Understanding PLC Programming Methods and the Tag Database System

By Dennis Sloatman

Published: 2016-09-16 · Archived: 2026-04-05 15:32:48 UTC

We discuss some advanced topics, including tag database vs. fixed memory

In part one of this series, we looked at the PLC's internal architecture, briefly discussed Ladder Logic programming, Human-Machine Interfaces, advantages of the PLC's ability to react consistently with the outside world and some possible applications. In part two of this series, we looked at some specifics such as where to buy a PLC, getting started with programming, a small sample program using ladder logic, some ladder logic functions and provided some links to instructional "how-to" videos. In this article, we move on to discuss some advanced topics, including tag database vs. fixed memory.

PLCs vs. LEGACY CONTROL SYSTEMS

In order to underscore and to drive home the impact of the PLC, let's consider the legacy method of creating a system controller. Not so long ago the engineer would construct a device employing dozens of relays, diodes, relay sockets, possibly resistors and capacitors for timing circuits along with power supplies, wiring, some sort of wiring termination (i.e., barrier strips) and an enclosure. Should any changes to such a controller need be made or any enhancement added, wiring harnesses would have to be undone, moved, parts added or removed — all of which gets messy and is quite time consuming.

Obvious negatives to such a system controller are maintenance and serviceability, aside from the difficulty in making changes.

And let's face it: The moment you leave the station, those custom controllers are scary to your successors at best, and at worst, removed and tossed out as soon as possible.

Consider the control system shown here. It was part of the New York City subway system.

Also consider the conditions at many broadcast transmitter sites: Dirt and dust, temperature extremes, humidity and EMI/RFI are common.

In part 1 of this series, I pointed out that devices such as the Arduino and Raspberry Pi aren't good choices in these sorts of environments. Their lack of capacity to drive or sink the current necessary to work with industrial loads means that they need to be supplemented with custom enclosures, relays, power supplies for the relays and connectors — thus putting them right back in the legacy category I previously mentioned, largely negating their benefits.

PROGRAMMING METHODS

In previous articles, I have discussed ladder logic, which is very much a sort of electronic schematic symbolization of the control system you are designing. This makes programming the PLC accessible to the broadcast engineer who has a solid electrical/electronic background.

In the next illustration, you see an example of ladder logic drawn from an AM station antenna controller I designed some years ago. Contacts are at the left on the ladder with Coils on the right (recall that these are internal memory locations, except when using true electrical outputs like relay contacts or open-collectors). Logically, the flow of the system is from left to right (input to output) and then down to the next rung of the ladder.

While I believe ladder logic is the optimal programming language for the PLC, there are at least two other methods of which the reader should be aware: Functional Block Diagram and Structured Text (which is somewhat similar to Pascal and will not be discussed here).

FBD is an interesting programming method which you should consider as it presents a sort of block diagram look and feel. FBD may bring forth memories of a digital logic class you may have had in college.

My advice is to try both of these methods, as you may find that your design is easier to visualize using FBD instead of ladder logic.

Not all PLC programming software provided by vendors supports FBD, but all (of which I'm aware) support ladder logic.

In some cases, the vendor-provided programming software features a simulation mode that permits the designer to debug a program without connection to a PLC. This is a powerful feature for testing various designs without purchasing several controllers (CPUs).

SOME DEEP TECH NOTES REGARDING PLC PROGRAMMING

I once more wish to drive home the point with respect to contacts and coils.

When I teach PLC design to engineers who are newcomers to the topic, I often find that there is a common misconception (and understandably so) with respect to the ladder logic objects.

The PLC is programmed with so-called coils (outputs) and contacts (inputs) and even complex function blocks, such as mathematic operators, analog comparators or timers/counters are referred to as coils. Anything that *is* an output or *has an* output, is referred to as a coil.

Any object on the left of the ladder diagram (see Fig. 1) is a contact — whether or not it's an actual physical contact. Similarly, anything on the right side of the ladder is coil whether or not it's an actual physical contact.

To those readers who have had programming courses in college (such as C, C++, Assembler, etc.) or otherwise have programming experience in such languages, it helps to consider most of these objects (except for the actual

hardware coils and contacts) as operating system variables stored in the CPU memory. This, in fact, is precisely what these objects are.

In some PLC architectures, the programmer is allowed to freely use an object, or in programming terms, to instantiate the coil or contact and set the attributes of that object to whatever is required. You can declare whether the object is a binary (on/off, digital I/O), 16-bit integer, 32-bit integer, 32-bit floating point, signed or unsigned and ASCII "on the fly." This type of programming approach is similar to unstructured programming languages.

PLC TAGS AND THE TAG DATABASE

In more advanced PLC designs, a more structured programming approach is taken with the use of "tags," which are stored within the PLC's memory in a tag database.

With a Tag Database, all function blocks, including contacts, coils, program variables (e.g., as a timer value called "Transmitter_ RF_ Mute_Timer"), as well all other objects, are stored as variables with attributes such as initial value, float, string, integer, Boolean (on/off), ASCII text, discreet inputs and discreet outputs. While this may seem confusing at first-glance, it's a superior approach for more complex designs but does require (as in structured programming languages) the designer to declare these variables "tags," as well as the data type in advance of their use in the program.

One salient features of this design is that all of this information (ladder program, variables and comments) is stored within the PLC, so it's rather self-documenting — that is to say, anyone making modifications can download the entire PLC program into the programming software, even if they do not have access to the original programmer's file. (As you might expect, PLCs can be password-protected.)

Much as in programming languages, you can also assign an initial value to the variable or tag. Data arrays may also be defined in the tag database and in fact, this is how I chose to construct a table of pattern change times for an AM station directional controller with the months as the row data and the pattern change times as the column data.

In summary, Tags are *names* that you assign to variables of any type stored in the PLC memory. Some examples of tag names might be: Ant_Sw_Delay, RF_Loss_Delay, Overtemp_Alarm_Delay, etc.

A LOOK AT SOME SAMPLE PLC SPECIFICATIONS

PLCs vary in design. In some, you will see in the device spec sheet a tabulation of how many timers, counters, comparators, registers, real time clock functions, internal contacts and other function blocks, such as Proportional, Integral, Differential (PID) you have available for use.

Generally, the number of these provided will be well in excess of your needs, but these are limitations, nonetheless.

In more advanced PLC designs, the PLC has a given amount of memory for you to use however your design requires. For example, if your design needs 134 timers but only 60 counters, 188 internal contacts, but only 20 comparators, you have that flexibility to use the available system memory as dictated by your design.

Recall that I said that all of these ladder objects are essentially variables stored in memory? This is at the very core of the concept for flexibility of resource allocation.

Let's look at a representative "fixed-resource" PLC specification sheet (Fig. 3).

Just a quick examination of the Click PLC data sheet will show the quantity of available resources. This $69 PLC has quite a number of objects available and for most designs, is more than enough to suffice. Look at the number of timers: 500! There are 1000 System Control Relays (which are internal CPU-defined memory flags such as real time clock functions and internal timers and system flags), 250 counters — each of which can be user-defined as up/down, up or down with presets. Common programming constructs such as for/next loops, interrupt-handling and subroutines are also available.

As I mentioned, more advanced designs using tags allow you to allocate the available resources as your needs require, but you would be hard-pressed to use all the control objects which are available in the representative specification sheet above.

In part 1 of this series, I touched upon deterministic operating systems, and I stated that Windows does not lend itself well to applications which require real-time control.

PLCs can scan the entire ladder logic program in *milliseconds* (see specification for Typical Scan in the data sheet). The PLC is as close to "real-time" as you'll likely ever get and with repeatable results. This feature of the PLC can be further enhanced through program segmentation through the use of subroutines and interrupt-handling; that is, initialization of a program can be handled upon startup only (say in a power-up set to defaults routine) with timing-critical control handled by interrupt routines (which work in the PLC much as they do in computer CPU's with stacks, "push" and "pop" action).

Now that we've covered the essentials of the PLC, such as programming methods, "underneath the hood" topics and have discussed some of the specifications, next time, we move to building a broadcast-related application of an easily available and inexpensive PLC.
