PLC Basics

PLC programs do not behave like scripted language programs used by most computer applications. A basic understanding of control application fundamentals is crucial for designing an effective support-focused system. Link to parts 1, 2, 3, online.

Before effectively producing a support-focused design that can efficiently interact with control applications, the designer must first have a basic understanding of control application fundamentals. However, before introducing control application fundamentals, this section of the article focuses on the basics of PLC programming. The material presented provides strategists with the rudimentary knowledge needed to recognize:

1. Common programming instructions
2. Control application terms
3. Potential of machine controller programs
4. Hard-coded and hardwired logic circuits
5. Programming techniques that affect circuit styles
6. Circuit substitution techniques.

It is often easier for computer programmers to think of ladder logic programs as lists of Boolean equations. Individual groups of equations make up separate controller applications. The machine controller constantly scans and evaluates each equation in each application.

If the examinable equation conditions are true, the programmed output instruction acts to enable or disable an internal signal, output signal, or function variable. Each application shares a fraction of each program scan. The net effect is equivalent to parallel processing all resident machine controller applications.

Circuits used to control the movement of a mechanism or object can be a logic circuit, which is a line of code that is part of a machine controller application, or an external circuit, which is any electric, pneumatic, optic, or hydraulic circuit that reacts to a machine controller's output signal to move an object or mechanism.

External circuits are physical circuits, while logic circuits are internal to a machine controller. The "external circuit" term refers to the arrangement of real-world devices that react to a controller's enabled output signal. Externally, output signals are connection points on a controller's output module. External circuits use wires from output connection points to enable relays, valves, and switches to activate a hydraulic, pneumatic, optic, and/or electric circuit. Logic circuits are lines of code with examinable elements and programmable resultants. Elements are discrete signals or application variables.

Each logic circuit enables one or more resultants. A resultant is typically an enabled or disabled signal or a changeable register-stored value. It is common to have logic circuits that unconditionally enable resultants. Resultants for some logic circuits are output signals, which when enabled will activate external circuits. Designers often place examinable elements in series on a logic circuit to denote AND conditions. Instead of parenthesis, programmers use branches with one or more parallel elements to denote OR conditions.

The following definitions describe some commonly used logic circuit programming terms:

- **Bit:** a machine controller's smallest Boolean variable having only two values, either one (1) or zero (0)
- **Word:** a machine controller's multi-bit (16, 32, 64, etc.) register used to store the contents of an application variable
- **Instruction:** a single, programmable operation used to examine, compare, enable, disable, move, set, reset, or manipulate a bit or word value
- **Bit instruction:** an operation used to examine, enable, or disable a bit value
- **Word instruction:** an operation used to compare, move, set, reset, or manipulate a word value
- **File instruction:** an operation used to compare, move, set, reset, or manipulate a multi-word data array
- **Special instruction:** a unique programmable operation designed to support a specific application purpose.

Figure 1 shows some instruction symbols and their associated bit, word, file, and special instruction names. These common instructions are compatible with most application variables. Bit instructions represent examinable or changeable discrete signals. These signals denote the

Key concepts

- The engineer must have a basic understanding of control application fundamentals.
- PLC operating systems emphasize the continuous and high-speed scanning of all lines of code.
- Each machine controller manufacturer is likely to provide a programming environment with differing sets of instructions.

state of an internal, input, and output variable.

An internal signal is a variable enabled by one circuit and examinable by others. An input signal typically represents the enabled or disabled state of a sensor, pushbutton, selector switch, or device. An output signal generally represents the activated or deactivated state of a light, external relay, or solenoid valve. One-shot, time-on timer, and retentive timer instructions are examples of special programming instructions.

A one-shot instruction enables designers to develop trigger circuits. A time-on timer instruction enables control applications to accumulate time when rung conditions are true. Unlike the time-on timer, a retentive timer holds the accumulated time value when the instruction is disabled. Word instructions are compatible with instructions that compare or manipulate multi-bit instructions.

File instructions support manipulation of contiguous groups of words. Most machine controller instructions allow applications to examine and manipulate variables associated with eight-bit bytes, four-bit nibbles, and multi-bit application variables. When a precondition transitions to a false state, the machine controller disables the signal related to the output instruction.

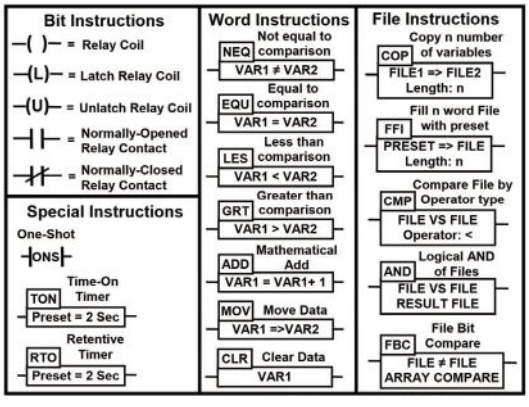

Figure 1 *shows some instruction symbols and their associated bit, word, file, and special instruction names. All images courtesy: Daniel Cardinal*

Relay coil and latch/unlatch coil instructions, when used, produce five types of circuits: 1) Setup circuit: a logic circuit that examines one or more serial or parallel conditions to enable the coil assigned discrete signal; 2) Seal circuit: a setup circuit that examines the rung's relay coil assigned discrete signal as a parallel condition around one or more other conditions that are expected to change state when the circuit is enabled; 3) Latch circuit: a logic circuit that examines various serial and/or parallel conditions before setting a signal; 4) Unlatch circuit: a logic circuit design that examines various serial and/or parallel conditions before resetting a signal; 5) Timer circuit: a logic circuit that examines various serial and/or parallel conditions before enabling a timer instruction.

Setup and seal circuits differ in two important ways. Unlike a seal circuit, a setup circuit will never include the rung's coil signal as an examinable precondition. A seal circuit always includes at least one condition needed to break the seal. Designers sometimes refer to a setup circuit as a summation circuit because it sums together all the serial and parallel conditions needed to enable the coil-assigned signal. Designers use a setup circuit when they expect any conditions to change state, and they want to keep the signal enabled until one does.

For seal circuits, designers simply OR the conditions they expect to change state with a normally opened contact enabled by the coil-assigned signal. Designers also add at least one other AND condition that must change its signal state or variable value when it is time to disable the sealed signal.

Various timer circuit terms sometimes cause confusion among control system designers. A circuit that uses a time-on instruction accumulates an elapsed time value when the rung conditions are true. When the circuit is false, the accumulated value automatically resets. Some designers use math-based add or retentive timer instructions to accumulate and hold elapsed time information. These circuit designs need special word instructions that reset or clear retained time information.

Designers usually apply timer circuits to produce one of the following two signal conditioning circuits: 1) Dwell circuit: a timer circuit that times the steady-state condition of a signal before enabling another circuit to examine the signal; 2) De-bounce circuit: a timer circuit that times a steady-state condition of a signal before it enables another circuit to react to the signal's new state.

Designers usually apply dwell and de-bounce circuits to time the state of a sensor's input signal. The difference in each timer circuit is what each declares when an accumulated elapsed time value is greater than a preset value. A dwell circuit declares the sensor input signal stabilized and ready for use, whereas a de-bounce circuit declares the sensor input signal qualified to change state for the next sensed object or mechanism.

Each machine controller manufacturer is likely to provide a programming environment that is equipped with differing sets of instructions. However, most environments use bit instructions, while others force programmers to create special add-on instructions (AOI) to manipulate word data or enable complex algorithms.

Most controller manufacturers provide an environment with a standard set of instructions that allow programmers to enhance the ability of simple relay applications. Programmers use relay instructions to control the movements of objects and mechanisms.

Programmers use bit instructions to enable other internal and external processes. For the most part, programmers use word and file instructions to develop ancillary applications that manipulate data. Shift-register, reader applications, get-next, and communication drivers are examples of an ancillary application.

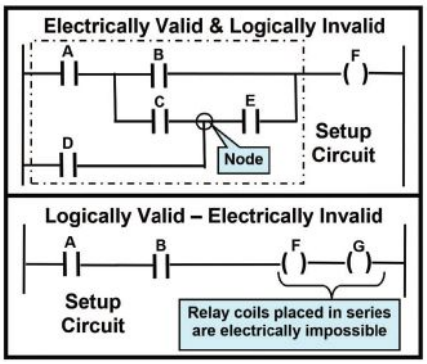

Figure 2: *Designers usually apply dwell and de-bounce circuits to time the state of a sensor's input signal.*

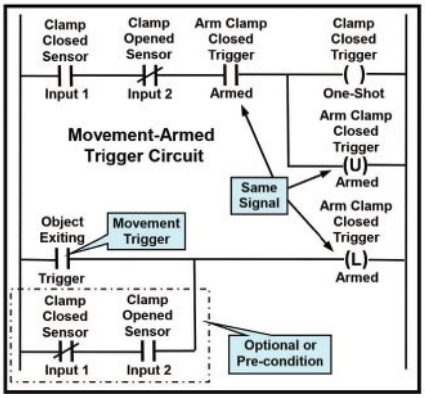

Figure 3 *shows an extremely reliable alternative to a latch-protected trigger circuit.*

Although latch-protected trigger circuit designs work for most programs, they still can re-enable the trigger for the same part. This happens when someone deliberately moves a mechanism to unlatch the block signal. Depending on the actual application, re-enabling the trigger may not be desirable. If re-firing the trigger does not cause any undesirable affects, these circuits do not create application anomalies. If repeat triggering creates anomalies, designers often leave applications to the chance someone will manually move the mechanism.

Figure 3 shows a reliable alternative to a latch-protected trigger circuit. Without the optional parallel branch, the base circuit does not allow the re-enabling of the trigger. The circuit's preconditions force the design to rely on a movement detection trigger to re-enable the arming signal.

If it is desirable to repeat fire the trigger, the programmer merely includes the OR branch to provide an alternate way to re-arm the trigger. The base circuit design ensures the ancillary trigger will only fire once per part.

Technical benefits to using instruction-based circuits over coil-blocked circuits would have nothing to do with decreasing the chaotic nature of control applications. The latch-protected circuit provides some added false trigger protection without using a one-shot instruction. The movement-armed circuit ensures the highest degree of reliability.

The instruction-based method provides programmers with a redundant way to generate a coil-blocked trigger. It is easy to conclude that the reaction by machine controller manufacturers to add a one-shot instruction to their suite of bit level instructions increases the chance designers will arbitrarily use them to produce less reliable triggers. Their use in control applications only increases the chaotic nature of control applications.

Signal-less one-shot circuits do not produce trigger signals that are examinable by other circuits. These circuits typically latch or unlatch a discrete signal. Some circuits simply change or move data when the preconditions are correct.

The lack of an examinable signal makes it difficult for other designers, support personnel, or controls integrators to discover the trigger conditions used. In most cases, signal-less one-shot circuits increase the probability that someone will create additional trigger circuits to accommodate their own applications. Their failure to reproduce the same circuit behavior guarantees the increased chaotic nature of the application. **ce**

- Daniel B. Cardinal is systems engineer with InSyte Inc.; edited by Joy Chang, digital project manager, Control Engineering, *jchang@cfemedia.com.*

Go Online

Go online to read the full article and part 1-3 of the series.

PART 1: Support-focused enterprise controls series

PART 2: Support-focused enterprise controls: Object detection for automotive automation

PART 3: Support-focused enterprise controls: Control system triggers

Consider this...

Did controller manufacturers react to their customers properly by providing programmers with one-shot instructions?