Encyclopedia of Systems and Control
DOI 10.1007/978-1-4471-5102-9_257-1

Programmable Logic Controllers

Georg Frey*
Saarland University, Saarbrücken, Germany

Introduction

Since the 1970s, the programmable logic controller (PLC) has been the primary workhorse of industrial automation. For a long time, it has provided a distinct field of research, development, and application, mainly for control engineering. This area has produced its own design methods and programming languages. Due to its importance for industrial application, a lot of these methods have been standardized by the International Electrotechnical Commission (IEC). Currently the most influential standards are IEC 61131 (John and Tiegelkamp 2010) and IEC 61499 (Vyatkin 2011). While the latter one is dedicated to distributed systems, IEC 61131 covers the PLC as such. This standard consists of several parts. The most important ones are:

Part 1 : General information. This part covers the *CONCEPT* of PLCs. It describes the general idea and typical functionalities, most importantly, the cyclic processing of the application program working on a stored image of the input and output values.

Part 2 : Equipment requirements and tests. Here requirements on the PLC *HARDWARE* (electrical, mechanical, and functional) and corresponding tests are defined.

Part 3 : Programming languages. This is the most important part of the standard. Based on already existing PLC programming languages, a harmonization of the *SOFTWARE* structure was achieved. This includes a general software model together with a set of different standardized programming languages. IEC 61131-3 paved the way from proprietary programming solutions to a set of well-accepted languages, allowing easier training of PLC programmers and – to some extent – the reuse of application solutions on different hardware platforms.

While Part 2 is of importance for PLC manufacturers only, Parts 1 and 3 contain relevant information for PLC users, especially for designers of PLC control applications. Before discussing these points, the definition of PLC from IEC 61131-1 is reproduced and discussed:

> A PLC is a digital electrical system used in manufacturing. It utilizes programmable memory to store practice-oriented control programs. Thus is suitable for implementation of specific functions such as combinatorial control, sequence control, time-, count- and arithmetic functions. Due to its special arrangement of digital or analog input/output, it is used for controlling various machines and processes. (. . .)

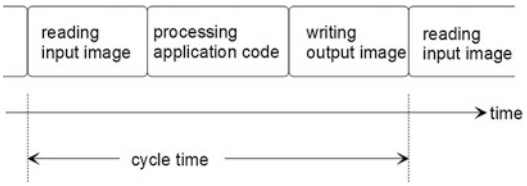
This definition is focused on the usage of the device and would – taken out of the context – also cover industrial PCs or microcontroller-based control solutions. The specifics of PLC hardware are discussed in Part 2 of the standard. However, much more important for distinguishing a PLC from other control hardware are the properties of the execution model described in Part 2 and discussed in the following.

*E-mail: georg.frey@aut.uni-saarland.de

Encyclopedia of Systems and Control
DOI 10.1007/978-1-4471-5102-9_257-1

Execution Model

In designing PLC applications, the execution model has to be considered. The main idea is the cyclic execution together with an I/O image. While microcontrollers and PCs typically use an event-based execution model (the application waits for external events from the environment – interrupts – and reacts accordingly), the PLC follows a time-based scheme (the application scans the environment at instances in time – often a fixed cycle time – and reacts on the new status of the input ports).



A PLC cycle consists of three iterated steps: input reading, program execution, and output writing. Together with the concept of the process image – a reserved memory space where input and output variables are stored – this execution model leads to the following:

(a) During one cycle, input and output values are kept fixed, i.e., a change in input signal values during a cycle will not be seen by the program executed. This means that a temporal change in an input signal value that is shorter than the cycle time may not be registered by the PLC at all.
(b) Changes in output signal settings by the program will be switched to the actual output ports only after execution of the complete program. This actually means that for an output signal where the value is changed several times during one program execution, only the last change will be set to the hardware output of the PLC.
(c) The response time of a PLC, i.e., the time between a change in an input signal and the corresponding reaction at the output port of a PLC, lies between one and two PLC cycles, depending on when the change at the input port occurs relative to the PLC cycle.

While the time needed for input reading and output writing is constant over all cycles, the time for program execution may vary due to conditional execution of some program parts. However, normally the PLC is operated with a fixed cycle time set high enough to allow for the worst-case execution time of the application program.

The advantage of the described concept is the deterministic behavior of the resulting system with a very simple way to determine the timing behavior. This is important for most PLC applications:

(a) Open-loop control, where the reaction to a change of an input signal has to be reached in a limited time, especially in safety-critical applications.
(b) Closed-loop control, where the design of a discrete-time control algorithm is based on the assumption of a fixed sample-time.

To realize control functions, an application program has to be written for the PLC. To this end, Part 3 of the IEC 61131 defines a software model together with a set of programming languages.

Encyclopedia of Systems and Control
DOI 10.1007/978-1-4471-5102-9_257-1

Software Model and Programming

The original idea that led to the development of the first programmable logic controller (PLC) in 1968 was to replace hardwired control equipment at machines. Back then, the controllers of machines, for example, lathes or grinders, typically consisted of a cabinet of interconnected relays. The size of such a controller could be considerable and its failure rate was high due to mechanical defects of single relays. Furthermore, the initial setup was very time-consuming and error prone, because the relays (often hundreds of them) had to be wired by hand. The biggest drawback of this technology, however, was the problems arising if a controller had to be changed, employing a new function or adjusting to a new production task. Then the hardwired structure had at least partially to be disassembled and rewired. Here was the main advantage of a controller that could be adjusted by changing software instead of hardware.

Since the first PLCs in the early seventies reached the market, graphical programming methods are used to develop the control algorithms. These are ladder diagram (LD, sometimes also referred to as ladder logic) and later Function Block Diagram (FBD). The implementation of LD on the very first PLC (the Modicon 084) was intended to allow an easy access for the people doing hardwired relay logic until then. (More on the history of PLCs can be found on the website of Dick Morley, commonly known as the father of the PLC (http://www.barn.org/FILES/historyofplc.html).)

LD, at least in its early forms, is basically the graphical representation of its hardwired forefather. The name ladder comes from the fact that on both sides of the drawing, there is a power rail and horizontally between those rails, like rungs on a ladder, sequences of logical element are drawn. The basic of these elements are relays (switches), depending on input signals or internal variables, and coils (memories to store variables and set output signals). The ladder is processed in a top-down and left-right fashion.

Figure 1 shows an example of an LD. Every rung can be read as an IF THEN ELSE statement. The first rung of the ladder means IF (Var1 = 1 AND Var2 = 1) THEN (Var3 := 1; Var4 := 0) ELSE (Var3 := 0; Var4 := 1). The second rung is IF (Var3 = 1 OR Var4 = 0) THEN (Var1 : = 1) ELSE (Var1 := 0).

While LD resembles relay logic, FBD is a graphical mimicking of the wiring of simple logic gates, like AND, OR, NOT, or FLIP-FLOP. Both languages (LD as well as FBD) are still part of the IEC 61131-3. However, they are not well suited for the description of sequential and concurrent algorithms because they have no means for the visual description of the control flow in a program.

The IEC 61131-3 standard also contains a language that is intended for the graphical description of sequential and concurrent behavior: the sequential function chart (SFC). The SFC is based on Grafcet (David 1995) and represents a form of Petri net (with very special dynamics and functionality). Due to its high functionality, SFC can be easily applied for the structuring of a PLC program on a high level. However, it is cumbersome (and by the standard also not intended)

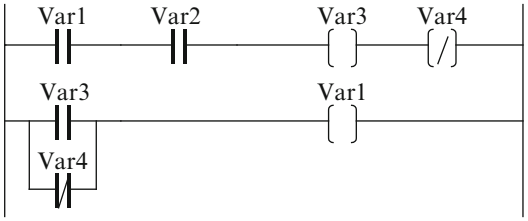


Fig. 1 Example of a PLC program written in Ladder Diagram (LD)

DOI 10.1007/978-1-4471-5102-9_257-1

to use for the specification of a low-level sequential algorithm, as, for example, the alternative switching between two motors.

In addition to the three graphical languages, there are also two textual languages in the standard: the assembler-like Instruction List (IL) and the Pascal-like Structured Text (ST).

The decision for one of the languages is based on functional aspects of the application to be realized (high-level languages SFC and ST vs. low-level languages LD, IL, and FBD) but also on traditions in the application domain (e.g., LD in automotive manufacturing vs. FBD in process industry), the geographical region (e.g., LD in the US vs. IL in Germany), and the preferences of the programmer (graphical vs. textual). To allow for flexible solutions and the optimal choice of languages, IEC 61131-3 allows the use of different languages for different parts of the control application.

An application in IEC 61131-3 is structured into Program Organization Units (POUs). Each of the POUs contains a header in a unified syntax for parameter and variable definitions and a body for the actual program code. This body can be written in any one of the defined PLC languages.

There are three types of POUs: Program, Function Block, and Function. A program is the top-level POU of a PLC application. Only in a program, variables can be linked to actual input and output ports. A program can call Function Blocks which in turn may call other function blocks. Programs and Function Blocks can also call Functions. A POU of type function has no internal memory while a Function Block has memory.

IEC 61131-3 introduced the type and instance concept into PLC programming. A Function Block is always the instantiation of a Function Block Type. Each instantiation gets its own name and variable space. This concept is similar to – but much older than – the class-object instantiation idea of object-oriented programming languages. The exclusive use of symbolic variables without direct references to hardware addresses or ports in Function Blocks allows their easy reuse in one or more applications and the definition of widely applicable Function Block (Type) Libraries.

Summary and Future Directions

PLCs are a proven technology in industrial automation. They follow a simple but deterministic execution and software model. This is the main reason why PLCs are still here and will be here for quite some time to come even if there is faster and fancier technology like embedded PCs available.

Currently the third edition of IEC 61131-3 is nearly ready for publishing. In addition to minor corrections, this new edition adds some concepts from object-oriented programming to the existing software model. First tools on the market already support these extensions.

For the future, two trends can be seen. First, there is a growing trend to integrate PLC programming into model-based software development processes: either by generating PLC code from existing model-based toolchains or by integrating model-based approaches, especially from the object-oriented domain, into PLC programming environments. Either way this is due to the fact that the complexity in PLC application is rising while the development time should be decreased.

Second, there is a growing interest in the use of formal methods in the PLC domain. In recent years, a lot of interdisciplinary work was aimed in this direction. This work results in the formalization of different steps in the control design process depending on what problems are to be solved (Frey and Litz 2000):

DOI 10.1007/978-1-4471-5102-9_257-1

1. The demand for reduced development time and the possible reuse of existing software modules result in the need for a formal approach to the development of the PLC programs.
2. The demand for high-quality solutions and especially the application of PLC in safety-critical processes result in the need for validation procedures, i.e., formal methods to prove specific static and dynamic properties of the programs.
3. The large numbers of already installed PLC programs, together with the high expense of programming, lead to the search for verification and validation methods that can be applied directly to programs written in PLC-specific programming languages such as ladder diagram.

To conclude, more than 50 years after its invention, the PLC is still an industrial success story, and due to ever-increasing demands on the complexity and correctness of its applications, it also still provides much room for further research and development.

Bibliography

David R (1995) Grafcet: a powerful tool for specification of logic controllers. IEEE Trans Control Syst Technol 3:253–268

Frey G, Litz L (2000) Formal methods in PLC programming. In: Proceedings of the IEEE conference on systems man and cybernetics SMC 2000, Nashville, Tennessee, pp 2431–2436

John K, Tiegelkamp M (2010) IEC 61131-3: programming industrial automation system: concepts and programming languages, requirements for programming systems, aids to decision-making tools, 2nd edn. Springer, Berlin

Vyatkin V (2011) IEC 61499 as enabler of distributed and intelligent automation: state-of-the-art review. IEEE Trans Ind Inform 7:768–781