PLC MASTERY

PLC Programmer's Guide

What You Should Know When Starting PLC Programming

Table of Contents

Introduction

PLCs or Programmable Logic Controllers are the backbone of the industrial automation system we rely so heavily on today. More and more, they are included in today's technical curricula. Understanding what they are, how they work, and how to program them is critical to fully embracing the world of automation.

PLCs are a combination of units and theoretical models that control input and output modules. All PLCs must have the following four components to be considered a full system:

- **CPU Module -** The central processor and memory needed for information storage and task execution. Inputs are received, processed and computed, and pushed out as an output.
- **Power Supply –** Every PLC module requires power, typically receiving AC power and converting it to DC.
- **Programming Device –** Programming software is required to provide the system with control logic.
- **Input/Output Modules –** These are the backbone, collecting data from the actuators and sensors, feeding it to the PLC system, and producing easily understandable information.

Being able to program these systems or units is a huge bonus. No longer do technicians have to disassemble and reassemble units; they simply reprogram them to do what's needed.

This guide will take you through everything you need to understand PLCs, including their history, hardware, programming languages, actuators and sensors, and HMI.

You don't need any previous experience in coding PLCs, so let's get started!

Chapter 1: The History of PLC

We can date the very first PLC back to the 1960s when the automobile industry was the first to use them, aiming to replace hardwired timers and relays with flexible, programmable controllers. Since then, PLCs have gone on to become widely adopted as the standard in many manufacturing industries.

Inception

1968 was the year the first PLC was developed by General Motors, who wanted a Standard Machine Controller specification. They distributed it to other vendors for a price, and some of their specification elements were:

- Use of modular, expandable solid-state components
- Minimum of 16 units, expandable to 256
- Minimum of 16 outputs, expandable to 128
- Easy to program and reprogram
- Minimum 1K memory, expandable to 4K, to ensure no stored programs could be lost during a power outage.

To meet the Standard Machine Controller requirements, Richard E. Morley of Bedford Associates designed the Modular Digital Controller. When General Motors tested it, they experienced a significant reduction in downtime, around 60%.

Modicon PLC

This success led Bedford Associates to change its name to Modicon PLC and begin producing the first PLC – the Modicon 084. Programming technique differentiated it from other similar products.

Most computer scientists were manipulating equipment using Boolean Statements consisting of AND, OR, and NOT expressions – the same logic used by all computers. However, these were not easy for plant engineers to use. Used to working with relay control systems employing ladder diagrams, Morley decided to introduce ladder logic into the system. Nothing more than a Boolean Logic representation, ladder logic changed the game. Engineers found it much easier to understand and implement, significantly easing their workload.

1970s

As the 1970s drew to a close, other competitors developed rival systems, and innovation became the key to gaining market share. PLCs were gradually becoming

more powerful and faster, with documentation tools and programming rapidly evolving.

The first PLCs didn't include program documentation platforms, resulting in programs needing to be drawn or hand-written before being entered. Allen Bradbury developed the Data Highway, and Modicon developed Modbus, allowing PLCs to talk and exchange information.

Programming terminals were developed, allowing programmers to enter logic programs remotely, recording the final program on cassette and downloading it to the PLC. And printouts could be generated, removing the need to draw or hand-write programs and saving time.

1980s

The 1980s heralded the era of personal computers in offices. While their speed was nothing compared to what we have today, it was still faster than drawing the programs. By the time the 1980s ended, virtually all designers had replaced drafting boards with personal computers.

But PCs weren't just adopted in offices. The shop floor also began to adopt them, using them to interface directly with PLCs. With software improvements, machine motions became much easier to monitor.

By now, PLC programs had become widely recognized as the best diagnostic tools, making troubleshooting more effective. However, machine diagnostics were still primitive.

Perhaps the most important milestone was the International Electrotechnical Commission (IEC) 61131-3 specification introduced in 1982. This became the standard by which PLC software development was measured, and, in 1993, it was published as IEC 1131 International Standard for Programmable Controllers.

This was necessary to ensure consistency in software products, allowing technicians and engineers to understand program flow and logic in any PLC software.

1990s

By the time the 1990s arrived, end-users were starting to make requests, such as new machinery coming complete with industrial terminals and built-in PLC monitoring software. Plant managers wanted troubleshooting done by technicians, ensuring the PLC programs were relatively simply designed. They wanted the machines to tell them when something was wrong, which led to the HMI – Human-Machine Interface.

Prototype HMIs were simple pushbutton replacers but were not considered economical where an application had 20 or fewer pushbuttons. However, they grew

in popularity as more uses were found and machine monitoring information became more critical.

Information such as time in auto, machine problems, production counts, and so on were monitored, showed on an HMI screen, and sent to the central computers.

Towards the end of the 1990s, logic control functions were a tiny part of a PLC program's capabilities. HMIs contained enough data that technicians no longer needed to look at program logic.

A new generation of PLCs was introduced at the end of the 1990s, devices that resulted in internet connectivity arriving on the factory floor. Megabytes were the new measurement standard for processor memory, and user-defined data types were introduced, allowing machine data to be manipulated and shared in multiple ways.

PLCs Today

Right from the start, the primary aim has always been to make maintenance and support simpler, and this is achievable only by reducing the automation system size. Today, we see the following trends in modern PLC technology:

- **Smaller, Faster, Better –** processors, circuit boards, and other components are shrinking, influencing PLC design. However, acceptance of these changes is influenced by stability, reliability, and ruggedness. Thus, speed is the industry's prevalent enhancement, brought about by faster processors which improve cycle time, offer better memory capacity, and new communication features. As the market demands continue, more functions and features found in high-end PLCs appear in low-end products, and the expectation is for smaller PLCs with top-tier features.

- **Memory Size –** modern PLCs are taking full advantage of the decline in cost and size in solid-state memory. This allows for better local storage and enables PLC to be used in applications that otherwise need costly systems to acquire data. The same system also allows additional utilities to be included, including on-board information storage, reducing the time needed to expedite troubleshooting.

- **Memory Devices –** portable memory devices are also now appearing in the industrial controls market, providing PLC users with huge storage capabilities in small packages. For example, today's microSD cards can provide a PLC with an extra 32 GB of memory

Ladder Logic Isn't Going Anywhere

Around 50 years ago, ladder logic diagrams replaced hardwired relay logic but, while it made life easier for technicians and engineers, it isn't without its downsides. The

most notable one is its inability to handle data and process control efficiently. This led to other programming languages being developed, and the IEC 6113 standard now covers these:

- Ladder Logic
- Structured Text
- Function Block
- Instruction List
- Sequential Function Charts

Each has its own place, i.e., sequential flow charts better control processes while structured texts suit data manipulation. While all the languages have their strong points, ladder logic remains the most popular of all PLC programming languages.

Chapter 2: PLC Hardware

PLCs consist of software and hardware components. The PLC software is the operating system and the application program, stored in the memory, while the hardware is all the PLC system's physical components. Each hardware component is specially designed to do a specific job and, without the right hardware components, a PLC cannot do its job.

Think of it this way – a car is made up of several physical components, such as the gearbox, engine, steering system, wheels, electrical system, body, and so on. Without all of these things, a car cannot do what it should. However, some components used in manufacturing a BMW, for example, cannot be used in manufacturing a Ford. PLC hardware works similarly.

Most PLC manufacturers have developed their software and hardware components, creating proprietary systems. This means the hardware components will be different between manufacturers, and, most of the time, they cannot be used interchangeably with another brand of PLC hardware or software.

PLC Hardware Components

Now you know what we mean by PLC hardware, let's look at what those components are. A PLC should include the following hardware components:

- Power supply
- Input/output modules
- Processor
- Programming device

Let's look at these in more detail:

1. **Power Supply**

The PLC power supply requires mains voltage and is connected to AC mains. The power supply's output is DC voltage, which is used to power all other associated modules. However, the power supply will not provide any power for field devices.

Voltage

The most important thing to consider is the type of power supply that goes to the element. Look for certain keywords on the instructions, such as "supply voltage," "input power," and "power supply."

When we look at the data on our equipment, we can see that every element is supplied with 24C DC power. We do not consider the communication board module because its power comes straight from the CPU and doesn't need a separate power supply connection. It is also clear that the manual provides details on the voltage tolerance range for normal operations. In PLC and HMI, there is a ± 20% tolerance.

Current and Power

You can delve further into this by looking for the words "current consumption" and "power consumption," where you will find more information regarding current consumption.

2. **Input/Output Modules**

These modules connect to analog or digital field devices and include encoders, switches, and transmitters. Typically, output field devices include proportional valves, lamps, and relays.

Different Types of IO

All PLCs need a way of receiving a signal from a real-world source and interpreting it. They also need to have some control over elements such as motors, valves, and solenoids. This is called IO, or Input/Output. Brick PLCs, otherwise known as monoliths, contains a fixed amount of this capability built-in. A rack plc, otherwise known as modular, takes individual circuit boards, known as cards, to customize their IO capability to the user's needs.

The latter has many advantages, and the most obvious is that these individual cards can be replaced should they fail, without the need to replace the whole PLC. Depending on your application, you can choose specific cards with a bias towards discrete cards where the application has multiple ON/OFF IOs and analog cards where the application has multiple 4-20mA signals, or similar.

Some PLCs even feature hot-swappable cards. This means that new cards can replace existing ones without de-energizing the PLC processor and rack power. However, unless you have built it yourself, you should never assume a PLC has these hot-swappable cards – if you try to replace a "live" card in a system without the feature, you can damage the card and potentially the entire unit.

Some PLCs can also connect remote racks that don't have remote access but do have extra IO modules or cards. That way, the number of IO channels can be increased beyond the base unit's capacity. Typically, the host PLC connects to these remote racks through a digital network that can cover huge distances.

System expansion can also be done by networking several PLCs, where each one has a dedicated processor and rack. Communication instructions allow a PLC to be programmed to read and/or write data to another one, effectively using the second one to extend its IO. Although it is costlier than remote IO, it does provide stand-alone capability should the connection be lost between the PLC processors.

There are three basic types of IO for PLCs – Discrete, Analog, and Network, and we'll look at each one in turn now.

Discrete IO

Discrete data points only have two states – ON and OFF. Some examples of a discrete sensing device are:

- Limit switches
- Process switches
- Proximity switches
- Pushbutton switches

For a PLC to know the state of the sensor, a discrete input channel must send a signal to it from the sensor. Each discrete input typically contains a set of LEDs (light-emitting diodes) and, when the corresponding sensing device is switched on, these LEDs are activated. Each LED shines its light on a photo-sensitive device in the module, such as a phototransistor. This will then activate a 'bit,' a single digital data element, in the PLC memory. This is known as an opto-coupled arrangement, resulting in each of the PLC's input channels being rugged, isolating the PLCs sensitive circuitry from voltage spikes and anything else that can cause electrical damage.

In a typical discrete input module, a single input channel has an opto-coupler and can write to its own memory register 'bit' in the PLC memory. Discrete input cards typically have more than one channel, usually 4, 8, 16, or 32.

Assemblies that consist of overload protection devices and discrete control devices, such as solenoid valves, indicator lamps, and motor starters, are all good discrete control device examples. Like discrete inputs, PLCs can connect multiple types of discrete control devices via a discrete output channel.

Typically, a discrete output module will use the same opto-isolation, allowing the computer circuity in the PLC to provide loads with electrical power. For example, the internal circuitry drives an LED, which activates a phot-sensitive switching device. Or instead of an opto-isolating semiconductor switching element, you can use electromechanical relays.

In a typical discrete output module, a single channel also has an opto-coupler like the input module, but this one is driven by the opto-coupler's memory register 'bit' located in the PLC memory. These cards also usually have 4, 16, or 32 channels.

There is one critical concept you need to master when you are working with DC discrete IO – how to distinguish between current-sinking and current-sourcing devices. The terms "sinking' and 'sourcing' reference the current's direction in or out of the control wire in a device. A device that sends current to another device through the control terminal (conventional flow) sources a current. In contrast, a device that accepts the current into the control terminal sinks a current.

You can only truly understand these terms when viewing electric current from the conventional flow perspective. In this, the DC power supply's positive terminal is said to be the current's source, where the current goes down to the power supply's negative terminal or ground. Where a PLC drives a discrete control device, the output channel forms circuits, and the same happens when a discrete sensing device drives a PLC input channel. One element in each circuit must source current, and the other sinks it.

Perhaps a simpler way of describing it would be "blowing" and "sucking." Where a device sources current to another device, it 'blows' the current. Where a device sinks current, it "sucks" it out of the other device. It may not be the most professional way of describing it, but if it helps you understand the difference, then it's all good.

If a non-polarity-sensitive discrete device is connected to the PLC, you can use either PLC IO module type. For example, a mechanical limit switch could connect to a sourcing or sinking PLC input. You should also note the polarity difference between the sinking and sourcing cards' common terminals. The sinking card indicates a positive input channel terminal and a negative common or com terminal, while the sourcing card has a negative input channel terminal and a positive common or VDC terminal.

However, you should be aware that some discrete sensing devices, such as an electronic proximity sensor with transistor outputs, will be polarity sensitive. Sourcing proximity switches may interface only with sinking PLC input channels and vice versa. In every case, the current is sent out of the sourcing device's signal terminal and taken into the sinking device's signal terminal.

Virtually all PLC discrete IO modules have a standard feature – LED indicators to provide a visual indication of each 'bit" or discrete channel's status. However, each will be different. For example, on an SLC 500 model, the LEDs are clustered close to the top of the module in eight numbered squares. On a Koyo DL06 model, things are somewhat different. The labeling shows a letter/number code designation starting with 'Y' on each output channel terminal. It also shows the common terminals, with a "C" label servicing output channel clusters. Each of these common terminals is common to just four output channels, which means this particular PL has sixteen output channels but with four different common terminals. This might seem odd, but the idea is to allow different types of DC power supplies to service different output channels.

With the AC discrete IO, electrical polarity is not an issue. AC polarity reverses regularly, but we must consider whether the discrete PLC module's common terminal connects to an ungrounded (hot) or grounded (neutral) AC power conductor.

Another example of a discrete AC output module is the Allen-Bradley SLC 500 PLC. The power switching devices for this module are TRIACS and not the transistors we find with SC discrete output modules. This is an eight-channel module with two TRIACS sets that switch power to AC loads. The AC power is received in each set from a hot terminal, either VAC 1 or VAC 2, while the load device's other side is connected to the AC power source's grounded or neutral conductor.

Thankfully, each PLC comes with a reference manual containing diagrams showing how the discrete output and input channels are connected to the field devices. Never neglect to look at these diagrams – it can save you a whole host of heartache.

Analog IO

In the early days of the PLC, memory, processor speed, and logic controllers were somewhat limited and could not support anything other than discrete control functions – ON/OFF. As a result, early PLCs only feature discrete IO. However, the more modern PLCs are powerful and fully support measuring, processing, and outputting continuously variable signals, i.e., analog.

At heart, every PLC is a digital device, which means that, for a successful interface with control devices or analog sensors, a little translation between the analog world and the digital one is required. All analog input modules contain an Analog-to-Digital Converter (ADC). This is a circuit that turns analog electrical signals into

multi-bit binary words. In contrast, all analog output modules have a Digital-to-Analog Converter (DAC), which turns the digital command words into analog electrical units.

Commonly, analog IO can be used in modular PLCs for several analog signals, such as:

- **Voltage** – 0-10 volt, 0-5 volt
- **Current** – 0-20 mA, 4-20 mA
- **Thermocouple** – millivoltage
- **RTD** – millivoltage
- **Strain Gauge** – millivoltage

Network IO

PLCs can communicate between themselves and between field devices using a number of digital network standards. Modbus was one of the earliest examples of a PLC communication digital protocol originally made for Modicon. However, it was soon adopted by some other industrial device and PLC manufacturers as the gold standard and is still a universal protocol for digital devices today.

Profibus is another digital standard developed and adopted as a gold standard, and Siemens developed it.

3. **Processors**

Processors are made up of two components – the CPU or central processing unit and the memory. This section is where the necessary decisions to observe field devices connected to the input/output modules and operate them.

Those decisions are based on programs created by users and stored in the memory, where data that represents the input field device condition is also stored, with data that tells output field devices what they need to do.

How to Choose a Processor

When it comes to choosing a processor for your PLC, there are several things to take into account:

- How much memory does your system need?
- How many devices will be connected to your system? This determines the amount of data memory you need.
- How big is the program? What instruction types will be included in the program? This determined the program memory.
- How fast do you need your scan time to be?

Why is all this important?

Data memory is the memory you need to store and manipulate data dynamically in the system. For example, if your program uses timer or counter instructions, the data memory stores the current values, setpoints, and other internal flags. Let's say your program requires historical data retention, like measuring device values over a lengthy period. In that case, the required data table size will determine what model of CPU you need.

Program memory stores the program instruction sequence selected for the specific application. Each different instruction type requires a certain amount of memory, which is usually defined in the programming manual.

Where your applications are sequential, they can rely on the rule of thumb that estimates an iOS device's program memory. That rule states that each IO device can have five words of memory. If your application is more complex, it's harder to judge.

If you consider scan time to be important, you need to look at the speed of the CPU processor and the speed at which instructions are executed.

Some CPUs are not so fast at instructions related to data handling but are much faster at Boolean logic. If you need PID or other special instructions, you need to look at a CPU that makes it easier to perform those types of functions.

To work out your program memory, follow this guideline – each discrete device should have five words of memory, and each analog device should have twenty-five words.

From this, you should be able to work out the requirements of your system and purchase the right processor.

4. **Programming Device**

In modern industrial applications, a programming device is typically a desktop or laptop that allows decision-making programs for PLC systems to be created. Examples include SIMATIC Step-7 on the Siemens PLCs and Studio 5000 on Allen Bradley PLCs.

Summary

- PLCs consist of a laptop or desktop computer with extra hardware. The hardware may be a part of the system, or it may be attached at any time.
- Modern PLCs have taken over from old-fashioned hardwired circuitry found in older industrial applications. However, most of the physical devices in the older system are connected to the PLC.
- The PLC's hardware components are a processor, power supply, input/output modules, and programming device.

- The power supply is AC mains connected, supplying the PLC modules with DC power. However, it doesn't supply field devices with power.
- The input/output modules are connected to field devices that may be analog or digital.
- Processors are made up of a CPU and memory.
- In modern industrial applications, the programming device is typically a computer of some description.

Chapter 3: PLC Programming Languages

Today, there are five popular PLC programming languages:

- **ST –** Structured Text
- **SFC –** Sequential Function Charts
- **LD –** Ladder Logic Diagram
- **FBD –** Function Block Program
- **IL –** Instruction List

Each has advantages, disadvantages, and best-use cases, but, as a PLC programmer, you need to be aware of what each one does so you know where to use it. It is worth bearing in mind that some languages may not be available or only available at a premium, depending on your PLC program.

Let's look at these languages in a bit more detail.

Ladder Logic

Before PLCs gained popularity, most manufacturing sites used relay controls, driving loads based on a simple logic implemented via hard-wiring. This wiring was specified in hand-drawn drawings, assuming a ladder-like layout of the logic. Ladder logic programming was then used in the early PLCs, mimicking that previous layout. It was the first PLC programming language and remains the most popular today.

Over the years, ladder logic has evolved, but the basic operational principles remain the same. Each rung is evaluated in sequence to assess conditional instructions. A TRUE result ensures the execution of the output instructions.

Advantages:

- **Simple implementation and troubleshooting –**visual, providing most instructions with status confirmation. That way, anyone can walk through a program, understanding the logic, with little knowledge.
- **Modular –** easy to modify, add and subtract logic because each rung is an individual condition.
- **Resilient and consistent –**allows the implementation of multiple functions. However, because the language is standardized, full flexibility is not provided. Between the different implementations, the code is kept consistent.

Disadvantages:

- **Steep learning curve** –a simple language, Ladder Logic isn't intuitive for those with a background in other languages, such as Python or Java.
- **Slow to Deploy** –Ladder Logic is visual, but creating the logic takes longer. Elements need to be bragged and dropped, slowing the development process compared to modern languages.
- **Not intuitive for complex applications** – great for sequential Boolean tasks, but not easy to decipher and implement for modern control theory with flow control, PIDs, feedback loops, and analog sensors.

Structured Text

Close to C or Assembly programming languages, Structured Text requires users to enter code lines that:

- Execute in a sequence
- Evaluate specified functions
- Do Boolean checks
- Energize the PLC's output

If you have a background in Python, Java, or other traditional languages, Structured Text is easy to transition to and is easy to manipulate in text processors. As such, it doesn't require hardware for implementation.

Advantages:

- **Intuitive -** especially to those transitioning from programming languages like C and Java, as it follows many of the same paradigms, functions, and structures.
- **Complexity** – more flexible than other languages, and advanced functionality is much easier to implement when you master Structured Text.
- **Transferability** – standard among many PLC systems, migration between platforms is easy. And, although the languages have significant differences between platforms, Structured Text can easily be implemented on software and hardware.

Disadvantages:

- **Troubleshooting is hard** – compared to Ladder Logic, Structured Text is far harder to troubleshoot. You don't get many visual aids, no visual cues, and each line has more code. Process flow is hard to figure out.

- **Error-Prone** – although it is more flexible, the cost comes in standardization. Users must catch potential failures and create safe fallbacks with software engineering best practices.

Typically, you should learn Ladder Logic before moving on to Structured Text unless you are already proficient in other languages. Although it isn't often seen in production environments, it is ideal for data manipulation, FOR loops, and other structures.

Function Block Diagrams

This language was developed for chemical processes, allowing users to create visual process flows and representations with the right transitions between instructions. The visual editor is simple to implement and use and provides a natural way to implement flows. Commonly, it is used for establishing PID controllers.

Advantages:

- **Flexible Visual Editor** – it is user-friendly and allows users easy layout creation.
- **Perfect for Complex Structures** – Ladder Logic requires multiple rungs to accomplish what one page of FBD can do. Users can bring instructions into complex PLC instructions implementing Motion control, PID loops, and AOIs (Add-On Instructions).
- **User-Friendly** – Easy to use, and a drag-and-drop methodology helps create process layouts.

Disadvantages:

- **Not Easy to Standardize** – because the layout is flexible, FBD doesn't make it easy to standardize programs. All PLC programmers have different approaches, and later programmers may struggle to understand the information flow.
- **Doesn't Scale Well** – while FBD works well on small implementations, the more complex the program gets, the troublesome it becomes.

FBD is critical for PID loops, analog scaling, and Motion Control, but you must master Ladder Logic first.

Sequential Function Charts

Sequential function charts are perfect for subsequent processes, for example, chemically transforming raw materials into the end product. One example is a brewing process. A beer brewing facility would have many tanks, pressure sensors,

valves, and heating elements, not forgetting the packaging section. When production of a new batch is initiated, the process will go through a sequence – these have been simplified:

1. Verify that the system is ready – are the ingredients there? Tanks empty? Valves in the right place> If yes to all, the system goes ahead. If not, it is aborted.
2. A tank filling sequence is initiated, which may require several ingredients. The state is validated and goes ahead when the tank is full.
3. The brewing process is initiated. The temperature is raised and maintained for a set period, and the tank pressure is monitored. Ingredients are added as required.
4. Transfer to the holding tank is initiated. Verify the valves are set properly, the tank is empty, and start the process
5. The batch is sent to the bottling facility.

The steps are executed in a sequence that could be implemented via Ladder Logic but works better with SFC.

Advantages:

- **Mimics Chemical Process Flows –** batching is common, taking raw ingredients and turning them into a product.
- **Combined with ST –** SFC editors use ST to create advanced logic flows.

Disadvantages:

- **Doesn't Apply to Most Applications –** sequential function charts are challenging to apply in non-sequential processes.
- **Not Easy to Implement Parallel Flows –** while there is no limit to the number of process flows you can implement, it gets harder to do where process paths are split into several flows. It is also harder to troubleshoot.

SFCs are good for some use cases but don't work for non-sequential cases. Before learning SFC, it is better to familiarize yourself with the processes, understand the product flow and build your model on paper first.

Instruction Lists

These are often mixed up with Structured Lists because of the similarity in their editors. However, they are not typically used on the same platforms because of flow similarity.

Each code line specifies the instructions and the execution conditions and outcome where program flow is concerned. Instruction Lists are closer to Ladder Logic than Structured Text, but both languages can create the same flows.

Advantages:

- **Standardized –** follow tight structures where users must explicitly create variables, specify conditions, and list all instructions. Program implementation doesn't differ, so the code is easier to understand.
- **Instruction-Focused –** instructions are more important than data flow, creating clarity on data processing in the program.

Disadvantages:

- **Not Available on all PLC Platforms –** not the most popular programming method and not usable on every platform.

Chapter 4: Steps to Set Up a PLC Project

Setting up a PLC doesn't have to be difficult, but neither is it that simple. This chapter will walk you through the steps needed to develop an industrial PLC project.

You've taken the time to research PLC programming and all its different aspects and are now ready to start your first project. The good news is that there are plenty of products and resources to help you in your quest, but you must keep in mind that using a PLC to automate systems, machinery, and equipment is multi-faceted. If you want to be successful in executing your automation project, you need to be prepared to do the following:

- Fully understand the equipment you need and the process functions.
- Account for real-world mechanical interfaces and behaviors.
- Properly instrument the equipment.
- Design your control panels.
- Design the IO and electrical power wiring.
- Develop the necessary documentation and drawings.
- Plan for issues surrounding installation and construction.
- Determine the correct communication interface and implement it.
- Create the PLC programming and test it.
- Configure the right HMI displays for the system operators.
- Test the system before it goes live to ensure its durability.

Here are some tips to help you with your first PLC project:

- **Planning –** the system functionality must be documented so that everyone understands it. This could be a sequence of operations written down,

descriptions of the system function, flow charts, steps, and any other method used. This documentation gives the developer and/or team the requirements they need to follow.

- **Plan for the Unexpected –** systems should never be designed for a perfect scenario, only real-world scenarios. While it is relatively easy to plan for normal operation in most applications, it isn't quite simple when you need a resilient system to cope with problems. You need to consider all the ways your equipment can fail, jam up, be slow in its operation, or face operator errors and then build in the provisions or manual features to cope with these situations.
- **It's all About the IO –** control systems can only be truly effective when they monitor the correct conditions and command the proper actions. Right from the start, you need to develop an IO list that indicates how all control points will operate to ensure everyone is working to the same understanding. It's wise to add more IO and sensors than you need because it is far easier to remove them if you don't need them than to add them later on.
- **Don't Forget the Crew –** control panel and schematic design is technical and detailed, subject to NEC and UL requirements. As such, it requires qualified engineering and design support. However, it makes sense to remember the people who are likely to be using the systems in the future. Ensure the drawings and documents are sufficient for personnel to work out how the system goes together and works. Ensure sufficient tagging, working space, and troubleshooting aids so that your operators, installers, and maintenance staff can do their work.
- **Always Use Spares –** when you create a new design, you should always build future space spaces to the largest possible extent. As a rule of thumb, target around 25% installed spares to ensure potential changes are addressed. If you continuously produce a typical system, those spaces can be reduced when it goes to production. Make sure you order additional consumables, such as fuses, and have devices like terminal blocks and circuit breakers in stock so you can change them out quickly and with minimal disruption.
- **Check the Tech Beforehand –** while classic IO and hardwiring are relatively simple, some of the more complex connections that use industrial communication may need more work to ensure everything works together. Most products will play together nicely, but you should always check these links beforehand to ensure everything works as you expect.
- **Don't Forget the HMI and PLC Programming –** this is the most critical part of your automated system, and not everyone can determine the software status throughout its progress. Hold regular meetings, ensure the code is exhaustively documented, and ensure that every variable and IO point is described in simple terms. You could also add these descriptions and requirements into the programming as comments.

- **Test Early, Test Often –** As you develop your software, test each piece and test each section as you combine it. Create plans to test it based on the documents drawn up for the system requirements and test it in shop conditions before entering production. Test it on every scenario you can possibly think of no matter how crazy they may seem. Then, once your system is field-installed, test it again.

Programming a PLC

This can sometimes be quite daunting, and it is easier to break the task into smaller ones. Using a die stamping application as an example, these are the best steps to take. You can apply these to all applications:

Step One - Define the Task

Write down what needs to happen and summarize it. Speak to every person involved in operating the machine and ask for their input. When a conflict arises on part of the operation, go back to the people concerned and clarify it. There is a good reason why this is the first step in developing a PLC program

The process is started and shut down using a master switch. There are two sensors:

- An upper limit switch – indicates that the piston is retracted fully
- A lower limit switch – indicates that the piston is extended fully

When the master switch is activated, the piston moves between the retracted and extended positions. An up-and-down solenoid is used to do this. When the master switch is deactivated, the piston returns to the retracted position, and every solenoid is switched off.

Step Two – Define the Inputs and Outputs

Using the information from the first step, you can determine which inputs and outputs to the PLC are needed to make things happen:

Inputs:

Master Switch – On/Off

Upper Limit Switch – On/Off

Lower Limit Switch – On/Off

Outputs:

Down Solenoid – On/Off

Up Solenoid – On/Off

Step Three – Develop the Logical Operational Sequence

Step three is where most of your development time will be spent. The first two steps allow you to express what the program needs to do. Step three is where you can make changes to the programming based on whether your program sequence and/or inputs and outputs need to be modified.

You can do this with a sequence table or flow chart, which makes it easier to understand the operation's logic before you program it. Most people jump right over this step and go straight to the programming, which can cost you in terms of frustration and time.

Step Four – Develop Your Program

Now you can write your PLC program in any of the available languages. We use ladder logic for our die stamping application and a sequence table to help us work out how it all goes. Let's say we are using Set and Reset conditions. The solenoid is activated when the master switch is turned on. The first run in the ladder logic directly correlates to this and so on up the ladder and through the process.

One of the most critical parts of this step is to document everything. This cannot be stressed enough as it will save you a ton of time and potentially money if you need to go back to the program in the future.

Step Five – Test Your Program

The last step is to test the program logic. Again, use all the previous steps as testing is the most important step to test for every condition in the logic, i.e., safety, sensors fail, power cycle, and so on.

Use a real machine or a simulator to test your program. Modify where needed and check with anyone else involved to ensure the program does what is expected of it. Ask if anything else is needed and, after a time, follow up to see if anything else needs to be addressed.

These tips and steps should help you plan your system right through to PLC programming and going live.

Conclusion

PLCs are here to stay, whether we like it or not. We can expect further integration as time goes on. The most important integration that will produce the most impact is Enterprise Resource Planning, not to mention synchronization with high-level computing systems. Extracting data in the past, and feeding to these systems, used to be a massive task, but future technologies will include functions, features, and hooks to simplify the entire integration process.

Over the years, the industrial landscape in terms of connectivity has seen significant changes and most factories now use Industrial Ethernet as their choice of network. This can handle data in much larger amounts at lightning-fast speeds, making it the best network for handling data-heavy and high-end applications usually found in a PLC.

Another significant reason why Industrial Ethernet has seen such high adoption rates is the Industrial Internet of Things (IIoT).

IoT makes it possible for manufacturers to connect equipment, ensuring it all works as one module using sensors and connectors. These are fitted to industrial devices and PLCs to make the data gathering process more efficient and provide managers with real-time views of the goings-on on the factory floor. This means they can spot the problem areas immediately they arise and find the solution to fix them.

This book showed you how PLCs came into being and how they have progressed over the decades and looked at the most popular PLC programming languages. The most popular, and certainly your starting place, is Ladder Logic. As you become more proficient, you should consider learning the other languages and which languages are best for each process.

Thank you for taking the time to read my guide. I hope you now have a better understanding of what PLCs are, how they work, and some of their many uses.