlying control features. Thermocouple inputs for example are linearized and converted to degrees in Centigrade or Fahrenheit by the brain board. When the processor reads the input, there is no additional host processing needed to obtain the temperature. Other functions are enabled by selecting the appropriate function from the Features list box when a point is added to the program.

3. Event/Reactions

An event/reaction is a powerful and unique feature that allows users to “off-load” or distribute control logic to an intelligent I/O brain board. That is, some of the logic in a strategy can be run on the I/O unit independently of the controller.

As the name suggests, an event/reaction consists of an event and a corresponding reaction. Each time an event becomes true, its corresponding reaction is executed once. The event is a user-defined state that the I/O unit can recognize. The defined state can be a combination of values, inputs, and outputs. On a digital multifunction I/O unit, for example, any pattern of input and output states (on and off) can constitute an event. On an analog I/O unit, an event could occur when an input channel attains a reading greater than a selected value. Examples of reactions include turning on or off a set of outputs, ramping an analog output, and enabling or disabling other event/reactions.

Intelligent I/O also has the capability to react to a specified event by generating a hardware interrupt at the host through a separate communication link. When the host processor receives an interrupt, it usually starts a suspended task called the Interrupt Chart, where user-defined logic in that chart is immediately executed. This is can be extremely useful for handling alarm conditions and situations where an event on one I/O unit must quickly cause a reaction on another I/O unit.

Control Options (cont.)

Event/reactions are stored in each I/O unit. As soon as power is applied to the I/O unit, all event/reactions for which scanning is enabled are scanned continuously in the same order in which they were configured in OptoControl. Since each I/O unit can be configured with up to 256 event/reactions, complex tasks and sequences can be performed.

Files, Addresses and Tagnames

One of the primary differences between RLL and OptoControl is the way program data is organized and accessed. RLL segments program information into files in memory with each file type having a specific function. Addressing is then used to get at specific data within each file. Addressing is a combination of the file type and the register location within the file. For example, Allen Bradley PLCs use the following address convention: F8:100 means - file F8 (File 8 or Floating Point file) at register 100. All program data is then manipulated by specifying the necessary file address and item (register, bit, etc.).

OptoControl abstracts control data by allowing the user to provide descriptive tagnames for all data elements. Understanding how low level memory storage operates is not necessary because configuring and manipulating data is transparently handled by OptoControl. To create a floating point variable, the programmer pulls down the *Configure Variable* menu, selects *Float* and gives it a name. All access to that variable is provided by referencing the tagname.

Mnemonic Instructions and Commands

In Relay Ladder programming functions such as reading inputs and writing output, timers and counters are handled by Mnemonic Instructions. Unless special I/O modules are used (i.e. Hardware Counter) these functions are performed in software by the PLC's processor.

With Opto 22 systems, many of these functions are imbedded into the intelligent I/O and accessed by Action or Condition Block commands. For example any intelligent input can act as a high-speed counter and count asynchronously from the host processor. Understanding how these functions are implemented will allow the programmer to take advantage of the added I/O features and benefits. Please take a look at the Overview chapter in the *OptoControl Command Reference* manual for more detailed explanations on using these features.

We will examine a typical PLC's implementation of these instructions and then explain how each is accomplished with Opto 22 control systems.

Digital Inputs and Outputs

RLL

Inputs and internal registers used as control bits are placed on the left-hand side of the rung and evaluated with the -| |- symbol (read as *Examine if Closed)* and the -|/|- symbol (read as *Examine if Open).*

In RLL the -( )- symbol represents an output coil. Coils toggle bits in memory locations and are used to control physical outputs and internal registers. Coils are placed on the right-hand side of a rung and are only executed if the input logic on the left-hand side of the rung is True.

Mnemonic Instructions and Commands (cont.)

OptoControl

The status of inputs is evaluated by using a condition block with the *On?* or *Off?* instruction.

Controlling outputs is accomplished by using an Action block with the *Turn On* or *Turn Off* command. One of the major differences in controlling I/O between RLL and OptoControl is that **all** Outputs and Variables are retentive and hold their state until instructed to change.

Analog Inputs and Outputs

Analog I/O is handled much in the same way with both systems. To read I/O status, mathematical compare functions are used (i.e. *Equal, Greater Than, Less Than,* etc.). If analog information needs to be logged to memory or transferred from an input to an output, the Move instruction is used. Writing analog outputs is also similarly handled with the *Move* instruction. Analog outputs are always retentive.

Latches

RLL

A latch in RLL is an output function. It is used to make digital outputs and internal registers retentive. The output holds its state until changed using an unlatch instruction.

OptoControl

Latches in OptoControl are hardware input functions executed by the intelligent I/O units. Latches are used to sense momentary high-speed signals. If a latch is enabled, it will read False until it sees a transition. Once the input sees a pulse, it will read True until it is cleared.

Examples

Included are a few examples of common PLC programs and a corresponding flowchart program.

Timers and Delays

As the name indicates, Timers are used to measure the time period between two events. In RLL, Timers are also used to provide pauses between logic steps. OptoControl provides Timers and also the Action Command *Delay* for logic pauses, which is more efficient than a Timer.

RLL

Timers are implemented as instructions in RLL and typically work by counting up from zero to a user-defined value. They are treated as output devices and are placed on the right-hand side of a rung.

Configuration

RLL timers typically have two configuration settings: Time Base - timer resolution typically in seconds, and the Preset - number of timer ticks (in time base resolution) to count. Total Timer Duration = Time Base x Preset.

Using PLC Timers

An RLL timer starts counting as soon as the rung circuit it is placed in becomes true. There are typically two runtime parameters used to utilize timers: Timer Done - a bit that is set TRUE when the timer has finished, and the Accumulator - a register that is used to check the current number of counts. Some RLL implementations also have a CLR or Clear Accumulator instruction that allows the Timer's Accumulator to be reset during program execution.

OptoControl

Timers

Timers in OptoControl are a special type of Numeric Variable and work by Moving a number (float or integer) into them. The timer then counts down from that number and stops when it reaches zero.

Configuration

Timers are created in OptoControl by selecting *Timer* from *Type* list box in the *Add Variable* dialog. Timers are measured in seconds with 1 millisecond resolution and offer a range of 0.001 to 4.611686 x 1015 seconds.

Using OptoControl Timers

A timer is automatically started by using the *Move* command to place the desired time value into the timer variable. Any command that references numeric variables can be used to examine or manipulate timers. The *Timer Expired?* condition command is used to determine when the timer has finished and tests True when the timer reaches zero in its countdown.

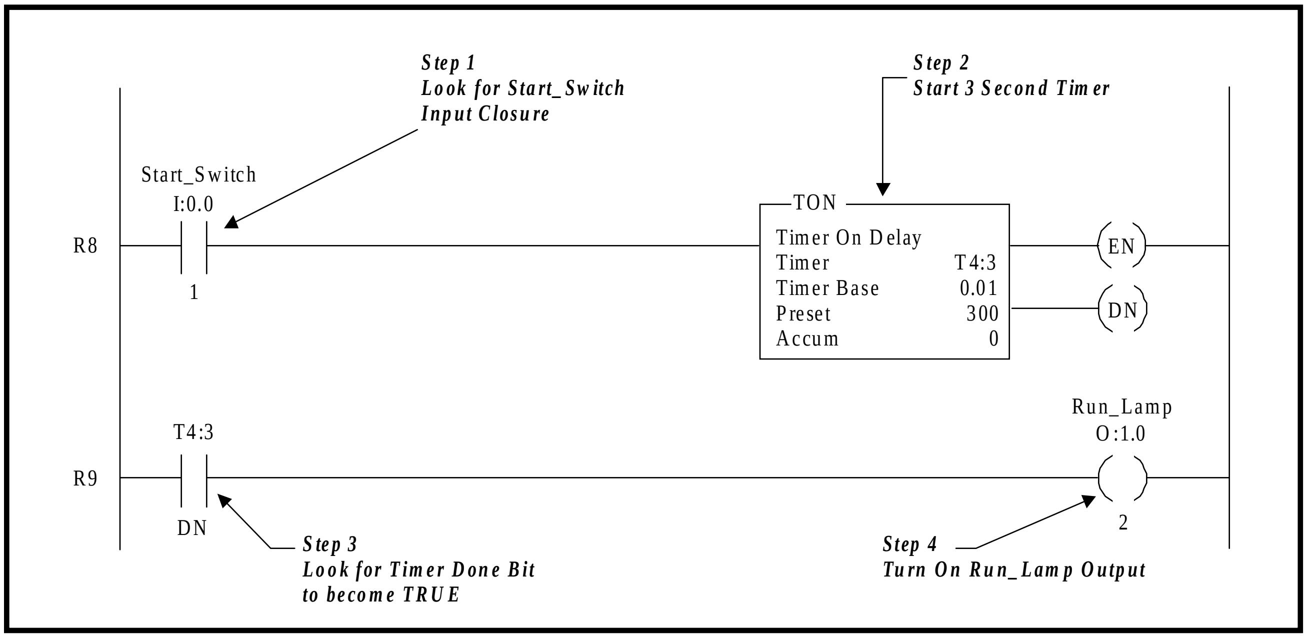
Delays

The *Delay* Action command in OptoControl can be used in place of a Timer to provide a time-specific pause between logic steps. This command provides better processor efficiency and is simpler to implement than a timer.

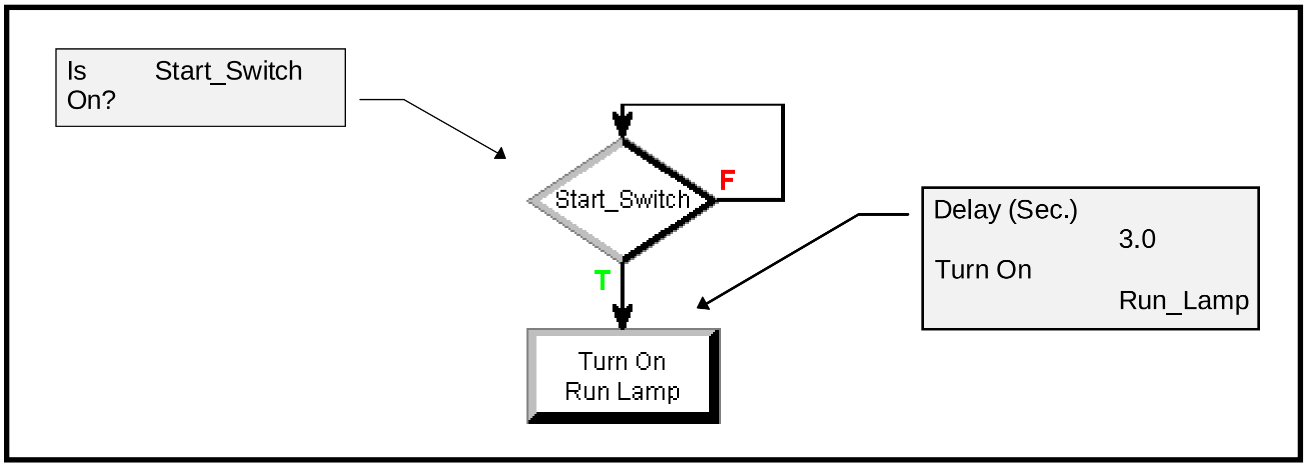
RLL Timer and OptoControl Delay Comparison

A simple application requires that an output is turned on 3 seconds after an input closure. This example shows how an RLL timer and the OptoControl *Delay* command provide the 3-second delay.

Relay Ladder Logic



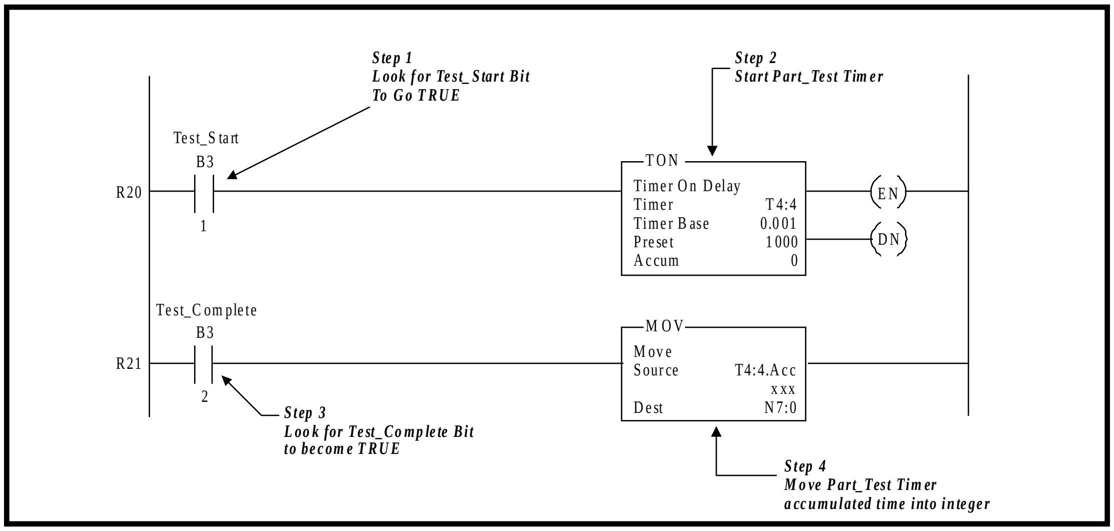
OptoControl



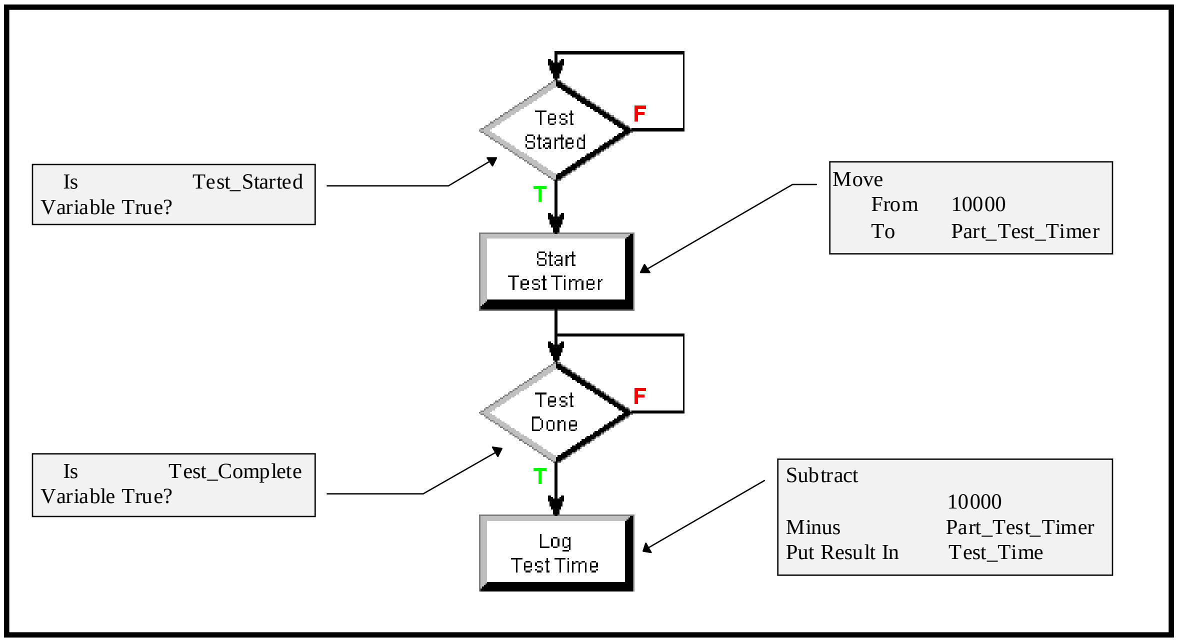
RLL and OptoControl Timer Comparison

This example illustrates the use of Timers to measure the time between two events. A timer is utilized to measure how long it takes a Widget tester to check a part.

Relay Ladder Logic



OptoControl


Test Started
F
T
Is Test_Started Variable True?
Start Test Timer
Move From 10000 To Part_Test_Timer
Test Done
F
T
Is Test_Complete Variable True?
Log Test Time
Subtract 10000 Minus Part_Test_Timer Put Result In Test_Time

Note: OptoControl Timers count down from a preset. To obtain the expired time, subtract the current time from the original preset.

Counters

RLL Counters

Counters typically come in two types: a Count Up and a Count Down. They are treated as output devices and are placed on the right-hand side of a rung.

Configuration

RLL Counters typically have one configurable setting: Preset - the desired number of counts to trigger the Done bit.

Using PLC Timers

There are typically four runtime parameters used to utilize counters:

1. Counter Done Bit - set TRUE when the Preset has been reached.

2. Enable Bit - follows Rung - set True when the Rung is True and False when rung is False.

3. Overflow (Count Up) and Underflow (Count Down) Bit - Overflow Bit is set True when Accumulator reaches the positive Counter limit on an Count Up counter. Underflow Bit is set True when Accumulator reaches the negative limit of possible Counter values on a Count Down counter.

4. Accumulator - a register used to hold the current number of counts.

Some RLL implementations also have a CLR or Clear Accumulator instruction that allows the Counter's Accumulator to be reset during program execution.

A *Count Up* Counter starts with the Accumulator set to 0. For each On to Off transition, the Accumulator is then incremented by one. The Done bit is set when the number of counts in the Accumulator equals the Preset.

A *Count Down* Counter starts by first moving the Preset into the Accumulator. For each On to Off transition, the Accumulator is then decremented by one. When the Accumulator reaches 0, the Done bit is set.

OptoControl Counters

Counters

Counters in OptoControl are implemented by intelligent I/O. Counters start from 0 and count up.

Configuration

Counters are created in OptoControl by selecting *Counter* from the *Features* list box in the *Add Digital Point* dialog. Although the input now has the *Counter* feature enabled, the *On?, Off?, and Latch* commands that operate on regular digital inputs are always available.

Using OptoControl Counters

A Counter is manipulated with six commands:

1. Start Counter - starts counter with no effect on previously accumulated counts

2. Stop Counter - stops counter with no effect on previously accumulated counts

3. Get Counter - moves current accumulated counts to a variable

4. Clear Counter - resets counter to 0 with no effect on Counter run status

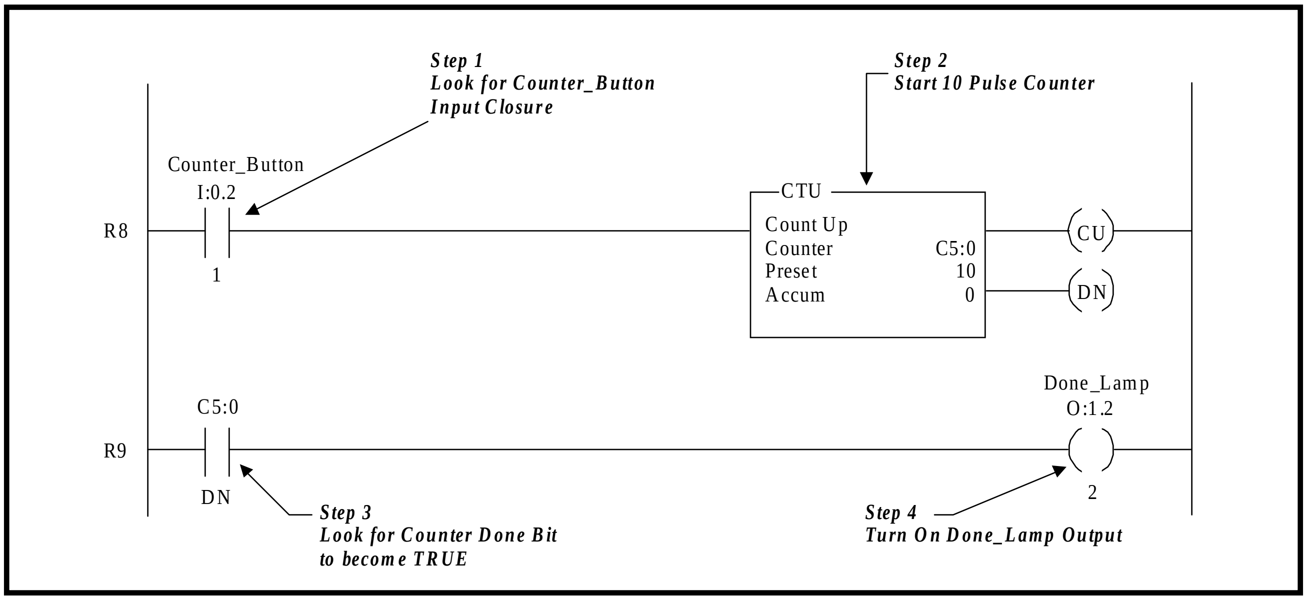
5. & 6. Get and Clear Counter - moves current accumulated counts to a variable and clears the counter.

In addition to these Counter *only* commands, any counter can be accessed directly via many of the common Variable and I/O commands. For example, a counter can be tested for a desired value by using the *Is Equal?* Condition command or the current input status can be checked using the *On?* command.

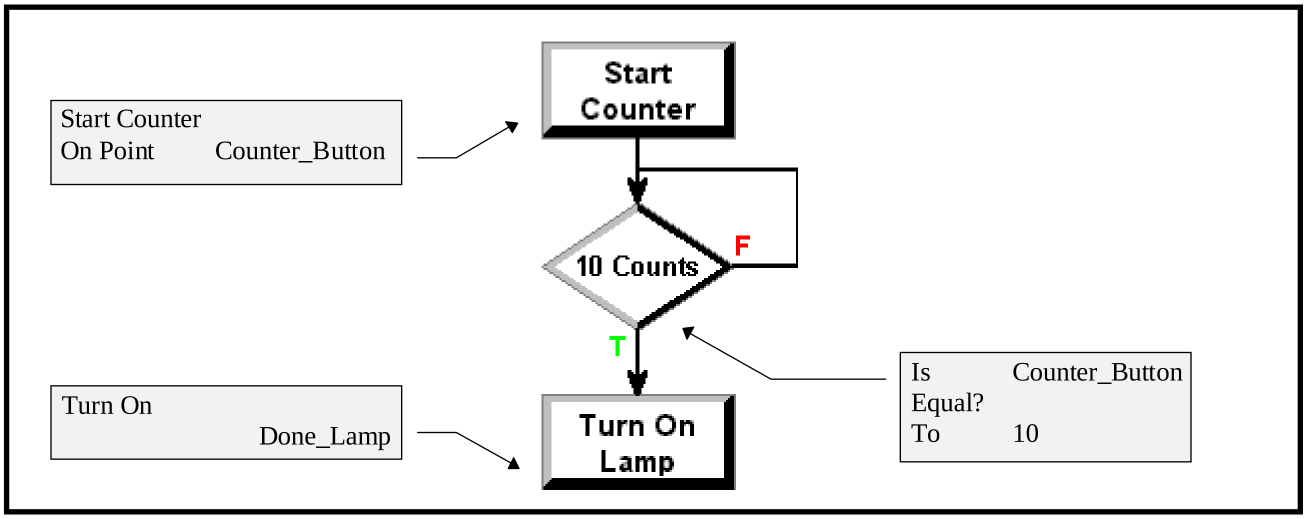
RLL and OptoControl Counter Comparison

An application requires that an output is turned on after a push button has turned on and off 10 times. This example shows how use a RLL Counter and an OptoControl Counter to count the button pushes.

Relay Ladder Logic



OptoControl


Start Counter
On Point Counter_Button
Start Counter
10 Counts
F
T
Turn On Lamp
Turn On
Done_Lamp
Is Counter_Button
Equal?
To 10