KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Politechnika Gdańska
Wydział Elektrotechniki i Automatyki

PROGRAMMABLE CONTROLLERS

Part 1

Ireneusz Mosoń

Gdańsk, 2010

Publikacja jest dystrybuowana bezpłatnie

Materiał został przygotowany w związku z realizacją projektu pt. „Zamawianie kształcenia na kierunkach technicznych, matematycznych i przyrodniczych – pilotaż" współfinansowanego ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego
Nr umowy: 46/DSW/4.1.2/2008 – zadanie 018240 w okresie od 21.08.2008 – 15.03.2012

Materiały pomocnicze do przedmiotu „Sterowniki programowalne” prowadzonego w języku angielskim w semestrze zimowym w roku akademickim 2010/2011 dla V semestru specjalności zamawianej „Technologie informatyczne w elektrotechnice” na studiach pierwszego stopnia na kierunku Elektrotechnika

Preface

Complex automation in all branches of industry is one of the dominating tendencies in technical development nowadays. Programmable controllers are among the most important devices, implementation of which makes possible this complex automation . For the first time programmable controllers were designed, produced and implemented in control systems of production lines in General Motors automotive plants over forty years ago. At that time they were named “programmable logic controllers”. Today’s programmable controllers are not only “logic” as they were while used for the first time. Enormous changes in information technology have influenced and caused such changes in programmable controllers technology that increased controllers potential of signal processing, enabled easier programming, improved controllers reliability, and network functions have been developed. Therefore the number of programmable controllers implemented in industry increases constantly. Concurrently there is a rising demand for engineers prepared for designing, programming, starting-up and supervising control systems with programmable controllers.

The above listed trends clearly show that “Programmable controllers” is a subject that engineering students have to be taught during their first level of study (B.Sc). One of the groups of engineering students for whom at least basic knowledge of programmable controllers is vital in their future professional activity are electrical engineering students. The basic knowledge that electrical engineering students are expected to acquire comprises not only programming aspects (the ability of writing, testing and commissioning control programs) but also a variety of technical aspects – including mainly: systems design and development, networking, structuring of user programs etc. Moreover, as far as complex automation (with networking from a shop-floor to the management level) is concerned, a more system approach to programmable controllers and their use is necessary. Such an approach has been offered to students that study speciality “Information technologies in electrical engineering” in the “Electrical engineering” field of study.

The course “Programmable controllers” – according to the Study programme for full-time students in the field of “Electrical engineering” – is an obligatory course and is taught during the 5-th semester (3-rd year of study). The course comprises: 30 hours of lectures, 15 hours of tutorials and 15 hours of laboratory works during the semester (two hours of lectures, one hour of tutorials and one hour of laboratory per week). The course “Programmable controllers” can be characterised by its syllabus and expected learning outcomes. There are 5 ECTS credit points allocated to this course.

In the academic year 2010/2011 for the speciality “Information technologies in electrical engineering” the course is taught in English to help these students to improve their communication skills in English, to prepare them to international exchange of students and possible future professional work in an international team.

These materials (Part 1) written in English comprise basic information about programmable controllers and their programming, according to the syllabus of the course. Two programming software packages (Sucosoft S40 ver. 4.24 and Easy Soft CoDeSys ver. 2.3.5.) have been used to prepare examples and some software features that are included into the materials. Additionally, because the basic source of information about programmable controllers, their hardware and software, is the International Standard IEC 61131 (which is also the European Standard EN 61131, as well as Polish Standard PN-EN 61131), selected information from the standard has been included into this materials. Paragraphs and sentences that have been taken directly from the standard are written in blue instead of using parenthesis.

Part 2 of these materials is written in Polish to give students from the speciality “Information technologies in electrical engineering” an opportunity to learn terminology related to programmable controllers and control systems with programmable controllers in both languages: English and Polish. In the second part such problems as: structuring of user programs, control of sequential processes and networking programmable controllers are presented.

In the list of references (the list is common for both parts of the materials) not only references quoted in this materials have been included; there are also other references that can be helpful in understanding lectures, tutorials and laboratory tasks.

Contents

Syllabus and learning outcomes of the course "Programmable controllers"

Syllabus

LECTURE

Programmable controllers in control systems. Types, structure and principle of operation. Execution of the user program. Process image memory. Hardware characteristics. Interaction with a controlled process. Digital, analogue and special input/output circuits.

Fundamentals of programming. PN-EN 61131-3 standard. Programming model. Programming languages. Data types and declaration of variables. Addressing. Program organization units - programs, functions and function blocks. Creation of user functions and function blocks. Structuring of user programs. Factors of a program quality.

Networking programmable controllers. Network structures. Communication interfaces and transmission media. Methods of media access control. Communication protocols (Suconet K, Modbus RTU, Profibus DP, AS-i). Industrial Ethernet (protocols: Modbus TCP, Powerlink, Profinet).

Design of programmable controllers based control systems. Selection of a programmable controller depending on an application. Realization of a human - machine interface (HMI). SCADA programs.

TUTORIALS

Number systems used in programmable controllers. Data types and functions of their conversion. Creation of control algorithms; graphical elements of the algorithms. Programming software Easy Soft CoDeSys. Creation of control programs (in IL, LD, FBD, ST, CFC languages) and their debugging with the use of program simulator (virtual controller). Creation of visualisation applications. Programming of control of sequential processes in SFC language.

LABORATORY

Programming software Sucosoft S40 (structure; configuring control systems; editing, debugging, testing and documenting programs). Program for a conveyor control - I and II. Conversion functions and arithmetic operators. Counting events and compiler options. Creation of the user function block. Modifying programs and changing variable values in On-line mode. Programming PS4-200 and PS4-150 series controllers in the network (master - active slave).

Learning outcomes

Student describes types and structures of programmable controllers. Explains principle of programmable controller operation and principle of execution of the user program. Student selects programmable controllers for specific applications. Analyses requirements of control tasks and creates control algorithms. Writes, debugs and tests programs of low and middle complexity for control of different control objects. Creates user functions and function blocks. Creates simple visualisation applications.

1. Programmable Controller Basics

1.1. What is a programmable controller?

In brief it can be said that a programmable controller is a programmable electronic device used in control systems. More precise explanation is published in the International Standard IEC 61131-1 [40] (see also Chapter 5 of this materials).

The definition in the standard says that a programmable controller is a digitally operating electronic system, designed for use in an industrial environment, which uses a programmable memory for the internal storage of user-oriented instructions for implementing specific functions such as logic, sequencing, timing, counting and arithmetic, to control, through digital or analogue inputs and outputs, various types of machines or processes.

It also says that the programmable controller and its associated peripherals are designed so that they can be easily integrated into an industrial control system and easily used in all their intended functions.

The definition, especially when reading it fast, may seem not very clear, but is precise and refers to all programmable controllers; from the first programmable controller to the newest controllers that are produced currently. It should be mentioned that the definition is general (as it obviously should be) and it doesn't mention that a programmable controller is a microprocessor-based electronic system. This is so because first programmable controllers were developed, produced and implemented some years before the first microprocessors were produced and implemented in electronic systems.

In practice not definition from the standard but another definition referring to the performed functions and tasks can better explain what a programmable controller is.

According to definition of the authors of the references [16, 23] programmable controllers are industrial computers which, under the control of the real time operating system:

- with the use of input modules acquire results of measurements from digital and analog sensors and measurement devices;
- using acquired data from the controlled process or machine execute user programs that comprise coded data processing and control algorithms;
- in accordance to results of user programs execution generate control signals and, with the use of output modules, send them to actuators,

and additionally are able to:

- transmit data with the use of interfaces and communication modules;
- perform hardware and software diagnostic functions.

As mentioned in the standard, although for **P**rogrammable **C**ontrollers the abbreviation PC should be used, this abbreviation could lead to confusion because PC is commonly used as abbreviation for **P**ersonal **C**omputer. Therefore for programmable controllers the abbreviation PLC (**P**rogrammable **L**ogic **C**ontrollers) is used instead. Historically this is the first abbreviation used, because first programmable controllers were using and processing only logic (binary) signals. Despite the fact that currently programmable controllers operate not only on logic signals the abbreviation PLC is well recognized and still in common use in the automation industry and in the literature (in technical books, journals, documentations etc.).

In resent years however one more abbreviation – PAC – have become popular; this abbreviation stands for **P**rogrammable **A**utomation **C**ontroller. In the year 2002 the company National Instruments started to use this name for their programmable controllers. Very quickly also many other companies followed and named their new programmable controllers as programmable automation controllers (for example GE Fanuc and other companies). The intention of this change from PLC to PAC was to underline their increased functionality and processing capabilities.

PLC and PAC are not the only abbreviations that mean programmable controllers. Some companies use other names for their programmable controllers. For example the Austrian company B&R their programmable controllers names **P**rogrammable **C**omputer **C**ontroller – abbreviation PCC, whereas the Swiss company SAIA their programmable controllers names **P**rogrammable **C**ontrol **D**evice – abbreviation PCD.

1.2. Basic structure of a programmable controller

The basic functional structure of a programmable controller system is presented in the standard, and in these materials is shown in figure 5.1 in the 5-th chapter. However, such a detailed description of a programmable controller functions and hardware is not necessary in this chapter to explain the principle of operation of a programmable controller. A simplified hardware block diagram shown in figure 1.1 should be helpful instead.

This hardware block diagram consists of three blocks:

- inputs meaning input circuits;
- CPU (Central Processing Unit) in which a processor (microprocessor) and a memory are the most important parts to understand the programmable controller principle of operation;
- outputs meaning output circuits.

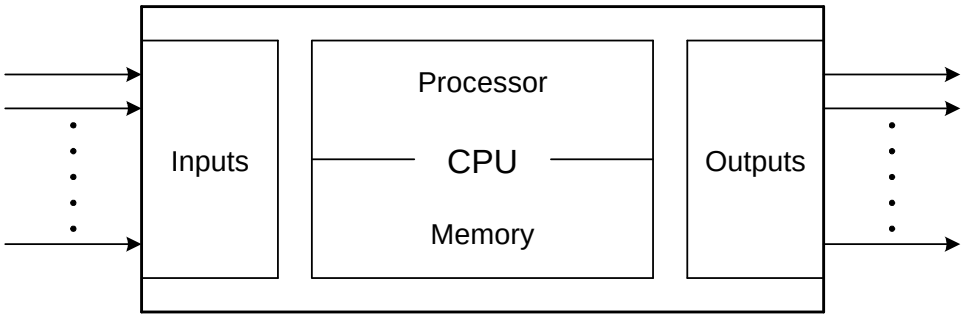

Fig. 1.1. Simplified hardware block diagram of a programmable controller

Input and output circuits are the programmable controller's interfaces through which it is connected to the controlled object – usually either a machine or machines in a production line/center or a technology process.

Each programmable controller has an internal memory (which is a RAM memory) that is used to store the application program and data. A part of the data memory is a process image memory. The process image memory consists of two parts. One part is a process input image memory – to store states of input signals during the program cycle. The other part is a process output image memory – to store states of output signals during the program cycle. Sometimes the process image memory is also called a process image register.

1.3. General principle of operation

The principle of operation of a programmable controller can be explained using a simplified diagram shown in figure 1.2 – and analysed together with the simplified hardware block diagram shown in figure 1.1.

The simplified diagram of a programmable controller operation consists of four blocks which names refer to their functions:

- reading inputs;
- execution of the application (user) program;
- writing outputs;
- operating system functions.

The three blocks on the left hand side in figure 1.2 refer to certain phases of the program cycle and are essential for explanation of the programmable controller operation.

Reading inputs

At the beginning of each cycle the processor reads the signal states of all inputs and stores them in the process input image memory. After reading states of the input signals do not change during the current cycle.

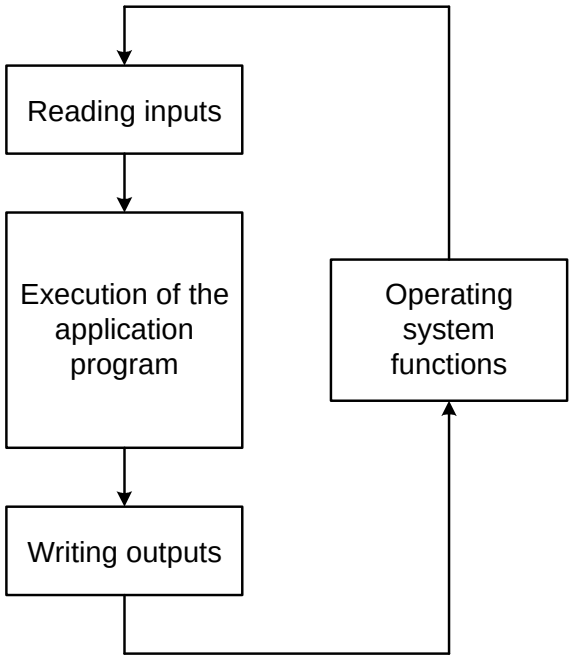

Fig.1.2. Simplified diagram of a programmable controller operation

If, during execution of the application program, an operand is the input signal its state is read from the process input image – not from the controller's physical (hardware) input. Therefore even if during program execution there will be a change in the state of the input signal, this will not be taken into account during the current program cycle.

The advantage of such a way of operation is that for a single cycle the whole program will be executed for the same states for each input signal and even if a change of an input signal occur this will not influence corresponding state in the process input image memory (until next reading of inputs in the next cycle).

Execution of the application program

Typically the application program is executed cyclically. The processor reads the application program memory locations one after another, from the beginning to the end of the program. This does not mean that in each cycle every instruction (every rung) is being processed. Some parts of the program can be omitted if their execution is conditional. Conditional execution of some program parts can depend on states of the controlled process or just on the program realization of the control algorithm.

The program performs all necessary logic and arithmetic operations on the data. Every time when an operand of the instruction is an input variable, its state is read from the process input image memory. The same rule applies to output variables. Every time when an operand of the instruction is an output variable, its state is changed in the process output image

memory. During a single program cycle the state of an output variable can be changed many times, but the valid state of this variable at the end of the program cycle is the last state of this variable written to the process output image memory.

In short it can be said that during execution of the application program the programmable controller's processor operates on images of physical (hardware) inputs and outputs stored in the process image memory.

Writing outputs

After execution of the application program states of output variables from the process output image memory are simultaneously written to the controller's physical (hardware) outputs. Thus all hardware outputs are updated simultaneously once in a cycle – at the end of the program cycle.

Operating system functions

All operating system functions can be divided into two groups.

The first group is performed before execution of the application program begins in a cycle or just after the end of execution of the application program in a cycle. Such functions are for example the watchdog timer resetting and diagnostic functions.

The second one is a group of functions that are performed simultaneously with the execution of the application program. They are event-driven functions because they are called up by various operating system interrupts. These interrupts make possible that several functions run virtually simultaneously. Functions belonging to this group are for example: monitoring the cycle time (watch dog), monitoring the power supply, processing the basic cycle pulses, communicating with a programming device – usually a personal computer with programming software or a hand-held programmer (however it is used very rarely nowadays).

As far as operating system functions are concerned it should be mentioned that apart from functions performed in every program cycle programmable controllers perform also operating system functions when the controllers are switched on or off.

Functions that are performed after switching on the controller are for example: self test when starting, initialisation of the CPU, determining the start-up behaviour (restart conditions), copying retentive data from the backup memory (in controllers with EEPROM) into the working memory.

A function that is performed after switching off the controller is for example backup of the retentive data into the backup memory (EEPROM).

The above listed functions are performed only once, before the first reading inputs phase of the cycle, but they are not shown in figure 1.2, as being of a minor importance to understand general principle of programmable controllers operation.

Typical execution of the application program in programmable controllers is cyclical, what means that program cycles are performed one by one without any delay between them.

1.4. Types and series of programmable controllers

Different divisions of programmable controllers into groups can be made depending on what factor will be taken into account. Usually programmable controllers construction, maximum number of inputs and outputs, the scale of applications in which controllers can be implemented are these factors depending on which the classification can be made.

Taking into account programmable controllers' construction they can be divided into:

- compact controllers;
- modular controllers.

Taking into account maximum number of inputs and outputs programmable controllers can be divided into:

- small – with a few dozen inputs and outputs;
- medium size – up to several hundreds of inputs and outputs;
- big – over one thousand inputs and outputs.

Taking into account the scale of applications programmable controllers can be divided into:

- controllers used in small application tasks – usually control of an individual machine or a small technological process;
- controllers used in medium-size application tasks – usually control of several machines in a production line, technological processes in different branches of industry;
- controllers used in large and complex automation tasks – usually complex production and technological processes in some branches of industry (chemical, petrochemical, pharmaceutical etc.).

Only the first classification is based on a clearly technical factor. The other two classifications are rather relative and subjective.

Compact programmable controllers in their single enclosure have all necessary systems to perform all programmable controller functions (figure 5.1). They have fixed

number of inputs and outputs of particular types. This does not mean that they cannot be expanded. They can often be expanded up to hundreds or even thousands inputs and outputs. For example Moeller PS4-201-MM1 programmable controllers have 8 digital inputs, 6 digital outputs, 2 analog inputs and 1 analog outputs. After connecting to PS4-201-MM1 maximum possible number of local expansion modules LE4 and also maximum number of remote modules EM4 (using only the controller's build-in network interface RS485 with Suconet K protocol), the total number of digital inputs and outputs in such a system would be 1006.

Modular programmable controllers do not have fixed hardware architecture as compact controllers. They consist of a rack (or base) and some number of modules. Modules are mounted in the rack or fixed on the base. The number of modules and their types depend on the application. Some parts and modules are obligatory, other modules can be chosen or not. Apart from the rack (base), power supply module and of course central processing (CPU) module, other modules are optional. These optional modules can be:

- digital input modules;
- digital output modules;
- digital input/output modules;
- analog input modules;
- analog output modules;
- analog input/output modules;
- analog modules for temperature measurements;
- communication modules;
- high-speed counter modules;
- positioning modules,

and others less frequently used (co-processor modules, stepper motor control modules, etc.). Modular programmable controllers can be expanded by remote racks (bases) connected to the main rack (base) by a network connection. Expansion capabilities of modular controllers and the maximum number of inputs and outputs are bigger than in case of compact controllers.

Producers of programmable controllers usually have and offer whole series of controllers (sometimes they name them "families" of controllers).
For particular models of programmable controllers that belong to the same series it is characteristic that:

- the same or similar technical solutions in particular controllers (and their modules) construction have been used;

- usually controllers from the same series are programmed with the use of the same programming software (programming package);
- very often application programs written for one programmable controller from a series can be used either directly or after a small adaptation (or just only a new compilation) in another controller from the series.

Below there are examples of series of programmable controllers that have been (or had been) produced in the last twenty years.

Simatic S5 and S7 series of programmable controllers are typical examples of programmable controllers series and replacement in production of the older series (S5) by the new series S7. In the older series Simatic S5 there are two compact mini programmable controllers (S5-90U and S5-95U), one failsafe mini programmable controller (S5-95F), one modular mini programmable controller (S5-100U) and three modular programmable controllers (S5-115U, S5-135U and S5-155U – all of them with various CPU modules). In the new series Simatic S7 there are in general three programmable controllers: S7-200, S7-300 and S7-400, each of them with a wide variety if CPU modules.

Another example are Moeller (earlier Klockner-Moeller) programmable controllers. In the last twenty years there were two changes in produced series of controllers: from the PS3 series (with PS3, PS306 compact programmable controllers, and PS316, PS32 modular programmable controllers) to the PS4 series (with PS-100, PS-150, PS-200, PS-300 compact programmable controllers, PS-400 compact fuzzy-logic programmable controller, and PS416 modular programmable controller), and recently to XC series (XC100 and XC200 modular programmable controllers with various CPUs available). Currently only some of programmable controllers from the PS4 series are produced (PS4-141-MM1, PS4-151-MM1, PS4-201-MM1, PS4-271-MM1, PS4-341-MM1) and all controllers from the series XC. Additionally there are produced Easy Control EC4P programmable controllers, and XV operator panels with touch screen and integrated programmable controllers. The latter ones and controllers from XC series are under the common name XControl.

2. Principle of Operation in Details

2.1. Cyclical and periodic operation

As it was mentioned earlier, standard operation of a programmable controller is cyclical processing of the program cycle. In this case program cycles are performed one by one without any delay – or it can be said "as quickly as possible" for the particular controller. But cyclical is not the only possible way of operation; programmable controllers operation can be also periodic. In such a case each next program cycle starts after a constant period of time – this time is counted from the beginning of the previous program cycle. If, for periodic operation, execution of the application program is shorter than the time set for periodic operation, the operating system waits until the set time for periodic operation has elapsed, and only then starts the next program cycle. While waiting for the set time of period to elapse, the CPU is not idle – it performs operating system functions.

For example: if the cycle time equals 5 ms while the set time for periodic operation is 50 ms, the operating system waits 45 ms before the next program cycle is started.

In case of a single control task execution in a programmable controller there is not a big difference between cyclical and periodic operation – provided that the time set for periodic operation is reasonably short (taking into account time constants of the controlled process).

The situation is different if it is possible to program and execute in a programmable controller more than one control task. Lets assume that there are, for example, two control tasks. One to control a fast process (with a short time constants); the other one to control a slow process (with long time constants). In such a case there would be advantageous to configure execution of the latter control task as periodic with the set time of period significantly longer (for example n times longer) than the time of execution of the former control task that has been configured as cyclical. Control signals of the faster process would be updated then approximately n times more frequently than control signals of the slower process.

2.2. Operation with or without process image memory

Typically programmable controllers operate with the use of the process image memory. However, the operation without using the process image memory is also possible. If, as shown in figure 2.1, phases "Reading inputs" and "Writing outputs" are excluded from

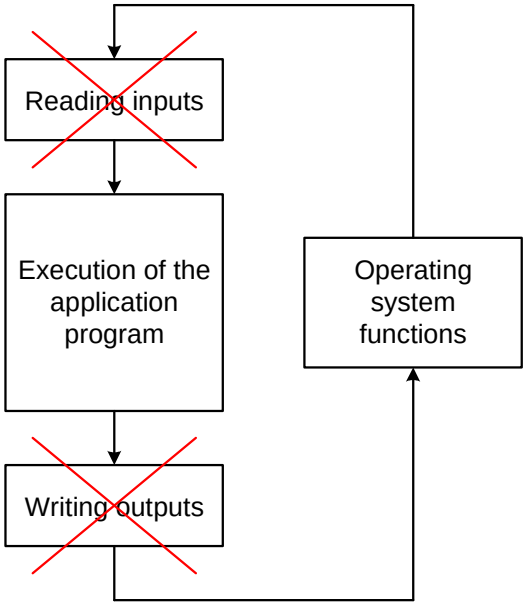

Fig.2.1. Excluding “Reading inputs” and “Writing outputs” in the simplified diagram

the program cycle, the controller operates without using the process image memory. In such a case the simplified diagram of a programmable controller operation consists only of two phases of the program cycle – as shown in figure 2.2.

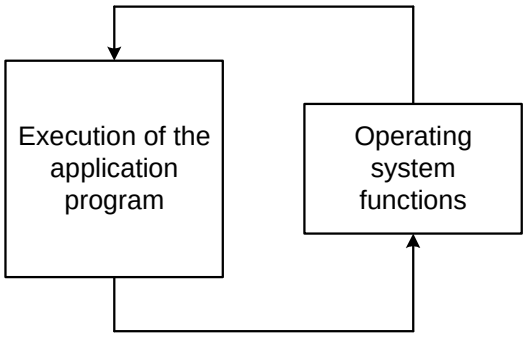

Fig. 2.2. Simplified diagram of a programmable controller operation without using the process image memory

In this case every time when an operand of the instruction is an input variable its state is read from the controller’s input (hardware input); analogously every time when an operand of the instruction is an output variable its state is changed in the controller’s output (hardware output).

An example of programmable controllers that work in such a way are PCD series controllers produced by SAIA. Operation without using the process image memory is their typical operation. Their outputs are updated at the point of time during the application

program execution when they are operands of the currently executed instructions. Writing programs for PCD series controllers it is necessary to take into consideration that parts of the application program can be executed for different states for each particular input variable. In some applications this necessity could be a disadvantage. On the other hand updating outputs during execution of the program is an advantage because the control system reaction is faster. If the user of the PCD series controllers wants to work with a process image memory, such a memory area have to be programmed by him.

Despite the fact that majority of programmable controllers use the process image memory, it is usually possible to read input signal states directly and to write output signal states directly. This is called direct access and this is necessary when signal states have to be processed immediately in the control program. The purpose of direct access to inputs and outputs is to shorten a control system reaction time.
Following are some examples of direct access to inputs and outputs in programmable controllers from different manufacturers.

In Siemens S5-115U modular programmable controllers digital module signal states are stored in two memory areas; the second one is a process image memory. It is possible to set parameters of this controller (changing one bit in the system data word SD120) that it will not use the process image memory. In this case exchange of information with digital modules takes place directly.

In GE Fanuc 90-30 series programmable controllers a special function block Do I/O (DO_IO) can be used to read selected input signal states directly, or to write selected output signal states directly. This function block is used during execution of the program cycle. Depending on the function block parameters the input signal states that have been read using this function block either modify the input signal states that had been read at the beginning of the program cycle (process image) or are written to a different memory area (for example markers area). Similarly programmable controllers outputs can be updated during the program cycle with signal states taken either from the process output image or from a specified memory area (for example markers area). If necessary this function block can be used many times in a program cycle.

In Koyo DL405 modular controller, in order to shorten the programmable controller reaction, so called “immediate inputs and outputs” that are special Boolean instructions can be used. This immediate instructions are used to read directly input states and write directly signal states to outputs during execution of the application program. The immediate

instructions take longer time to execute the program execution is interrupted while the CPU reads or writes signal states from a digital module.
Immediate input and immediate output instructions in a different way influence signal states that are stored in the process image memory.
Although an immediate input instruction reads the most current status of the input signal, it only uses the result to solve that one instruction. It does not update the process input image memory. Therefore, any regular (not immediate) instructions that follow will still use the input states (not updated) from the process input image.
The immediate output instruction writes the current status of the output signal to the corresponding output in a digital output module, and simultaneously updates this output signal state in the process output image memory.

In Moeller PS4 compact programmable controllers direct access to inputs and outputs is possible using special addressing. However, direct access to inputs and outputs is possible only to inputs and outputs of the controller itself (direct access is not possible to inputs and outputs of local expansion modules and remote expansion modules connected to the controller). Direct access is possible for single inputs, single outputs, input bytes and output bytes (for bigger controllers like for example PS4-341-MM1 also for input words and output words).
For example, standard addressing (when the process image memory is used) is %IB0.0.0.0 for the input byte, and %QB0.0.0.0 for the output byte; direct addressing (the process image memory is not used) is %IP0.0.0.0 for the input byte, and %QB0.0.0.0 for the output byte. Single inputs and outputs can also be accessed – as it was mentioned earlier; for example %IP0.0.0.0.5 means direct reading of the 5-th input in the 0 input byte, whereas %QP0.0.0.0.3 means direct writing to the 3-rd output in the 0 output byte.
For programmable controllers from the series PS4-150 and PS4-200 direct reading of inputs and writing outputs during execution of the application program do not change states of the corresponding inputs and outputs in the process image memory. In contrast to behaviour of these controllers, the process image of the PS4-300 series controllers is updated immediately during execution of the application program.

Almost all considerations so far about usage of the process image memory referred to digital (binary) inputs and outputs. For majority of programmable controllers using process image memory for digital inputs and outputs is a standard. The situation is different as far as other types of inputs and outputs are concerned. Some programmable controllers do not use the process image memory for analog inputs and outputs.

For example in Siemens S5-115U modular programmable controller analog module signal states are not written to the process image memory. They are read in or transferred to an analog output module directly.

As far as usage of process input image in programmable controllers is concerned it can be concluded that majority of programmable controllers, as a standard solution, use the process input image. On the other hand programmable controllers enable usage of direct access to their inputs and outputs. However, technical solutions of how this can be achieved are quite different for programmable controllers of different producers. Sometimes they can be different even for different programmable controllers of the same producer. Therefore detailed study of technical documentation of a particular programmable controller should always be recommended.

2.3. Control system and programmable controller response time

The control system response time is the period between the occurrence of an event and the reaction of the control system to this event.

The programmable controller response time is the period between the change of a signal state in the controller's input and the change of a signal state on the controller's output.

The latter response time is of course shorter than the former one. The difference is the sum of the following:

- time necessary for a sensor output signal change;
- time needed to transfer this signal to the controller's input;
- time needed to transfer controller's output signal to an actuator;
- time necessary for an actuator for its reaction.

The first time is not negligible because it is the response time of a sensor (for example an inductive proximity switch).

The second and the third time are negligible in case of standard wiring of the control system, but they are not negligible in case of usage of a bus system for sensors and actuators (for example AS-interface).

The forth time is also not negligible because it is the response time for an actuator (for example a contactor).

The sum of these times does not depend on the programmable controllers.

What should be analysed in this chapter is the programmable controller response time. This response time (often called the I/O response time) depends on:

- the delay of the controller's input circuit;
- the program cycle time;
- the delay of the controller's output circuit,

but is not just a sum of the listed factors, because the I/O response time depends also on the point in the program cycle when the field input changes states.

In practice the delay of the controller's input circuit is longer than the delay of the controller's output circuit. The delay of the controller's input circuit can be relatively big (in comparison to the program cycle time) if the function of the input circuit is also hardware filtering of input signals. The delay of the controller's output circuit is often negligible in comparison to the program cycle time.

Two cases should be analysed: when the input signal state changes just before the reading inputs phase of the program cycle (the best case) and when the input signal state changes just after the reading inputs phase of the program cycle (the worst case).

The I/O response time is the shortest when the input signal state change takes place just before the reading inputs phase of the program cycle. The I/O response time for this case is shown on the time diagram presented in figure 2.3.

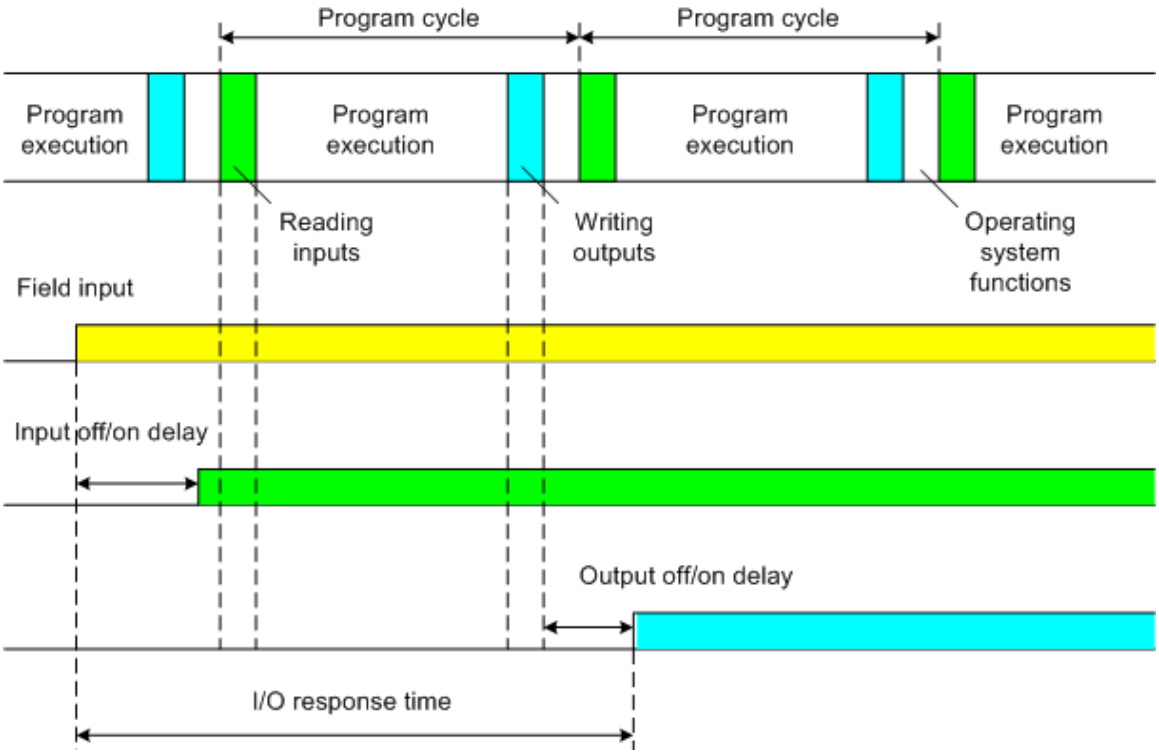

Fig. 2.3. The I/O response time – better case

In the first case the I/O response time equals (approximately):

Response time = Input delay + Program cycle time + Output delay

The I/O response time is the longest when the input signal state change takes place just after the reading inputs phase of the program cycle. The I/O response time for this case is shown on the time diagram presented in figure 2.4.

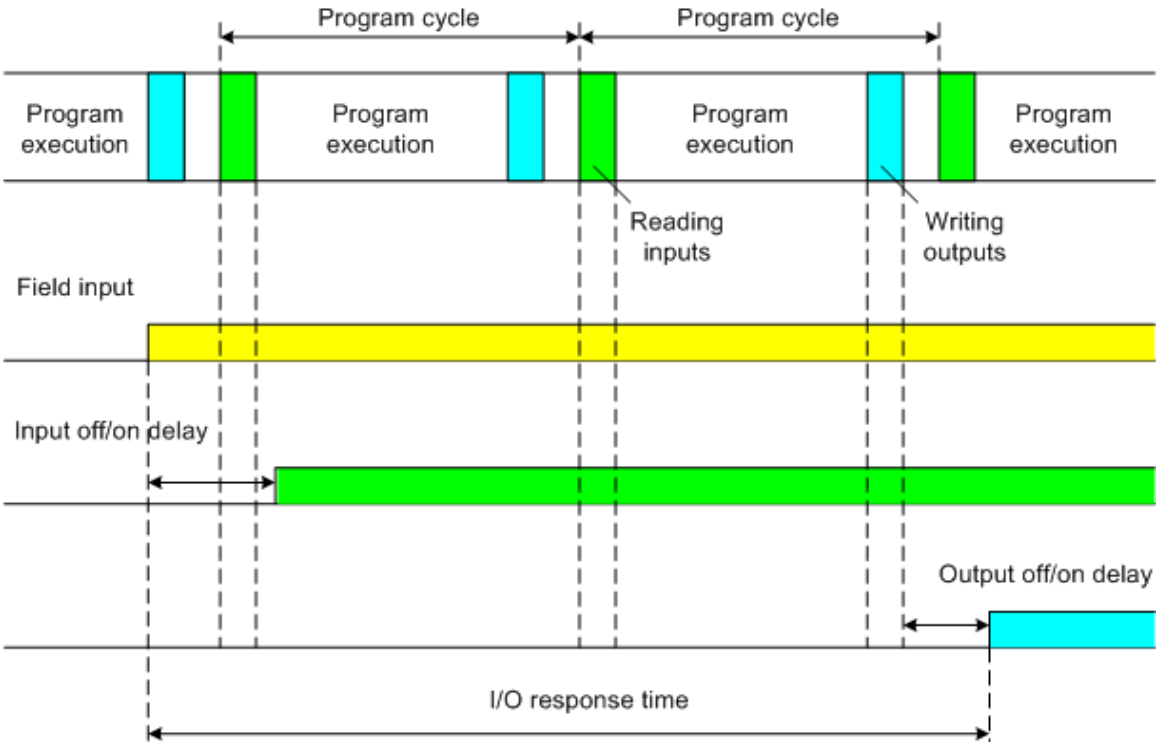

Fig. 2.4. The I/O response time – worse case

In the second case the I/O response time equals (approximately):

Response time = Input delay + 2 x (Program cycle time) + Output delay

Taking into account that input delay and output delay are usually substantially shorter than the program cycle time it can be concluded, that the maximum response time is (approximately) twice the program cycle time.

In many applications even the maximum I/O response time is such, that the timing analysis is, in fact, not relevant. However, some applications do require extremely fast responses. In such cases direct access to controller's inputs and outputs can be used.

Using direct access with Boolean instructions during execution of the application program not always is effective and shortens the response time. It depends on the point in time when the change of the input signal took place; if before the instruction with direct

access then the response time is shorter, whereas if after the instruction with direct access then it does not influence the response time.

Instructions with direct access are always effective and make the response time shorter, if used in special function blocks. For example in Moeller PS4 series controllers there are three so called alarm function blocks. These are: CounterAlarm, EdgeAlarm and TimerAlarm. The first one is associated with the high-speed counter input, the second one with the interrupt input, and the third one with internal time interrupts. Direct access to inputs and outputs, if used in these function blocks are always effective and the response is always fast.

If, for example, the interrupt input (with the address %I0.0.0.0.1 in PS4-201-MM1 programmable controller) with the function block EdgeAlarm is used, outputs with addresses %QP… will update the controller's output signals at the point in time they are used in this function block (FB). The I/O response time for this case is shown on the time diagram presented in figure 2.5.

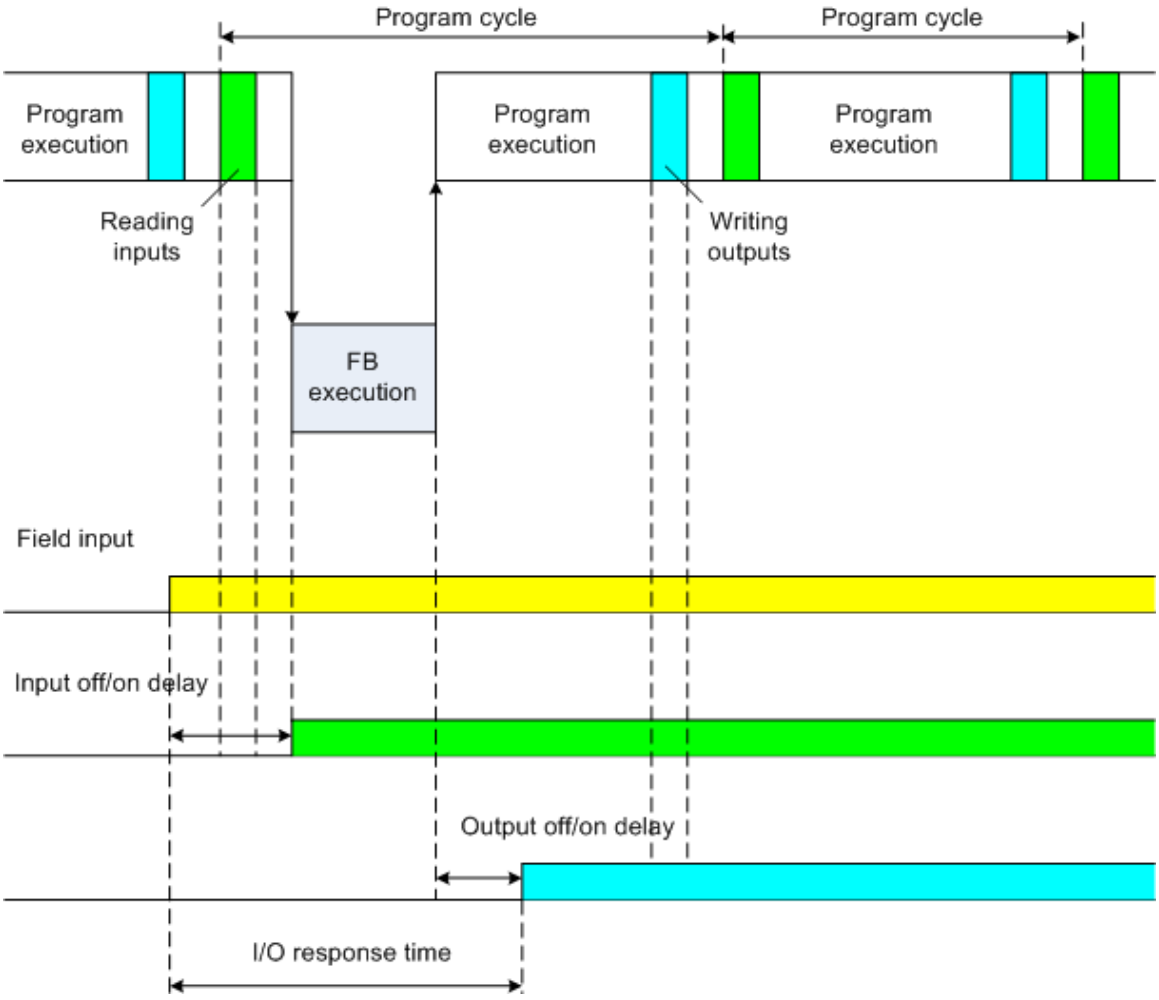

Fig. 2.5. The I/O response time when using the EdgeAlarm function block

The EdgeAlarm function block can be called in the program as a result of the input signal rising edge in the controller's interrupt input, or as a result of the input signal falling edge.

There is one more problem concerning digital input signals that has not been discussed yet. In the preceding analysis of programmable controllers' input signals it was always assumed (however never underlined) that the duration of an input signal is longer than the program cycle time. Such signals can always be detected, because for at least one reading inputs phase of the cycle their state is high.

The situation is different in case of signals which duration is shorter than the program cycle time. They are detected in a stochastic manner; if it happens that they exist during the reading inputs phase of the program cycle – they are detected; if opposite – they are not detected.

However there are programmable controllers that have inputs on which such short signals are always detected in a fully deterministic way. For example, Telemecanique TSX07 compact programmable controller has inputs from which short changes of input signal states (shorter than one program cycle and existing between consecutive reading inputs phases) can be stored and will have the high state during the next program cycle.
The time diagram of the described case is shown in figure 2.7.

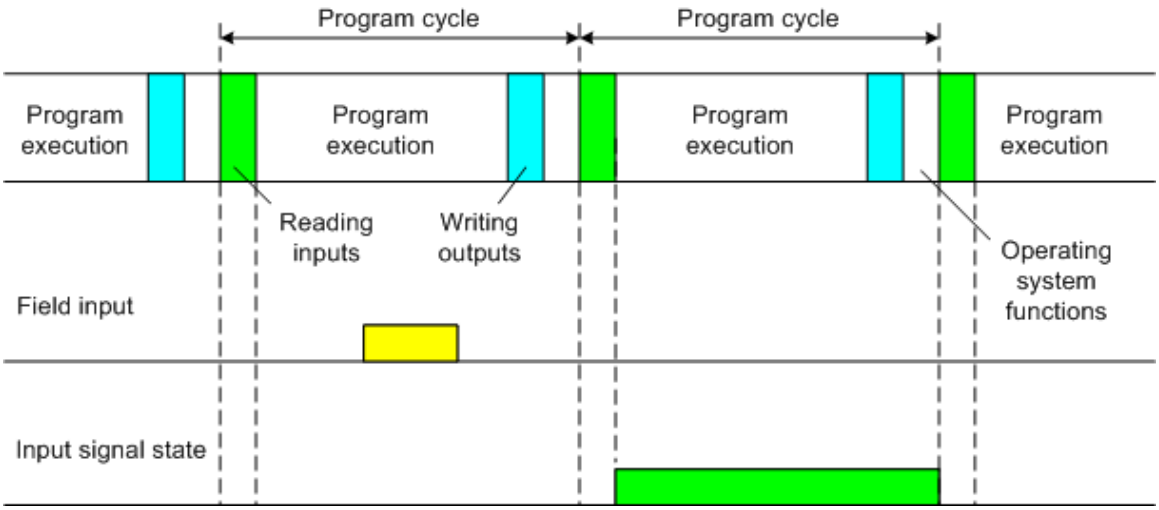

Fig. 2.7. Detection of very short input signals in TSX07 programmable controllers

As far as methods of shortening of the I/O response time are concerned it can be concluded again that apart from common methods, specific technical solutions applied in programmable controllers of particular producers can be found in their technical documentations.

3. Examples of the Program Cycle Description

3.1. First example – Moeller PS4 series

First example refers to PS4 series compact programmable controllers produced by Moeller Company. These programmable controllers can be in three different operating states: Run, Ready, and Not Ready. Only when the controller is in Run state it executes the user program. The program cycle for these programmable controllers is shown in figure 3.1.

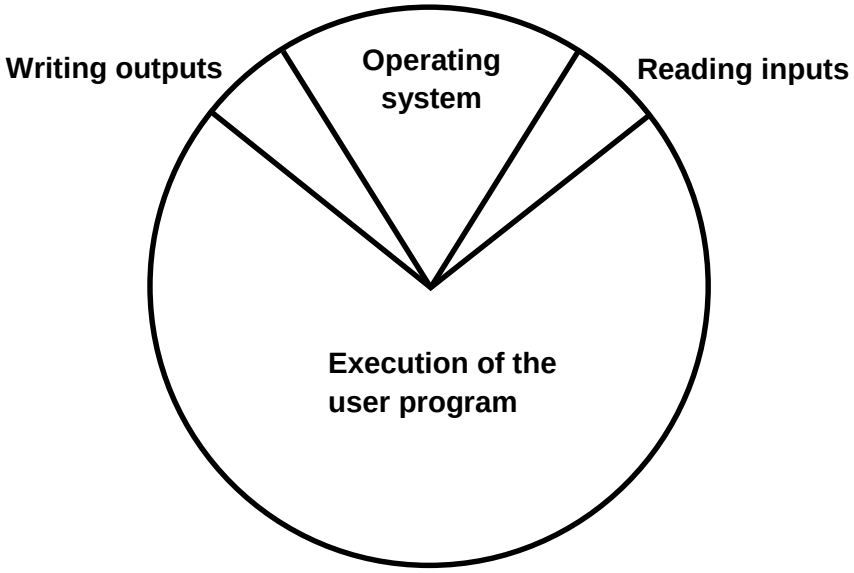

Fig. 3.1. The program cycle for Moeller PS4 series programmable controllers

The CPU operating system activities during particular phases of the working cycle are as follows:

Reading inputs

Before the user program execution begins, during the reading inputs phase of the cycle the CPU reads (simultaneously) the states of all inputs and stores these values in the input image register as high (H) or low (L) – for digital signals. The status image of all inputs remains constant for a whole program cycle. Changes in status of the inputs which can take place within processing of the program are therefore ignored by the controller.

Execution of the user program

During this phase of the program cycle the user program is executed. Each instruction of the program is distinguished by its memory location number, the address. The program counter has to select these addresses step-by-step. It starts from the zero address and ends on the address with the instruction of the end of the program. During execution of the user program

every time when an input is the operand of an instruction its status is read from the input image register. The output results of the program sequences are stored temporarily in the output image register.

Writing outputs

At this phase (writing outputs) of the program cycle, after executing the whole user program, states of the outputs from the output image register are allocated together (simultaneously) to the controller physical outputs.

Operating system

During this phase of the program cycle, the controller's operating system activities are carried out. These system activities are independent of current program processing. For example signal transfer between the programmable controller and a programmer. The programmer can be a PC with programming software (Sucosoft S40) or a hand-held programmer (during on-line operation with the programmer connected).

Looking at the program cycle for Moeller PS4 series programmable controllers it can be concluded that it comprises the same phases as the diagram in figure 1.2. The names of the cycle phases are almost the same, and the only difference is its circular form.

3.2. Second example – Hitachi E series

Second example refers to E series compact programmable controllers produced by Hitachi Company. The E series programmable controllers have different number of digital inputs/outputs (20, 28, 40, 64), are locally expandable (by additional 64 inputs/outputs), and can have relay or transistor outputs. There are three modes of operation for these controllers: PROG – programming mode, TEST – test mode, and RUN – operation mode. The program cycle for these controllers is shown in figure 3.2.

The CPU operating system activities during particular phases of the program cycle are as follows:

Input processing

During this phase the ON/OFF status of the external inputs is taken into the data RAM of the E series. For each input its status in the data RAM remains unchanged even if the ON/OFF status of any external input changes during the arithmetic processing of the program. The status change will be taken into the data RAM when input processing is made for the next

scan. To be sure that the change of the input signal status will be taken into the data RAM, the pulse width should exceed one scan time.

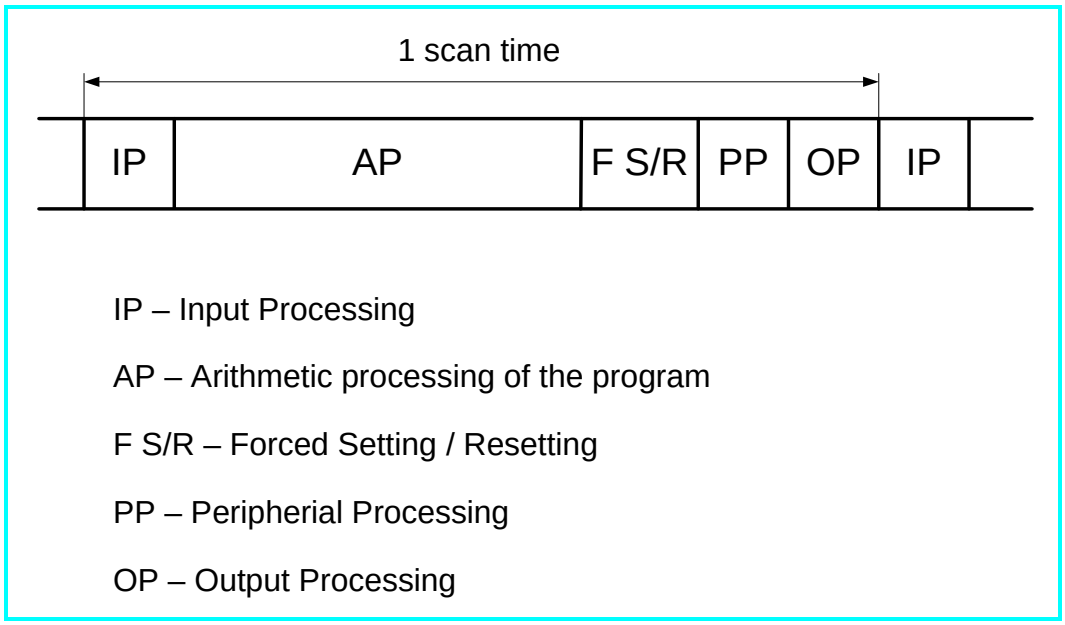

Fig. 3.2. The program cycle for Hitachi E-series programmable controllers

Arithmetic processing of the program

A written program is arithmetically processed starting from the first step according to the commands of the program. The arithmetic processing causes the contents of external outputs and internal outputs (i.e. markers and registers) to be changed from time to time in the data RAM.

Forced setting/resetting

During operation of the programmable controller, forced setting will be made after completion of all operations of the program. Once this function is activated, the power failure-protected internal outputs, timers and counters in the data RAM can be set or reset.

Peripheral processing

In case the ON/OFF status of input/output or the current value of timer/counter are monitored via the programmer, the content of the data RAM is displayed during the peripheral processing phase.

Output processing

During this phase of the scan the ON/OFF status of each external output in the data RAM is sent out to the external output terminal (driving the output relays).

Looking at this example it can be concluded that despite different names of the program cycle phases in figure 3.2 they are very similar to those at the simplified diagram shown in figure 1.2: reading inputs is named input processing, execution of the user programme is named arithmetic processing of the program, writing outputs is named output processing, and finally forced setting/resetting and peripheral processing are the operating system tasks.

3.3. Third example – Koyo DL series

Third example refers to DL205 and DL405 modular programmable controllers produced by Koyo Company. These programmable controllers' CPU has two operating modes that allow different types of operations. These modes are: Program Mode and Run Mode. The program cycle for these controllers is shown in figure 3.3.

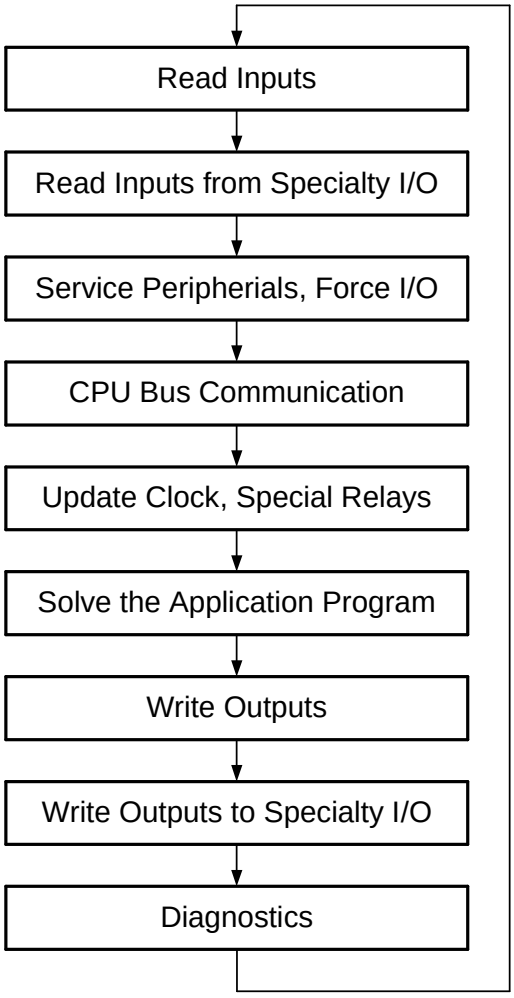

Fig. 3.3. The program cycle for Koyo DL series programmable controllers

The CPU operating system activities during particular phases of the program cycle are as follows:

Read Inputs

The CPU reads the status of all input modules present in the local CPU base and local expansion bases. The status of each input is stored in the process input image register.

Read Inputs from Specialty I/O

After the CPU reads the inputs from the input modules, it reads ant input point data from the specialty modules, that are installed, such as High Speed Counter (HSC) modules, etc. In this phase of the cycle also the input status from Remote I/O are read.

Service Peripherals and Force I/O

After the CPU reads the inputs from the Specialty I/O modules, it reads any attached peripheral devices. This is primarily a communication service for a programming device to see if any input or output status needs to be modified. Or, the programmer may be requesting a Run-Time Edit or a change to Program Mode. The CPU also gets any output forcing information during this phase.

One of the most popular requests is forcing an input on, even though it is really off. This allows the programmer to change the point status that was stored in the process image register. This value will be valid until the process image register location is written to during the Read Inputs phase of the next cycle. This forced value is used in solving the application programme, but only during the current cycle. In the next cycle if there is no another request for a force, the process input image register maintains the status obtained during the reading of the inputs.

CPU Bus Communication

During this phase of the cycle the CPU communicates with intelligent modules (like, for example, Data Communication Modules) that must be placed in the CPU base. The CPU performs both read and write requests.

Update Clock, Special Relays

During this phase of the cycle special memory locations that hold the Clock and the Calendar (i.e. the Real Time Clock – RTC) information are updated on every cycle. This information can be used therefore in the application programme.

There are also several different Special Relays, such as diagnostic relays, etc., that are also updated during this phase of the cycle.

Solve the Application Program

The CPU evaluates each instruction in the application programme during this phase of the cycle. The instructions define the relationship between the input conditions and the desired output response.

The CPU uses the process output image register to store the status of the desired action of the outputs. The actual outputs are updated during the Write Outputs phase of the cycle.

If the CPU have obtained and stored forcing information when it serviced peripheral devices, the process output image register also contains this information, but if an output point was used in the application programme, the results of the programme solution will overwrite any forcing information that was stored.

Write Outputs

After solving the application programme the CPU sends current output states of the process output image register to the output modules – located in the local CPU base or the local expansion bases. Forced outputs that are not used in the programme are also sent and corresponding outputs continue to be turned on until their forcing will be released.

Write Outputs to Specialty I/O

After updating the outputs in the local and expansion bases the CPU sends the output information that is required by any specialty modules which are installed. In this phase of the cycle also the information from the process output image register that is intended for the Remote I/O racks is updated.

Diagnostics

During this part of the cycle, the CPU performs all system diagnostics and other tasks such as calculating the scan time, updating special relays and resetting the watchdog timer. There are many various error conditions that are automatically detected.

One of the most important CPU activities during this phase of the cycle is the scan time calculation and the watchdog timer control. The watchdog timer stores the maximum time allowed to solve the application programme and complete a cycle. If this time is exceeded the CPU will enter the Program Mode and will turn off all outputs. The default value (for the DL405 CPU) set from the factory is 200 ms. It is possible to view the minimum, maximum and current scan time. An error is automatically signalised.

Looking at this example it can be concluded that despite different names of the program cycle phases in figure 3.3 they are very similar to those at the simplified diagram shown in figure 1.2: reading inputs is named read inputs and read inputs from specialty I/O,

execution of the user programme is named solve the application program, writing outputs is named write outputs and write outputs to specialty I/O, and finally service peripherals and force I/O, CPU bus communication, update clock and special relays, diagnostics are all the operating system tasks.

3.4. Fourth example – GE Fanuc 90-30 series

The forth example refers, among others, to 90-30 series and VersaMax programmable controllers produced by GE-Fanuc Company. In documentations of these controllers the sequence of operations necessary to execute a program one time is called a sweep. The sweep includes: obtaining data from input devices, executing the logic program, sending data to output devices, performing internal housekeeping, diagnostics, servicing the programmer, and servicing other communications. The program cycle (the sweep) is shown in figure 3.4.

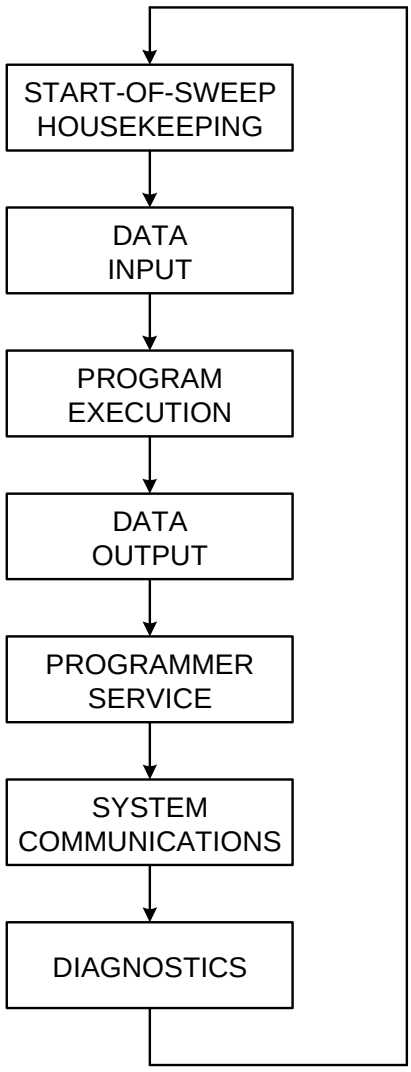

Fig. 3.4. The program cycle (the sweep) for 90-30 series programmable controllers

Programmable controllers of this series normally operate in *standard program sweep* mode. They can also operate in *constant sweep* mode. In both listed above modes the controller executes the logic program. Other operating modes include *stop with I/O disabled* mode, and *stop with I/O enabled* mode. In the latter two modes the logic program is not executed. Each of the listed modes is controlled by external events and application configuration settings. The programmable controller makes the decision regarding its operating mode at the start of every sweep.

The CPU operating system activities during particular phases of the program cycle (for standard program sweep) are as follows:

Housekeeping

During this phase of the sweep (cycle) all of the tasks necessary to prepare for the start of the sweep are performed. Among the most important tasks are: calculation of the sweep time, determination of an operating mode of the next sweep, updating fault reference tables, updating timer values, and reset of the watchdog timer. If the controller is in constant sweep mode, the sweep is delayed until the required sweep time elapses. If the required time has already elapsed, the system variable meaning oversweep is set, and the sweep continuous without any delay.

Data input

During this phase of the sweep all input modules are scanned, and the obtained data are stored in discrete inputs memory (%I) or analog inputs memory (%AI) accordingly. Any data received from Genius communication modules are stored in the part of memory for global data (%G). If the CPU is in the stop with I/O disabled mode, the input scan is skipped.

Program execution

The application logic execution starts immediately after the data input phase of the sweep. Solving the program logic provides a new set of output signals which are stored in discrete outputs memory (%Q) or analog outputs memory (%AQ) accordingly. In some CPU models there are logic co-processors; the logic co-processor executes the Boolean instructions. In such a case the time needed to solve the application program is shortened.

Data output

During this phase of the sweep, following the program execution, outputs are updated according to data from discrete outputs memory (%Q) and analog outputs memory (%AQ). Global data (%G) are sent to Genius communication modules. This phase continues until all outputs in output modules are updated. If the CPU is in the stop with I/O disabled mode, the

output scan is skipped.

Programmer service

This phase of the sweep is performed if a programmer is connected to the programmable controller. The programmer is either a PC with a programming package (VersaPro or Logicmaster) or the Hand-Held Programmer, however usage of the latter is very rare nowadays. During this phase of the sweep the CPU executes the programmer communication window. The programmer can communicate with the CPU or with intelligent option modules. In the default limited window mode the CPU performs one operation for the programmer each sweep. If the programmer makes a request that requires more than 6 ms to process (for some CPU models 8 ms), the request processing is spread out over several sweeps – lengthening each of these sweeps by no more than 6 (or 8) ms.

System communications

In this phase of the sweep communication requests from intelligent option modules are processed. Communication with intelligent option modules is performed in the order of appearing requests, but according to the rule – communication with one intelligent option module in a sweep. In the default *run-to completion* mode, the length of the system communications window is limited to 50 ms. If an intelligent option module makes a request that requires more than 50 ms to process, the request is spread out over multiple sweeps – lengthening each of these sweeps by no more than 50 ms.

Diagnostics

Diagnostics is made as a checksum calculation performed on the user program at the end of every sweep. The number of words to be checked can be specified by the user (from 0 to 32). If the calculated checksum does not match the reference checksum, the system variable meaning the program checksum failure is set. This causes a fault entry to be inserted into the programmable controller fault table and the controller mode to be changed to stop. The default number of words for checksum calculation is 8.

In the standard program sweep, each sweep executes as quickly as possible, with a varying duration of a sweep. An alternative to this is a *constant sweep* mode; in this mode duration of each sweep is the same. A value from 5 to 200 ms (for some CPU models up to 500 ms) can be set for the constant sweep timer.

Constant sweep mode is used if it is necessary to update input and output variables at a constant frequency (as required in many control algorithms). Constant sweep mode is also used to ensure that a certain amount of time elapses between outputs update in current sweep

and reading inputs in the next sweep (this time delay is necessary to be sure that particular input signals have changed their states after the output signals that influence these input signals had been changed).

Looking at this example it can be concluded that despite small differences in names of the program cycle phases in figure 3.4 they are very similar to those at the simplified diagram shown in figure 1.2: reading inputs is named data input, execution of the user programme has almost exactly the same name – program execution, writing outputs is named data output, and finally housekeeping, programmer service, system communications, and diagnostics are all the operating system tasks.

4. Programming of Programmable Controllers

In this chapter basic information about programming programmable controllers is presented. This chapter is based on the third part of IEC 61131 and covers main terms and ideas – presented with more details during lectures. In order to learn the whole material Readers should refer to the standard itself or to one of the references [16, 23], in which the content of the standard is presented more widely.

The third part of the standard is tightly connected to the first part of the standard, in which definitions are established and characteristics of programmable controllers and programmable controller systems are presented. Therefore a lot of information (all necessary), as the selected information from IEC 61131-1, is placed in the last chapter of this materials. All definitions and programmable controllers characteristics to which there are references in the programming model are in this last chapter.

In this materials the main stress is put on these elements of the programming model according to IEC 61131-3 that have been implemented and are used in programming packages Sucosoft S40 and Easy Soft CoDeSys (that are used during tutorials and in the laboratory). Presentation of other elements of the programming model (for example: tasks, access paths) is less detailed.

4.1. Programming model according to IEC 61131-3

The third part of IEC 61131 specifies syntax and semantics of programming languages for programmable controllers. Syntax describes language elements and the way they can be used whereas semantics – their meaning.
Semantics is the relationship between the symbolic elements of a programming language and their meaning, interpretation and use.

The standard defines basic terms, general rules, programming model, communication model as well as basic types and data structures.
In IEC 61131-3 the elements of programmable controller programming languages are classified as follows:

- Data types;
- Variables;
- Program Organization Units (POUs):
  - Functions;
  - Function Blocks;

  - Programs;
- Configuration elements:
  - Global variables;
  - Resources;
  - Tasks;
  - Access paths;
- Sequential Function Chart elements.

The software model from the standard is shown in figure 4.1. Definitions and explanations to the software model elements are as follows.

A configuration is a language element which corresponds to a programmable controller system as defined in IEC 61131-1.

A resource corresponds to a “signal processing function” and its “man-machine interface” and “sensor and actuator interface” functions (if any) as defined in IEC 61131-1.

A configuration contains one or more resources, each of which contains one or more programs executed under the control of zero or more tasks. A program may contain zero or more function blocks or other language elements as defined in this part.

Configurations and resources can be started and stopped via the “operator interface”, “programming, testing, and monitoring” or “operating system” functions defined in IEC 61131-1. The starting of a configuration shall cause the initialization of its global variables according, followed by the starting of all the resources in the configuration. The starting of a resource shall cause the initialization of all the variables in the resource, followed by the enabling of all the tasks in the resource. The stopping of a resource shall cause the disabling of all its tasks, while the stopping of a configuration shall cause the stopping of all its resources.

A task is an execution control element providing for periodic or triggered execution of a group of associated POUs.

The standard describes also mechanisms for the control of tasks and mechanisms for the starting and stopping of configurations and resources via communication functions.

Yet, for majority of programmable controllers (and programmable controller systems) not all language elements that are present in the programming model, have been implemented in operating systems of controllers and their programming packages. Still, the most common situation for programmable controllers working in different control systems is that the controller performs one application program and there is only one resource in the

configuration (which is usually not explicitly declared). For programmable controllers that are used during tutorials and in the laboratory instead of configurations, topologies of control systems are being created. Other language elements of the programming model are created either for the program (for example global variables) or for the topology (for example communication functions).

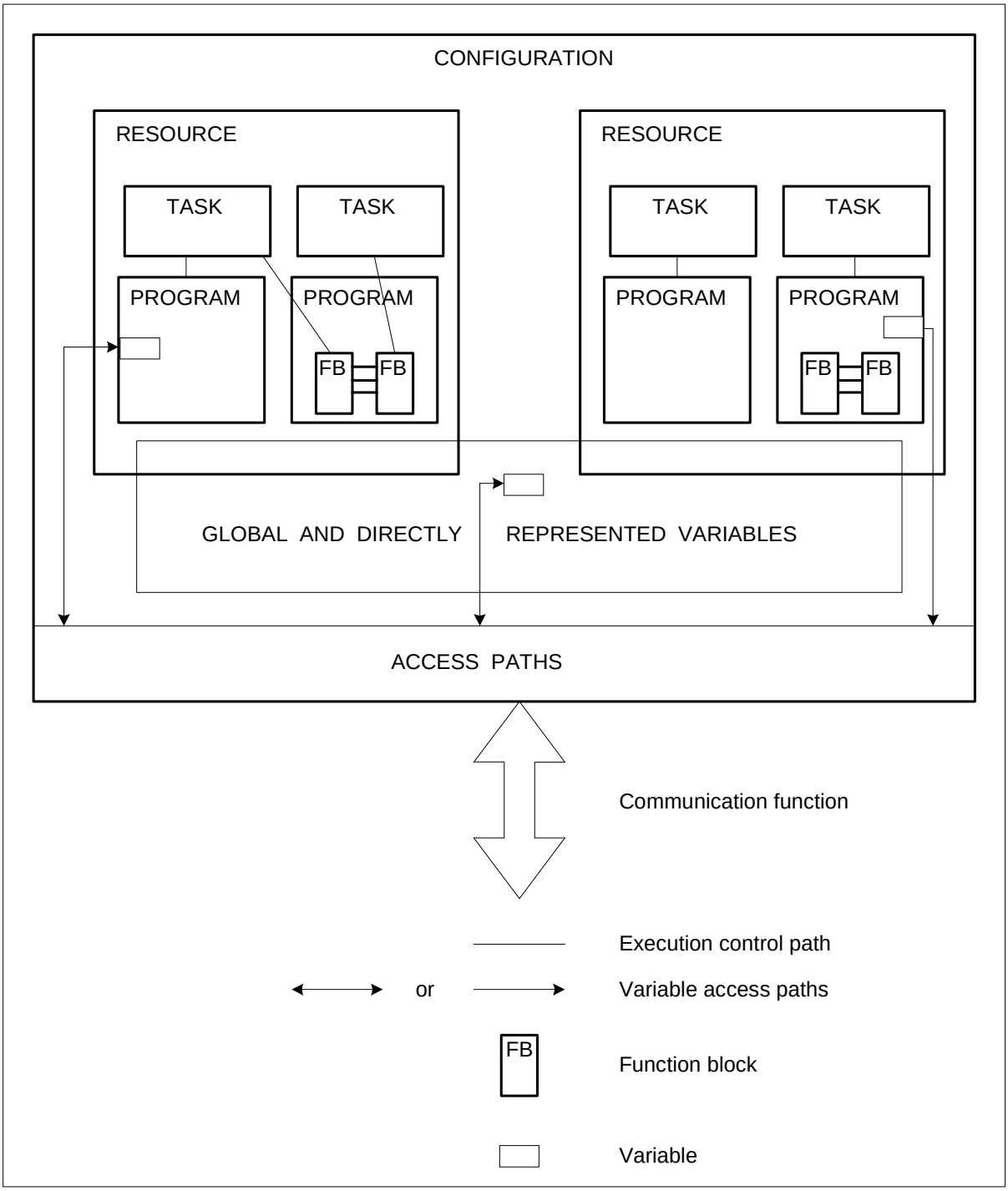

Fig. 4.1. Software model

Communication functions are also described in IEC 61131-3 while describing the communication model. In the communication model the ways that values of variables can be communicated among software elements are presented.

Data exchange between program elements in programmable controller systems can take place between:

- elements of a single program (for example, between function blocks with the use of input and output variables);
- programs within a single configuration (for example, with the use of global variables);
- different configurations (for example, using declared access paths or with the use of communication functions).

Communication functions are executed by communication function blocks and can perform data exchange not only between different configurations (topologies) but also within a single configuration or even within a single program.

4.2. Data types and variables

A number of elementary (pre-defined) data types are recognized by the standard. Additionally, generic data types are defined for use in the definition of overloaded functions. and a mechanism for the user or manufacturer to specify additional data types (derived data types) are also defined.

A data type is a set of values together with a set of permitted operations.

A generic data type is a data type which represents more than one type of data.

Overloaded: with respect to an operation or function, capable of operating on data of different types.

Certain data types can be grouped together to form generic data types. Generic data types are identified by the prefix “ANY”.

Generic data types and their hierarchy:

ANY ⇒ **(1)** ANY_BIT (BOOL, BYTE, WORD, DWORD)

⇒ **(2)** ANY_NUM ⇒ **(2a)** ANY_INT (SINT, INT, DINT, USINT, UINT, UDINT)

⇒ **(2b)** ANY_REAL (REAL, LREAL)

⇒ **(3)** ANY_DATE (DATE, TIME_OF_DAY, DATE_AND_TIME)

⇒ **(4)** TIME

⇒ **(5)** STRING

⇒ **(6)** Derived data types

Among other groups of data types there is also a group of derived data types.
Derived data types are special manufacturer or user defined data types derived from elementary data types and which have been assigned a new name. They are declared with the keywords TYPE … END_TYPE and can thus be used with the new names in variable declarations.
Derived data types can be divided into:

- Enumerated data type;
- Sub-range data type;
- Array data type;
- Structured data type.

Examples of derived data type (with their characteristic) are shown later in the current chapter.

Variables provide a means of identifying data objects whose content may change, for example, data associated with the inputs, outputs, or memory of the programmable controller. A variable can be declared to be one of the elementary types or one of the derived types.
Declaration is the mechanism for establishing the definition of a language element. A declaration normally involves attaching an identifier to the language element, and allocating attributes such as data types and algorithms to it.
All variables can be divided into:

- directly represented variables;
- symbolic variables.

Direct representation is a means of representing a variable in a programmable controller program from which a manufacturer-specified correspondence to a physical or logical location may be determined directly.

The manufacturer shall specify the correspondence between the direct representation of a variable and the physical or logical location of the addressed item in memory, input or output. When a direct representation is extended with additional integer field separated by periods, it shall be interpreted as a hierarchical physical or logical address with the leftmost field representing the highest level of the hierarchy, with successively lower levels appearing to the right.
For example:

%I0033 – if this digital input variable refers to 90-30 series modular programmable controller in which the first three modules are digital input modules (with 16 digital inputs each), this is the first input in the third module.

%I1.2.3.1.7 – if this digital input variable refers to PS4 series compact programmable controller with EM4 network stations and LE4 local expansion modules, this is the seventh digital input in the first byte, in the third module, in the second network station, in the first communication line.
In this case the physical hierarchy is: controller – network – station – module – input byte – single digital input.

If the hierarchy is logical it is usually with no relation to the physical structure of the programmable controller's inputs, outputs, memory.

For directly represented variables their memory location is always known. Depending on the variable it can be stored in the controller's memory or in the communication buffer (network variables). It should be remembered that digital input and output variables are stored in the process image memory.

The available addresses depend on the programmable controllers hardware.
The following symbols can be used to specify a variable length:

Symbol	Meaning	Length	Examples
X or none	Bit	1 bit	%I0.0.0.0.3 ; %M0.0.0.1.7
B	Byte	8 bits	%IB0.0.0.0 ; %MB0.0.0.1
W	Word	16 bits	%IW0.0.0.0 ; %MW0.0.0.2
D	Double word	32 bits	%ID0.0.0.0 ; %MD0.0.0.8

Remark: Examples shown in the table are input and memory addresses used in PS4 series programmable controllers. Double word applies only to PS416 modular programmable controller.

Apart from direct representation of variables there is also symbolic representation of variables.
Symbolic representation means the use of identifiers to name variables.
An identifier is a string of letters, digits and underline characters which shall begin with a letter or underline character.
The case of letters shall not be significant in identifiers; the underline characters positions in identifiers are significant. Identifiers are usually called variable names.

In contrast to directly represented variables there is automatic memory allocation of symbolic variables.

Symbolic names usually are but do not have to be assigned to directly represented variables. In this case (symbolic names not assigned) the operands are addressed with their physical addresses – starting with the % character. In the declaration the keyword AT is entered before the physical address – separated by a space. However even with directly represented variables the use of symbolic names is recommended because this enables faster modifications in POUs while programming, debugging, testing and starting-up programs.
An operand is a language element on which an operation is performed.
An operator is a symbol that represents the action to be performed in an operation.

For both – direct and symbolic – variable representation an initial value can be assigned to the variable.
An initial value is the value assigned to a variable at system start-up.

If an initial value is not assigned during declaration of a variable, the variable will be assigned its default value at the programmable controller start-up. The default value depends on the data type of the variable. For variables of ANY_BIT, ANY_NUM and TIME data types their default initial value is zero. For variables of STRING data type their default initial value is empty string. For variables of ANY_DATE data types their default initial value depends on the manufacturer.

There are different types of variables. The types of all variables are defined in the declaration section of a program organisation unit. Variable types are identified with keywords. Variables of the same type are stored in the declaration block. A declaration block starts with a keyword that depends on the variable type and ends with a keyword that depends on the variable type.

VAR … END_VAR
Local variables are declared in this declaration block. They are valid only within the POU where they were declared.

VAR_GLOBAL … END_VAR
Global variables are declared in this declaration block. A variable is declared as a global variable if it is to be used in a program and in POUs which can be called by this program. A global variable called up in a program is known within this program and within the POUs that are called up by this program. The variable must be declared with the same identifier (name) as external variable in all invoked POUs in which it is used.

VAR_EXTERNAL … END_VAR

External variables are declared in this declaration block. If a global variable is used within a function block, it must be declared there as external variable with the same identifier (name).

VAR_INPUT … END_VAR

Input variables are declared in this declaration block. A variable is declared as an input variable if it is to be read only within a function or within a function block, or if it is to be used for transferring parameters in a function or a function block. This variable value cannot then be changed in this POU.

VAR_OUTPUT … END_VAR

Output variables are declared in this declaration block. Output variables are used as outputs in function blocks.

VAR_IN_OUT … END_VAR

Input-output variables are declared in this declaration block. An input-output variable is read, processed and output under the same name by the function block in which it is used. Since operations on an input-output variable have direct effect on its value, this variable cannot be of a type that does not permit write operations – for example: variables with the attribute CONSTANT.

TYPE … END_TYPE

Derived data types are declared in this declaration block.

Declaration of various variable types and the range of their usage:

Keywords – beginning of the declaration block		
VAR	⇒	P , F , FB
TYPE	⇒	P , F , FB (locally within the POU)
VAR_GLOBAL	⇒	P
VAR_EXTERNAL	⇒	FB
VAR_INPUT	⇒	F , FB
VAR_OUTPUT	⇒	FB
VAR_IN_OUT	⇒	FB
Keyword – end of the declaration block		
END_VAR or END_TYPE		

where: P – means program; F – means function; FB – means function block.

Derived data types with examples

Declaration of a data type as enumeration

The declaration of an enumerated data type defines a list of identifiers whose values may contain a data element. In order to declare the enumerated data type the variable list is entered in parenthesis, with individual elements separated with a comma. An initialisation value can be assigned to a variable of this data type. If an initialization value has not been assigned during declaration, the variable will be assigned the value of the first element in the enumeration list at the controller start-up.

Declaration of a variable “SYGNALIZACJA” of a data type as enumeration with the initial value “Zolte” is shown in figure 4.2 as an example.

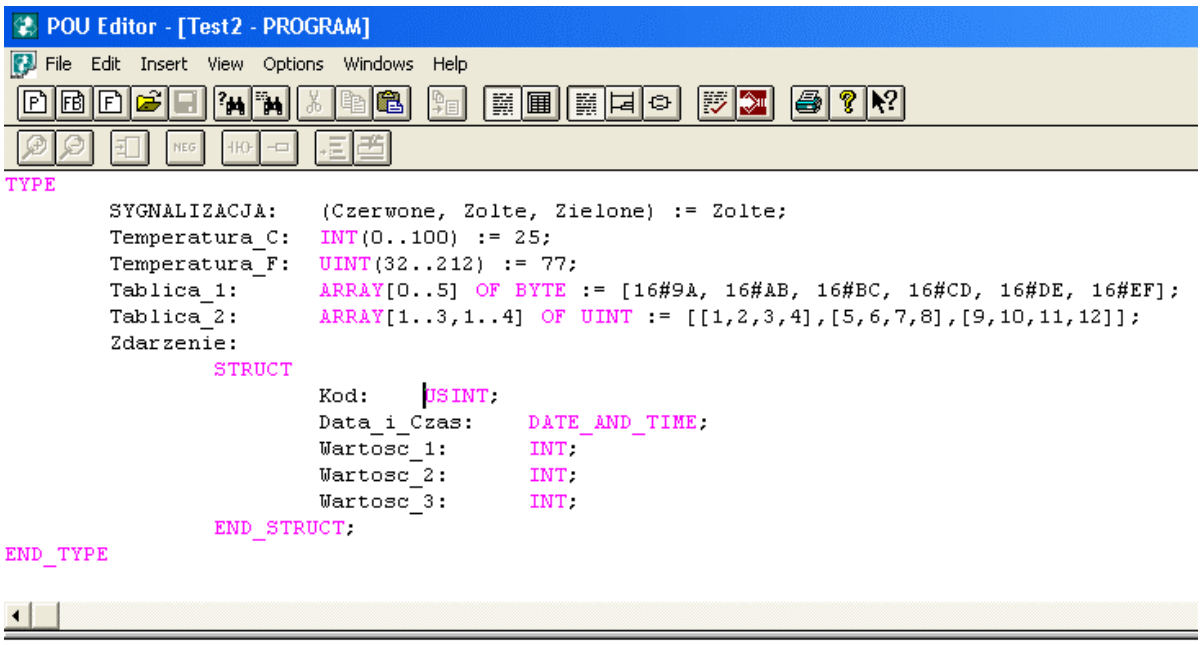

Fig. 4.2. Declaration of derived data types – examples

Declaration of a sub-range data type (restricted value range)

The use of derived data types enables the user to restrict the value range for the used data type. If values beyond the value range are assigned, the limit value next to it will be entered automatically. This rule applies also to initial values.

Declaration of a variable “Temperatura_C” of integer data type with restricted value range from 0 to 100 with the initial value 25, and declaration of a variable “Temperatura_F” of unsigned integer data type with restricted value range from 32 to 212 and the initial value 77 are shown in figure 4.2 as examples.

Declaration of a data type as array

Various data elements of the same type can be combined into an array which can consist of elementary or derived data types. This data type is declared with the keyword ARRAY and the definition of the number of array elements and their data type. Array variables may be assigned with a direct address which is an address of the first element of the array.
Declaration of the array "Tablica_1" as a vector of 6 variables of byte data type and initial values written in hex format (data delimited on the left by 16#), and declaration of the two-dimensional array "Tablica_2" of 12 variables (three rows and four columns) of unsigned integer data type with initial values are shown in figure 4.2 as examples.

Declaration of a data type as structure

Several data elements of different data types can be grouped together into a single structure. A structure may consist of elementary or derived data types. A structure is declared with the keywords STRUCT … END_STRUCT and a list of the structure elements specifying their data types.

A structured data type is an aggregate data type which has been declared using a STRUCT or FUNCTION_BLOCK declaration.
An aggregate is a structured collection of data objects forming a data type.

Declaration of a structure is shown in figure 4.2 as an example. "Zdarzenie" is a structure comprising 5 variables of different data types; "Kod" is of unsigned short integer data type, "Data_i_czas" is of DATE_AND_TIME type, whereas "Wartosc_1", "Wartosc_2" and "Wartosc_3" are of integer data type.

The declaration of local and global variables can be supplemented with the following attributes (qualifiers): RETAIN and CONSTANT.

RETAIN

The attribute RETAIN is used to declare a retentive local or global variable. Retentive means that with a warm start the variable declared as retentive keeps the last valid value it had before the stop. The attribute RETAIN is written behind the keyword VAR or VAR_GLOBAL after a space.
Remark: In some programmable controllers non retentive variables are called volatile, and retentive variables are called non volatile.

CONSTANT

The attribute CONSTANT is used to declare a local or global variable if its value cannot be

changed. The attribute CONSTANT is written behind the keyword VAR or VAR_GLOBAL after a space.

Additionally there are two other attributes that refer to BOOL data type and can only be used as input variables (VAR_INPUT) for user function blocks. These are: R_EDGE (rising edge) and F_EDGE (falling edge). The user function block input declared as R_TRIG or F_TRIG is high for one program cycle if the variable associated to its input changes its value from low to high (for R_TRIG declaration) or from high to low (for F_TRIG declaration). If the user function block is not executed every cycle, the user function block input will remain high until the next execution of this user function block.

4.3. Program organization units

There are three Program Organisation Units that have been distinguished in the programming model:

- F – *Functions* ;
- FB – *Function Blocks* ;
- P – *Programs* .

A function (procedure) is a program organization unit which, when executed, yields exactly one data element and possibly additional output variables (which may be multi-valued, for example, an array or structure), and whose invocation can be used in textual languages as an operand in an expression.

A function block type is a programmable controller programming language element consisting of: 1) the definition of a data structure partitioned into input, output, and internal variables; and 2) a set of operations to be performed upon the elements of the data structure when an instance of the function block type is invoked.

An input variable (input) is a variable which is used to supply an argument to a program organisation unit.
An output variable (output) is a variable which is used to return result(s) of the evaluation of a POU.
An argument is synonymous with input variable, output variable or in-out variable.

An instance is an individual, named copy of the data structure associated with a function block type or a program type, which persist from one invocation of the associated invocation to the next.

An instance name is an identifier associated with a specific instance.
An instantiation is the creation of an instance.

Before the standard IEC 61131 had been created such program organisation units as functions and function blocks were usually called sub-programs. Sub-programs could be called from programs or other sub-programs (nested calls). According to the standard there are some clear distinctions between functions and function blocks. Differences between functions and function blocks are shown comparatively in table 4.1.

Declaration of a function block is called its instantiation.
Each instantiation means that a copy of the function block with a new name has been created. The name of a function block assigned to it during creation is called the instance name.

During function block declaration a part of the programmable controller's memory is reserved to store all internal variables of this function block. These internal variables can be either allocated dynamically in the controller's memory or it is necessary to know how many registers are needed to store all internal variables and point out the address of the first register (n consecutive registers are used to store the function block internal data).

Table 4.1. Differences between Functions and Function blocks

Functions	Function blocks
Functions are not declared.	Function blocks need to be declared.
Each function can be used many times.	Each function block can be used many times, but each time it need to be declared with a different name.
Function can have many inputs, but only one output.	Function block can have many inputs and many outputs.
Function has not internal variables.	Function block has internal variables.
Function is a static language element; that means, that each time for the same combination of values of input signals the value of the output signal is always the same.	Function block is a dynamic language element; that means, that for the same combination of values of input signals, the values of the output signals can be different.

Although a function can have only one output, the output variable can consist of many elements – as, for example, an array type variable.

4.4. Programming languages

There are two groups of programming languages defined in the standard: textual and graphical.

Textual languages:

- IL – Instruction List ;
- ST – Structured Text .

Graphical languages:

- LD – Ladder Diagram ;
- FBD – Function Block Diagram .

In the IEC 61131-3 standard a method of creation of a program structure as a Sequential Function Chart (SFC) is also presented. The SFC enables description of sequential control tasks by means of graphs consisting of steps and transitions between these steps. Transitions, and actions that are associated with the particular steps, are programmed in one of the programming languages. Sequential control is presented in Part 2 of this materials.

An additional programming language (available in some programming packages) is Continuous Function Chart (CFC). This is also a graphical language.

Instruction List (IL)

The Instruction List language looks like and is similar to an assembler language. The IL language consists of a series of instructions. Each instruction begins in a new line and contains an operator and, depending on the type of operation, one or more operands separated by commas.

In front of the first instruction in a program sequence there can be a label followed by a colon.

A POU written in the IL language can have also comments. Comments are placed between parenthesis with asterisks; for example: (* This is a comment *).

Operators and modifiers that can be used in IL language are shown in table 4.2 [X,X].

In the table CR means “current result” and this is the value which is stored in the accumulator; N means Boolean negation of the operand, modifier (means that the result of consecutive operators until the operator) is met, is an operand; C means condition, whereas CN means negation of the condition; for example: JMPC means jump conditional, JMPCN means jump conditional not; ES1 is the label name; NFB1 is the function block name.

Table 4.2. Operators and modifiers in the IL language

Operator	Modifiers	Operand	Description
LD	N	*	Operand is loaded into the accumulator (CR)
ST	N	*	CR is stored as the operand value
S		BOOL	(Set) if CR = 1 , then the operand is TRUE
R		BOOL	(Reset) if CR = 1 , then the operand is FALSE
AND	N, (, N(	BOOL	Boolean AND (CR and operand)
OR	N, (, N(	BOOL	Boolean OR (CR and operand)
XOR	N, (, N(	BOOL	Boolean XOR (CR and operand)
ADD	(	*	Addition (CR and operand)
SUB	(	*	Subtraction (CR and operand)
MUL	(	*	Multiplication (CR and operand)
DIV	(	*	Division (CR and operand)
GT	(	*	(Greater Than) comparison: CR > operand
GE	(	*	(Greater than or Equal) comparison: CR > = operand
LT	(	*	(Less Than) comparison: CR < operand
LE	(	*	(Less than or Equal) comparison: CR < = operand
EQ	(	*	(Equal) comparison: CR = operand
NE	(	*	(Not Equal) comparison: CR < > operand
JMP	C, CN	ES1	Jump to the label ES1
CAL	C, CN	NFB1	Call FB with the name NFB1
RET	C, CN		Return (from F or FB)
)			Delimiter of the modifier (

An example of a program sequence written in the IL language (with the use of Sucosoft S40 programming package) is shown in figure 4.3.

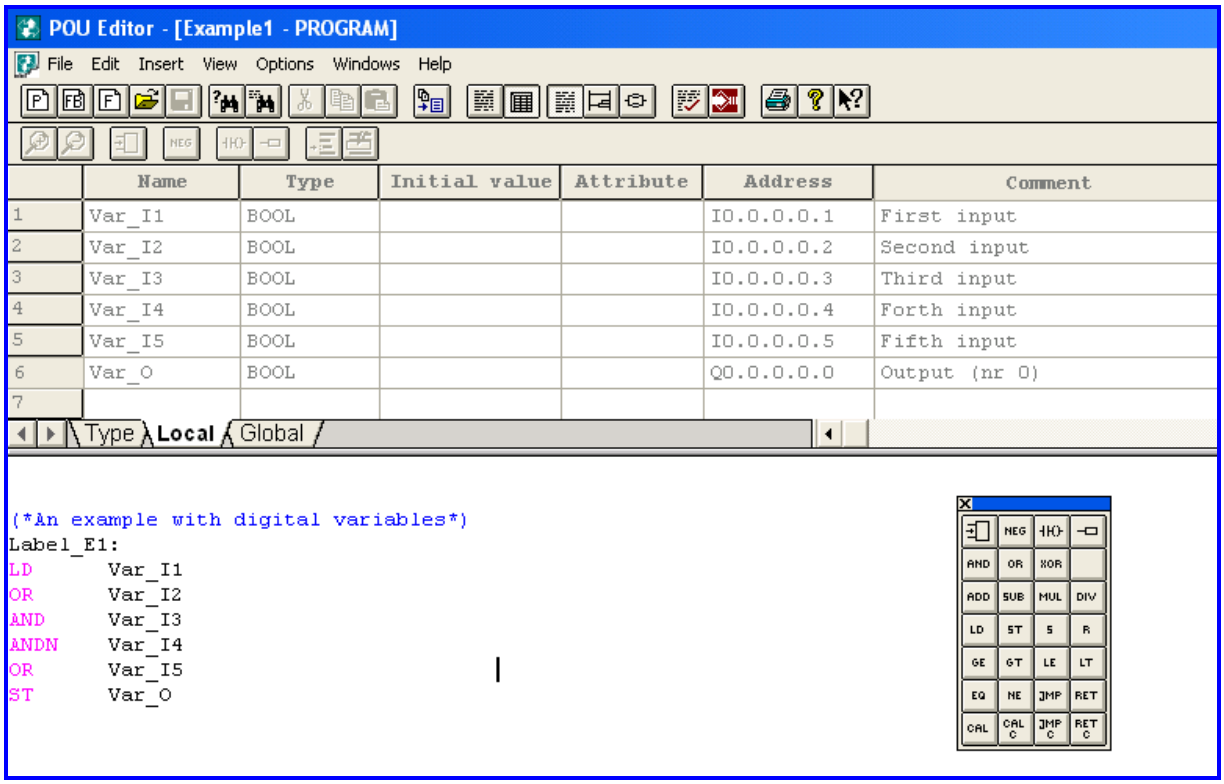

Fig. 4.3. An example of a program sequence written in IL language

Structured Text (ST)

The Structured Text language is similar to high level languages. The instruction section of a POU written in the ST language consists of at least one ST statement or a sequence of statements. Each statement must be terminated with a semicolon. This separator allows a statement to cover several lines, and several statements can be written on the same line.

Comments can be positioned at any place as required. They are placed exactly as in the IL language – between parenthesis with asterisks.

An expression is not an independent statement, but a construction which, after its evaluation, returns a result (a value) for further processing in the statement. An expression consists of one or several operands that are linked together by means of operators. An operand can be a constant, a variable, a function call, or another expression.

The evaluation of an expression takes place by means of processing the operators according to certain binding rules. The operator with the strongest binding is processed first, then the operator with the next strongest binding, etc., until all operators have been processed. Operators with equal binding strength are processed from the left hand side to the right hand side.

Operators that can be used in ST language are shown in table 4.3 [41, 16].

Table 4.2. Operators in the ST language

Operation	Symbol	Binding strength
Parenthesization	( )	1 (high)
Function call	Name ( )	2
Two's complement	-	3
Two's complement used twice	+	3
One's complement	NOT	3
Exponentiation	**	4
Multiplication	*	5
Division	/	5
Modulo	MOD	5
Addition	+	6
Subtraction	-	6
Comparison	>, > =, <, < =	7
Equality	=	8
Inequality	< >	8
Logic AND	AND, &	9
Logic EXCLUSIVE-OR	XOR	10
Logic OR	OR	11 (low)

Instructions of the ST language are listed below [41, 16].

Elementary statements

Value assignment

Syntax: Data element := Expression ;

Empty statement

Example: ; (a single semicolon can be used as an empty statement)

Branching within a POU

Conditional statement

Syntax:
```
IF Expression THEN StatementSequence END_IF ;
```

Single alternative statement

Syntax:
```
IF Expression THAN StatementSequence1
ELSE StatementSequence2 END_IF ;
```

Multiple alternative statement

Syntax:
```
IF Expression1 THEN StatementSequence1
ELSIF Expression2 THEN StatementSequence2
ELSIF Expression3 THEN StatementSequence3
…
ELSE StatementSequenceN END_IF ;
```

Multiple selection (CASE statement)

Syntax:
```
CASE Expression OF
ValueList1 : StatementSequence1
ValueList2 : StatementSequence2
…
ELSE StatementSequenceN
END_CASE ;
```

WHILE loop

Syntax:
```
WHILE Expression DO StatementSequence END_WHILE ;
```

REPEAT loop

Syntax:
```
REPEAT StatementSequence UNTIL Expression END_REPEAT ;
```

FOR loop

Syntax:
```
FOR LoopVariable := ExpressionInitialValue TO ExpressionEndValue BY
StepWidth DO StatementSequence END_FOR ;
```

Loop exit

Syntax:
```
EXIT ;
```

Branching within the application

POU exit

Syntax:
```
RETURN ;
```

Call of a function block (with assignment of inputs and outputs)

Remark: Function calls are not separate statements but expressions; therefore they can only be used within expressions.

An example of a POU written in the ST language (with the use of Easy Soft CoDeSys programming package) is shown in figure 4.4.

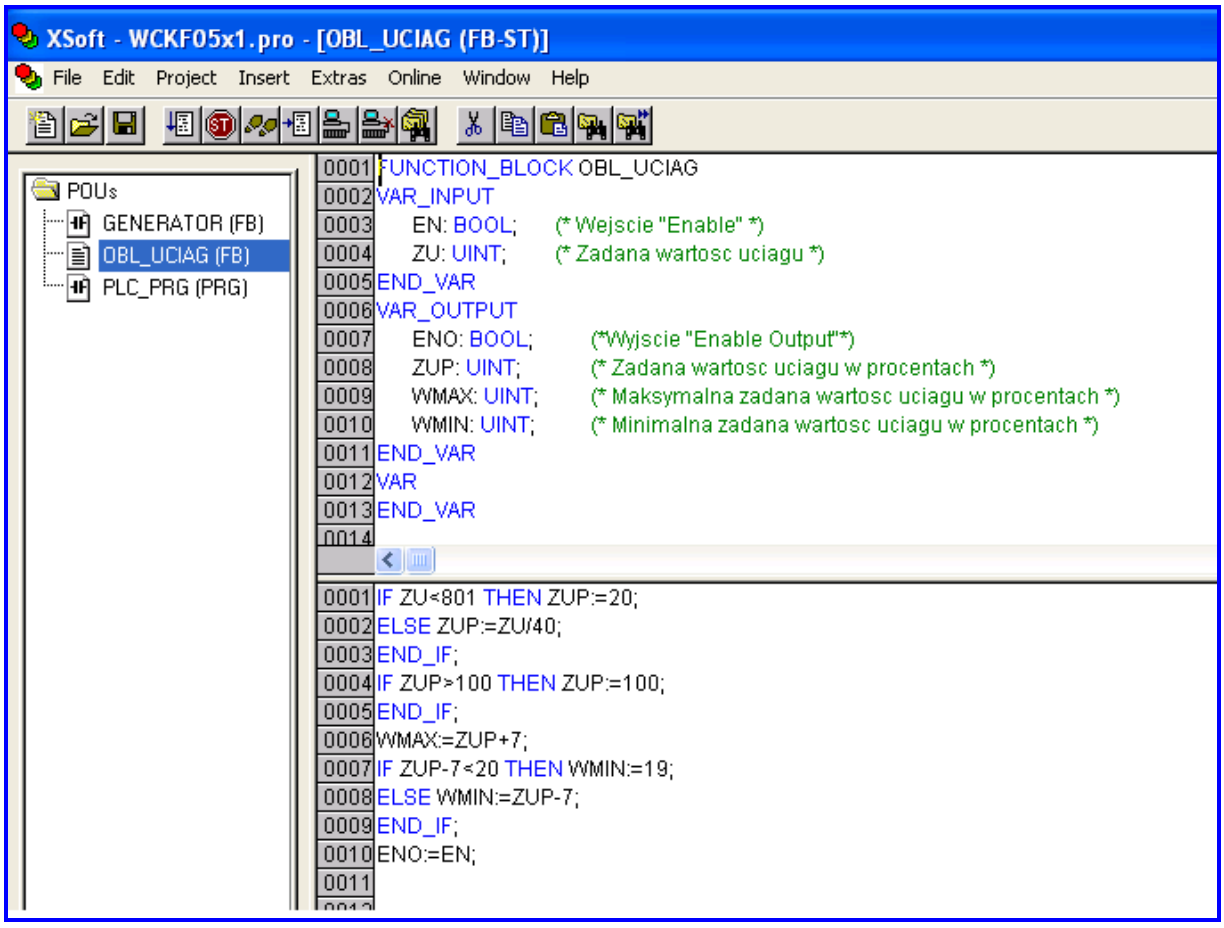

Fig. 4.4. An example of a POU written in the ST language

Function Block Diagram (FBD)

The Function Block Diagram is a graphically oriented programming language. It works with a list of networks. Each network contains a structure which represents either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

An example of a network written in the FBD language (with the use of Sucosoft S40 programming package) is shown in figure 4.5.

Ladder Diagram (LD)

The Ladder Diagram is a graphically oriented programming language which resembles the structure of an electric circuit. A program written in the LD language consists of a series of networks. A network is limited on the left hand side and on the right hand side by vertical lines. In the middle there is a circuit diagram made up of contacts, coils and connecting lines.

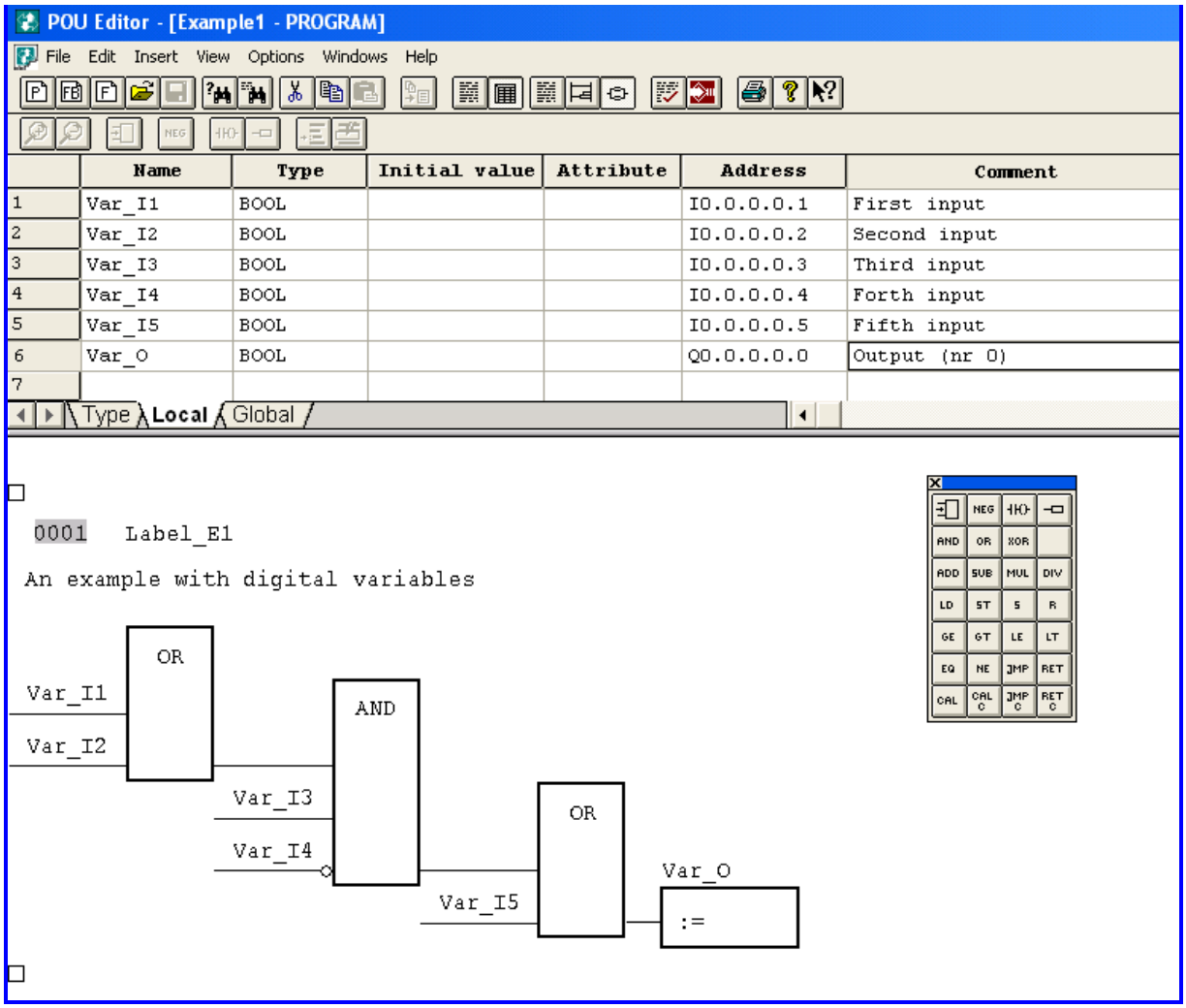

Fig. 4.5. An example of a network written in the FBD language

The left vertical line can be treated as a power supply line, usually 24 V DC in a control electric circuit, whereas the right vertical line can be treated as a 0 V line. Networks are analysed in a similar way as electric circuits – whether they pass signal (current) from the left hand side to the right hand side or not.

An example of a network written in the LD language (with the use of Sucosoft S40 programming package) is shown in figure 4.6.

Contacts

Each network on the left hand side consists of a number of contacts (contacts are represented by two parallel lines). A Boolean variable is associated with each contact. The value of a Boolean variable can be TRUE or FALSE. If the value of the variable associated with the contact is TRUE, then the signal is passed on through this contact. If opposite – i.e. the value of the variable associated with the contact is FALSE, then the signal is not passed on through this contact.

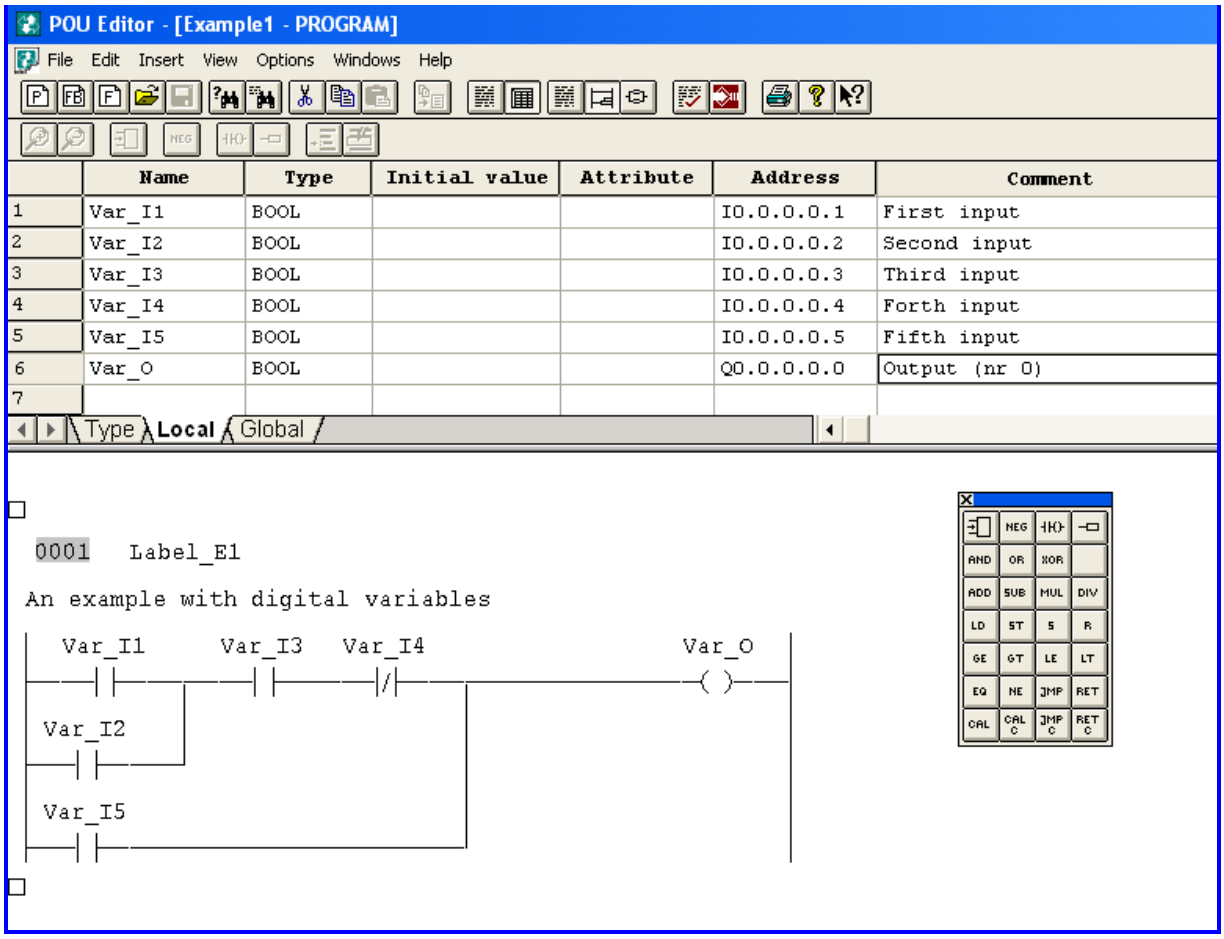

Fig. 4.6. An example of a network written in the LD language

A contact can be negated (negated contact is represented by the slash in the contact symbol). If the value of the variable associated with the negated contact is FALSE, then the signal is passed on through this contact. If opposite – i.e. the value of the variable associated with the negated contact is TRUE, then the signal is not passed on through this contact.

Contacts can be connected in parallel, then one of the parallel branches must transmit the signal so that the parallel branch transmits the signal; or contacts can be connected in series, then all contacts must transmit the signal. This therefore corresponds to electric circuits: parallel and series circuit.

Coils

On the right hand side of a network in LD there can be any number of so-called coils (coils are represented by parenthesis). They can only be in parallel.

A coil transmits the signal (is energized) if the signal in the network has been passed on to the entry line (on the left) of the coil. Then the Boolean value of the variable associated with this coil takes the value TRUE. Otherwise the Boolean value of the variable associated with this coil is FALSE.

Coils can also be negated (negated coil is represented by the slash in the coil symbol). A negated coil is energized if the signal in the network has not been passed on to the entry line of the coil; the variable associated with this coil is then TRUE. Otherwise the negated coil is not energised and the variable associated with this coil is FALSE.

Set / Reset coils

Coils can also be defined as set or reset coils. A set coil can be recognized by the “S” in the coil symbol. A reset coil can be recognized by “R” in the coil symbol.
Boolean variables are associated with set and reset coils. In a POU the same Boolean variable is associated with at least one set coil and with at least one reset coil.
A set coil never writes over the value TRUE of the associated Boolean variable. That is, if the variable was once set at TRUE, then it remains so. A reset coil never writes over the value FALSE of the associated Boolean variable. If the variable has been once set on FALSE, then it remains so. Therefore it is rather pointless to have a Boolean variable associated with either only a set coil (set coils) or only a reset coil (reset coils).

Function blocks in the LD network

Apart from contacts and coils function blocks can also be entered in an LD network. In the network they must have an input and an output with Boolean variables and can be used at the same places as contacts, that is on the left hand side of the LD network.
In some LD editors however, function blocks can be entered in a POU only as separate rungs in the ladder. Only single variables can be associated with their inputs and outputs. If a value of an input variable depends on a complex condition, this should be evaluated in a separate rung. Such an example is shown and discussed in chapter 4.6.

An example of a network in the LD language (with the use of Easy Soft CoDeSys programming package) with two function blocks connected in parallel (a signal generator and an up-counter) is shown in figure 4.7.

Continuous Function Chart (CFC)

The Continuous Function Chart editor does not operate like Function Block Diagram with networks, but rather with elements that can be placed, quite freely, in different positions. This allows, for example, feedback connections.

An example of a POU written in the CFC language (with the use of Easy Soft CoDeSys programming package) is shown in figure 4.8.

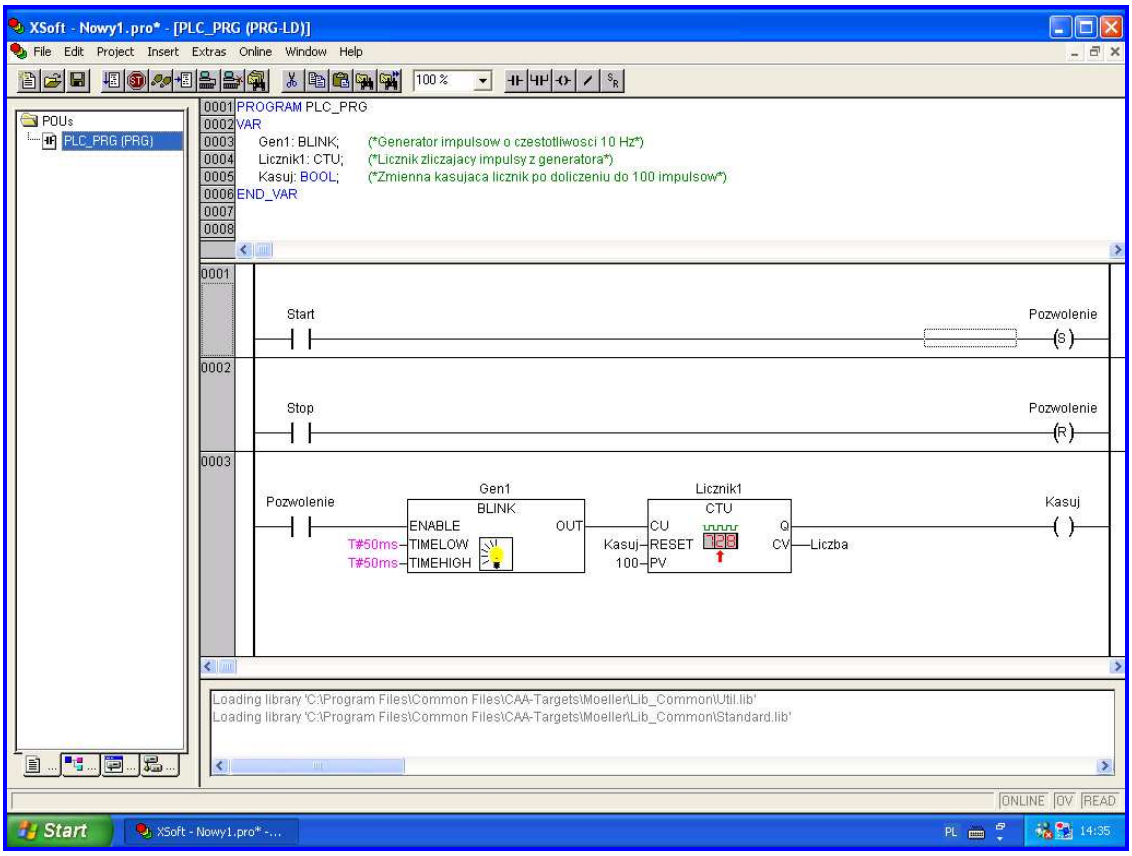

Fig. 4.7. An example of use of function blocks in the LD language

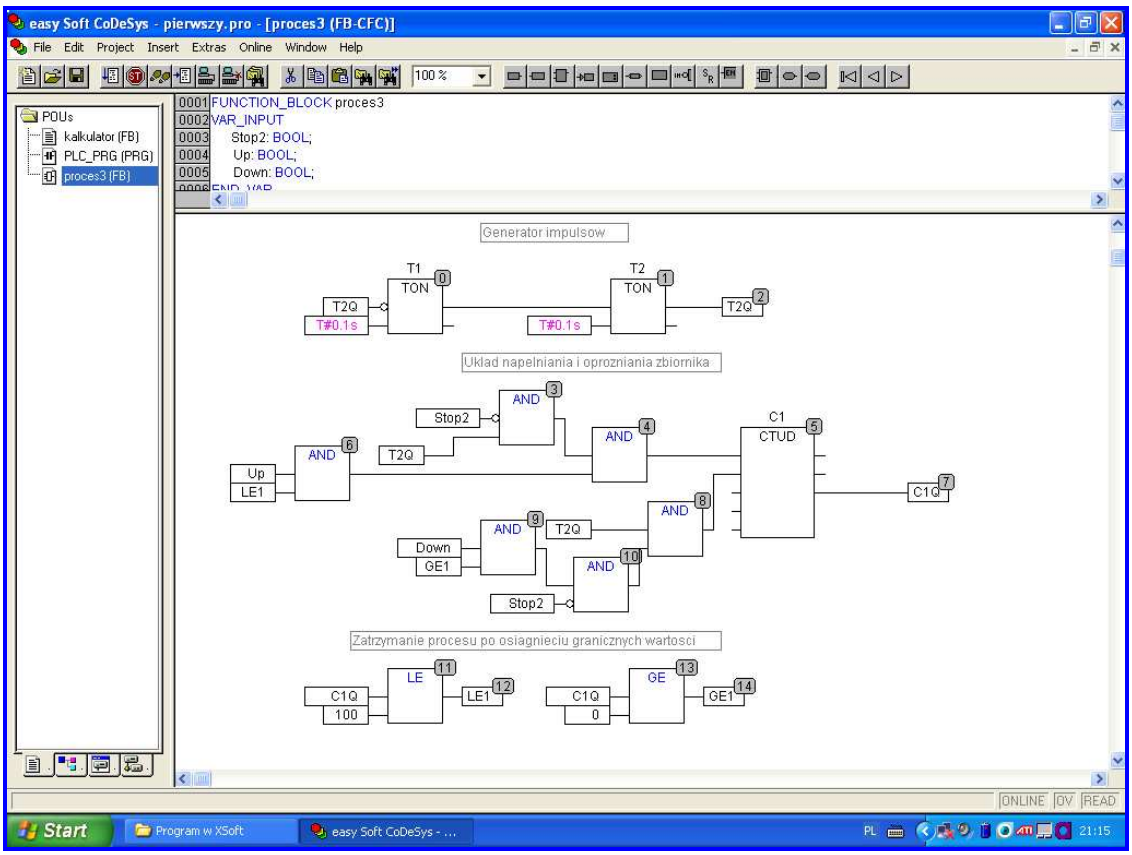

Fig. 4.8. An example of a POU written in the CFC language

4.5. Standard functions and function blocks

In the International Standard IEC 61131-3 there is a number of standard functions declared. All these functions are divided into seven groups [41, 16]:

- Type conversion functions;
- Numerical functions;
- Bit string functions;
- Selection and comparison functions;
- Character string functions;
- Functions of time and data types;
- Functions of enumerated data types.

All function blocks that are mentioned in the International Standard IEC 61131-3 are divided into four groups [41, 16]:

- Bistable elements;
- Edge detection elements;
- Counters;
- Timers.

Functions and function blocks belonging to the listed above groups are not discussed in these materials because there is a detailed description of all these functions and function blocks in the standard IEC 61131-3. Description of these functions and function blocks can be also found in some references, for example in [16]. In these materials only some important remarks have been made.

Manufacturers of programmable controllers usually provide quite a lot of diverse functions and function blocks, more than those described in the standard IEC 61131-3. This is so because their programming packages have been evolving during past years and contain functions and function blocks which have been developed before the International Standard IEC 61131 have been established. The use of such functions and function blocks should not be recommended if analogous IEC functions and function blocks exist, but is fully acceptable if these functions and function blocks are somehow unique, for example, because they can implement specific programmable controllers' hardware features.

The prototypes of IEC and manufacturer functions and function blocks often have generic data types (ANY) associated with their inputs and outputs. Generic data types cannot be used for declaration of user function blocks. Overloaded data types of this kind are only

possible with input and output parameters of IEC or manufacturer functions and function blocks.

In graphical languages POUs (functions and function blocks) can have an additional input EN and an additional output ENO. The EN input and the ENO output are always of the BOOL type. The EN meaning is: the POU with EN input is evaluated if the EN input has the value TRUE. The ENO meaning is: after correct evaluation of the POU with EN input the ENO output has the value TRUE. Otherwise the output has the value FALSE. These special inputs and outputs are used to control the application program execution.

An example how the EN input of the function can be used (or not) in a program to control the program execution is shown in figures 4.9 and 4.10. The intention was to add the value 526 to the variable "Wartosc" only once – in the first program cycle after the programmable controller has been switched on.

In the first program (part of which is shown in figure 4.9) this was realized in the following way. There is a Boolean variable "Pom" declared with an initial value TRUE. In the first program cycle the condition for the conditional jump to the label "skok" is therefore not fulfilled. The next rung with the ADD function is executed (any time the function ADD is executed the value on the EN input is TRUE) and 526 is added to the variable "Wartosc". In the next rung the Boolean variable "Pom" is reset by itself. In the following program cycles, until the variable "Pom" will remain FALSE the jump condition will be fulfilled and the rung with the ADD function will be skipped.

In the second program (part of which is shown in figure 4.10) this was realized with the use of EN input. There is also a Boolean variable "Pom" declared with an initial value TRUE. In the first program cycle when the value of the Boolean variable "Pom" is TRUE the ADD function is executed (because the value on the EN input is TRUE) and 526 is added to the variable "Wartosc". In the next rung the Boolean variable "Pom" is reset by itself. In the following program cycles, until the variable "Pom" will remain FALSE the ADD function will not be executed.

4.6. User functions and function blocks

Apart from IEC and manufacturer functions and function blocks also user functions and user function blocks can be used in application programs. User POUs (functions and function blocks) are created using appropriate to particular programmable controllers programming packages.

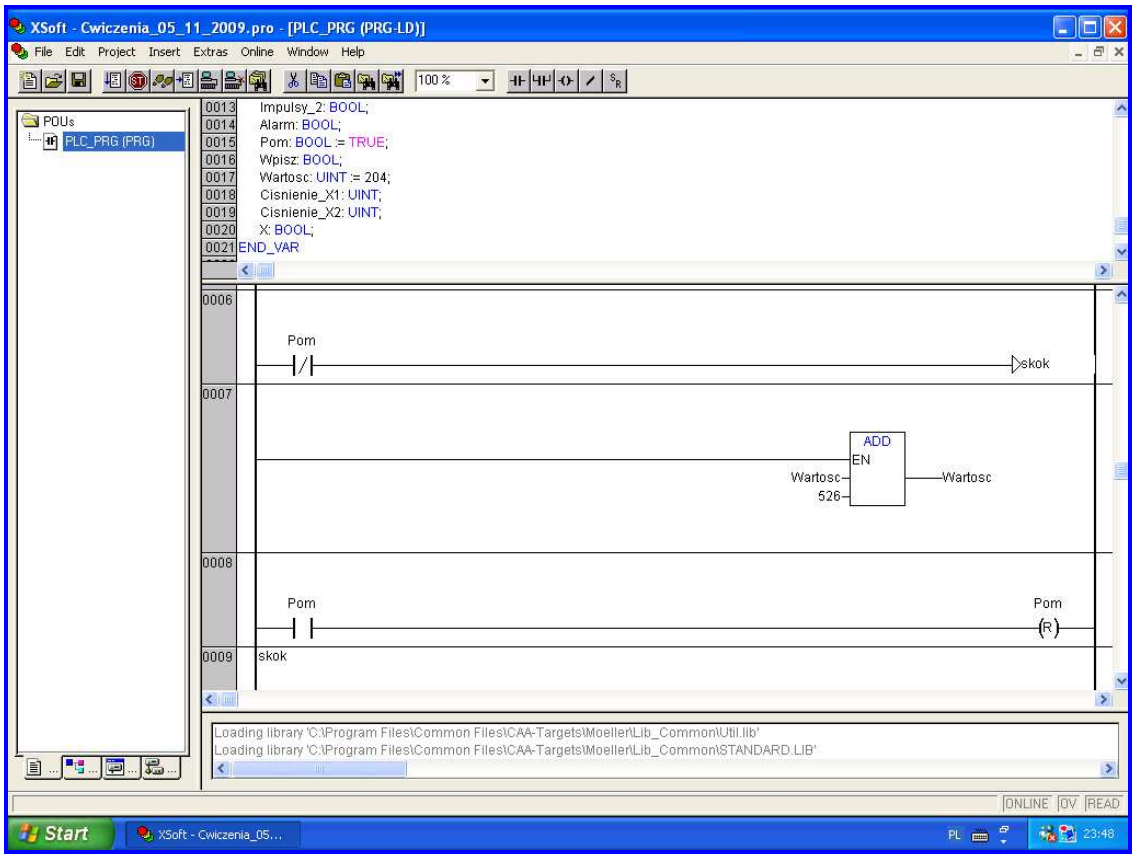

Fig. 4.9. Unconditional execution of the function ADD

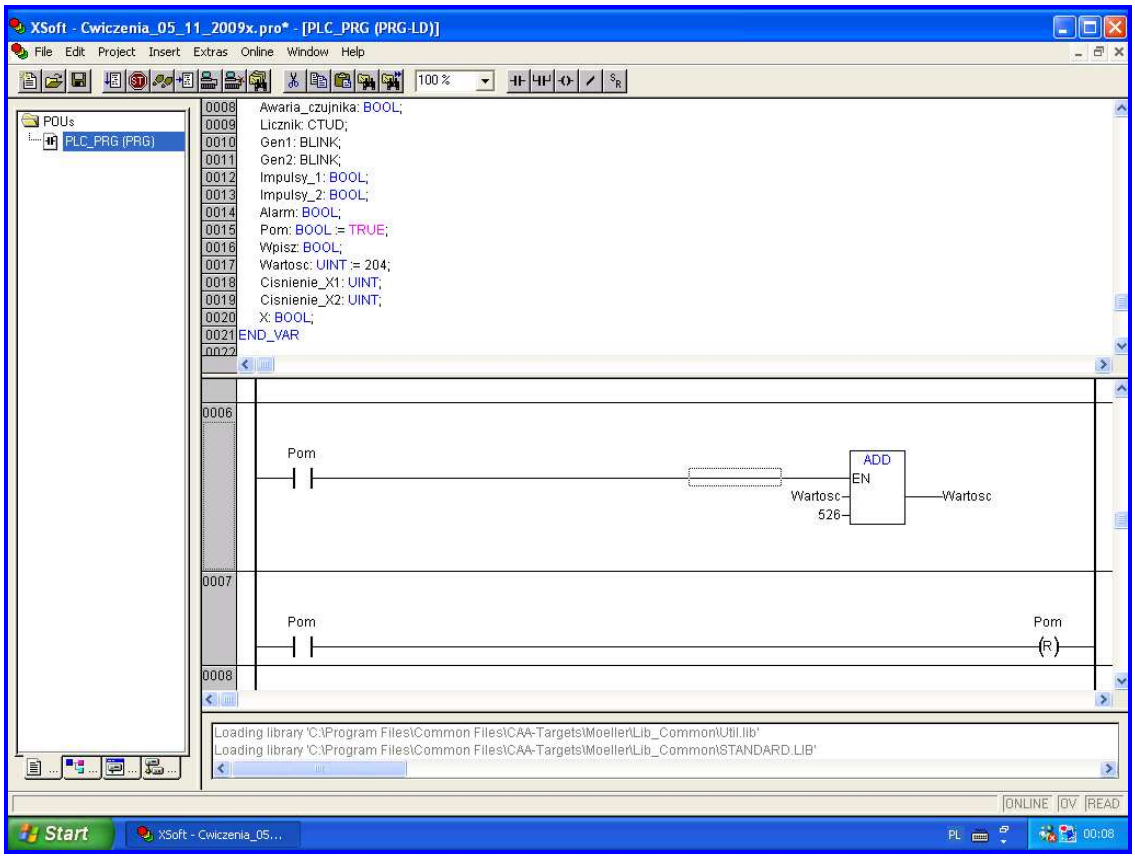

Fig. 4.10. Conditional execution of the function ADD

Creation and usage of user POUs in application programs, especially user function blocks, has many advantages:

- helps in structuring application programs;
- important application program quality features (verifiability, modifiability and reusability) are improved;
- creation of application programs can be less time consuming, as function blocks can be declared and used many times;
- it is possible to create user function blocks that refer to specific hardware features;
- once user function blocks has been thoroughly tested it is then easier to test and start-up application programs.

Usage of user function blocks helps in structuring application programs, because its structure becomes more clear. Part of the application program complexity is covered by user function block bodies. This in turn positively influences many important application program quality features: verifiability, modifiability and reusability. It is easier to analyse such a structured program and make changes if necessary during the control system life time.

A user function block once created can be declared (with different names) and used in an application program many times. Therefore repeated section of a program can be programmed as a user function block and used many times. Moreover, even if there are no repeated sections in a program and a user function block will be created, it can be used in other programs. In such a way a programmer can create its own library of user function blocks, to save time in writing other application programs in future.

The use of tested user function blocks helps in testing an starting-up application programs. If there are errors in a program it is easier to find them debugging the program, because the programmer can be quite sure that they are not inside its user function blocks.

The other important difference between IEC or manufacturer function blocks and user function blocks is the fact, that user function blocks can refer to specific features of the sensors, actuators and other automation equipment that operates in a programmable controller based control system. Unlike standard function blocks, user function blocks can be created as specific hardware functions oriented function blocks.

Such a user function block created (with the use of Sucosoft S40 programming package) for control of a specific hardware element is shown in figure 4.11. This is a user function block named IZM_1T to control a circuit breaker of IZM type. This function block has 4 digital inputs, 2 time inputs and 6 digital outputs. As it can be seen in the declaration

section this function block has been used in the program several times (with the following names: CB_TR1, CB_TR12, CB_TR2, CB_TR3, CB_TR4). Each time different variables have been associated with its inputs and outputs.

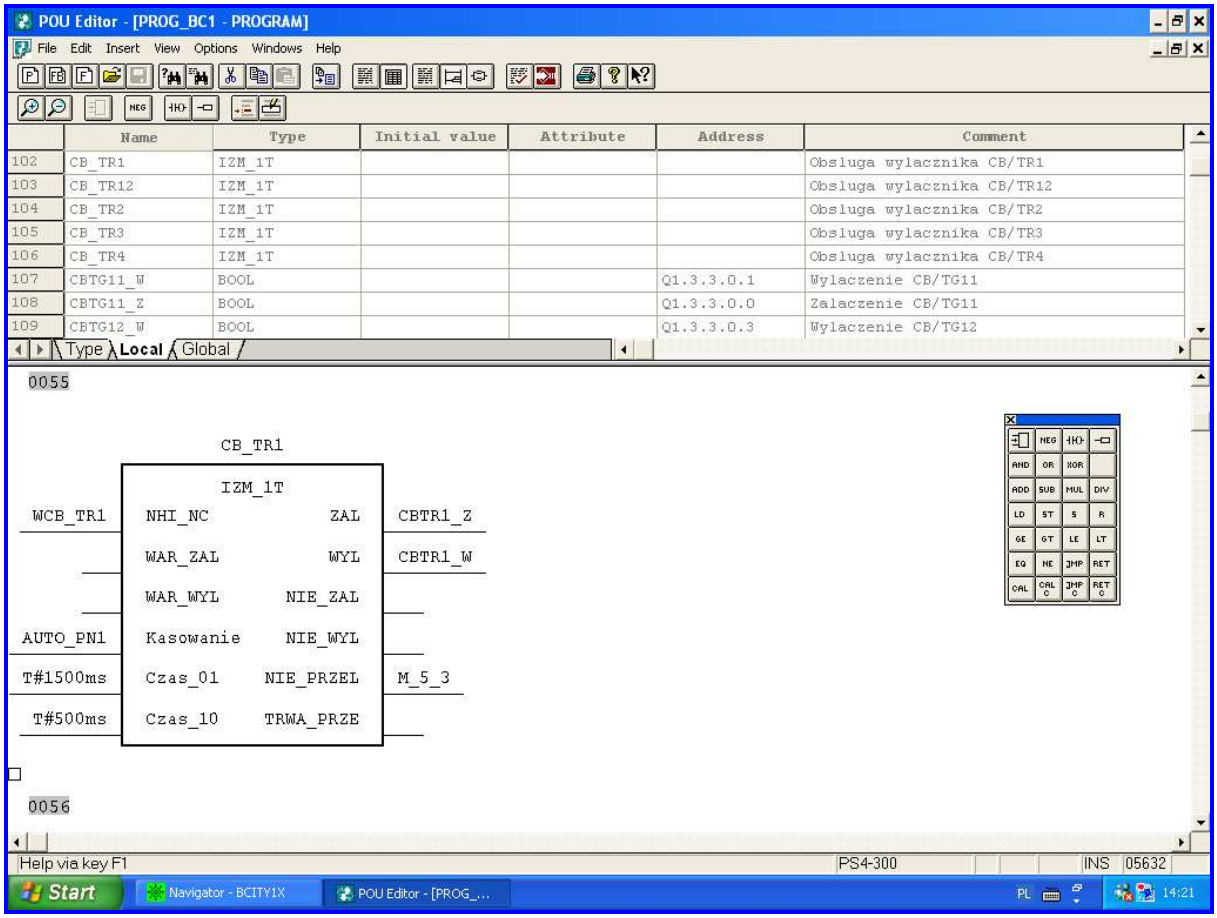

Fig. 4.11. User function block IZM_1T for control of the ICM circuit breaker

An interesting function block declaration feature is, that after declaring a function block its inputs and outputs can be used in the program without any additional declaration. Even if there are no variables associated with some inputs and outputs of the function block, that does not mean that these inputs and outputs are not used in the program. For example, even if there is no variable on the “WAR_ZAL” function block input it is used in the program. This is shown in figure 4.12.

In the lower rung shown in the figure there is an output variable (coil) with the name “CB_TR1.WAR_ZAL”. The first part of this name refers to the function block CB_TR1, whereas the second part of the name means the particular input of this function block, which is WAR_ZAL. Such usage of inputs and outputs of function blocks can limit the number of auxiliary variables that need to be declared in the program (or other POU). This is also a proof

that during declaration of a function block a part of the programmable controller's memory is reserved to store the function block input, output and internal variables.

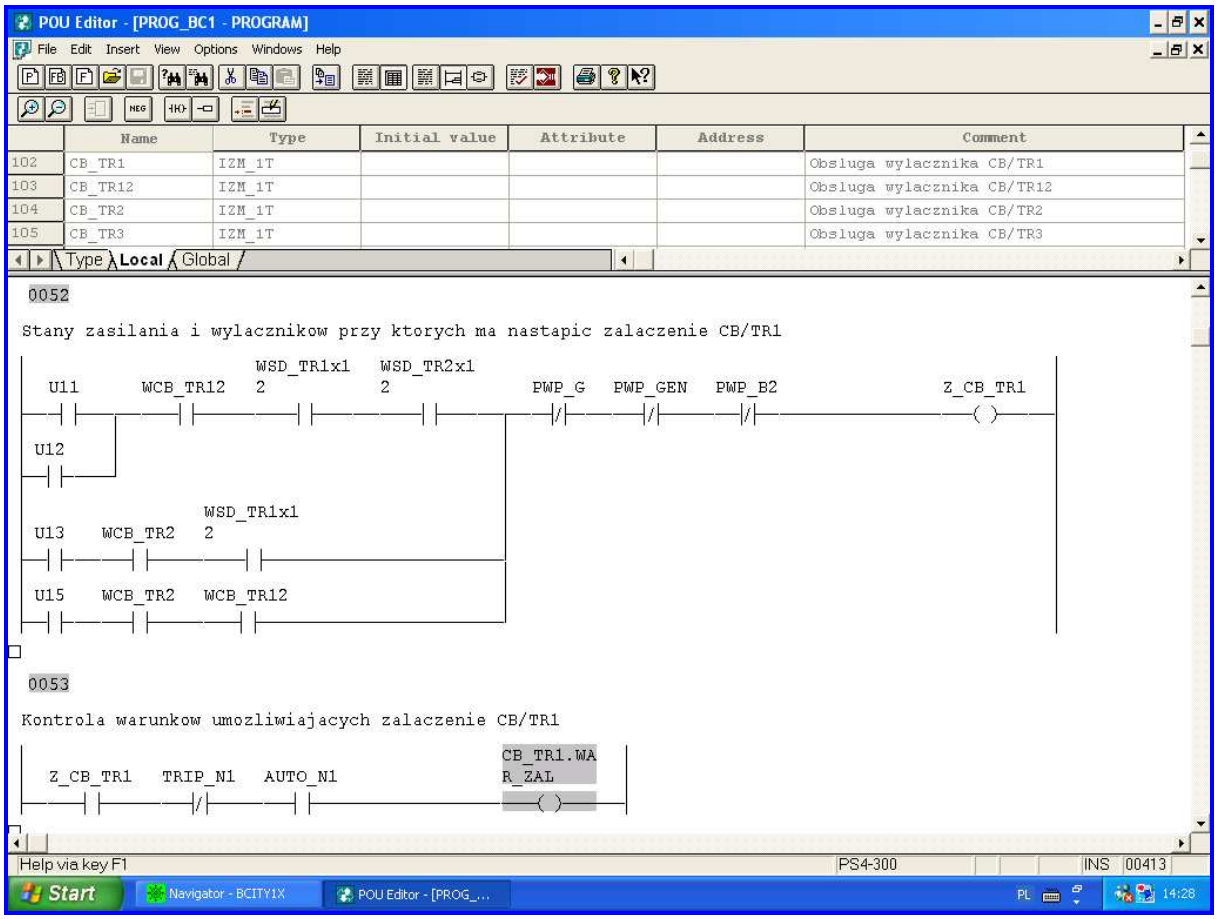

Fig. 4.12. Usage of the function block input variable as a program variable that do not need to be declared

Another example of a user function block (created with the use of Easy Soft CoDeSys programming package) is shown in figure 4.13. This is the user function block "OBL_UCIAG" which has been declared with the name "ZAD_UCIAG". It has one input of UINT data type and three outputs of UINT data type.

Additionally it has an EN input (BOOL type) and an ENO output (BOOL type) but these input and output are not used in the program.

The OBL_UCIAG function block has been programmed in the ST language and the body of this user function block has been shown in figure 4.4.

In the declaration section declaration of some variables can be seen. It is worth to mention that apart from symbolically represented variables FST_SCAN and ALWAYS_ON (which are declared with an initial value TRUE) other variables are directly represented

variables (the keyword with the variables addresses). Comments in (* … *) have been associated to all variables shown in figure.

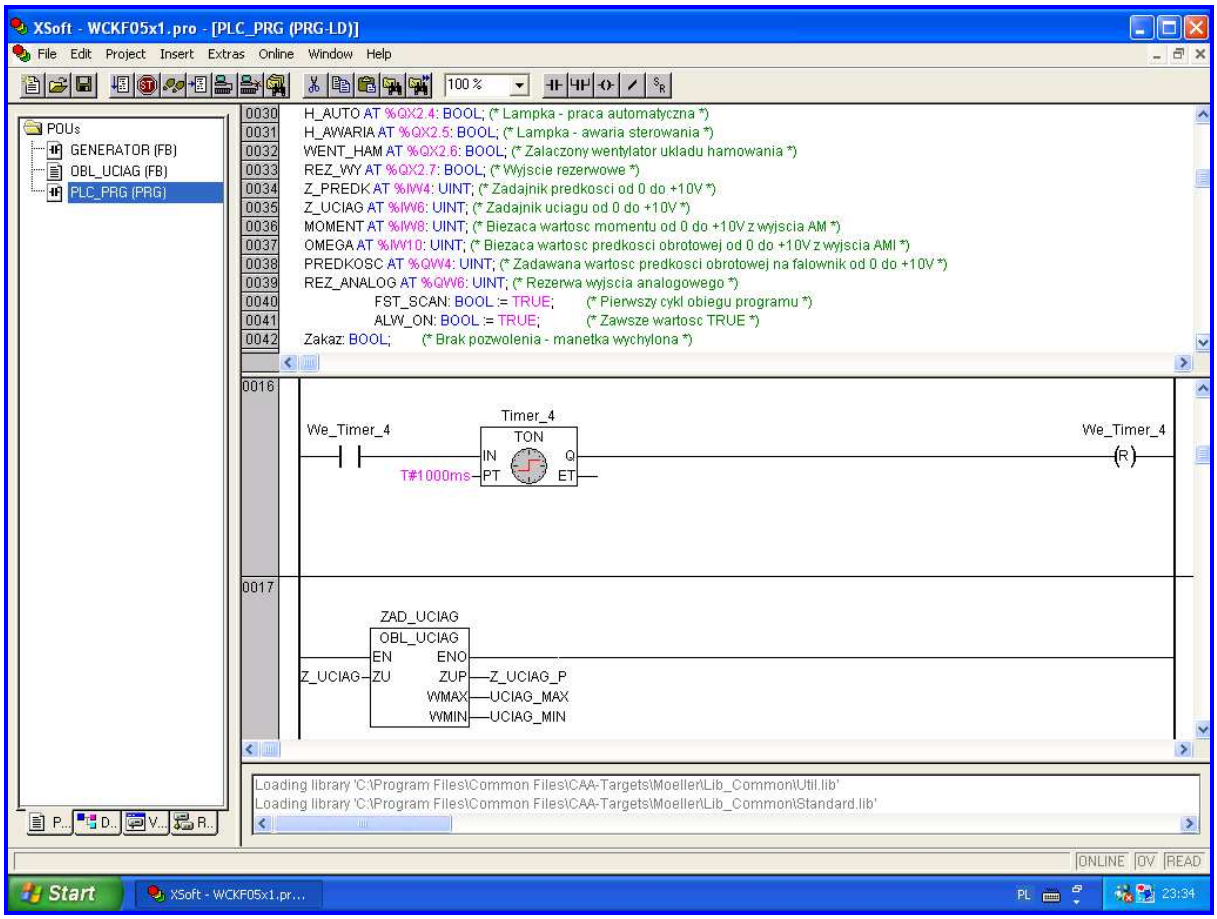

Fig. 4.13. An example of a user function block

Basing on the above shown examples it can be definitely concluded that usage of user function blocks helps to structure and reduces the complexity of application programs.

5. International Standard IEC 61131 – Selected Information

International Standard IEC 61131 consists of the following parts under the general title: Programmable controllers.

Part 1: General information
Part 2: Equipment requirements and tests
Part 3: Programming languages
Part 4: User guidelines
Part 5: Communications
Part 6: Reserved
Part 7: Fuzzy-control programming
Part 8: Guidelines for the application and implementation of programming languages for programmable controllers

The purposes of this standard are:

Part 1 establishes the definitions and identifies the principal characteristics relevant to the selection and application of programmable controllers and their associated peripherals;
Part 2 specifies equipment requirements and related tests for programmable controllers (PLC) and their associated peripherals;
Part 3 defines, for each of the most commonly used programming languages, major fields of application, syntactic and semantic rules, simple but complete basic sets of programming elements, applicable tests and means by which manufacturers may expand or adapt those basic sets to their own programmable controller implementations;
Part 4 gives general overview information and application guidelines of the standard for the PLC end-user;
Part 5 defines the communication between programmable controllers and other electronic systems;
Part 6 is reserved;
Part 7 defines the programming language for fuzzy control;
Part 8 gives guidelines for the application and implementation of the programming languages defined in Part 3.

The first part of the standard (IEC 61131-1) applies to programmable controllers (PLC) and their associated peripherals such as programming and debugging tools (PADTs), humane

machine interfaces (HMIs), etc., which have as their intended use the control and command of machines and industrial processes.

PLCs and their associated peripherals are intended to be used in an industrial environment and may be provided as open or enclosed equipment. If a PLC or its associated peripherals are intended for use in other environments, than the specific requirements, standards and installation practices for those other environments must be additionally applied to the PLC and its associated peripherals.

The functionality of a programmable controller can be performed as well on a specific hardware and software platform as on a general-purpose computer or a personal computer with industrial environment features. This standard applies to any products performing the function of PLCs and/or their associated peripherals. This standard does not deal with the functional safety or other aspects of the overall automated system. PLCs, their application programme and their associated peripherals are considered as components of a control system.

Since PLCs are component devices, safety considerations for the overall automated system including installation and application are beyond the scope of this Part. However, PLC safety as related to electric shock and fire hazards, electrical interference immunity and error detecting of the PLC-system operation (such as the use of parity checking, self-testing diagnostics, etc.), are addressed. Refer to IEC 60364 or applicable national/local regulations for electrical installation and guidelines.

This Part of IEC 61131 gives the definitions of terms used in this standard. It identifies the principal functional characteristics of programmable controller systems.

Two other parts are indispensable for the application of the IEC 61131-1 part of the standard:

- IEC 61131-2, Programmable controllers – Part 2: Equipment requirements and tests;
- IEC 61131-3:2003, Programmable controllers – Part 3: Programming languages.

For the purposes of the first part of the standard the following terms and definitions apply.

Application programme or user programme

Logical assembly of all the programming language elements and constructs necessary for the intended signal processing required for the control of a machine or process by a PLC-system.

Automated system

Control system beyond the scope of IEC 61131, in which PLC-systems are incorporated by or for the user, but which also contains other components including their application programs.

Field device

Catalogued part to provide input and/or output interfaces or to provide data pre-processing/post-processing to the programmable controller system. A remote field device may operate autonomously from the programmable controller system. It can be connected to the programmable controller using a field bus.

Ladder diagram or relay ladder diagram

One or more networks of contacts, coils, graphically represented functions, function blocks, data elements, labels, and connective elements, delimited on the left and (optionally) on the right by power rails.

Programmable (logic) controller (PLC)

Digitally operating electronic system, designed for use in an industrial environment, which uses a programmable memory for the internal storage of user-oriented instructions for implementing specific functions such as logic, sequencing, timing, counting and arithmetic, to control, through digital or analogue inputs and outputs, various types of machines or processes. Both the PLC and its associated peripherals are designed so that they can be easily integrated into an industrial control system and easily used in all their intended functions.

Programmable controller system or PLC-system

User-build configuration, consisting of a programmable controller and associated peripherals, that is necessary for the intended automated system. It consists of units interconnected by cables or plug-in connections for permanent installation and by cables or other means for portable and transportable peripherals.

Programming and debugging tool PADT)

Catalogued peripheral to assist in programming, testing, commissioning and troubleshooting the PLC-system application, programme documentation and storage and possibly to be used as HMIs. PADTs are said to be pluggable when they may be plugged or unplugged at any time into their associated interface, without any risk to the operators and the application. In all other cases, PADT are said to be fixed.

Remote input/output station (RIOS)

Manufacturer’s catalogued part of a PLC-system including input and/or interfaces allowed to operate only under the hierarchy of the main processing unit (CPU) for I/O multiplexing/demultiplexing and data pre-processing/post-processing. The RIOS is the only permitted limited autonomous operation, for example, under emergency conditions such as

breakdown of the communication link to the CPU or of the CPU itself, or when maintenance and trouble shooting operations are to be performed.

FUNCTIONAL CHARACTERISTICS

Basic functional structure of a programmable controller system

The general structure with main functional components in a programmable controller system is presented in figure 5.1. These functions communicate with each other and with the signals of the machine/process to be controlled.

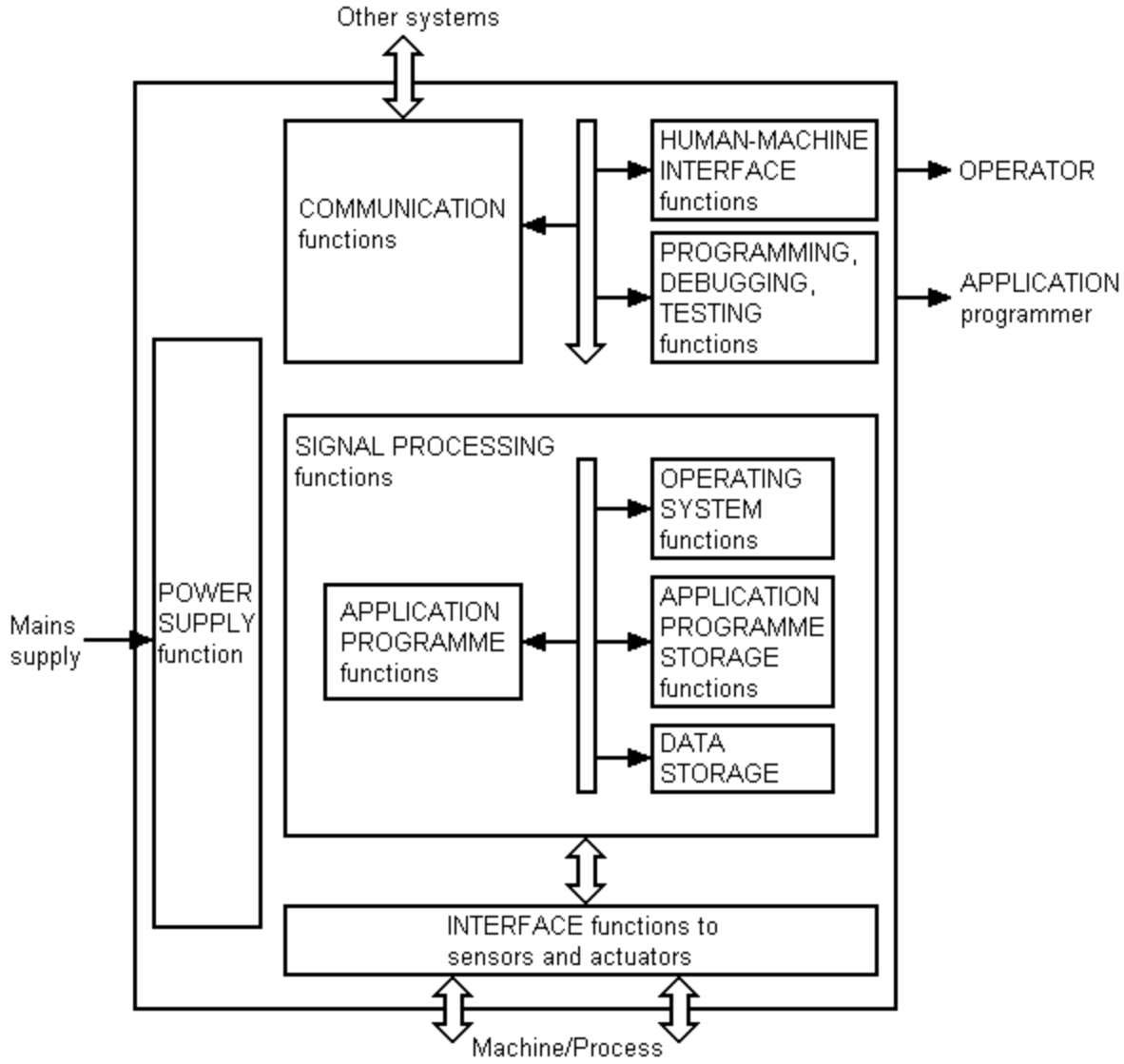

Fig.5.1. Basic functional structure of a PLC-system

Signal processing functions

The CPU function consists of the application programme storage, the operating system, and

the execution of the application programme functions.

The CPU processes signals obtained from sensors as well as internal data storage and generates signals to actuators as well as internal data storage in accordance with the application programme.

Interface function to sensors and actuators

The interface function to sensors and actuators converts

- the input signals and/or data obtained from the machine/process to appropriate signal levels for processing;
- the output signals and/or data from the signal processing function to appropriate signal levels to drive actuators and/or displays.

The input/output signals to the interface functions may be coming from special modules which pre-process external sensor signals according to the defined functions contained in the special modules themselves. Examples of such special modules include PID module, fuzzy-control module, high-speed counter module, motion modules and others.

Communication function

The communication function provides data exchange with other systems (third-party devices) such as other PLC-systems, robot controllers, computers, etc.

Human-machine interface (HMI) function

The HMI function provides for interaction between the operator, the signal processing function and the machine/process.

Programming, debugging, testing and documentation functions

These functions provide for application programme generation and loading, monitoring, testing and debugging as well as for application programme documentation and archiving.

Power-supply functions

The power-supply functions provide for the conversion and isolation of the PLC-system power from the mains supply.

Characteristics of the CPU function

The capabilities of the programmable controllers are determined by programmable functions which, for ease of use, are subdivided into application oriented groups:

- logic control (logic, timers, counters);
- signal/data processing (mathematical functions, data handling, analogue data processing);
- interfacing functions (input/output, other systems, HMI, printers, mass memory);

- execution control;
- system configuration.

Operating system

The operating system function is responsible for the management of internal PLC-system interdependent functions (configuration control, memory management, application programme execution management, communication with peripherals and with the interface functions to sensors and actuators, etc.).

After a power-down or a distortion, the PLC system can restart in three different ways.

Cold restart

Restart of the PLC-system and its application programme after all dynamic data (variables such as I/O image, internal registers, timers, counters, etc., and programme contexts) are reset to a predetermined state. A cold restart may be automatic (for example, after a power failure, a loss of information in the dynamic portion(s) of the memory(ies), etc.) or manual (for example, push-button reset, etc.).

Warm restart

Restart after a power failure with a user-programmed predetermined set of remnant data and a system predetermined application programme context. A warm restart is identified by a status flag or equivalent means made available to the application programme indicating that the power failure shut-down of the PLC-system was detected in the run mode.

Hot restart

Restart after power failure that occurs within the process-dependent maximum time allowed for the PLC-system to recover as if there had been no power failure.

All I/O information and other dynamic data as well as the application programme context are restored or unchanged.

Hot-restart capability requires a separately powered real-time clock or timer to determine elapsed time since the power failure was detected and a user-accessible means to programme the process-dependent maximum time allowed.

Memory for application data storage

Application programme storage

The application programme storage provides for memory locations to store a series of instructions whose periodic or event-driven execution determines the progression of the

machine or the process. The application programme storage may also provide for memory locations to store initial values for application programme data.

Application data storage

The application data storage provides for memory locations to store I/O image table and data (for example, set values for timers, counters, alarm conditions, parameters and recipes for the machine or the process) required during the execution of the application programme.

Memory type, memory capacity, memory utilisation

Various types of memory are in use: read/write (RAM), read-only (ROM), programmable read-only (PROM), reprogrammable read-only (EPROM/UV-PROM, EEPROM). Memory retention at power failure is achieved by a proper selection of the memory type where applicable (for example, EPROM, EEPROM) or the use of memory back-up for volatile memories (for example, a battery).

Memory capacity relates to the number of memory locations in Kbytes, which are seserved to store both the application programme and the application data. Memory capacity measurements are:

- capacity in the minimum useful configuration;
- size(s) for expansion increments;
- capacity(ies) at maximal configuration(s).

Each programmable function used by the application programme occupies memory locations. The number of locations required generally depends on the programmable functions and the type of programmable controller.

Application data storage requires memory capacity depending on the amount and format of data stored.

Execution of the application programme

An application programme may consist of a number of tasks. The execution of each task is accomplished sequentially, one programmable function at a time until the end of the task. The initiation of a task, periodically or upon the detection of an event (interrupt condition), is under the control of the operating system.

Characteristics of the interface function to sensors and actuators

Types of input/output signals

Status information and data from the machine/process are conveyed to the I/O system of the programmable controller by binary, digital, incremental or analogue signals. Conversely,

decisions and results determined by the processing function are conveyed to the machine/process by use of appropriate binary, digital, incremental or analogue signals. The large variety of sensors and actuators used requires accommodating a wide range of input and output signals.

Characteristics of the input/output system

Various methods of signal processing, conversion an isolation are used in input/output systems. The behaviour and performance of the PLC-system depend on the static/dynamic evaluation of the signal (detection of events), storing/non-storing procedures, opto-isolation, etc.

Input/output systems in general display a modular functionality which allows for configuration of the PLC-system according to the needs of the machine/process and also for later expansion (up to the maximum configuration).

The input/output system may be located in close proximity to the signal processing function or may be mounted close to the sensors and actuators of the machine/process, remotely from the signal processing function.

Characteristics of the communication function

The communication function represents the communication aspects of a programmable controller. It serves the programme and data exchange between the programmable controller and external devices or other programmable controllers or any devices in an automated system.

It provides functions such as device verification, data acquisition, alarm reporting, programme execution, and I/O control, application programme transfer, and connection management to the signal-processing unit of the PLC from or to an external device.

The communication function is generally accomplished by serial data transmission over local area networks or point-to-point links.

Characteristics of the human-machine interface (HMI) function

The HMI function has two purposes.

- To provide the operator with the information necessary for monitoring the operation of the machine/process.
- To allow the operator to interact with the PLC-system and its application programme in order to make decisions and adjustments beyond their individual user scope.

Characteristics of the programming, debugging, monitoring, testing and documentation functions

These functions are implemented as either an integral or an independent part of a programmable controller and provide for code generation and storage of the application programme and application data in the programmable controller memory(ies) as well as retrieving such programmes and data from memory(ies).

Language

For the programming of the application, there is a set of languages defined in IEC 61131-3.

Textual languages

Instruction list (IL) language

A textual programming language using instructions for representing the application programme for a PLC-system.

Structured text (ST) language

A textual programming language using assignment, sub-programme control, selection and iteration statements to represent the application programme for a PLC-system.

Graphical languages

Function block diagram (FBD) language

A graphical programming language using function block diagrams for representing the application programme for a PLC-system.

Ladder diagram (LD) language

A graphical programming language using ladder diagrams for representing the application programme for a PLC-system.

Sequential function chart (SFC)

A graphical and textual notation for the use of steps and transitions to represent the structure of a program organisation unit (program or function block) for a PLC-system. The transition conditions and the step action can be represented in a subset of the above-listed languages.

Writing the application programme

Generating the application programme

The application programme may be entered via alphanumeric or symbolic keyboards and, when menu-driven displays are, or a graphical programme entry is, used via cursor keys, joystick, mouse, etc. All programme and data entries are generally checked for validity and

internal consistency in such a way that the entry of incorrect programmes and data is minimized.

Displaying the application programme

During application programme generation, all instructions are displayed immediately, statement by statement or segment by segment (in the case of a monitor or other large display). In addition, the complete programme can generally be printed. If alternative representation of programming language elements is available, than the display representation is generally user-selectable.

Automated system start-up

Loading the application programme

The generated programme resides either in the memory of the programmable controller or in the memory of the PADT. The latter requires a programme transfer via down-loaded or memory cartridge insertion into the programmable controller before start-up.

Accessing the memory

During start-up or trouble-shooting operations, the application programme and application data storage are accessed by the PADT as well by the processing unit to allow programme monitoring, modification and correction. This may be done on line (i.e. while the PLC-system is controlling the machine/process).

Adapting the programmable controller system

Typical functions for adapting the PLC-system to the machine/process to be controlled are:

- test functions which check the sensors and actuators connected to the PLC-systems (for example, forcing the outputs of the PLC-system);
- test functions which check the operation of the programme sequence (for example, setting the flags and forcing the inputs);
- setting or resetting of variables (for example, timers, counters, etc.).

Indicating the automated system status

The ability to provide information about the machine/process and the internal status of the PLC-system and of its application programme facilitates the start-up and debugging of a PLC application. Typical means are:

- status indication for inputs/outputs;
- indication/recording of status changes of external signals and internal data;
- scan time/execution time monitoring;
- real-time visualization of programme execution and data processing;
- fuse/short-circuit protection status indicators.

Testing the application programme

Test functions support the user during writing, debugging and checking the application programme. Typical test functions are:

- checking the status of inputs/outputs, internal functions (timers, counters);
- checking programme sequences, for example, step-by-step operations, variations of programme cycle time, halt commands;
- simulation of interface functions, for example, forcing of I/Os, of information exchanged between tasks or modules internal to the PLC-system.

Modifying the application programme

Functions for modification provide for changing, adjusting and correcting application programmes. Typical functions are search, replace, insert, delete, and add; they apply to characters, instructions, programme modules, etc.

Documentation

A documentation package should be provided to fully describe the PLC-system and the application. The documentation package may consist of

- description of the hardware configuration with project-dependent notations;
- application programme documentation consisting of
  - programme listing, with possible mnemonics for signals and data processed;
  - cross-reference tables for all data processed (I/Os, internal functions such as internal stored data, timers, counters, etc.);
  - comments;
  - description of modifications;
  - maintenance manual.

Application programme archiving

For rapid repair and to minimize down-time, the user may want to store the application programme in non-volatile media such as flash, PC-cards, EEPROM, EPROM, discs, etc. Such a record needs to be updated after every programme modification so that the programme executed in the PLC-system and the archived programme remain the same.

Characteristics of the power-supply functions

The power supply functions generate voltages necessary to operate the PLC-system and generally also provide control signals for proper ON/OFF synchronization of the equipment. Various power supplies may be available depending on supply voltages, power consumption, parallel connection, requirements for uninterruptible operation, etc.

AVAILABILITY AND RELIABILITY

Every automated system requires a certain level of availability and reliability of its control system. It is the user's responsibility to ensure that the architecture of the overall automated system, the characteristics of the PLC-system and its application programme will jointly satisfy the intended application requirements.

Architecture of the automated system

Techniques such as redundancy, fault tolerance and automatic error checking, as well as machine/process diagnostic functions can provide enhancements in the area of availability of the automated system.

Architecture of the programmable controller system

A modular construction in conjunction with suitable internal self-tests allowing rapid fault identification may provide enhancements in the area of maintainability of the PLC-system and therefore of the availability of the automated system. Techniques such as redundancy and fault tolerance of the availability of the automated system. Techniques such as redundancy and fault tolerance may also be considered for special applications.

Design, testing and maintenance of the application programme

The application programme is a key component of the overall automated system. Most programmable controllers provide enough computing power to permit implementation of diagnostic functions in addition to the minimum control function. Machine/process behaviour modelling and subsequent identification of faulty conditions should be considered.

Adequate testing of the application programme is mandatory. Every modification implies proper design and testing so that the overall availability and reliability are not impaired. The programme documentation shall be maintained and annotated accordingly.

Installation and service conditions

PLC-systems are typically of rugged design and intended for general purpose service. However, as for any equipment, the more stressing the service conditions, the worse is the reliability, and benefit in this area may be expected when permitted service conditions are better than the normal service conditions specified in IEC 61131-2. Some applications may require consideration of special packaging, cooling, electrical noise protection, etc., for reliable operation.

Conclusion

In Part 1 of these materials information about programmable controllers, their principle of operation and programming is presented. It was intended to start presentation of each problem starting from its basics, and end with its detailed analysis. Any time it was appropriate, necessary comparisons in relation to various programmable controllers have been made and discussed. A special effort have been made to present all material step-by-step in a well organised manner – especially Chapter 4 about programming. In the last chapter in this part of materials selected information from the International Standard IEC 61131 is presented. It should help to get acquainted with programmable controllers and programmable controller based control systems basics, as well as with programmable controllers nomenclature.

These materials are auxiliary materials for students to better understand the course "Programmable controllers". Completion of this course should give them a good basis to start their engineering activities in the field of programmable controllers applications.

References

1. Automation Systems and Drives. Main Catalogue 2001/2002. Moeller GmbH, Bonn, 01/2001, HPL0213-2001/2002GB.
2. Automation systems. Main Catalogue 2004/2005. Moeller GmbH, Bonn, HPL0213-2004/2005GB-INT MM/We 10/04.
3. Bauernfeind D., Dung O., Skupin J.: CM4-505-GS1 AS-Interface Master: Gateway for Suconet K – Actuator Sensor Interface. Moeller GmbH, Bonn, AWB 27-1271-GB, 09/96.
4. Berger H.: Automatyzacja za pomocą sterownika SIMATIC S5-115U. SIEMENS AG, Erlangen, 1987.
5. Brock S., Muszyński R., Urbański K., Zawirski K.: Sterowniki Programowalne. Wydawnictwo Politechniki Poznańskiej, Poznań, 2000.
6. Broel-Plater B.: Układy wykorzystujące sterowniki PLC. Projektowanie algorytmów sterowania. Wydawnictwa Naukowe PWN, Warszawa, 2010.
7. Cena G., Valenzano A., Vitturi S.: Hybrid Wired/Wireless Networks for Real-Time Communications. IEEE Industrial Electronics, Vol.2, Nr 1, 2008, pp. 8-20.
8. DL405 User Manual, 2nd Edition. Direct Logic Koyo, PLC *Direct*™ Inc., 1995.
9. Dirnfeldner M.: Automation with Programmable Controllers. An Introduction for Beginners. LS 27-064 GB, Klockner-Moeller 12/88.
10. Flaga S.: Programowanie sterowników PLC w języku drabinkowym. Wydawnictwo BTC, Legionowo, 2010.
11. Gomółka W.: Realizacja algorytmów sterowania sekwencyjnego za pomocą sieci działań GRAFCET. PAK,1990, Nr 7, s. 136-141.
12. Hitachi Programmable Controller - E Series. Operation Manual, No. NJI 008A.
13. Kamiński K.: Programowanie sterownika S7. Norkom, Gdańsk, 2000.
14. Kamiński K.: Programowanie w Step 7 MicroWin. Wydawnictwo GRYF, Gdańsk, 2006.
15. Kamiński K.: Podstawy sterowania z PLC. Wydawnictwo GRYF, Gdańsk, 2009.
16. Kasprzyk J.: Programowanie sterowników przemysłowych. WNT, Warszawa, 2006.
17. Kiełtyka L., Stoiński D., Fengler W.F.: Wykorzystywanie sieci Petriego do sterowania procesami sekwencyjnymi. Prace IX Sympozjum SPD-9 „Symulacja procesów dynamicznych", Polana Chochołowska, 10-14 czerwiec 1996, s. 125-129.
18. Konen P.L., Leuchs K.: Networking Programmable Controllers. An Introduction for Beginners. G 27-2100-GB, Klockner-Moeller 5/90.

19. Król A., Moczko-Król J.: S5/S7 Windows. Programowanie i symulacja sterowników PLC firmy SIEMENS. Wydawnictwo Nakom, Poznań, 2000.
20. Kwaśniewski J.: Programowalne sterowniki przemysłowe w systemach sterowania. J.Kwaśniewski & Fundacja Dobrej Książki, Kraków, 1999.
21. Kwaśniewski J.: Sterowniki PLC w praktyce inżynierskiej. Wydawnictwo BTC, Legionowo, 2008.
22. Kwaśniewski J.: Programowalny sterownik SIMATIC S7-300 w praktyce inżynierskiej. Wydawnictwo BTC, Legionowo, 2009.
23. Legierski T., Kasprzyk J., Hajda J., Wyrwał J.: Programowanie sterowników PLC. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 1998.
24. Lüder A., Lorentz K.: IAONA Handbook – Industrial Ethernet. IAONA e.V., Magdeburg, 2005.
25. Maczyński A.: Sterowniki programowalne PLC. Budowa systemu i podstawy programowania. Astor Sp. z o.o., Kraków, 2002.
26. Menden R., Petrick B.: Manual. Structuring of User Programs. Klockner-Moeller, Bonn 5/91, AWB 27-1011-GB.
27. Mikulczyński T., Samsonowicz Z.: Synteza sekwencyjnych układów sterowania zautomatyzowanych procesów technologicznych. PAK, Nr 7, 1994.
28. Mikulczyński T., Samsonowicz Z.: O syntezie sekwencyjnych układów sterowania ZPT. PAK, Nr 10, 1994.
29. Mikulczyński T., Samsonowicz Z.: Analiza wybranych metod modelowania i programowania dyskretnych procesów produkcyjnych. PAK, Nr 3, 1995.
30. Mikulczyński T., Samsonowicz Z.: Automatyzacja dyskretnych procesów produkcyjnych. WNT, Warszawa, 1997.
31. Mosoń I., Karkosiński D.: Programowalne sterowniki logiczne w dydaktyce. „Zastosowanie komputerów w dydaktyce ’94”, Cykl seminariów zorganizowanych przez PTETiS Oddział w Gdańsku, Zesz. Nauk. Wydz. Elektrycznego Politechniki Gdańskiej Nr 6, 1994, s.89-94.
32. Mosoń I.: Sterowniki programowalne PS4 w zdecentralizowanych systemach sterowania. Elektroinstalator (Automatyka) 1996, Nr 11, s.27-30.
33. Mosoń I., Karkosiński D.: Program sterowania, diagnozowania stanu technicznego i monitorowania parametrów pracy linii technologicznej okleinowania płyt meblowych. Opis algorytmu. 1998, 15 s. 2 rys. 3 tabl. bibliogr. 6 poz. maszyn.

34. Mosoń I.: Język GRAFCET a weryfikacja algorytmów i programów sterowania. Materiały: Seminarium Naukowo-Techniczne „Zastosowanie nowoczesnej aparatury Schneider Electric w urządzeniach i instalacjach elektroenergetycznych niskiego napięcia", Gdańsk, 9 czerwiec 1999, s. 4-14.
35. Mosoń I.: Wybrane aspekty wprowadzenia do dydaktyki przedmiotu Sterowniki Programowalne. „Zastosowanie komputerów w dydaktyce '99", IX cykl seminariów zorganizowanych przez PTETiS Oddział w Gdańsku, Zesz. Nauk. Wydz. Elektrotechniki i Automatyki Politechniki Gdańskiej Nr13, 1999, s.101-106.
36. Mosoń I.: Sterowniki programowalne – strukturyzacja programów sterowania. „Zastosowanie komputerów w nauce i technice 2001", XI cykl seminariów zorganizowanych przez PTETiS Oddział w Gdańsku, Zesz. Nauk. Wydz. Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 17, 2001, s.145-152.
37. Mosoń I., Żukowski K.: Symulator Sym-PS4 sterownika programowalnego PS4-201-MM1. „Zastosowanie komputerów w nauce i technice 2002", XII cykl seminariów zorganizowanych przez PTETiS Oddział w Gdańsku, Zesz. Nauk. Wydz. Elektrotechniki i Automatyki Politechniki Gdańskiej Nr 18, 2002, s.125-130.
38. Mosoń I.: Control systems with programmable controllers – teaching aspects. IX Konferencja Naukowo-Techniczna „Zastosowania Komputerów w Elektrotechnice" ZKwE'2004, Poznań/Kiekrz, 19-21 kwietnia 2004, s. 621-624.
39. Orłowski H.: Komputerowe układy automatyki. WNT, Warszawa, 1987.
40. PN-EN 61131-1:2004. Sterowniki programowalne – Część 1: Postanowienia ogólne.
41. PN-EN 61131-3:2004. Sterowniki programowalne – Część 3: Języki programowania.
42. Practical Aspects of Industrial Control Technology. Telemecanique Technical Collection. Editions CITEF, 1994.
43. Roersch P.: LE4-505-BS1 AS-I Network Module. Hardware and Engineering. Moeller GmbH, Bonn, AWB 27-1314 GB, 02/98.
44. Ruda A., Olesiński R.: Sterowniki Programowalne PLC. Wydawnictwo COSiW SEP, Warszawa, 2003.
45. Sałat R., Korpysz K., Obstawski P.: Wstęp do programowania sterowników PLC. Wydawnictwo Komunikacji i Łączności, Warszawa, 2010.
46. Schunemann U.: Programming PLCs with an Object-Oriented Approach. ATP International, Automation Technology In Practice. Nr 2, 2007, pp. 59-63.
47. Seta Z.: Wprowadzenie do zagadnień sterowania – wykorzystanie programowalnych sterowników logicznych PLC. Wydawnictwo NIKOM, Warszawa, 2002.

48. Simatic S5-115U Programmable Controller. Manual. Siemens AG, 1991, EWA 4NEB 811 6130-02a.
49. Stańczak W.: Przemysłowe sieci lokalne. Sieciowe systemy komunikacyjne integrujące automatyzację wytwarzania. Podręczniki Nr 6. PIAP, Warszawa, 1998.
50. Sterownik VersaMax. Podręcznik użytkownika. Astor Sp. z o.o., Kraków, 1999.
51. Tassin A.: Structured Programming and Utilities for the Series PCD. User's Guide. SAIA AG, 1992, Edition 26/732 E4 06.94.
52. Trybus L.: Regulatory wielofunkcyjne. WNT, Warszawa, 1992.
53. Volberg J.: Actuator-Sensor Interface. Networking I/O on the fieldbus. Klockner-Moeller, Bonn, 1996.