Designing an efficient Programmable Logic Controller using Programmable System On Chip

By Raja Narayanasamy, Product Apps Manager Sr, Cypress Semiconductor Corp.

A Programmable Logic Controller (PLC) is a microcontroller-based, general-purpose electronic device to control the operation of a machine or process. Contrary to conventional microcontroller based systems, PLCs are not programmed by the device manufacturer but by the machine builder or the end user. Some of the applications that use PLCs include:

- Automatic test equipment
- Pick and Place machines
- Machine controls
- Home automation
- Elevator systems
- Factory automation

PLCs are classified into following the main categories based on the number of input and outputs they can handle:

- Nano PLC: PLC that has less than 32 input/output (I/O) functions.
- Micro PLC: PLC that has more than 32 Input/Output (I/O) points but not more than 128 input/output (I/O) functions.
- Small PLC: PLC that has more than 128 Input and Output (I/O) points but not more than 256 I/O functions. It does not include any I/O enhancements fitted along with the basic system.
- Medium and Large PLC: PLC that controls a large number (>256) of discrete elements using very fast input/output (I/O) scan times.

According to a recent study conducted by the ARC advisory group, the PLC market reached nearly $9.0 billion in 2007 and is forecasted to grow to over $12 billion in 2012.

**Programming PLCs**

PLCs are programmed using a Ladder diagram and are shipped with custom front-end software that allows the final users to program them. These users are typically nonprogrammers who do not have any C or Verilog/VHDL programming knowledge.

Ladder logic programming looks similar to the wiring diagram of the electromechanical relay-based control circuits that PLCs replace. Consider the example of a staircase light control using two switches. The staircase bulb should come ON if any one of the switch is operated and if any one of the switch is operated again the bulb should go OFF. Figure 1 shows the control implemented as a wiring diagram using two switches (switch 1 is on the lower floor of the staircase and switch 2 is installed at the upper floor of the staircase) and a light bulb.

Switch 1 and Switch 2 are in the OFF position in Figure 1. If the Switch 1 is actuated, then the upper circuit will switch ON the bulb. If switch 2 is actuated it will open the upper circuit and close the bottom circuit. However, switch 1 is already open at the bottom circuit, so the bulb will not glow. Actuating switch 1 again will close the bottom circuit and the bulb will glow.
Figure 1: Wiring diagram of a staircase bulb control using two switches

Figure 2 shows the same control implemented using a PLC and ladder logic. The hardware switch contacts are replaced by \( \overline{|O|} \) (normally open contact) and \( \overline{|C|} \) (normally closed contact) symbols.

Figure 2: Ladder logic diagram of staircase bulb control implemented in a PLC

Figure 3 shows the same control implemented using logical gates (NOT, AND, OR) in a PLD, FPGA, or a System On Chip (SoC) with integrated PLD functionality. Note that all the gates shown in Figure 3 can be replaced by one XOR gate.

Fig.3 Staircase bulb control logic implemented using “gates”

PLC Principle of Operation

Most PLCs on the market are microprocessor-based circuits. These PLCs read the status of all the inputs (i.e., switches) and then execute the user-written ladder logic program to determine the final update for the outputs (i.e., bulb). Figure 4 shows the various components used to build a conventional PLC-based control system.

The PLC inputs are isolated and level translated using opto-isolators and connected to the ports of the microcontroller. The PLC outputs are connected to buffers and relays to enable connecting the same to output elements as shown in the picture below.
The flow chart in Figure 5 shows how the PLC operates.

**Limitations of Conventional PLCs**

The sequential program execution used in conventional PLCs takes around 10 ms or more to complete. At this rate, they are suitable for control applications where the input signal frequency is below 100 Hz. The scanning time is also limited by the length of the program. For example, if you want to read a speed sensor input to measure a speed at around 1200 Rotations Per Minute (1200/60 = 200 Hz signal frequency), a microcontroller-based PLCs cannot measure the speed correctly using this input. Such a system may need a custom input module with decoder or counter ICs that can read the signals at high frequency and convert them into a count value to pass to the microcontroller. Alternatively, consider controlling a flow control Solenoid value with a PWM signal at 10 Khz frequency. Due to the above mentioned limitation, the PLC cannot output this directly and needs a custom output module with PWM generators. Adding such high-speed counter modules and PWM generator modules will increase the PLC cost by 2 to 3 fold.
PLCs that have digital programmable logic like CPLDs or FPGAs can address frequency issues. FPGAs, however, lack built-in analog capabilities. For example if you need to measure temperature using an analog temperature sensor, you cannot interface directly to the FPGA. Moreover, FPGAs are too expensive a component to be used in these kinds of applications.

**SoC-based PLC**

To address the needs of low-cost PLCs, developers require a device that can handle high-speed digital input, high frequency output, and also to directly handle analog signals. Today, there are many System On Chip (SoC) devices that combine a microcontroller with integrated programmable logic and configurable analog blocks to achieve this in a cost-effective manner.

For example, these SoCs can implement an HDL-based quadrature decoder that uses two 90 degree phase-shifted sensor signals to detect the speed and direction of rotation of a motor by reading speed/position sensor signals at high speed. Likewise, a PWM waveform generator can be implemented using on-chip programmable digital resources. Any timer and counter modules required for normal machine control operations can also be implemented in the same way. These components (quadrature decoder, PWM generator, timers, counters, and so on) can be coded once by a programmer and made available as configurable library components. This allows end users with no HDL experience to program the PLC using these components with no low-level HDL/C programming or without having to learn the low-level architecture of the device. Apart from this, other control functions can be implemented using AND, OR, and NOT gates that are available as standard components in a SoC.

**An Example System**

Figure 6 shows the components of a SoC suited for PLCs, including CPU, configurable analog blocks (comparators, op-amps, ADCs, DACs), and programmable digital blocks (PLD-based). With these components integrated into a single device, developers can build a single-chip, low-cost PLC that can overcome the speed limitations of standard PLCs and still provide ease-of-use programming for end users.
PLD-based Universal Digital Blocks (UDBs) are used to implement gate-based logic while configurable analog components like ADCs, DACs, Op-amps, and comparators handle the analog signals. An integrated CAN controller block can be used to interface more than one PLC on a network to support more input and outputs. A built-in USB controller can be used as a programming and debugging interface for the PLC.

With a programmable SoC architecture such as the Cypress PSoC, any internal analog or digital signal can be routed to any GPIO (General Purpose Input/Output) pin. This flexibility enables PLC users to implement various machine control functions using a single PLC across designs. Consider a product line with Machine “A” (10 digital inputs, 2 analog inputs, 7 digital outputs, and 1 analog output) and Machine “B” (12 digital inputs and 8 digital outputs). A PLC with 12 inputs and 8 outputs can be configured by firmware to control either of these machines. This is the major advantage of using a mixed-signal programmable device as a controller for PLC applications.

For example, if a PID speed control loop needs to be built for controlling a DC motor using a quadrature velocity encoder for speed feedback, it is possible to use the same SoC-based PLC subsystem. The PLD-based Universal Digital Blocks (UDB) can be configured as a quadrature decoder using a pre-built component to read the velocity signal at very high speeds (> 100 kHz) and one more UDB can be configured as a PWM generator to generate a pulse width modulated signal at the required frequency (i.e., 32 kHz) to control the MOSFET H bridge to control the DC motor speed and direction. Together, these can be developed as a single component so that the end user who has to program the PLC can just drag and drop these components, configure the parameters for each, and begin using the PLC to control a motor.
**Networked PLC**

To control a machine that has many input and outputs requires the ability to network many PLCs to function as a single, large PLC. This can be achieved by supporting a CAN bus interface between PLC modules to support more inputs and outputs (see Figure 8).

![Figure 8: PSoC-based modular and scalable PLC network](image)

Such an architecture allows developers to build a networked PLC that can control a bigger system with many inputs and outputs. In addition, many industrial control applications utilize touchscreens for control panels for machine tools. Integrated touchscreen and configurable HMI (Human Machine Interface) capabilities can further consolidate cost by eliminating separate and expensive custom touch screen-based HMI panels.

**Ease-of-Use Programming**

Developers may also be able to utilize a silicon vendor’s development tools for end user PLC programming instead of having to create a custom ladder logic programming application. For example, Cypress’ “PSoC Creator” programming software allows users to visually program PSoC-based PLCs from a schematic-level view using the AND, OR, and NOT gate components. As a result, users do not need C or VHDL/Verilog programming knowledge to program the PLC to its full capabilities. Figure 9 shows the staircase switch logic implemented using the PSoC Creator. The gate-level implementation can be an alternate to ladder-based implementations, allowing users to take advantage of all of a system’s features.
Figure 9: Staircase switch implementation using PSoC Creator.