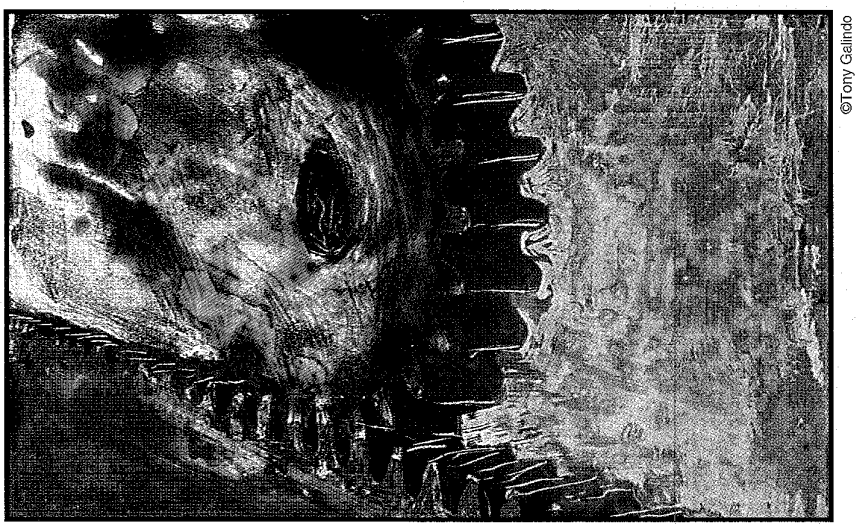
Programmable logic controllers

The workhorse of factory automation keeps things on track



©Tony Gelindo

Programmable Logic Controllers are at the forefront of manufacturing automation. Many factories use Programmable Logic Controllers to cut production costs and/or increase quality.

Since its predecessor was hard-wired relay panels, the Programmable Logic Controller uses a unique language called ladder logic. Although other languages are used, ladder logic presently remains the dominant language of automation. The Programmable Logic Controller (PLC) is sometimes called a Programmable Controller (PC), but the abbreviation PLC is preferred to distinguish it from the Personal Computer.

PLCs developed out of the need to replace the hard-wired relay panels. In the 1960s, a typical automated assembly or other manufacturing line had a cabinet of relays wired to control the operation. As one might expect, debugging relay failures could be time-consuming, and changing functionality by modifying the sequence of operations was time-consuming and costly because of the required rewiring.

In 1968, the Hydramatic Division of General Motors Corporation (GM) specified design criteria for the first PLC. (They had to rewire many relay panels annually for car model year changes.) Some major specifications were:

1. Easily programmed and reprogrammed, preferably in-plant to alter its sequence of operations.
2. Easily maintained and repaired—preferably with plug-in modules.
3. Capable of operation in a plant environment.
4. Smaller than relay equivalent.
5. Capable of communicating with central data collection system.
6. Cost-competitive with solid-state and relay logic systems then in use.

A handful of companies responded to develop the device we now call a PLC in late 1969 and early 1970. The first PLCs just basically replaced hard-wired relay logic.

Kelvin T. Erickson

Today, PLCs are available in a wide range of capabilities and cost. There are five general categories of PLCs available. The general capabilities of each category are:

Micro PLCs: Generally have the basic relay instructions, counters, and timers with up to 32 digital input/output (I/O) points (fixed number of each) and 2K words of program memory built into a compact unit.

Small PLCs: Added capabilities of analog I/O, expandable I/O of up to 128 points, 4K words program memory, shift register and sequencer instructions, and primitive communications with other PLCs.

Medium PLCs: Expandable I/O of up to about 1024 points and 32K words program memory, remote I/O, basic math and data handling instructions, subroutines, interrupts, functional block or high-level language, local area network connection.

Large PLCs: Expandable I/O of up to about 2048 points and 256K words program memory, enhanced math and data handling instructions, PID control.

Very Large PLCs: Expandable I/O of up to about 8192 points and 4M words program memory.

PLC architecture

The architecture of a general PLC is shown in Fig. 1. The main parts of a PLC are its processor, power supply, and input/output (I/O) modules. In a micro PLC, all three main parts are

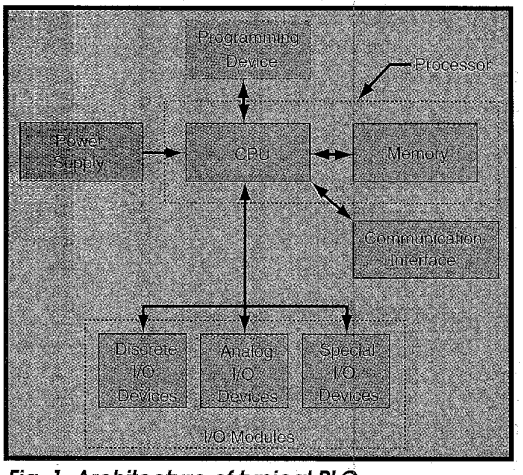


Fig. 1 Architecture of typical PLC

0278-6648/96/$5.00 © 1996 IEEE

Ford Ex. 1080

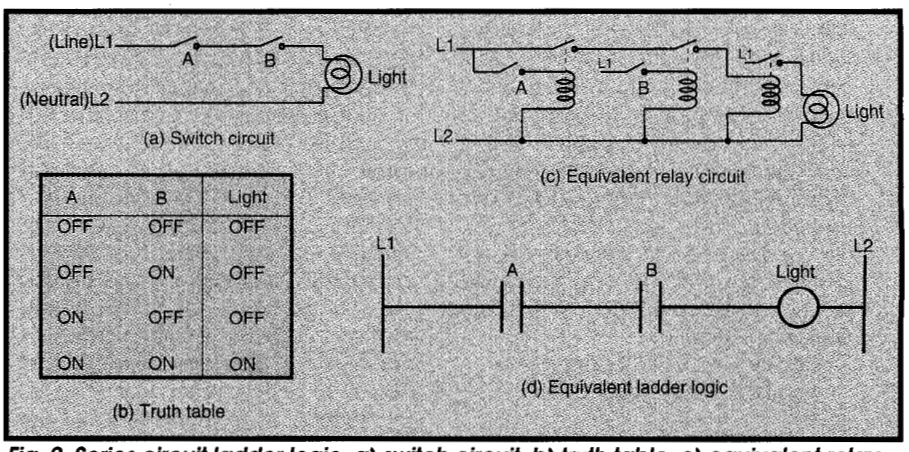
enclosed in a single unit. For larger PLCs, these three parts are separately purchased (depending on desired functionality), and combined to form a PLC. The programming device, often a personal computer, connects directly to the processor through a serial port or remotely through a local area network. Depending on the manufacturer, the local area network interface may be built into the processor, or may be a separate module. Many of the PLC local area networks are proprietary to one manufacturer. However, interfaces to standard networks, such as Ethernet, have recently been introduced.

The architecture of the PLC is basically the same as a general purpose computer. In fact, some of the early PLCs were computers with special I/O. However, some important characteristics distinguish PLCs from general purpose computers. They can be placed in an industrial environment that has extreme temperatures (typically up to 160°F), high humidity (up to 95%), electrical noise, electromagnetic interference, and mechanical vibration. They are easy to use by plant technicians. Hardware interfaces are easily connected. Modular and self-diagnosing interface circuits pinpoint malfunctions and are easily replaced. They are programmed using ladder logic, which is easy to learn. The PLC executes a single program in an orderly and sequential fashion. However, most medium to large PLCs have instructions that allow subroutine calling, interrupt routines, and the bypass of certain instructions. Also, many PLCs have modules that implement higher-level languages, such as C and BASIC.

Ladder logic

The IEC 1131 international standard defines four PLC languages: ladder logic, sequential function charts, function blocks, and a text language. By far, ladder logic is the most prevalent language. The ladder logic symbology was developed from the relay ladder logic wiring diagram. In order to explain the symbology, simple switch circuits will be converted to relay logic and then to PLC ladder logic.

Consider the simple problem of turning on a lamp when both switches A and B are closed, Fig. 2(a). Fig. 2(b) is a truth table of all possible combinations of the two switches and the consequent lamp action. To implement this function using relays, the switches A and B do not connect to the light directly. Instead, control relay coils, whose contacts are normally open, control the light, Fig. 2(c). The switches appear as inputs to the circuit.



A	B	Light
OFF	OFF	OFF
OFF	ON	OFF
ON	OFF	OFF
ON	ON	ON

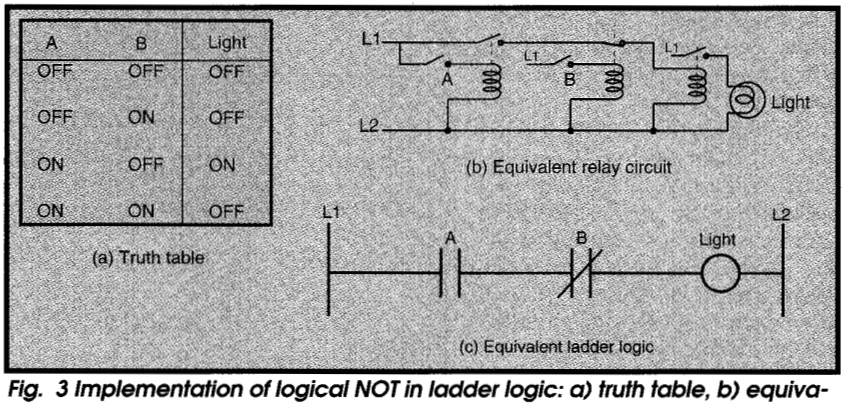
Fig. 2 Series circuit ladder logic: a) switch circuit, b) truth table, c) equivalent relay circuit, d) equivalent ladder logic

The output (lamp in this case) is not driven directly, but driven by another relay to provide voltage isolation from the relays implementing the logic. The switches control relay coils so that the inputs are isolated from the logic. Also, this way one input can be used multiple times by using the multiple poles (contacts) on the relay for that input. The ladder logic symbology is in Fig. 2(d).

Notice how the notation is shortened to show only the relay contacts and the coil of the output relay. It is assumed that the inputs (switches in this example) are connected to relay coils; that the actual output is connected to a set of normally open contacts controlled by the rightmost coil. The label shown above the contact symbol *is* not the contact label. It is the control for the *coil* that controls the contact.

Also note that the output for the rung occurs on the extreme right side of the rung and power is assumed to flow from left to right. One would interpret the rung symbology as: "When input (switch) A is ON and input (switch) B is ON then the lamp is ON." If the example is changed to turn on a lamp when either switch A or B is closed, then the two contacts are placed in parallel.

Now consider the implementation of a logical NOT function. Suppose one wants to turn on a lamp when switch A is on (closed) and switch B is off (open). Figure 3 shows the truth table, relay implementation and ladder logic for this example. The logical NOT is accomplished with the normally closed (NC) contact in the ladder. One would interpret the rung symbology in Figure 3(c) as: "When input (switch) A is ON and input (switch) B is OFF then the lamp is ON." This particular example is impossible to implement with only two normally open switches.



A	B	Light
OFF	OFF	OFF
OFF	ON	OFF
ON	OFF	ON
ON	ON	OFF

Fig. 3 Implementation of logical NOT in ladder logic: a) truth table, b) equivalent-relay circuitry, c) equivalent ladder logic

A more complicated ladder logic diagram is shown in Fig. 4. This figure

Ford Ex. 1080

shows more obviously why it is called a ladder logic diagram. Each rung has a connection to the left (power) rail and a connection to the right (neutral) rail. In reality, the ladder logic diagram is only a symbolic representation of the computer program. So, power does not really flow through any actual contacts; however, the concept of power flowing through contacts is useful when explaining the program operation. The three basic ladder logic symbols are:

Normally open (NO) contact ⊣ ⊢

Normally closed (NC) contact ⊣ / ⊢ or ⊣/⊢

Output (relay coil) —( )— or —O—

The output is energized whenever any left-to-right path of input contacts is closed. For example, in Fig. 4, the output, OUT 1 is on whenever A and B and C are simultaneously on or D is off and E is on. Symbols are being used to avoid having to deal with I/O addressing, which is generally different for each PLC manufacturer.

There are two classes of ladder logic instructions: input instructions and output instructions. Input instructions are the contact instructions, or any instruction that can replace a contact instruction. These instructions are the conditions to turn on the output. In contrast, an output instruction always occurs on the extreme right side of the rung.

In the examples used so far, the only output instruction is —( )—. Depending on the particular PLC manufacturer, the other types of output instructions that may be available are: inverted output coil (output is deenergized if any left-to-right path of contacts is closed), latch output coil, unlatch output coil.

Not all instructions are contacts or coils. All other types of instructions are often called "box instructions" because that is how they appear in the symbology. Timers, counters, comparison, and computation are the most common box instructions, but sequencers, shift registers, and data move instructions are also box instructions. A generic rung with one box input instruction and a box output instruction is shown as the last rung in Fig. 4.

Depending on the manufacturer, box instructions may be classified as input or output instructions. For example, the Siemens TI505 PLC has only box input instructions. On the Allen-Bradley PLC-5's, comparison instructions are box input instructions and timers, counters and computation instructions are box output instructions.

During operation, the PLC repeatedly executes a scan, during which the input channels from all of the input modules are copied into the internal memory; the ladder logic is scanned, updating the outputs being held in internal memory, and then the internal outputs are copied to the actual output modules. After the actual outputs have been updated, the scan is repeated. The time to execute a scan, depending on the number of I/O channels and the length of the ladder logic program, is on the order of 1 - 10 milliseconds. Normally, the processor uses only the internal copy of I/O when executing the ladder logic. It does not read input channels or write output channels. However, some manufacturers do allow that option, which is useful in critical or emergency situations.

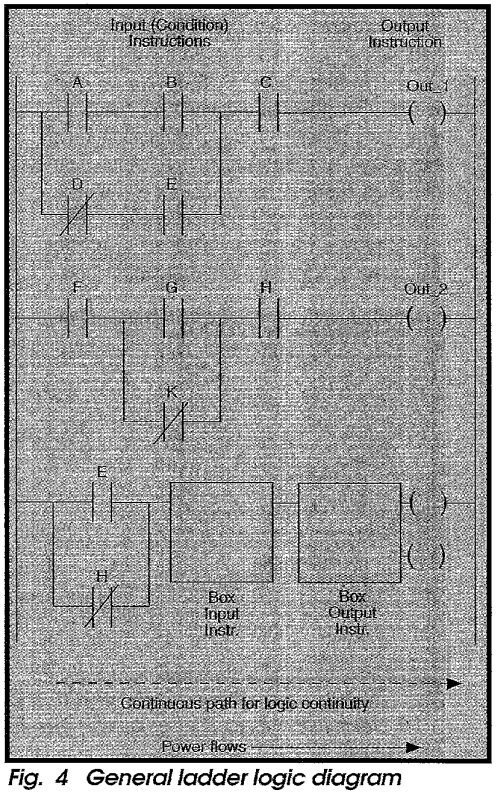


Fig. 4 General ladder logic diagram

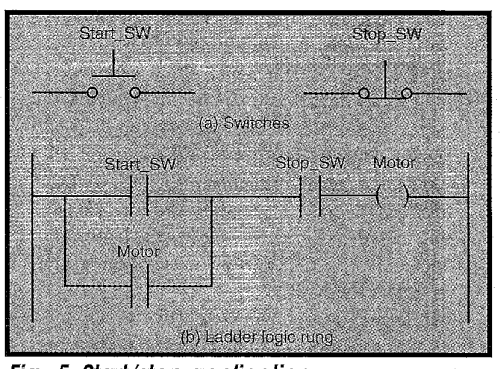


Fig. 5 Start/stop application

The previous examples used external (switch) discrete inputs and an external (lamp) discrete output. However, it is not required that all contacts be controlled by external discrete input devices. The contacts can also refer to an output (such reading the current state of an output). Many PLCs provide internal one-bit memory locations, often called internal coils, to store information that is not connected to any external output channel.

One aspect of ladder logic that is often confusing is the use of the NC contact. The contact symbol in the ladder does not necessarily correspond to the actual switch type used in the field. After all, the PLC does not know how the switch is wired in the field, only whether the switch is open (off) or closed (on). So, a NO switch does not require a ⊣ ⊢ in the ladder logic and a NC switch does not require a ⊣/⊢ in the ladder logic. Regardless of the type of switch in the field, when one wants "action" (something to be logically true, or on) when the switch is *closed* (on), use the ⊣ ⊢ symbol. When one wants "action" (something to be logically true, or on) when the switch is *open* (off), use the ⊣/⊢ symbol. One must eventually learn to read a ladder logic diagram as symbols and not as relay contacts.

One common application uses two momentary switches to control a device, for example, a motor. One switch, called START_SW, is a momentary normally-open switch that when pressed, starts the motor. The motor must continue to run after START_SW is released. The second switch, STOP_SW, is a momentary normally-closed switch, that when pressed, stops the motor. The switches are specified this way for safety reasons.

If there is any faulty wiring to START_SW, the motor cannot be started. In addition, the motor will automatically stop when there is any faulty wiring connected to STOP_SW. The ladder logic diagram that will accomplish the above function is shown in Fig. 5. Note the contact symbol used for STOP_SW. It is the NO contact even though STOP_SW is wired normally-closed. Remember, the PLC does not know how the

Ford Ex. 1080

switch is wired in the field, only whether the switch is open (off) or closed (on).

When START_SW is on and STOP_SW is on (not pressed) the motor is turned on. The contact labeled "MOTOR" in parallel with the START_SW contact ensures that the motor remains on, even after START_SW is released and the PLC reads it as off. When STOP_SW is pressed (turns off), the motor is turned off and remains off until START_SW is pressed again. This type of ladder logic rung is often called a "seal circuit" or "latching circuit." Often, in a real application, there will be multiple conditions that will have to be satisfied before the motor can be turned on. There will also be a multitude of conditions, any one of which will cause the motor to be turned off.

For the last example, consider an application where one wants to control a two speed motor. The specifications for the application are:

a) The motor can only be started in Speed 1.

b) The motor is then switched from Speed 1 to Speed 2 after a 10 second delay.

c) The motor cannot be switched from Speed 2 to Speed 1.

d) Speed 1 and Speed 2 cannot be ON simultaneously.

e) If excessive vibration occurs, the motor must stop and cannot be restarted (is locked out) until a reset button is pressed.

f) If the stop button is pressed when the motor is running in either speed, the motor will stop, but will not lock out. Assume the following input and output assignments (only symbols are used here to avoid explaining the I/O addressing scheme):

Inputs:

START_PB Start push-button, NO, ON when starting

STOP_PB Stop push-button, NC, OFF when stopping

RESET_PB Reset push-button, NO, ON (closed) when resetting

VIB_SENSE Vibration sensor, NC, OFF when vibration occurs

Outputs:

SPEED_1 Motor speed 1

SPEED_2 Motor speed 2

The two outputs are assumed to be inputs to a motor controller that directly controls the motor.

Unfortunately, it is hard to show a generic ladder for any application that uses timers, since their implementation varies among PLC vendors. One ladder logic that will fulfill the specifications for an Allen-Bradley PLC-5 is shown in Fig. 6. RUN and VIB_OCCUR are internal coils (one-bit memory locations) and are not output channels. The TON instruction is an on-delay timer instruction and is an output instruction. The timing interval is the product of the base and the preset values. When the input conditions to the left of the timer become logically true (continuity through contact), the timer accumulator is counted up once for each time base interval. When the accumulator equals the preset value, the timer "done" bit (addressed as TIMERA.DN) is set to true (ON). If the input condition to the timer becomes logically false at any time, the timer is reset and the accumulator is set to zero. It does not retain the accumulator value. Since the timer is an output instruction, the timer done bit must be used on another rung to turn on the SPEED_2 output. The first rung is the normal start/stop rung with an additional condition for stopping.

The RUN internal coil is used because there is not a single output that defines the motor operation. When vibration occurs (VIB_OCCUR turns on), RUN turns off. The second rung defines the delay timer operation. The third and fourth rungs drive the outputs that control the motor. As long as RUN is on and the timer has not finished the 10 second timing interval, SPEED_1 is on. When the 10 second interval has elapsed, SPEED_1 is turned off and SPEED_2 is turned on. The last rung implements a latch/unlatch for the vibration sensor (VIB_SENSE). It is different from the start/stop rung for safety reasons.

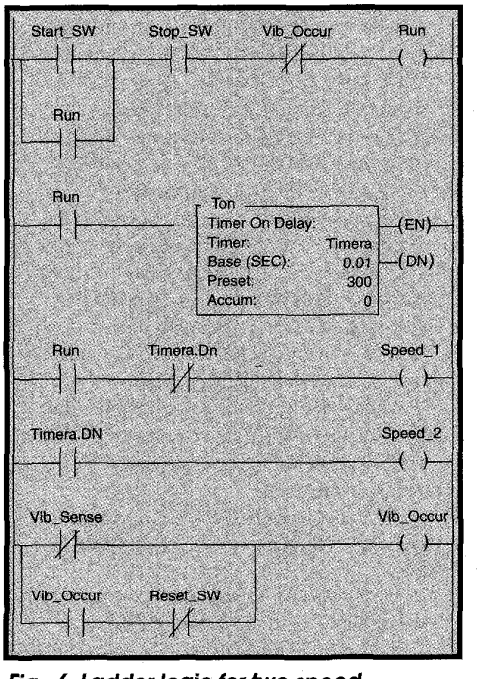


Fig. 6 Ladder logic for two speed motor application

If the RESET_PB NC contact is placed in series with VIB_SENSE on the upper part of the rung (in the same position as the STOP SW in the first rung), then holding the reset switch on (pushing the NO push-button) will override the vibration sensor. This will allow the motor to run even when vibration continues to occur. Obviously, a situation one would want to prevent for safety reasons.

Conclusion

Programmable logic controllers and their unique language, ladder logic, are the workhorses of factory automation. Higher-level languages, such as sequential function charts and function blocks, ease the programming task for large systems. However, ladder logic remains the dominant language at present. Any engineer working in a manufacturing environment will at least encounter PLCs and ladder logic, if not use them on a regular basis.

Read more about it

- Filer, Robert and Leinonen, George. *Programmable Controllers and Designing Sequential Logic*, Saunders College Pub., 1992.
- Bryan, L.A. and Bryan, E.A. *Programmable Controllers: Theory and Implementation*, Industrial Text, Chicago, IL, 1988.
- Simpson, Colin D. *Programmable Logic Controllers*, Regents/Prentice-Hall, Englewood Cliffs, NJ, 1994.
- Warnock, Ian G. *Programmable Controllers: Operation and Application*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
- Webb, John W. and Reis, Ronald A. *Programmable Logic Controllers: Principles and Applications*, 3rd Ed., Prentice-Hall, Englewood Cliffs, NJ, 1995.
- International Electrotechnical Commission. "Part 3: Programming Languages," *IEC Standard 1131-3, Programmable Controllers*. American National Standards Institute, Ref. No. CEI/IEC 1131-3, 1993.

About the author

Kelvin Erickson is an Associate Professor of Electrical Engineering at the University of Missouri-Rolla.

Ford Ex. 1080