



TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

<EIN BEITRAG ZUR PLC PROGRAMMIERUNG>

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-
Ingenieurs unter der Leitung von

Em.O.Univ.Prof. Projektass. Dipl.-Ing. Dr.h.c.mult. Dr.techn. Peter KOPACEK
E325

Institut für Mechanik und Mechatronik

Eingereicht an der Technischen Universität Wien
Fakultät für Maschinenwesen und Betriebswissenschaften

von

Yunus Emre CETIN
0325009

Untere Augartenstrasse 31/105 A-1020 Wien

Wien, am _____

TABLE OF CONTENTS

ABSTRACT	1
1. PLC (PROGRAMMABLE LOGIC CONTROLLER).....	2
1.1. Advantages of the PLCs	3
1.2. Disadvantages of the PLCs.....	3
1.3. History of PLCs (Melere, 2001).....	4
2. SIMATIC.....	6
2.1. S300 Station	7
2.1.1. Components of the Simatic Station	8
3. SIMATIC STEP 7	12
3.1. The Programming Languages in Step 7	12
3.1.1. Ladder Logic Diagram	13
3.1.2. Function Block Diagram	13
3.1.3. Statement List.....	13
3.2. Functions in Step7	14
3.2.1. AND Function	14
3.2.2. OR Function	14
3.2.3. NAND Function (Negation AND)	15
3.2.4. NOR Function (Negation OR)	15
3.2.5. XOR Function (Exclusive OR)	15
3.2.6. SR (Set-Reset) Flip Flop	16
3.2.7. Positive RLO (Result of Logic Operation) Edge Detection.....	16
3.2.8. Pulse S5 Timer	16
3.2.9. On-Delay S5 Timer	17
3.2.10. Off-Delay S5 Timer.....	18
3.2.11. Counting Up and Down S_CUD	19
4. INSTALLATION.....	20
4.1. Wiring.....	21

4.1.1.	Wiring the PS:	21
4.1.2.	Wiring the CPU:	21
4.1.3.	Wiring the signal modules:	22
4.2.	Configuring the Station	22
4.3.	Addressing	25
4.3.1.	Default channel addressing:	25
4.3.2.	User-defined Addressing:	26
4.4.	Connecting to a Programming Device	28
5.	PROGRAMMING WITH STEP 7	30
5.1.	Creating a program in OB1 (Organisation Block):	31
5.2.	Creating a Function Block (FB) and Function (FC):	32
5.3.	Downloading the Program:	33
6.	S7-PLCSIM (SIMULATION PROGRAM)	36
6.1.	CPU Operating Modes:	36
6.2.	CPU Indicators:	37
6.3.	Object viewing:	38
6.4.	Scan Mode Options:	39
6.5.	Opening and saving a simulated PLC:	39
7.	FUTURE ASPECTS	41
7.1.	Trends in automation systems	41
7.1.1.	Visual Control	41
7.1.2.	Decentralisation	41
8.	DIFFERENT EXAMPLES OF PLC APPLICATIONS	43
8.1.	Pump Motor Control	43
8.1.1.	Simulation of control pump motor	48
8.2.	Conveyor Belt Controller	51
8.2.1.	Simulation of conveyor belt controller	53
8.3.	Determining the Direction of the Rotating Shaft	55
8.3.1.	Simulation of the Rotating Shaft	59
8.4.	Sorting and Packing of Pipes	61
8.4.1.	Simulation of Pipe Sorting and Packing	67

8.5. Garage Door Controller	70
8.5.1. Simulation of Garage Door Controller	74
9. CONCLUSION	77
REFERENCES	80
LIST OF FIGURES	82
LIST OF TABLES	84

ABSTRACT

Firstly the work in this thesis aspires to present one of the most important notions in industrial automation: *Programmable Logic Control*. If we take a look at the evolution of the industrial automation, it is easily noticed that the development of the PLC has greatly influenced mass production techniques over forty years. After the short description of the PLC and its specifications, this thesis takes an interest in the history of PLCs from 1960's till today by mentioning important milestones and developments.

This thesis mainly focuses on S7-300 Simatic Station, which is a very popular PLC station around the world and belongs to Siemens, one of the biggest manufacturers of the PLC systems. Before the main components of a S7-300 station are described, an overview of the other Simatic stations such as S7-200, -400, Simatic TDC and Simatic WinAC has been provided.

Step 7 is used to configure and program the Simatic S7 automation systems. One section of this thesis is dedicated to the description of the programming languages and most used basic functions such as OR, AND, SR (Set Reset) flip flop, ON Delay timer and etc.

The aim of this work is to give different examples of industrial automation applications for educational purposes, characterise problems and find solutions. At first, the whole process is described step by step: how a S7-300 station should be installed, wired, configured and addressed properly. After preparing the station for use, this thesis shows also how you can control program flow and design structured programs. Additionally, each example is simulated by integrated simulation software S7-Plcsim to test user programs offline without using any hardware.

1. PLC (PROGRAMMABLE LOGIC CONTROLLER)

A programmable logic controller is a device which is used for automation of industrial processes, such as control of several machines which working on assembly lines. With the PLC many inputs and output ports can be easily configured and programmed with many different arrangements. Programmable logic controllers are designed for real-time use, because the outputs results have to be produced according to input values within a bounded time. Programs to control the machine operations are written mostly in logic ladder diagram, Basic or C. Programs are stored in non-volatile memory (They can retain the stored information even when they are not powered).

Since PLCs can read not only digital but also analog process variables such as rotations, temperature or pressure, they can be used almost in every automation process. Thus, controlling hydraulic and pneumatic components, electric motors, magnetic valves and other possible analog outputs is quite simple and easy. Maybe the most important thing that makes PLCs so popular in automation processes is that so many inputs and outputs can be wired and configured by one CPU and changing the configuration or developing the system is so easy and not comparable with the other automation controllers. Before PLC, the main problem in automation of industrial processes was to change the processes or programming, since all the devices on assembly lines were individually wired and these devices have to be connected with each others individually.

A PLC scans a program continually. It searches every input to determine whether the inputs are on or off and stores the inputs' value in its memory. Then PLC executes the program according to program structure and decides which outputs are to be turned on or off. Finally PLC updates the outputs' values. After that, PLC repeats the process continuously. Figure 1.1 shows how PLCs work:

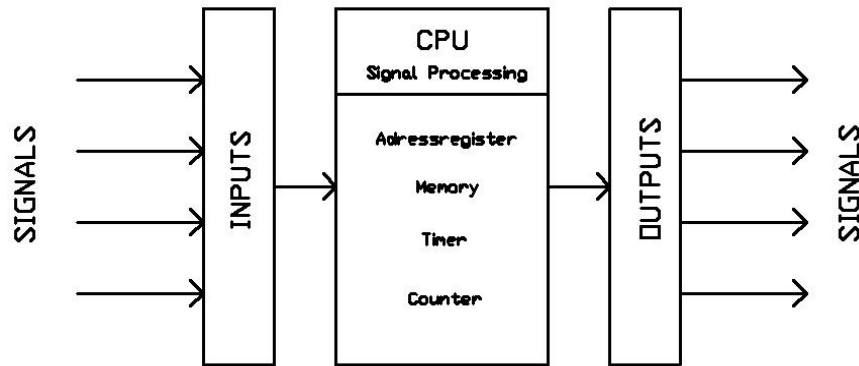


Figure 1.1 Data Flow of a typical PLC

1.1. Advantages of the PLCs

Flexibility: One single Programmable Logic Controller can easily run many machines.

Correcting Errors: Correcting errors in PLC are extremely short and cost effective.

Programming Language: PLCs offer different programming language, which are easy to understand and program.

Space Efficient: Thousands of timers and counters can be connected in a single PLC.

Modularity: Another component can be easily mounted and connected to the CPU.

Low Cost: Prices of PLC are very reasonable and cannot be compared to the prices of the contact and coils and timers that you would pay to match the same thing.

Testing: The program can be tested, validated and corrected saving very valuable time.

Visual Observation: When running a PLC program a visual operation can be seen on the screen.

1.2. Disadvantages of the PLCs

Redundant Cabling: Since every component communicate with others and CPU, too much cable connection in the whole system.

Fault Tolerance: If any problem occurs and system fails, downtime is not predictable, can take long.

Centralized Controller: If the master controller fails, whole system fails.



Figure 1.2 Typical PLCs

1.3. History of PLCs (Melere, 2001)

The evolution of PLCs began as a response to the demands of the American automotive manufacturing industry. First PLC is introduced in the late 1960's. Bedford Associates (Bedford, MA) produced its new product called a Modular Digital Controller (MODICON). MODICON was the Bedford's Associates' eighty-fourth project that brought the first commercial PLC into the automotive industry.

In the mid70's the dominate PLC technologies were sequencer state-machines and the bit-slice based CPU. The AMD 2901 and 2903 were quite popular in MODICON PLC's. Communications abilities began to appear in approximately 1973. The first such system was Modicon's Modbus. The PLC could now talk to other PLCs and they could be far away from the actual machine they were controlling. They could also now be used to send and receive varying voltages to allow them to enter the analog world. (Melere,

2001)

General Motor started to standardize communication with its automation protocol MAP in the 80's. The size of the PLC got smaller with the development of the electronic parts and circuits. Symbolic programming on personal computers became possible instead of programming terminals or handheld programmers.

The 90's have seen a gradual reduction in the introduction of new protocols, and the modernization of the physical layers of some of the more popular protocols that survived the 1980's (Melo, 2001). According to the latest standard (IEC 1131-3) plc programming languages are tried to be standardized in one international standard. PLC programming softwares offer now different programming languages, such as function block diagrams, instruction lists, ladder logic diagrams, C and structured text.

2. SIMATIC

The Simatic S7 series are electronic programmable logic controller from Siemens. Simatic is programmed with the help of STEP5 or STEP7 and softwares from other program developers. All controlling-functions are saved as a program in the memory card and read by CPU. Simatic series are modular design and many inputs, outputs and other modules can be connected with the CPU.

The name of Simatic was registered by patent office in Germany in 1958 and Simatic is the abbreviated form of **Siemens** and **Automatic**. First Simatic was designed as a hardwired controller (*VPS-Verbindungsprogrammierte Steuerung*). If any change in automation process is required, wiring and assembling of the components must be changed too. In 1973 Simatic S3, which can be called as the first programmable logic controller, was ready for end-user. But Simatic was not so popular, until Simatic S5 appeared on the markets in 1979. The Simatic S5 was developed continuously and consequently. The user programs could be written in three programming languages LAD (Ladder Logic Diagram), FBD (Function Block Diagram), STL (Statement List), which made programming much easier than ever. In 1996 with Simatic S7 Totally Integrated Automation-TIA has been developed and this strategy defines the interaction of extensive single components, tools and the services to achieve an automation solution. According to TIA, a whole automation process is customer-oriented and every single component can be integrated specifically that the limits of the automation solutions can be extended.

The interaction performs integration across the four automation levels of the automation pyramid:

1. Management Level
2. Operator's level
3. Controller's level
4. Field level (“Totally Integrated Automation”)

Simatic S7 series offer five different controllers:

1. S7-200: the first generation of Simatic S7, up to 24 Kbyte program memory and min. bit operation time 0,22 μ s. Communication with the other devices is possible over RS-485-Interface.
2. S7-300: the most popular serie of Simatic S7 controllers, up to 8 Mbyte program memory and min. bit operation time 0,01 μ s. Communication with the other devices is possible over RS-485, Profibus or Ethernet-Interface.
3. S7-400: most sophisticated serie of Simatic S7 controllers, up to 16 Mbyte program memory and min. bit operation time 0.03 μ s. Communication with the other devices is possible over RS-485, Profibus or Ethernet-Interface.
4. Simadyn D, Simatic TDC: special serie of Simatic S7 for the press machines, rolling mills, gas and oil conveyors and power plants.
5. Simatic WinAC: Simatic WinAC is used when high performace, high data volumes and at the same time hard real time are required for the automation task, especially if different tasks has to be integrated on a pc such as data processing, communication or visualization. (Siemens AG, 2008)

2.1. S300 Station

It is the second generation of Simatic S7 automation systems. The S7-300 controller is a modular design. The modules can be configured directly by the CPU or distributed modules can be connected to the CPU by PROFIBUS DP. With the centralized configuration up to eighth I/O modules can be plugged into the central rack. If this single-tier configuration is insufficient, then up to four-tier configuration can be chosen and each rack can be connected with eight modules.

Main specifications are written below (Siemens AG, 2007):

- CPUs are able to be integrated with Industrial Ethernet/PROFINET interface, integrated technological functions and fail-safe designs.
- The S7-300 can be set up in a modular configuration
- The Micro Memory Card can be used during operation for storing and accessing data
- Safety technology and motion control can also be integrated
- Extended temperature range (-25...+60°C)

2.1.1. Components of the Simatic Station

An S7-300 station consists of several modules that depend on the requirements of the automation system. The components of the S7-300 station which is to be installed are described below:

Mounting rack:

It accommodates and connects the modules. Its length is determined by the number of the modules.

Power supply (PS):

Provides internal supply voltage, converts power system voltage 120/230 V AC into 24 V DC for the Simatic stations.

Central Processing Unit (CPU):

Executes the user programs and communicates with the programming device and the

other stations. It controls the central and distributed I/O modules and supplies the backplane bus with 5 V.

The station has the *CPU 314C-2DP*:

- MPI interface onboard
- PROFIBUS DP master/slave interface
- Technological functions (Siemens AG,2008):
 - Counting
 - Closed Loop Control
 - Frequency Measurement
 - Pulse Width Modulation
 - Pulse Generator
 - Positioning

Signal Module (I/O Modules):

Signal modules are available as input/output modules for digital and analog signals. This module adapts the signals from process to the signal level or controls contactors, sensors, actuators, etc.

Digital and analog modules differ as regards the number of channels, voltage and current ranges, electrical isolation, diagnostics and alarm functions, etc. (Siemens AG, 2007)

The station has 4 different signal modules:

- DI 16 / DO 16 x DC24V (16 digital inputs and 16 digital outputs)
- SM 331 AI 8 x RTD (Analog input 8 channels with resistance temperatur detector)
- SM 331 AI 8 x 12 Bit
- SM 332 AO 4 x 12 Bit (Analog output 4 channels, 12 Bit)

Interface module (IM):

By means of this module the mounting racks can be connected with each other. The

station has no interface module.

Communication Processor (CP):

It connects the station with the subnets. It can be over integrated RS 485, Ethernet, Profibus or serial point to point connection.

The station has the standard CP 343-1 Industrial Ethernet.

A typical S7-300 Structure is shown in Figure 2.1

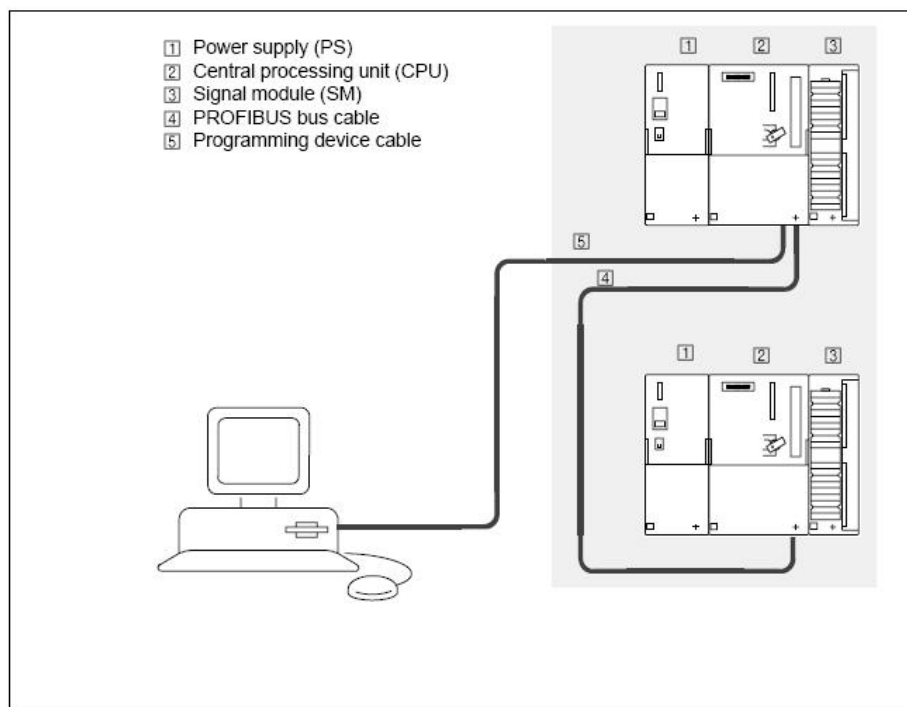


Figure 2.1 S7300 Structure with two racks

Micro Memory Card (MMC):

The memory module used on the CPU is a Simatic Micro Memory Card. User program can be saved in the MMC and read by CPU.

MPI (Multi Point Interface):

The Siemens MPI protocol is used by Siemens PLCs to communicate with external

devices like operator interfaces. MPI supports baud rates (state changes) of 187.5 kbps to 12 Mbps. The addresses of the MPI nodes must be unique and are set with the programming device PC. PROFIBUS bus line for building up the network or Repeater RS 485 for coupling segments can be used at MPI.

Digital and analog signal modules, interface module, communication processor and other possible components communicate with CPU and others through the backplane bus.

3. SIMATIC STEP 7

Simatic Step 7 is the standard software for the Simatic S7, Simatic C7 and Simatic WinAC automation systems. Step 7 enables the user to control the whole automation process with different functions. Designing the program structure, configuring and parameterizing the station, hardware and networks, programming automation process step by step, specifying the communication, simulating, testing and monitoring the program to diagnose possible errors are the most important features of Step 7. Step 7 offers the user basically three different programming languages, which are FBD (Function Block Diagram), LAD (Ladder Logic Diagram) and STL (Statement List).

3.1. The Programming Languages in Step 7

There are several programming languages and programming methods to write a user program, for instance LAD, STL, FBD, SCL and etc. The most known and used programming languages are LAD (ladder logic diagram), FBD (function block diagram) and STL (statement list). For the complex structures Structural Control Language (SCL) is preferred, since its language is like PASCAL and very suitable for programming complex algorithms.

STL is a text-oriented language, the functions are written in the form of a list. LAD and FBD are graphics-oriented languages; the structure will be described by connecting the function-blocks or -boxes in series or parallel arrangements. The program which is written in FBD or LAD can be completely converted into STL, but some expressions in STL cannot be converted into FBD and LAD. With these programming languages not only binary signals but also digital values in different formats can be processed.

Basically there are two kinds of functions:

Binary Functions: these kinds of functions process signals of data type BOOL which

means that a bit can have two states “1” or “0”.

Digital Functions: The digital functions operate on variable with digital values. These values can be the data types INT (Integer), DINT (Double Integer), and REAL.

3.1.1. Ladder Logic Diagram

On the left side of the diagram called rung a point is selected and there any program element can be inserted. These elements are called as inputs (contacts). As an output coils are used. For the elements with non-binary functions boxes are used. There are two types of standard boxes: boxes with EN/ENO parameters (e.g. data type conversion, MOVE and mathematical functions) and boxes without EN/ENO parameters (e.g. Set/Reset functions, timers, counters and compare boxes) (Berger, 2005)

3.1.2. Function Block Diagram

In the program editor the networks can be created from left to right and from top to bottom. The inputs are on the left and the outputs are on the right. Every function is shown with the boxes and the assigned logic operations are written in the boxes. Binary functions are used to check the signal states of bit addresses, like AND, OR, XOR and etc. Simple boxes operate on bit addresses, like outputs. Complex boxes are used for the non-binary functions, like time functions.

3.1.3. Statement List

In statement list the program is written in the form of a series of statements. Each statement contains a logical operation or instruction. For instance, AND functions is defined in STL as A, OR as O, NOR as NO, SET as S and so on.

3.2. Functions in Step7

The most common functions used in PLC programming language are below:

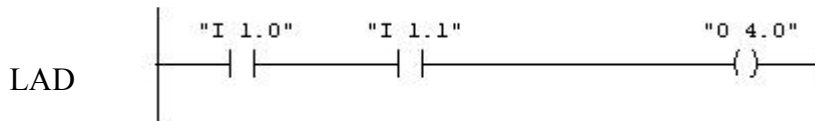
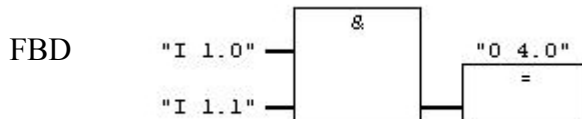
3.2.1. AND Function

The output of the AND-function has the value “1”, if every single input of the function has the value “1”.

STL A "I 1.0"
 A "I 1.1"
 = "O 4.0"

I 1.0	I 1.1	O 4.0
0	0	0
0	1	0
0	0	1
1	1	1

Table 3.1 AND-Function



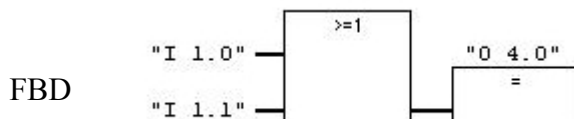
3.2.2. OR Function

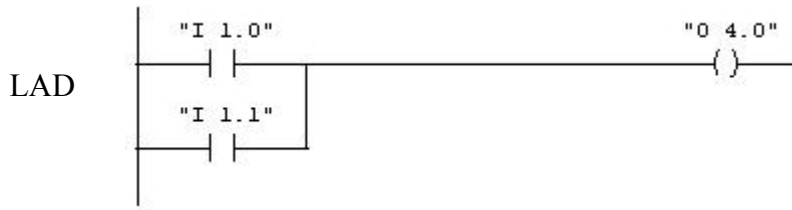
The output value of the OR-function is “1”, if any input of the function has the value “1”.

STL O "I 1.0"
 O "I 1.1"
 = "O 4.0"

I 1.0	I 1.1	O 4.0
0	0	0
1	0	1
0	1	1
1	1	1

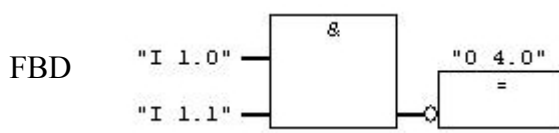
Table 3.2 OR-Function





3.2.3. NAND Function (Negation AND)

NAND-function has the output value “0”, if all of the input values are “1”.

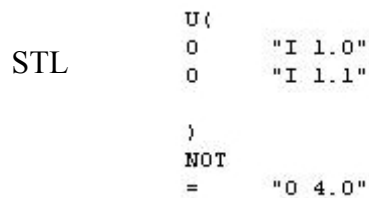


I 1.0	I 1.1	O 4.0
0	0	1
1	0	1
0	1	1
1	1	0

Table 3.3 NAND-Function

3.2.4. NOR Function (Negation OR)

Negates the OR-Function.

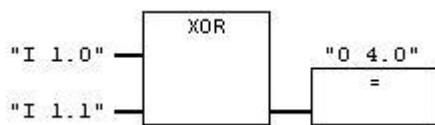


I 1.0	I 1.1	O 4.0
0	0	1
1	0	0
0	1	0
1	1	0

Table 3.4 NOR-Function

3.2.5. XOR Function (Exclusive OR)

The output value is 1, when the input value of one of the two specified addresses is 1.

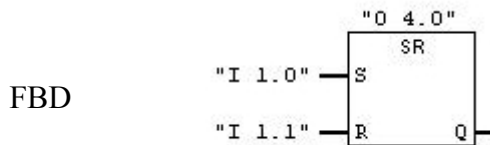


I 1.0	I 1.1	O 4.0
0	0	0
1	0	1
0	1	1
1	1	0

Table 3.5 XOR-Function

3.2.6. SR (Set-Reset) Flip Flop

Set-Reset Flip Flop is set when the signal state at input S is 1 and the signal state at input R is 0. If input S is 0 and input R is 1, the flip flop is reset. If the RLO at both inputs is 1 the flip flop is reset which means R input has the priority.



3.2.7. Positive RLO (Result of Logic Operation) Edge Detection

The positive edge detection function detects a change from 0 to 1 at the specified address and after it detects the rising edge; its value will be 1.

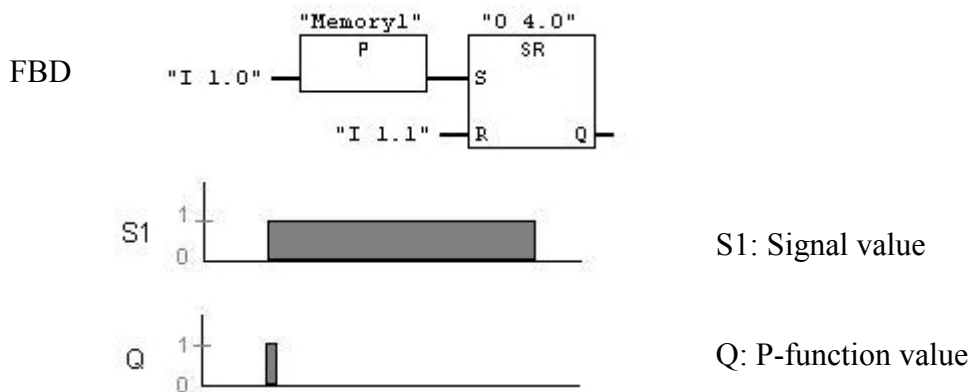


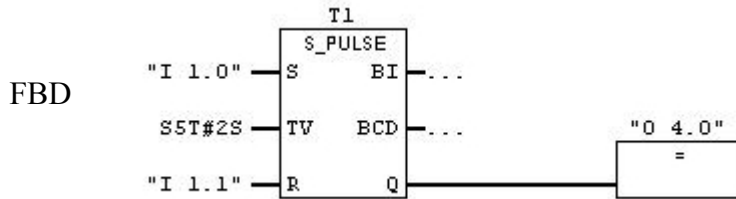
Figure 3.1 Positive Edge Detection

Memory1 stores the signal state and as a temporary data defined.

3.2.8. Pulse S5 Timer

Pulse Timer function starts a specified timer, if there is a change in signal state from 0 to 1 at the Start (S) input. The timer continues to run for the time defined at the time value TV until the defined time elapsed. While the timer is running the value of the signal is 1.

If there is a change from 1 to 0 at the S input before the time has elapsed, the timer is stopped. While the timer is running, a change from 0 to 1 at the Reset (R) input resets the timer.



As it is seen above in the diagram, output value will be 1 for two seconds, if there is a change from 0 to 1 at the S or R input.

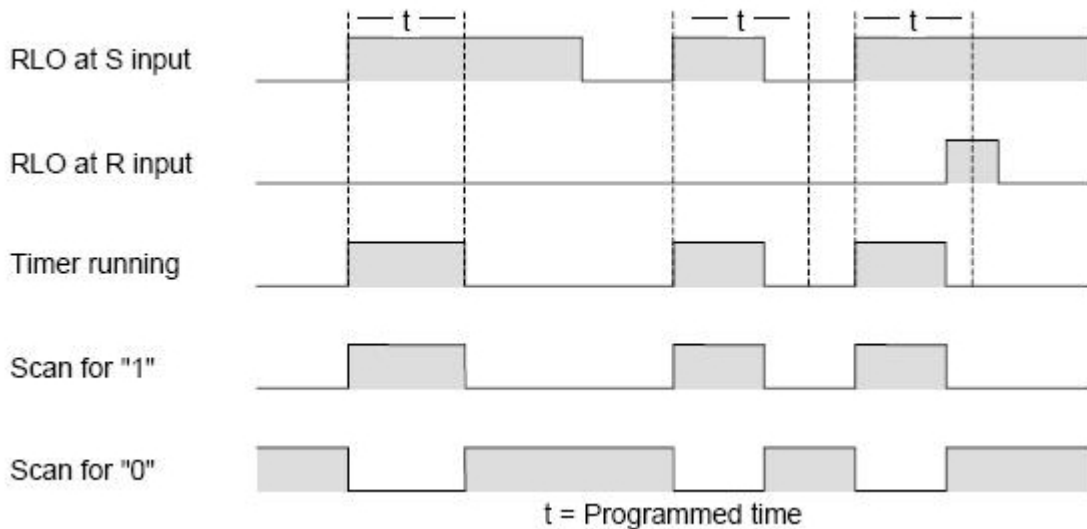
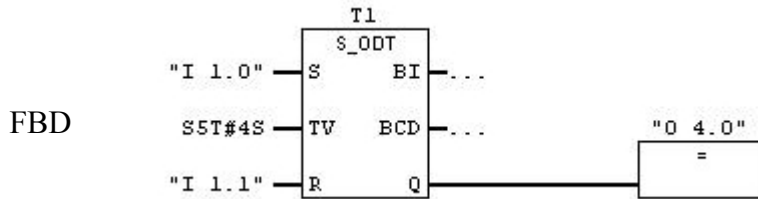


Figure 3.2 S_PULSE Time-Diagram

3.2.9. On-Delay S5 Timer

On-Delay timer starts a specified timer, if there is a change from 0 to 1 at the S input. The timer continues to run for the time defined at the time value TV until the defined time elapsed. After the defined time elapsed, output value will be 1. While the timer is running, a change from 0 to 1 or 1 to 0 at the S input resets the timer, just like a change from 0 to 1 at the R input.



After four seconds the output value will be 1, if there is no change at R and S input.

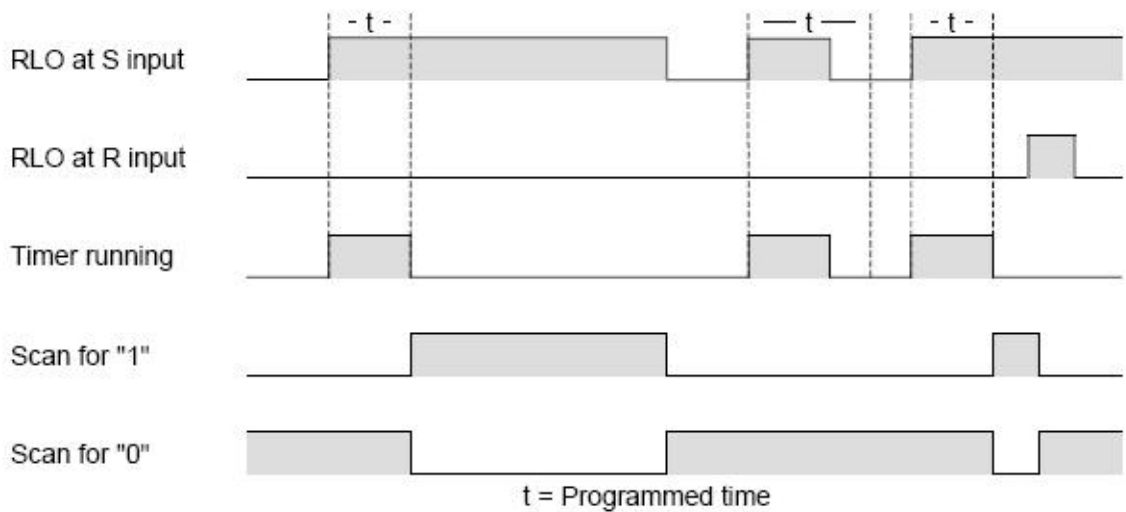
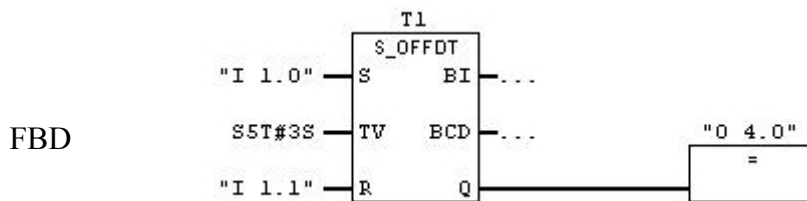


Figure 3.3 S_ODT Time-Diagram

3.2.10. Off-Delay S5 Timer

Off-Delay timer starts a specified timer, if there is a change in signal state from 1 to 0 at S input. The output value remains 1, until the defined time elapsed or input value at the S is 1. While the timer is running, a change from 0 to 1 at S input resets the timer and remains until input value at S from 1 to 0 changes. A change from 0 to 1 at R input resets the timer.



As shown above, after a change from 1 to 0 at S input, the output value remains for three seconds 1, then will be 0.

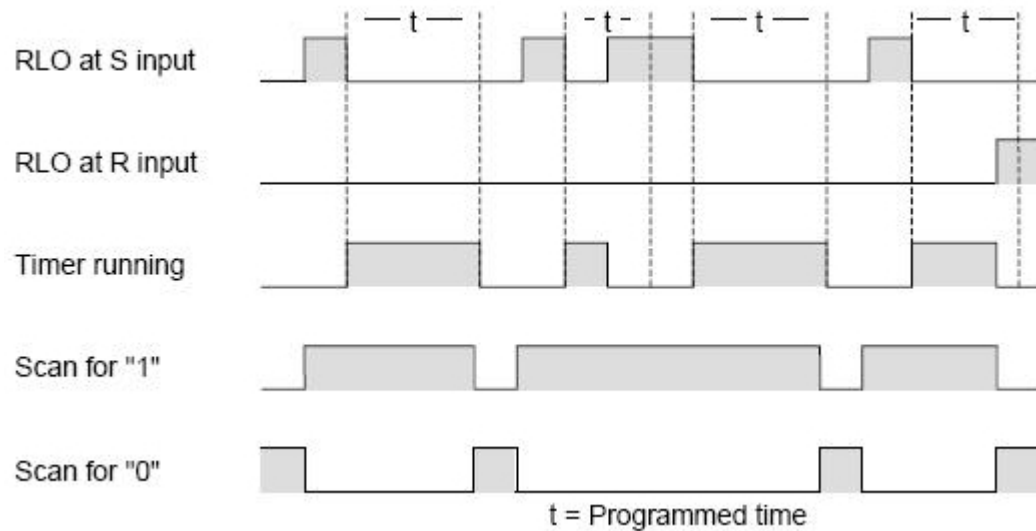
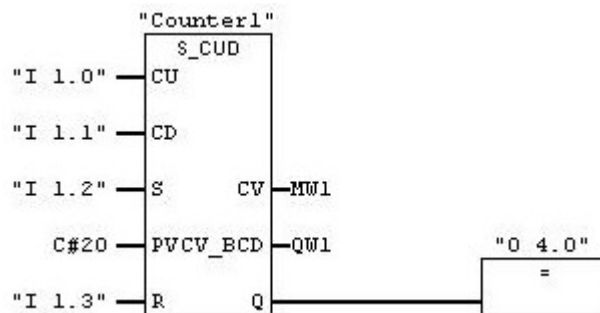


Figure 3.4 S_OFFDT Time-Diagram

3.2.11. Counting Up and Down S_CUD



With the counter function *S_CTUD*, it is possible to count up and down. The count value is changed by a rising edge on inputs. At input *I 1.0* (*CU*) the count value is incremented by 1 and at input *I 1.1* (*CD*) the count value is decremented by 1. The input *I 1.2* (*S*) presets the counter to the value defined by *PV* (in the example its value is 20). The input *I 1.3* resets the counter value to "0". The counter value can be read on output *O 4.0* or *CV* and *CV_BCD*.

4. INSTALLATION

S7-300 can be installed in a horizontal or vertical position.

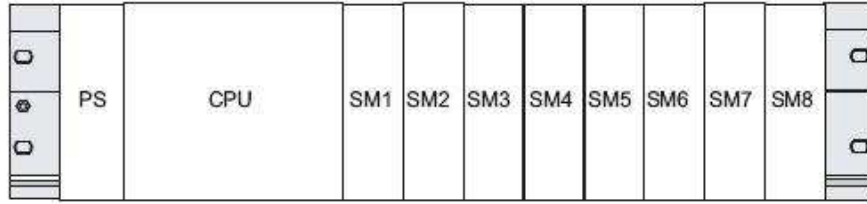


Figure 4.1 S7-300 Module Arrangement on a Single Rack

In the first row power supply (PS) is mounted and CPU is always next to PS. Signal modules, function modules and communication processor can be installed to the right of the CPU (more than eight modules are not allowed). All the modules are connected with each other and with the CPU by means of backplane bus connector. If a single rack is insufficient, it can be connected with other racks (up to four) by interface module (IM).

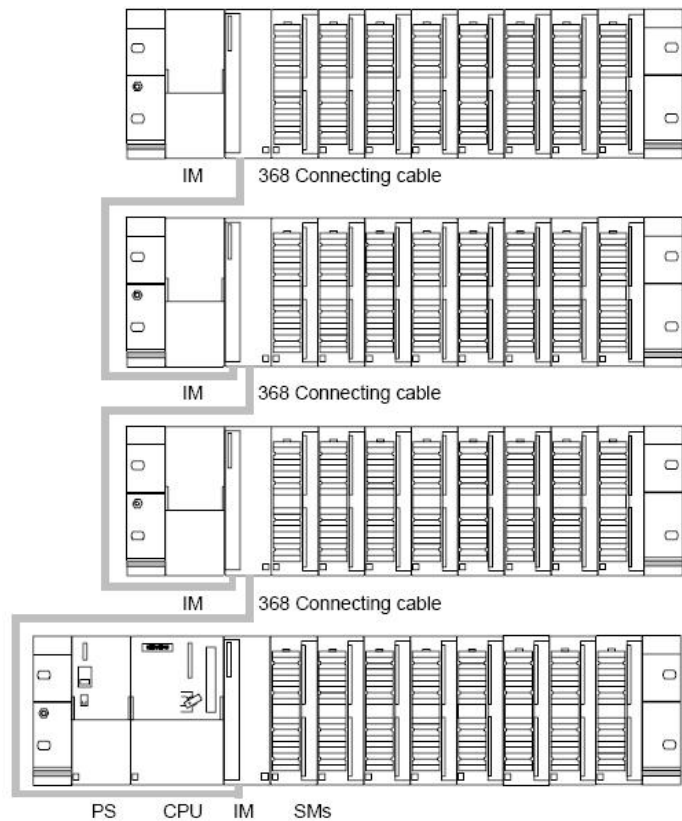


Figure 4.2 Module Arrangement on four Mounting Racks

4.1.3. Wiring the signal modules:

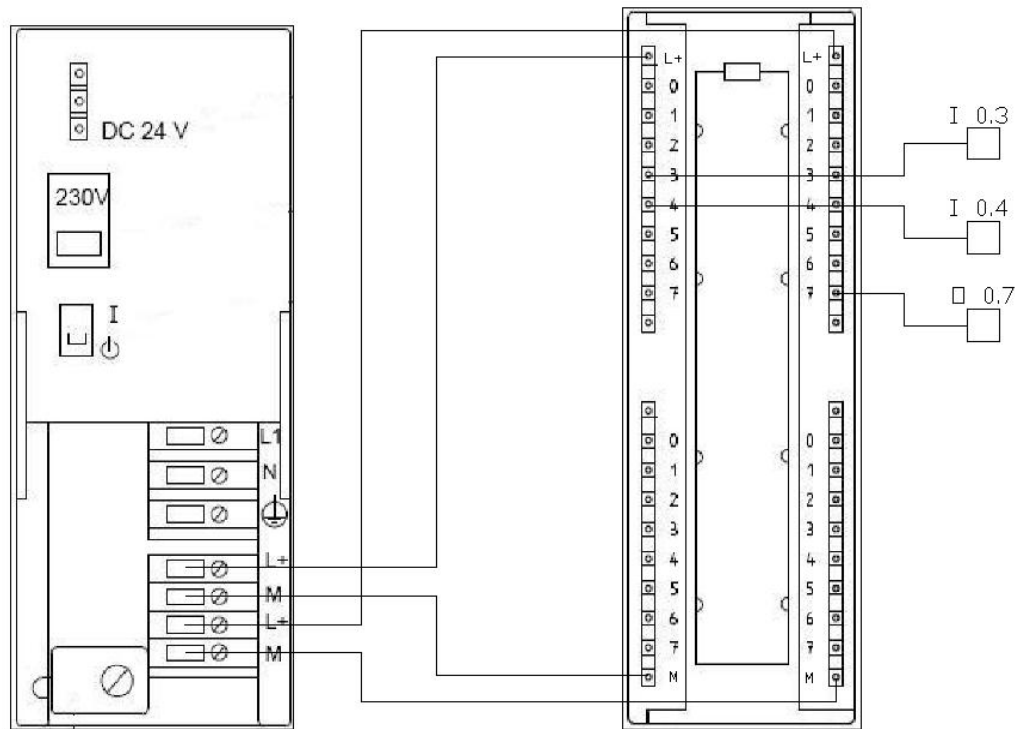


Figure 4.5 Wiring Signal Modules

As it is shown in the Figure 4.5, L+ terminal on PS307 is wired to the terminal L+ on the signal modules and M terminal on PS307 to the M terminal on signal modules. In the example two inputs (*I 0.3* and *I 0.4*) and one output (*O 0.7*) are wired to the signal module.

4.2. Configuring the Station

After power supply (PS), CPU and other modules are installed and wired properly; these have to be configured in Step 7. First we need to create a new project in Simatic Manager.

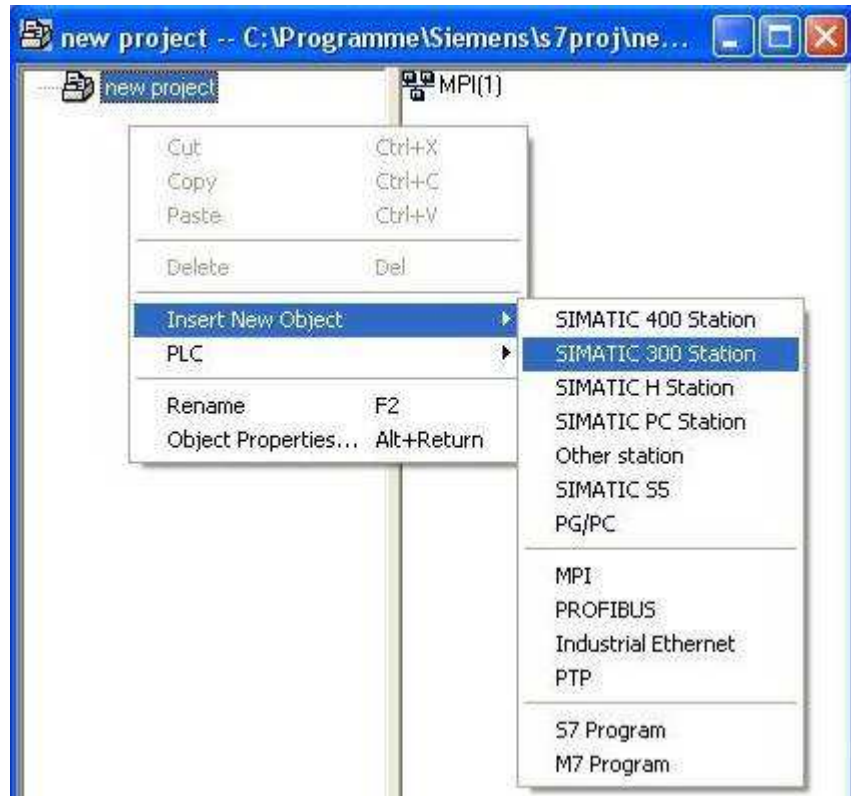


Figure 4.6 Creating S300 Station

S7-300 Station can be added to the project as shown in figure 4.6

After S7-300 Station is added to the project, we need to configure PS, CPU and signal modules. This can be done by clicking *hardware* and then *View>Catalog*. A catalog of the CPUs, PS, signal modules and other single components appears at the right side of the *station window*.

First a rail (can be found in the list RACK-300) should be added to the station window in order to insert PS, CPU and other required components. The appropriate types of CPUs, PS and signal modules are to be found in the hardware catalog and can be inserted on a mounting rail by drag&drop with right mouse click (shown in figure 4.7).

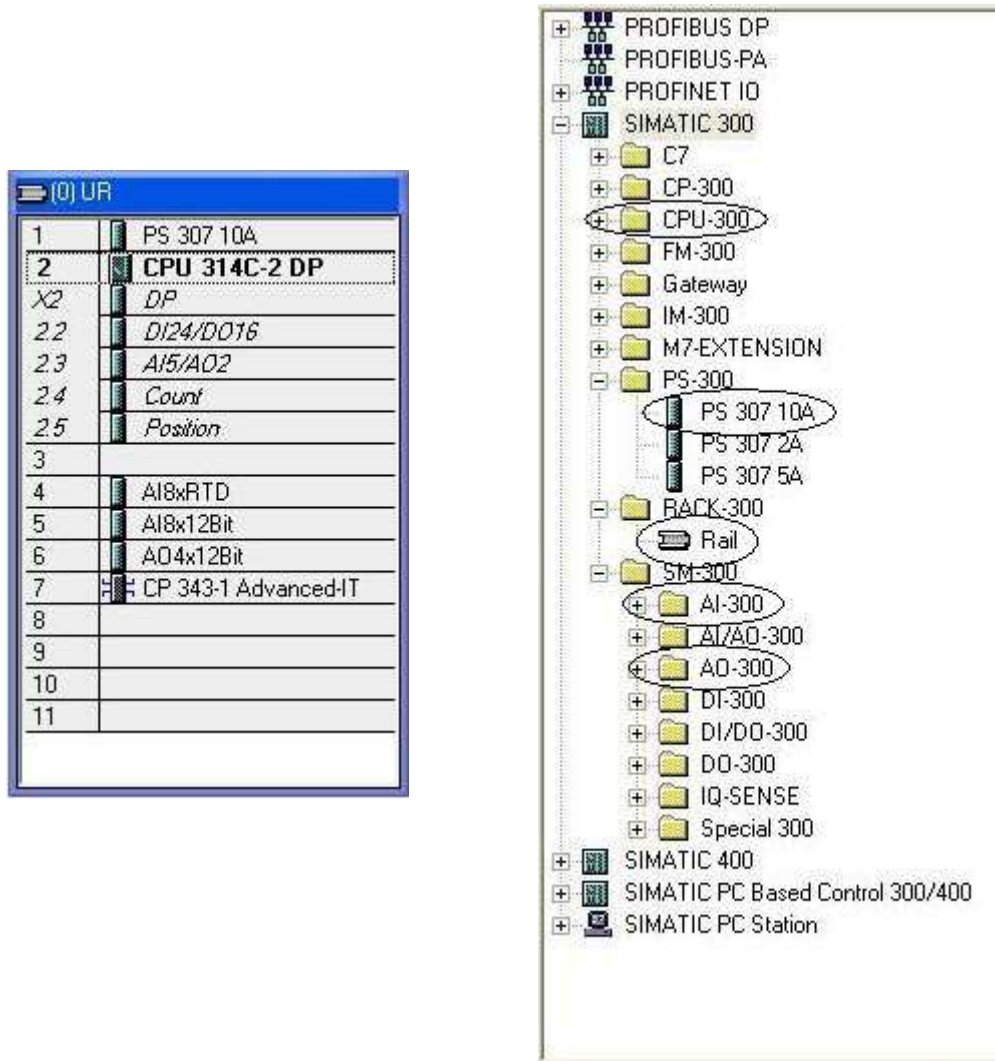


Figure 4.7 Adding components to the station

The slot rules for the S7-300 Station are below:

Slot 1: Only for power supply or empty. Three different types of PS are available, which work with 2A, 5A or 10A.

Slot 2: CPU only. The station has CPU 314C-2 DP. (If more than one rack is used, slot for the CPU should be always empty except central rack.)

Slot 3: Interface module only or empty.

Slots 4 through 11: Signal, function modules and communications processors or empty.

The lower part of the *station window* shows a detailed view of the inserted components. The order numbers, input-output addresses of the modules can be read below in table form (figure 4.8).

Slot	Module	Order number	Fir...	MPI ...	I address	Q address	Comment
1	PS 307 10A	6ES7 307-1KA00-0AA0					
2	CPU 314C-2 DP	6ES7 314-6CF02-0AB0	V2.0	2			
X2	DP				1023		
2.2	DI24/DO16				124...126	124...125	
2.3	AI5/AO2				752...761	752...755	
2.4	Count				768...783	768...783	
2.5	Position				784...799	784...799	
3							
4	AI8xRTD	6ES7 331-7PF00-0AB0			256...271		
5	AI8x12Bit	6ES7 331-7KF01-0AB0			272...287		
6	AO4x12Bit	6ES7 332-5HD01-0AB0				288...295	
7	CP 343-1 Advanced-IT	6GK7 343-1GX21-0XE0	V1.0	3	304...319	304...319	
8							
9							
10							

Figure 4.8 Station window with added components

By double clicking on the inserted components detailed information and properties of selected item are shown and in appearing window some settings can be changed.

After all of the components are inserted and configured properly, S7-300 Station can be saved and compiled by clicking on corresponding icon or under the menu *Station>Save and Compile*.

4.3. Addressing

4.3.1. Default channel addressing:

Step 7 automatically assigns a module start address and this can be viewed in the lower part of the station window. Default addressing is a slot based addressing. It means that

the modules get their start addresses according to their slot numbers. Start addresses for the signal modules on four-racks are shown below in the table (Berger, 2006):

Rack	module	Slot Number										
	start addresses	1	2	3	4	5	6	7	8	9	10	11
0	digital	PS	CPU	IM	0	4	8	12	16	20	24	28
	analog				256	272	288	304	320	336	352	368
1	digital	-		IM	32	36	40	44	48	52	56	60
	analog				384	400	416	432	448	464	480	496
2	digital	-		IM	64	68	72	76	80	84	88	92
	analog				512	528	544	560	576	592	608	624
3	digital	-		IM	96	100	104	108	112	116	120	124
	analog				640	656	672	688	704	720	736	752

Table 4.1 Default Channel Addressing

4.3.2. User-defined Addressing:

It means that the user is free to allocate any module an address of his choice. The user defines the start address of the module and the other addresses are based on this start address.

Example to digital modules:

The address of a digital module consists of a bit address and byte address.

E.g. Q 0.7

Q: Output 0: Byte address 7: Bit address

The byte address depends on the module start address.

The bit address is the number printed on the module.

Figure 4.9 shows how the addresses for a digital module are obtained:

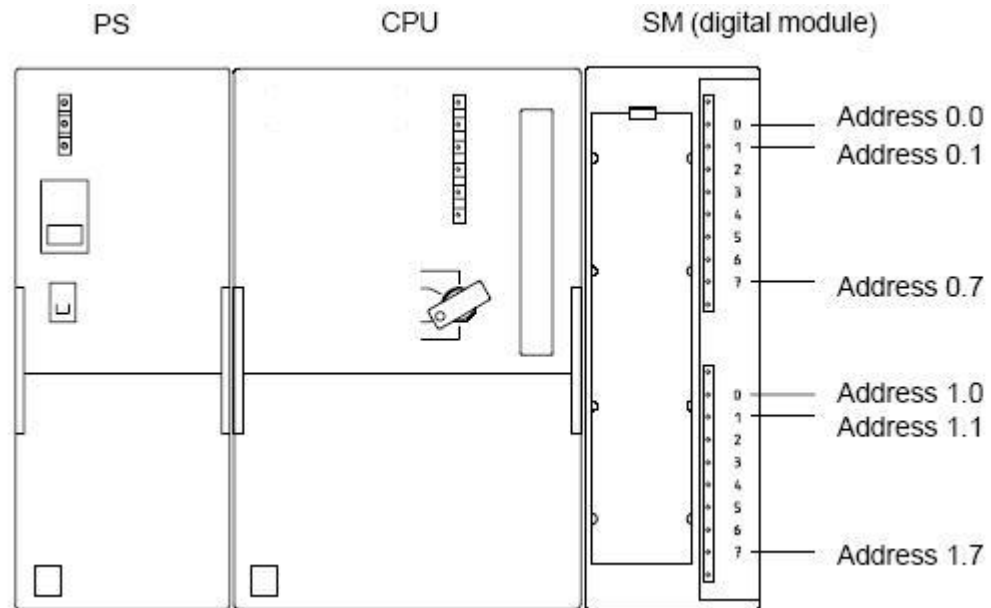


Figure 4.9 Addressing for digital modules

Example to analog modules:

Figure 4.10 shows how the addresses for an analog module are obtained:

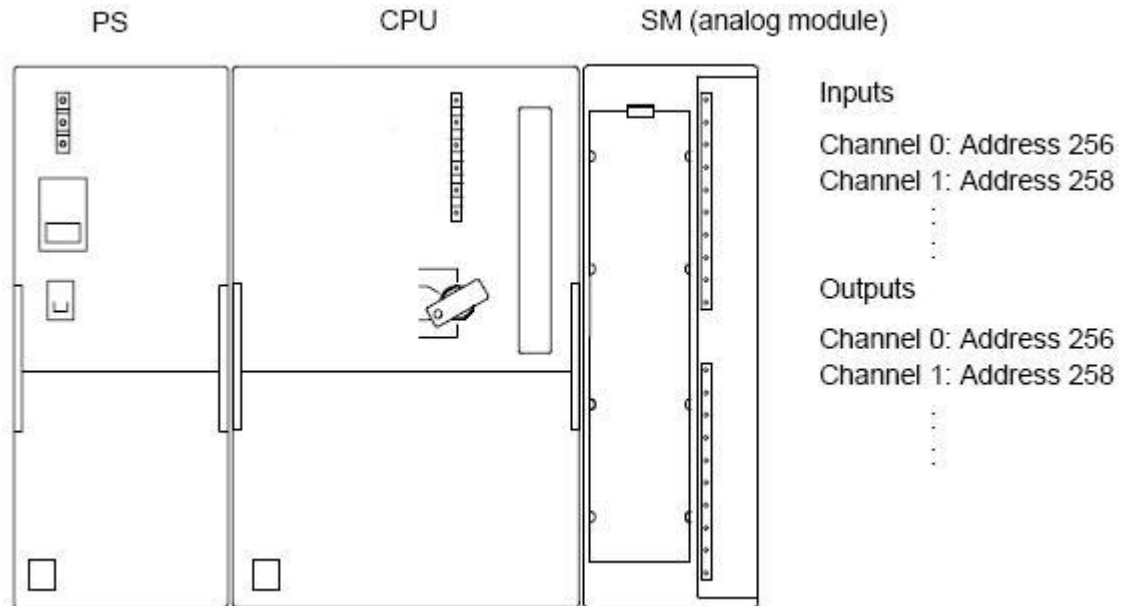


Figure 4.10 Addressing for analog modules

4.4. Connecting to a Programming Device

After the S7-300 Station is installed, wired and configured properly, it can be connected to a programming device. The programming device must have an integrated MPI interface or MPI card and connection can be established with a programming device cable. If more than one station (networked) is available, they can be connected to a programming device over PROFIBUS bus cable.

After power supply module is switched on:

- 24V DC LED on the power supply module comes on
- CPU 5V DC LED comes on
- CPU STOP LED comes on after the memory reset
- If there is no backup battery in the CPU, BATF LED comes on

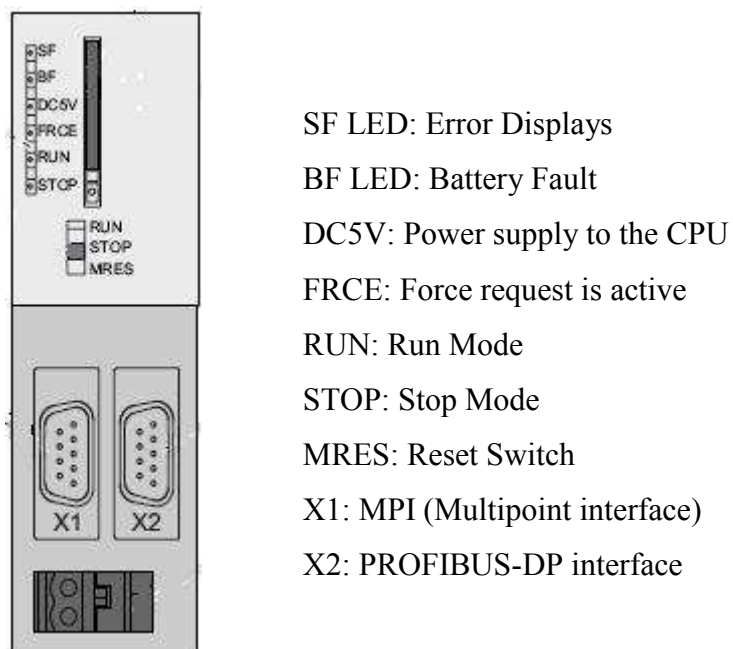


Figure 4.11 Control and Display Elements of CPU

CPU 314C-2DP does not have any integrated memory, but an external Micro Memory Card can be inserted to the station. If CPU requests a memory card reset (MRES-STOP

LED blinking at 1-second intervals), it can be done as follows:

- Switch is set to MRES and is hold in this position until the STOP LED lights continuously, it takes approximately 9 seconds.
- Within the next 3 seconds the switch has to be released and set again to MRES.

The STOP LED blinks during the delete procedure.

5. PROGRAMMING WITH STEP 7

First all of the inputs, outputs, timers, counters and other required functions must be defined in the symbol table. This can be found in the *Simatic Manager window* under CPU 314C-2DP>S7 Program>Symbols.

In symbol table symbolic names can be assigned to absolute addresses of the inputs and outputs.

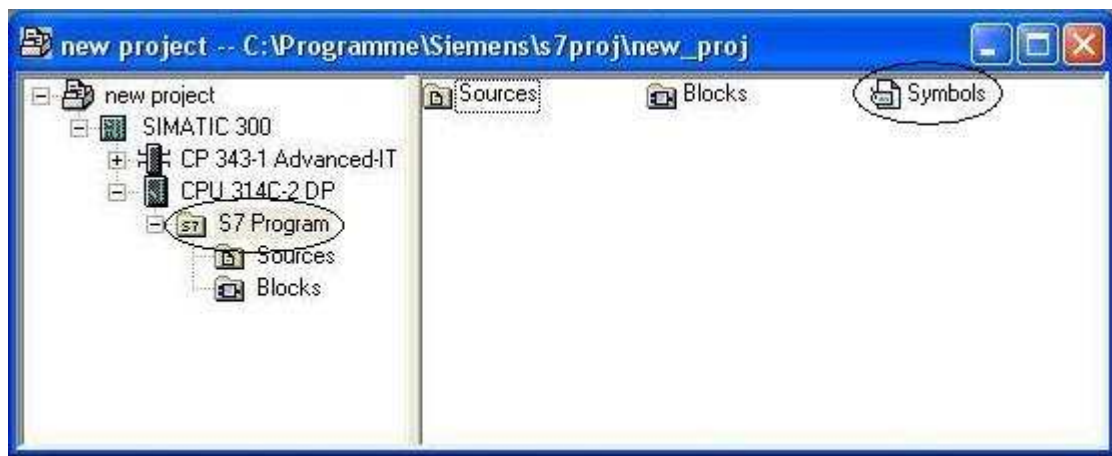


Figure 5.1 Opening Symbol Window

The screenshot shows the 'S7 Program (Symbols)' window. It contains a table with the following data:

	Status	Symbol	Address	Data type	Comment
1		Main Programm	OB 1	OB 1	
2		Input 1	I 0.0	BOOL	
3		Input 2	I 0.1	BOOL	
4		Input 3	I 0.2	BOOL	
5		Output 1	Q 0.0	BOOL	
6		Timer 1	T 1	TIMER	
7		Counter 1	C 1	COUNTER	
8		Memory 1	M 10.0	BOOL	

Figure 5.2 Editing Symbols


As it is shown above, under symbol we can assign names to the addresses, under address we can define the absolute addresses, under data type software shows automatically which data type is chosen and under comment we can describe the inputs, outputs and

other symbols so that automation process can be easily understood by other users. According to inputs, outputs, timers or counters Step 7 uses different data types, most used are shown below in the table 5.1:

Type	Size in Bits	Format Options	Example in STL
BOOL	1	Boolean text	TRUE
BYTE	8	Hexadecimal number	B#16#12
WORD	16	Binary number	2#0
		Hexadecimal number	W#16#1000
		BCD	C#55
		Decimal number unsigned	B#(100,200)
DWORD	32	Binary number	2#0
		Hexadecimal number	DW#16#0000_1111
		Decimal number unsigned	B#(0,13,43,127)
INT(integer)	16	Decimal number signed	2732
DINT	32	Decimal number signed	L#12345
REAL	32	IEEE Floating-point number	1,54E+30
S5TIME	16	S7 time	S5T#3M_15S

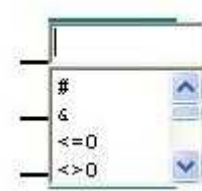
Table 5.1 Data Types

5.1. Creating a program in OB1 (Organisation Block):

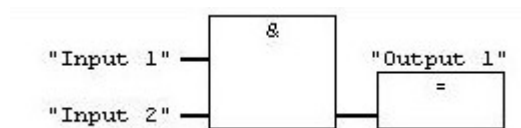
OB1 window can be opened by clicking on corresponding icon, this can be found in Simatic Manager under CPU 314C-2DP>S7 Program>Blocks>OB1. By clicking on *Empty Box* icon on the toolbar or by right mouse clicking  and then choosing insert

empty box, a new box can be added.

In appearing box a list of functions can be found. Here we can choose any function required, and then inputs and outputs can be inserted.



An example of AND-Function is shown below:



Program can be written not only in FDB, but also in STL or LAD. In *OB1 window* under *View* language settings can be changed.

A new network can be inserted by clicking on *new network* icon.



If any function must be negated, first a path should be selected, where negation is required, then by clicking on *negate binary input* icon, a function can be negated.



By clicking on *Bin.Input* icon, another input can be inserted to the box.



By clicking on *Branch* icon, one more output can be inserted to the box.



5.2. Creating a Function Block (FB) and Function (FC):

Function block (FB) is below the organization block in program hierarchy. Function block contains just a part of the main program that can be called in OB1. FB can be

added by clicking right mouse button on *Blocks* which can be found in Simatic Manager under *CPU 314C-2DP>S7 Program>Blocks*, and then *Insert New Object>Function Block*. The upper part of the appearing window, variable declaration window can be seen and there inputs, outputs and other variables can be defined. After programming FB, a data block (DB), which is assigned to a function block, has to be generated so that FB can be called by OB1.

When the control function does not have to store any of its own data, a function (FC) can be programmed. In contrast to a function block (FB), a function (FC) has no instance data block.


5.3. Downloading the Program:

The operating mode must be in STOP position, the red STOP led comes on.

The program can be downloaded by clicking in the *Simatic Manager window* under *PLC>Download* or on corresponding button  .

Under *View>Online* online status of the station can be observed. The offline window shows the situation on the programming device, the online window shows the situation on the CPU. The online and offline windows are indicated by the different colored headers.

After turning the operating mode to RUN, the green RUN led comes on and the red STOP led goes out. It means the CPU is ready for operation. If the red STOP led remains lit, an error has occurred.

All the automation process can be online observed by clicking on *Monitor*  button or under *Debug>Monitor* in the *OB1 window*. If any error occurs, monitoring is a good way to find out, where the process hangs. An example is given below:

For Input1 (I0.0):1, Input2 (I0.1): 0



For Input1 (I0.0):1, Input2 (I0.1): 1

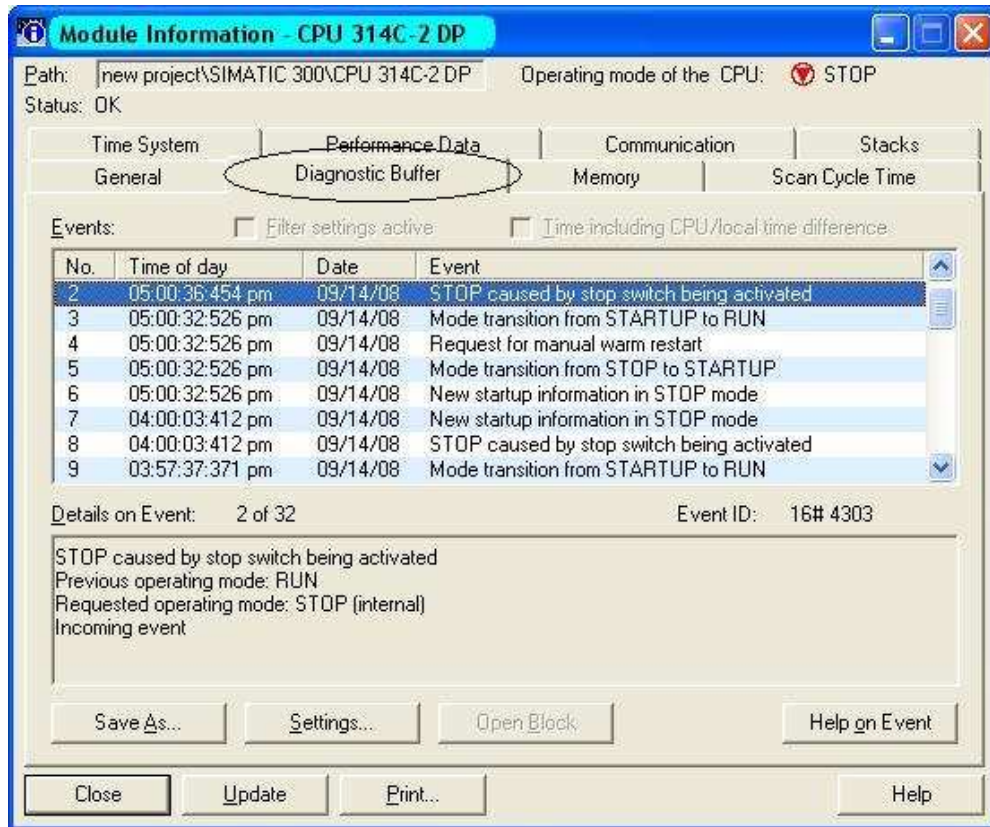


Figure 5.3 Diagnostic/Module Information

As it is shown in the Figure 5.3, if the CPU goes into STOP, while processing a program or if the CPU does not go into RUN mode after downloading the program, the reason of

the error can be determined in the diagnostic buffer. This can be found in the Simatic Manager under *PLC>Diagnostic/Setting>Module Information*.

6. S7-PLCSIM (SIMULATION PROGRAM)

Without additional hardware it is possible to test the user program with the S7-PLCSIM software. It simulates a PLC completely on the programming device and by means of it there is no need to be connected to any S7 hardware. S7-PLCSIM provides an interface to monitor how the input changes effect on the outputs, beside this bit memories, timer functions, counters can be monitored too.

S7-PLCSIM can be started by clicking on the simulation icon or by selecting the menu commands Options → Simulate Modules. After starting the application S7-PLCSIM user interface appears in a separate window. In the standard setting, CPU-subwindow appears in the main window, which shows the control switches and the LEDs of the CPU.

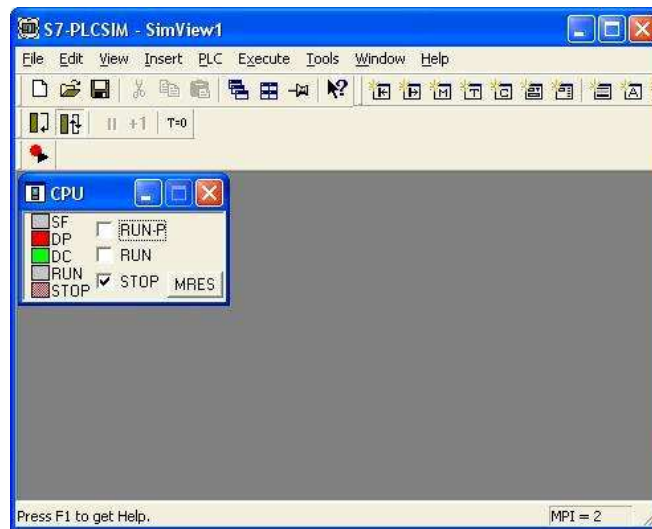


Figure 6.1 PLCSIM Simulation Program

6.1. CPU Operating Modes:

RUN mode: The CPU runs the program, by reading the inputs, executing the program, storing output values and updating the outputs. Downloading any program or changing any parameters is not allowed, when the CPU is in RUN mode.

RUN-P mode: The CPU runs the program and it is possible to change the program and its parameters, while the program is running.

STOP mode: The CPU does not work the program

MRES: Memory Reset Button

6.2. CPU Indicators:

SF: (System Fault) alerts that the CPU encountered a system error. These could be hardware faults, programming errors, timing errors, faulty memory card, battery fault, communication error etc.

DP: (Distributed Peripherals, or remote I/O) indicates the status of communication with distributed I/O.

DC: (Power Supply) indicates whether power to the CPU is on or off.

RUN: indicates that the CPU is in RUN mode.

STOP: indicates that the CPU is in STOP mode.

After the sub windows are inserted by clicking corresponding icons, the changes on outputs, timers, counters and bit memories can be monitored, as shown in the figure 6.2:

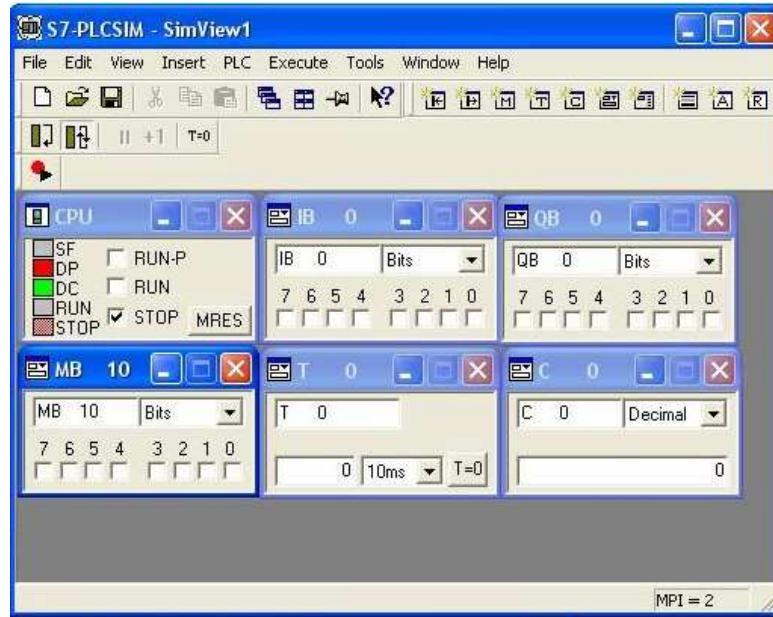


Figure 6.2 PLCSIM Window with Variables

Input Variable View Object:



→ Variable window for binary signals

→ Input-Addresses (e.g. I 1.x)

The CPU overwrites the I (Input) memory with the PI (Peripheral Input) memory at the beginning of every scan. If an I memory value is changed, the simulator immediately copies the changed value to the peripheral area. By this way, the desired change is not lost when the peripheral value overwrites the process input value on the next scan.

6.3. Object viewing:

Output Variable View Object: With this view object it can be monitored, how the input and other changes effect on the outputs.



Bit Memory View Object: This view object allows monitoring and modifying bit memory.

Counter View Object: Monitoring and modifying the counters used by the program is with this view object possible.


Timer View Object: It monitors the timers used by the program and displays the actual timer value and the time base.

6.4. Scan Mode Options:

S7-PLCSIM can be run in two different modes:

Single Scan: the CPU executes one scan and then CPU waits for next scan that is started by the user. Another scan can be started under *Execute>Next Scan* or by clicking corresponding item . To choose single scan mode, click  or under menu *Execute>Scan Mode>Single Scan*.

Continuous Scan: the CPU executes one scan and then starts another one. CPU reads the peripheral inputs, executes the program and writes the results to the peripheral outputs.

Continuous Scan can be chosen by clicking corresponding item  or under menu *Execute>Scan Mode>Continuous Scan*.

6.5. Opening and saving a simulated PLC:

When S7-PLCSIM is started, a new, untitled simulated PLC is opened. If you want to open an archived simulation, it can be found under *File>Open PLC*. Before you close a simulated PLC, the simulation program asks, whether it should be saved or not and it can be saved as *.PLC* or *.LAY* file.

.PLC File: This file contains information about the simulated PLC. If any changes are made in data, they will be saved in the .PLC file.

.LAY File: This file is used to save information about the physical layout of the simulated PLC. If you want the arrangement of the view objects in a certain order, save it as .LAY file and next time the view object will be displayed in that order.

7. FUTURE ASPECTS

So what brings the future? Over 20 years it has been predicted that PLC approaches its end as a control platform. Contrary to claims, PLCs sold \$6 billion worth around the world each year and sales are growing at over 5 percent each year, as Wardzel reported in Siemens User Conference in 2006. After all we can say that as long as PLCs continue to transform itself and satisfy the demands of the industrial automation, they will be still in need.

Since industrial automation requirements get complicated, some demands become more important than others. PLCs have to deal with following conditions for next years, if they continue to grow:

- Shorter Processing Time
- Strong Modularity
- More Intelligence
- Higher Quantity of DataStream
- Self-Configuration

7.1. Trends in automation systems

7.1.1. Visual Control

Image processing is used in many automation systems to bring more intelligence and simplicity. Instead of using several sensors, one camera can be integrated to the systems for detecting and checking machine parts or goods on a conveyor band.

7.1.2. Decentralisation

In the future more intelligence will be required in the automation systems. Longer processing time, higher quantities of DataStream are important impediments to bring the

intelligence in PLCs. Since every component, input and output are wired to a central controller; system suffers from the consumed time for the maintenance and unscheduled downtime. Main disadvantage of the centralized system is: if the central controller fails, rest of the system fails too.

Communications in the centralized systems are not very sufficient. The central controller sends signals or commands to each node and wait for a response. If two nodes needed to communicate with each other, it can be done through the central controller. Thus CPU will be fully engaged and reaction-time will be longer.

That's why decentralisation is the most mentioned trend in automation systems today. For instance, a sensor is equipped with a processor. Hence sensor will be no longer engaged with the central controller and processing time gets shorter. Also the actuators can be directly controlled by sensors. Having more controllers in the system allows us to build more complicated controlling strategies and processes. If one controller fails, rest of the system can still function. Decentralised controllers allow putting other plant parts into operation separately.

8. DIFFERENT EXAMPLES OF PLC APPLICATIONS

8.1. Pump Motor Control (Kopacek, 1993)

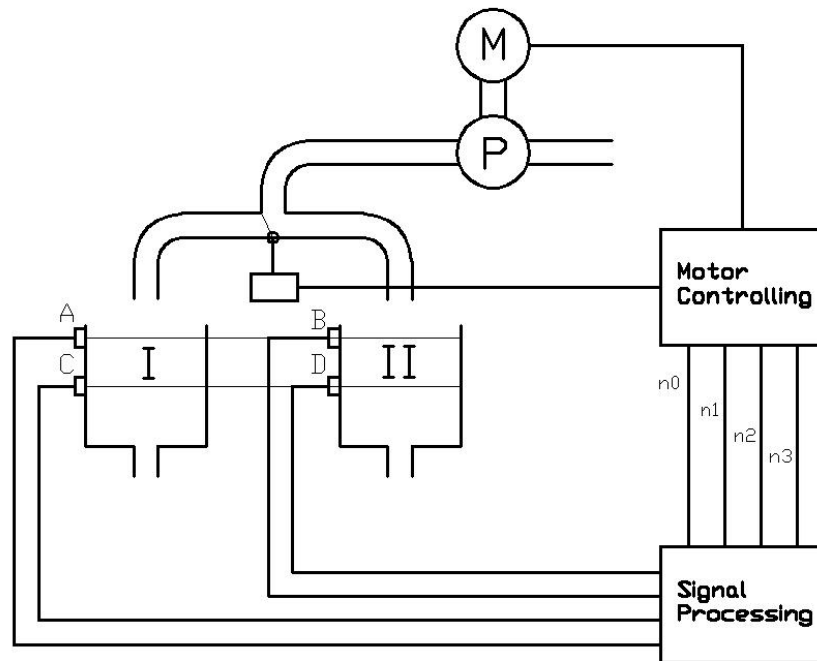


Figure 8.1 Plant Schema of Pump Motor Control

Two fluid containers will be filled by a pump. The pump motor can work with different speed values according to the fluid status of the containers. The conditions of these different speed values are described below:

If the containers are full: $n=0$

If one of the containers is full and the other one is more than half:

$$n=750 \text{ rpm}$$

If both of them are more than half or one of them more than half and the other one is less than half:

$$n=1450 \text{ rpm}$$

If both of them are less than half:

$$n=2850 \text{ rpm}$$

Changing the filling sequence of the containers is done by using two-way valve.

Inputs and outputs of the system:

I 0.1	Sensor A	The sensor at the top of the first container
I 0.2	Sensor B	The sensor at the top of the second container
I 0.3	Sensor C	The sensor in the middle of the first container
I 0.4	Sensor D	The sensor in the middle of the second container
Q 0.0	n=0	Motor does not work
Q 0.1	n=750	Motor's speed at 750 rpm
Q 0.2	n=1450	Motor's speed at 1450 rpm
Q 0.3	n=2850	Motor's speed at 2850 rpm
Q 0.4	Valve	for the value 1: filling the first container

Table 8.1 Input and Output Table of Pump Motor

The status of the sensor signals according to the container fluid level:

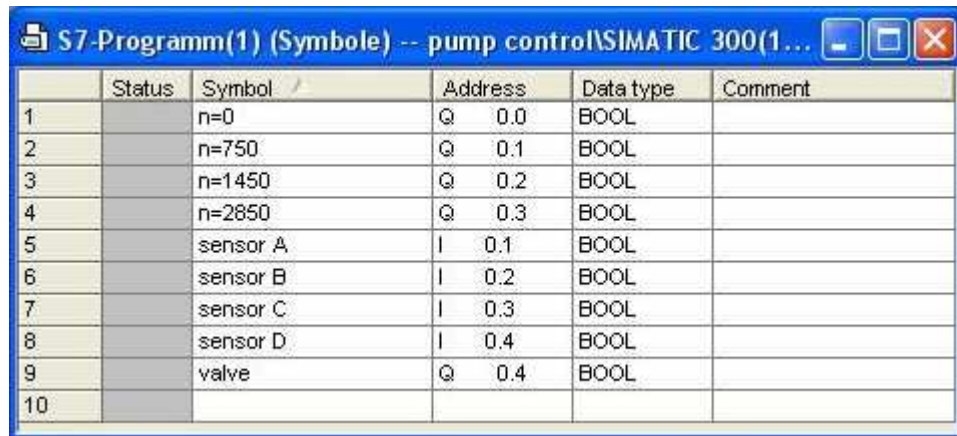
	First Container		Second Container	
	A	C	B	D
empty to half	1	1	1	1
half to full	1	0	1	0
full	0	0	0	0

Table 8.2 Sensor Signal Status of Pump Motor

The speed of the motor and status of the valve according to the sensor values:

A	B	C	D	n=0	n=750	n=1450	n=2850	valve
0	0	0	0	1	0	0	0	1
0	0	0	1	x	x	x	x	1
0	0	1	0	x	x	x	x	1
0	1	0	0	0	1	0	0	1
1	0	0	0	0	1	0	0	0
0	0	1	1	x	x	x	x	1
0	1	0	1	0	0	1	0	1
1	0	0	1	x	x	x	x	1
0	1	1	0	x	x	x	x	1
1	0	1	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0
0	1	1	1	x	x	x	x	1
1	0	1	1	0	0	1	0	1
1	1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	0	1
1	1	1	1	0	0	0	1	1

Table 8.3 Function Table of Pump Motor



	Status	Symbol	Address	Data type	Comment
1		n=0	Q 0.0	BOOL	
2		n=750	Q 0.1	BOOL	
3		n=1450	Q 0.2	BOOL	
4		n=2850	Q 0.3	BOOL	
5		sensor A	I 0.1	BOOL	
6		sensor B	I 0.2	BOOL	
7		sensor C	I 0.3	BOOL	
8		sensor D	I 0.4	BOOL	
9		valve	Q 0.4	BOOL	
10					

Table 8.4 Symbol Table of Pump Motor

After all the input and output variables are defined in the program by clicking the *symbols* icon, the networks can be created. The pump motor works with three different speed values (with the value $n=0$ four different values). Also each of these situations requires one network and one network for the status of the valve, totally five networks.

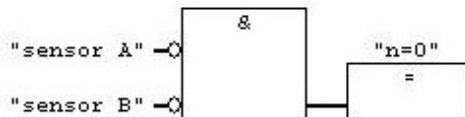
The networks are created in the organization block window by clicking new network.

The first network is for the situation that the motor does not work:

If the sensor values of *A* and *B* are “0”, this means that the containers are full, the motor does not work ($n=0$).

Network 1: Title:

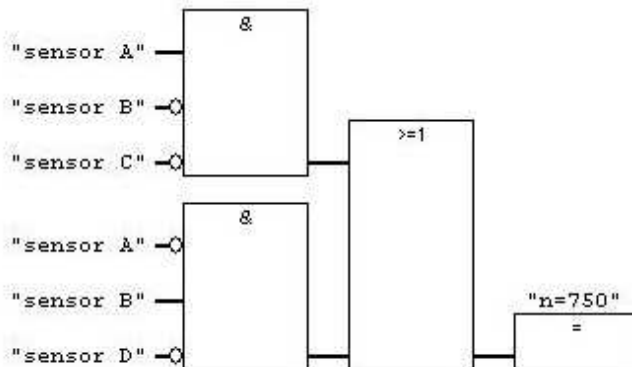
Comment:



The second network defines the situation when the motor works at 750 rpm:

Network 2: Title:

Comment:

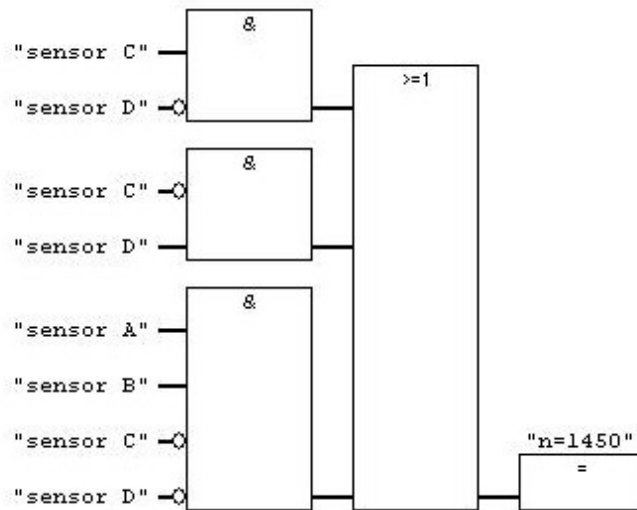


If one of the container’s fluid level is more than half (*A* or *B* is set to “1”), the motor works at 750 rpm.

The third network defines the situation that the motor works at 1450 rpm:

Network 3 : Title:

Comment:



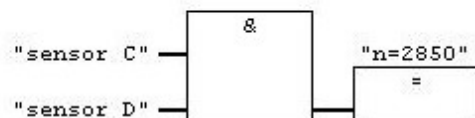
The motor works at 1450 rpm under these conditions:

- If one of the container's fluid level is less than half and the other one's more than half,
- If both container's fluid levels are more than half.

The fourth network defines the situation that the motor works at 2850 rpm:

Network 4 : Title:

Comment:

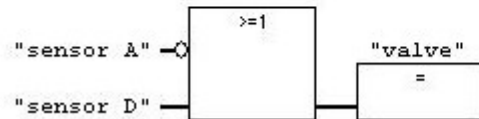


If both of the container's fluid levels are less than half, the motor works at 2850 rpm.

The last network is required to define the status of the two-way valve:

Network 5 : Title:

Comment:



For the value “1”: Filling the second container

For the value “0”: Filling the first container

Sensor D has here priority, as long as sensor D is set to “1”, the pump fills the second container. After the second container is half-filled (it means D is set to “0”), first container can be filled and then the second container can be fulfilled. If both of containers are full, valve status is set to “1”.

8.1.1. Simulation of control pump motor

If both containers are full, the program runs as shown in Figure 8.2:

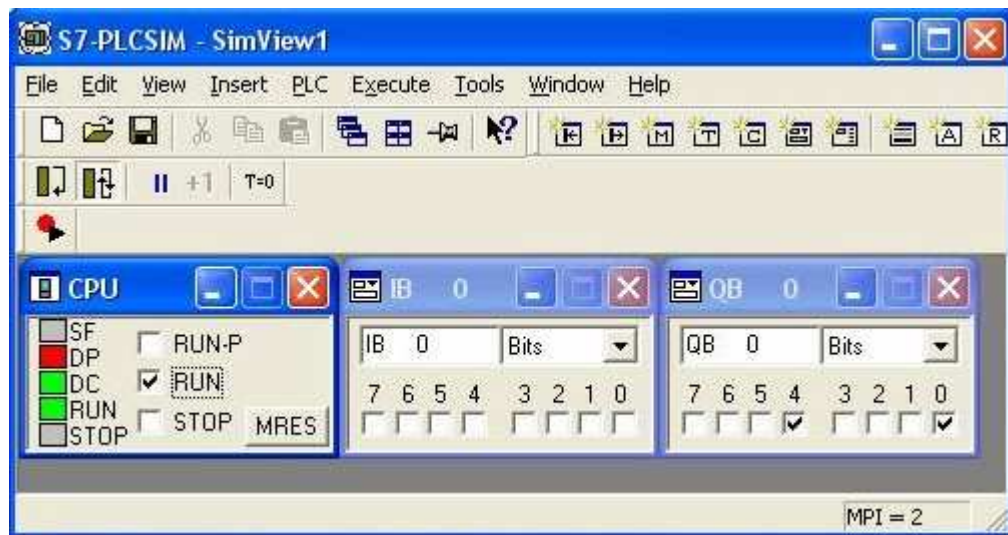


Figure 8.2 Simulation Pump Motor 1

All of the sensors are set to “0”, *valve* is set to “1” (*Q 0.4*) and *n=0* (*Q 0.0*) (Figure 8.3)

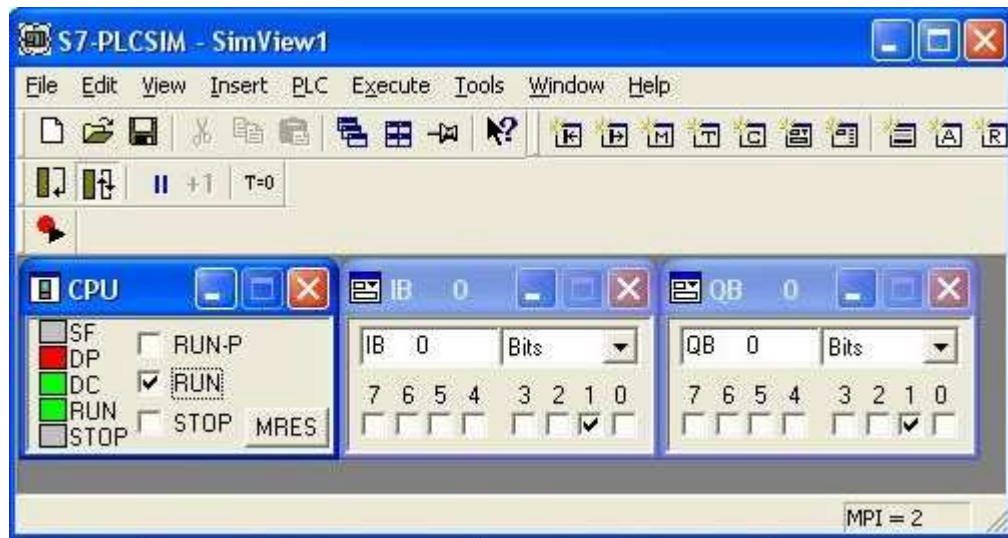


Figure 8.3 Simulation Pump Motor 2

If the first container’s fluid level is more than half and the other one full, motor works at 750rpm (*Q0.1*) and valve is set to “0” (it means first container is fulfilled.). (Figure 8.4)

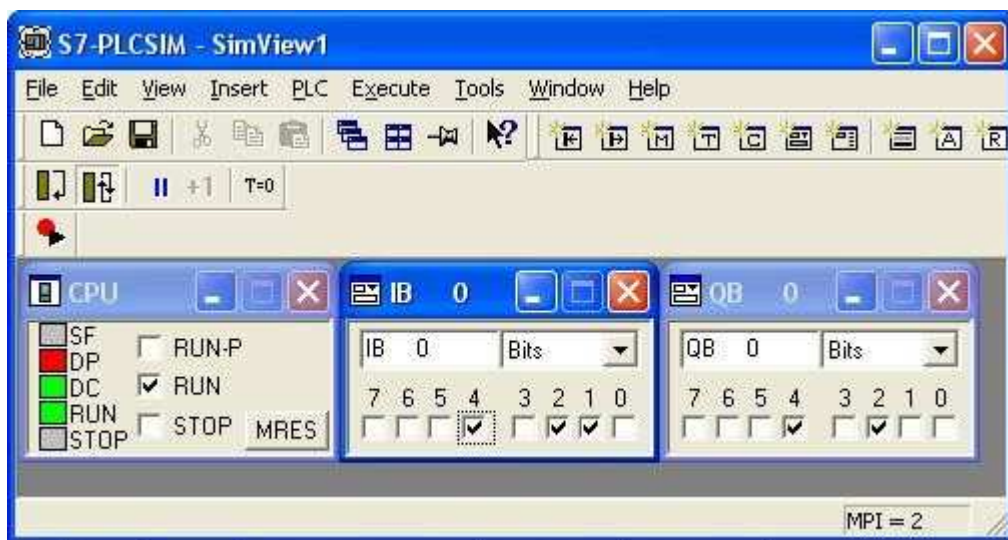


Figure 8.4 Simulation Pump Motor 3

Motor works at 1450rpm (*Q0.2*), if first container’s fluid level is more than half (*I0.1*) and the second one’s less than half (*I0.2* and *I0.4*) and valve is set to “1” (*Q0.4*). (Figure

8.5)

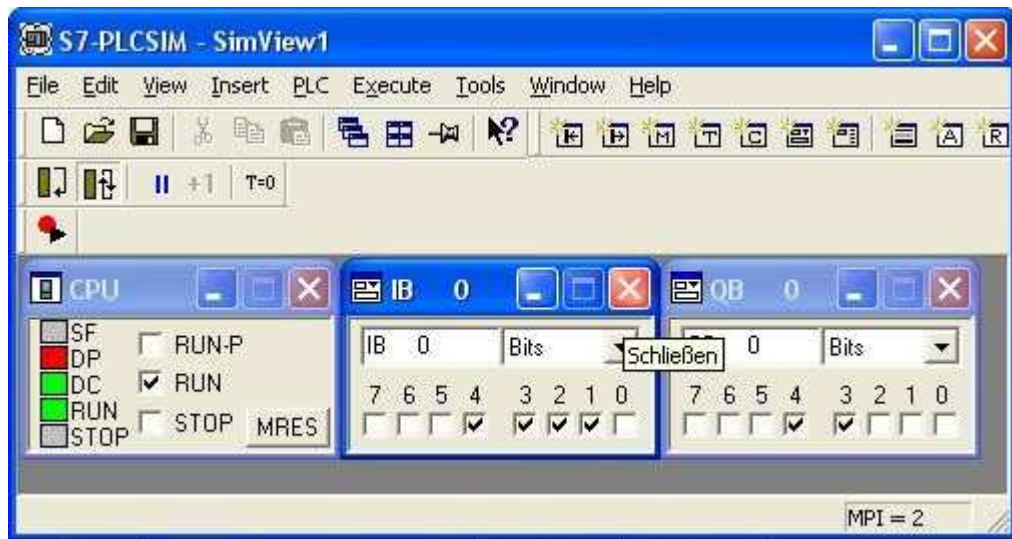


Figure 8.5 Simulation Pump Motor 4

If both container's fluid levels are less than half (all the input values are set to "1"), motor works at 2850rpm and valve is set to "1" (second container is filled.).

8.2. Conveyor Belt Controller (Kopacek, 2003)

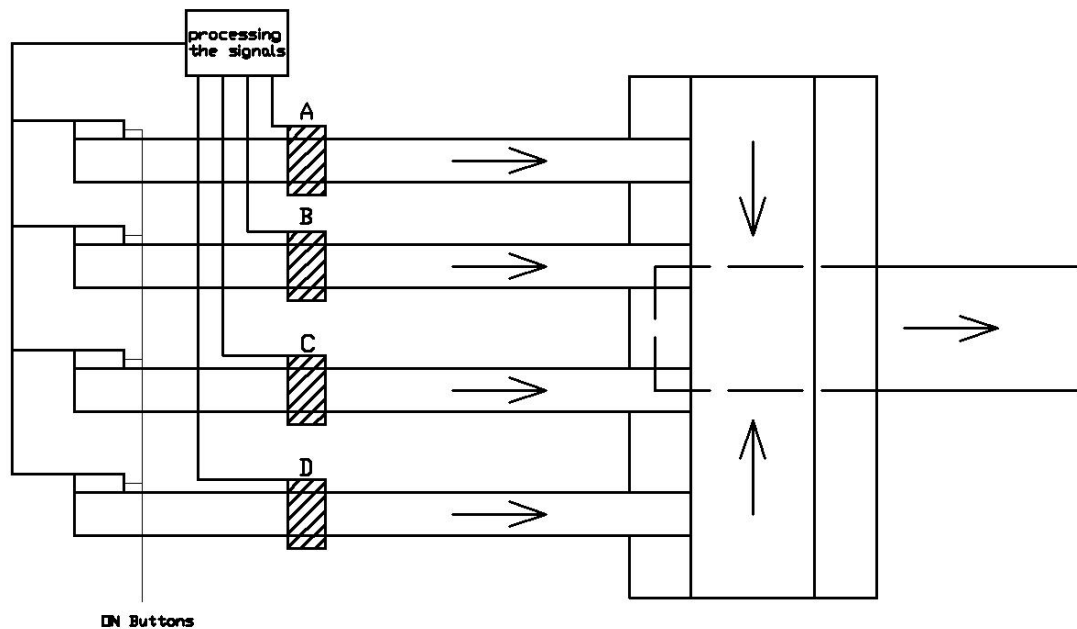


Figure 8.6 Conveyor Belt Controller

There are four conveyor belts available for transporting goods and they are all working with same speed and transport capacity. Only one or two conveyor belts should work at the same time, because more than two conveyor belts cause overload and the transporting process can be clogged. Turning on the third and fourth working conveyor belt has to be locked to avoid overloading.

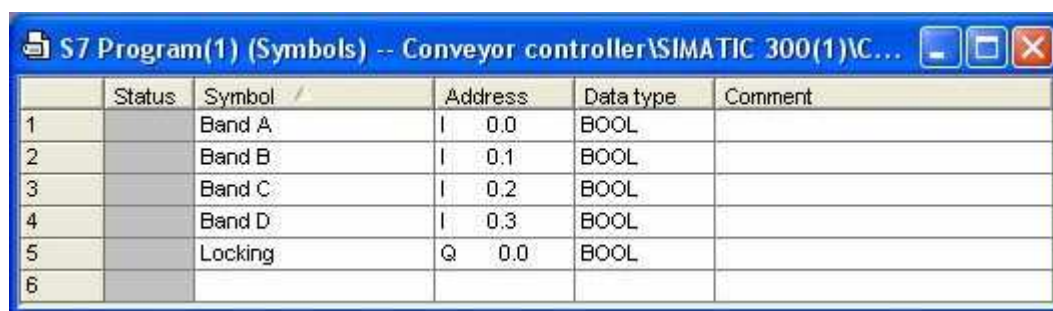
Inputs and outputs of the system:

I 0.0	Band A	First conveyor belt
I 0.1	Band B	Second conveyor belt
I 0.2	Band C	Third conveyor belt
I 0.3	Band D	Fourth conveyor belt
Q 0.0	Locking	To lock third and fourth working conveyor belts

Table 8.5 Input and Output Table of Conveyor Belt

Band A	Band B	Band C	Band D	Locking
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0
0	0	1	1	1
0	1	0	1	1
1	0	0	1	1
0	1	1	0	1
1	0	1	0	1
1	1	0	0	1
0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 8.6 Function Table of Conveyor Belt



	Status	Symbol	Address	Data type	Comment
1		Band A	I 0.0	BOOL	
2		Band B	I 0.1	BOOL	
3		Band C	I 0.2	BOOL	
4		Band D	I 0.3	BOOL	
5		Locking	Q 0.0	BOOL	
6					

Table 8.7 Symbol Table of Conveyor Belt

All of the inputs and outputs are defined in the symbol editor.

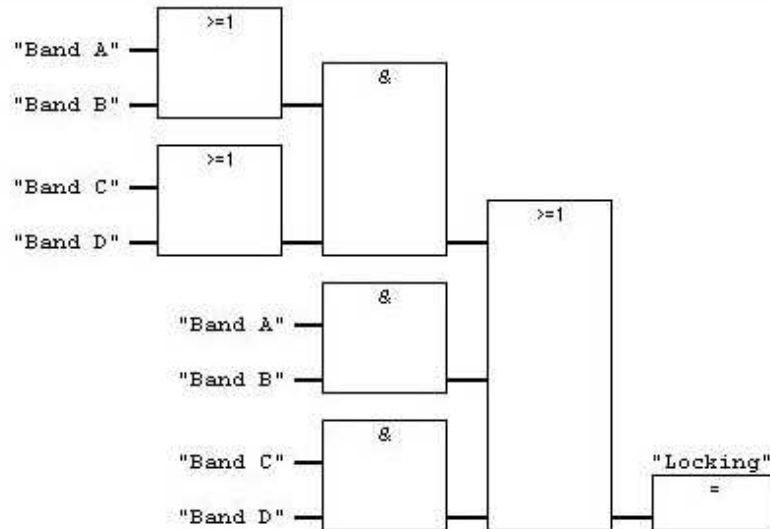
In this example we have just one output, which is used to lock turning on the conveyor belts. Thus one network is enough to define all of the conditions.

OB1 : "Main Program Sweep (Cycle)"

Comment:

Network 1: Title:

Comment:



After two conveyor belts are turned on, *locking* will be activated and turning on any other conveyor belt will not be possible. Thus overloading of feed belts is avoided.

8.2.1. Simulation of conveyor belt controller

As it is shown in Figure 8.7, Band B and Band D (Q0.1 and Q0.3) are working and locking (Q0.0) is activated to prevent turning on any other belt.

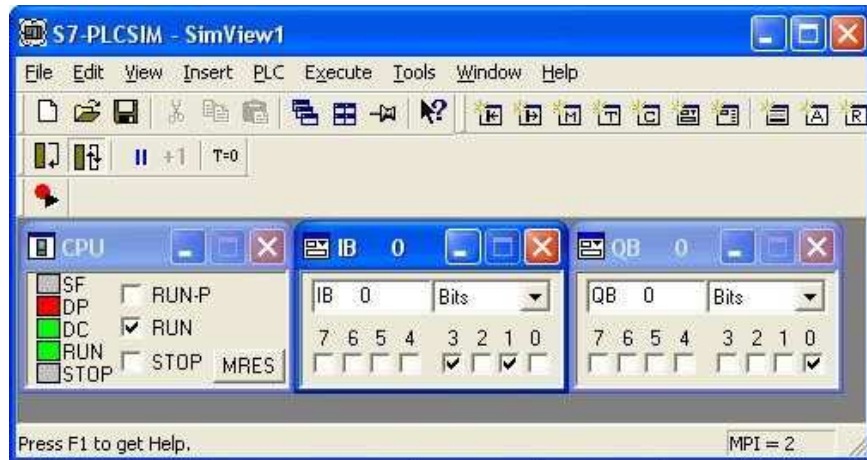


Figure 8.7 Simulation Conveyor Belt 1

In this situation Band A and Band B are working and locking (Q0.0) is activated.(Figure 8.8)

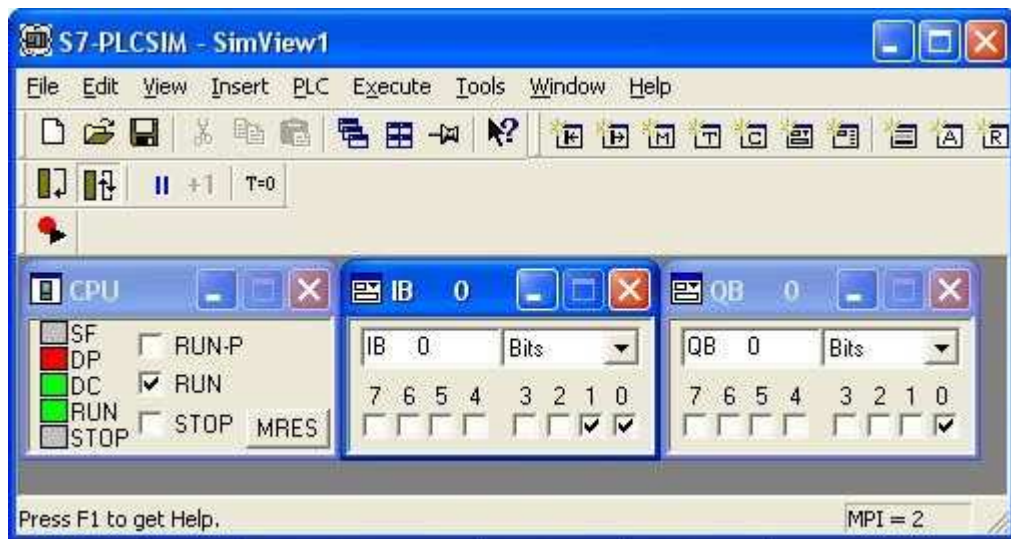


Figure 8.8 Simulation Conveyor Belt 2

8.3. Determining the Direction of the Rotating Shaft (Kopacek, 2003)

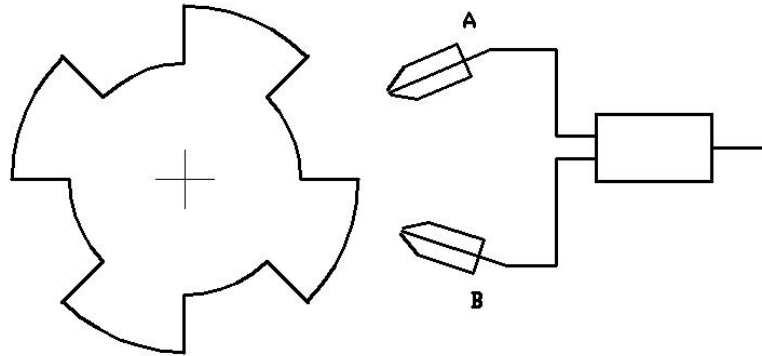


Figure 8.9 Rotating Shaft

The direction of the shaft can be determined by shaft-rotation due to the changes of the pneumatic sensor's signals. Within 45° rotation of the shaft the value of sensor A or B has to change "0" to "1" or "1" to "0". In order to observe the rotation of the shaft, a disc is mounted on the shaft and the sensors get the value "1" with the beginning of the rising edge of the disc.

Inputs and outputs of the system are shown in the table 8.8

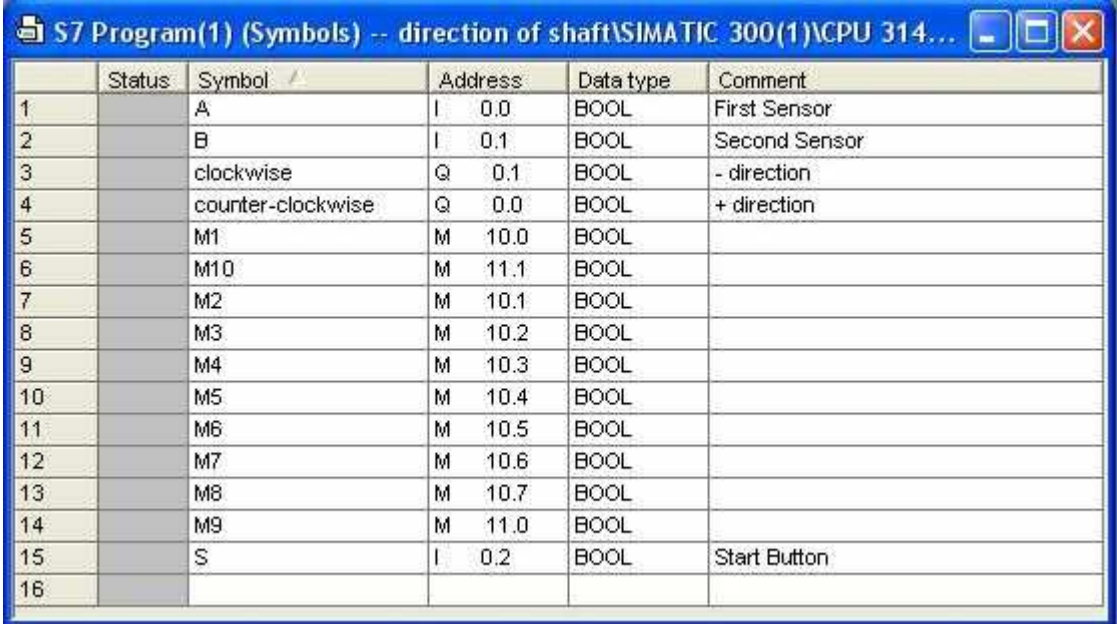
I 0.0	Sensor A	First Sensor
I 0.1	Sensor B	Second Sensor
Q 0.0	Clockwise	Shaft Rotates Clockwise
Q 0.1	Counter-clockwise	Shaft Rotates Counter-clockwise
S	ON-Button	Shaft Starts Rotating

Table 8.8 Input and Output Table of Rotating Shaft

Possible signal changes according to the rotation direction of the shaft:

AB 00 to 10
 10 to 11
 11 to 01
 01 to 00 ; these changes show that shaft rotates clockwise.

AB 00 to 10
 01 to 11
 11 to 10
 10 to 00 ; these changes show that shaft rotates counter-
 Clockwise.



	Status	Symbol	Address	Data type	Comment
1		A	I 0.0	BOOL	First Sensor
2		B	I 0.1	BOOL	Second Sensor
3		clockwise	Q 0.1	BOOL	- direction
4		counter-clockwise	Q 0.0	BOOL	+ direction
5		M1	M 10.0	BOOL	
6		M10	M 11.1	BOOL	
7		M2	M 10.1	BOOL	
8		M3	M 10.2	BOOL	
9		M4	M 10.3	BOOL	
10		M5	M 10.4	BOOL	
11		M6	M 10.5	BOOL	
12		M7	M 10.6	BOOL	
13		M8	M 10.7	BOOL	
14		M9	M 11.0	BOOL	
15		S	I 0.2	BOOL	Start Button
16					

Table 8.9 Symbol Table of Rotating Shaft

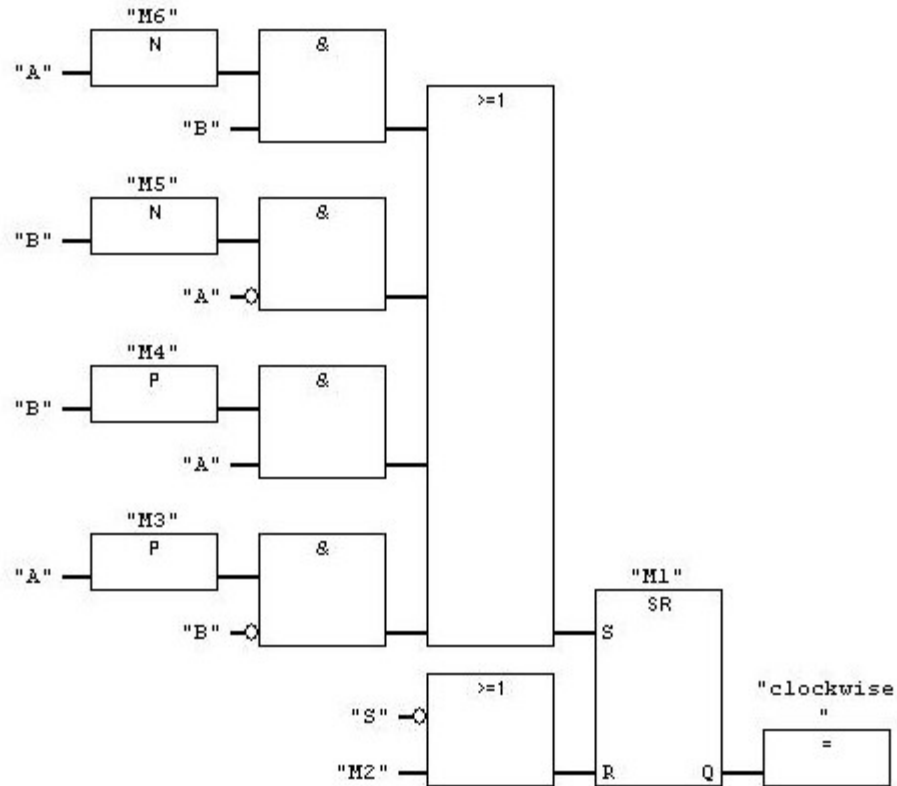
All inputs, outputs and memory variables are defined in the symbol editor.

The shaft can rotate clockwise or counter-clockwise. We have two outputs and for each output one network is required.

First network defines all situations that shaft rotates clockwise:

Network 1: - direction

Comment:

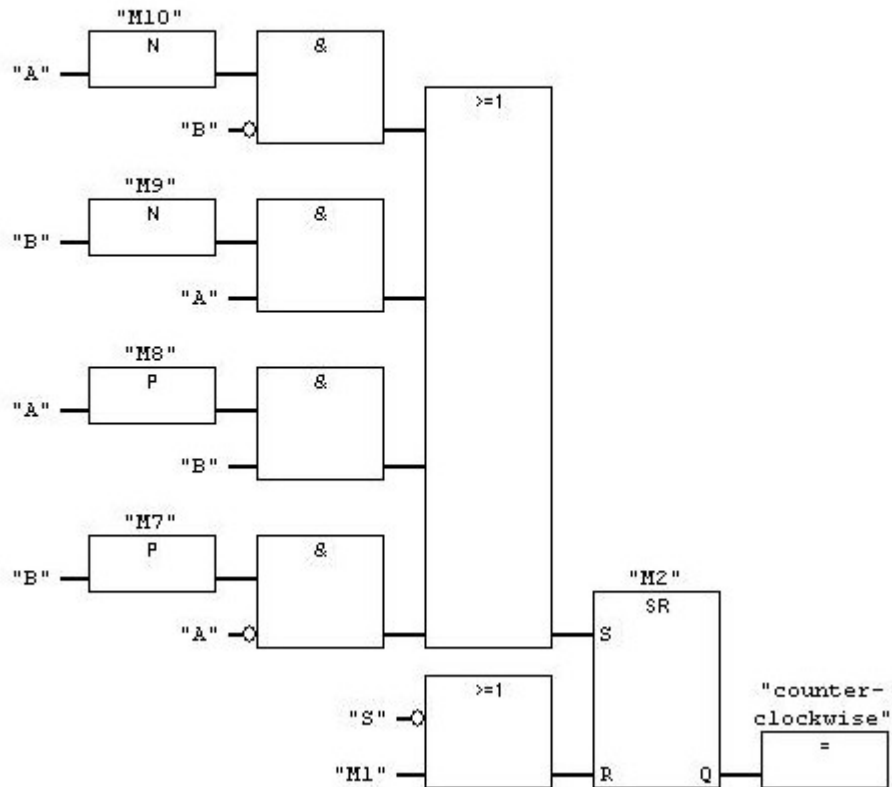


For instance, if sensor A is “1” and sensor B changes “0” to “1” (positive edge detection-*M4*) or sensor B is “1” and sensor A changes “1” to “0” (negative edge detection-*M6*), the output is set to “1” and shows that shaft rotates clockwise. If shaft stops rotating (*S* is set to “0”) or starts to rotate counter-clockwise (*M2* is set to “1”), the variables *S* or *M2* will reset *M1*.

Second network is required for all situations so that shaft rotates counter-clockwise:

Network 2 : Title:

Comment:



If sensor A is set to “0” and sensor B changes “0” to “1” (positive edge detection-*M7*) or sensor A is set to “1” and sensor B changes “1” to “0” (negative edge detection-*M9*), output is set to “1” and shows that shaft rotates counter-clockwise. If *S* is set to “0” or *M1* to “1”, *M2* will be reset.

8.3.1. Simulation of the Rotating Shaft

Sensor A (*I 0.0*) and *B* (*I 0.1*) are set to “0” and *S* is set to “1”. No output is activated yet. (Figure 8.10)

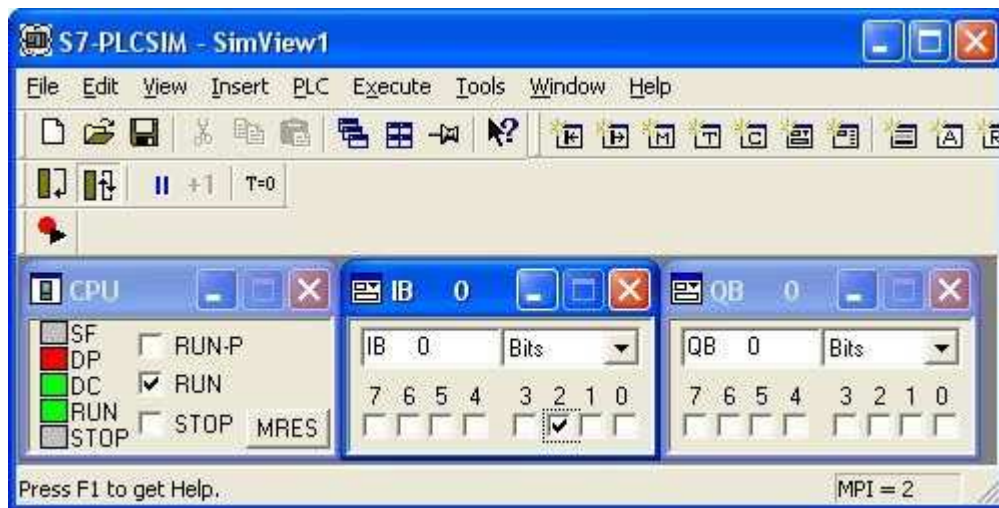


Figure 8.10 Simulation Rotating Shaft 1

Then the *sensor B*'s (*I 0.1*) signal value changes “0” to “1” (positive edge detection) and the output *counter-clockwise* (*Q0.0*) is automatically set to “1”. (Figure 8.11)

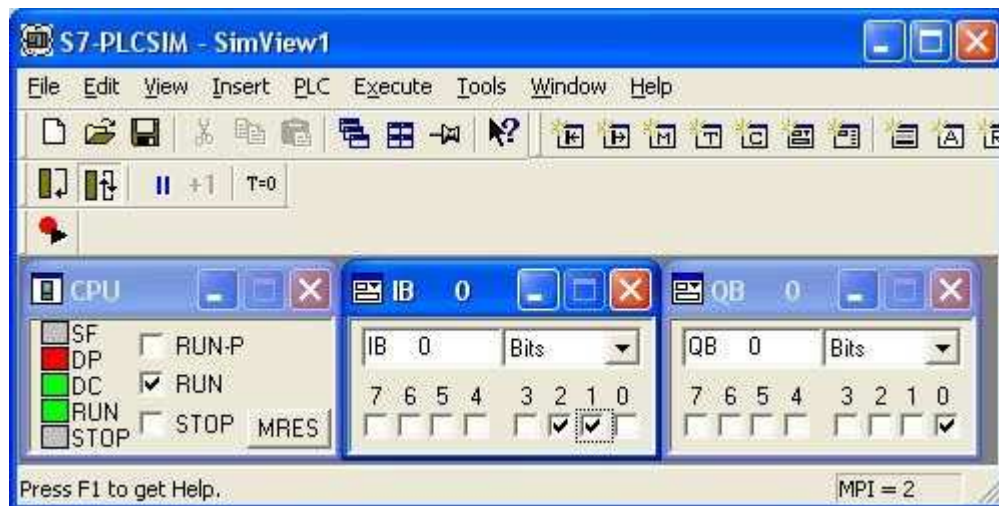


Figure 8.11 Simulation Rotating Shaft 2

S ($I\ 0.2$), sensor A ($I\ 0.0$) and B ($I\ 0.1$) are set to “1”, no output is activated yet. (Figure 8.12)

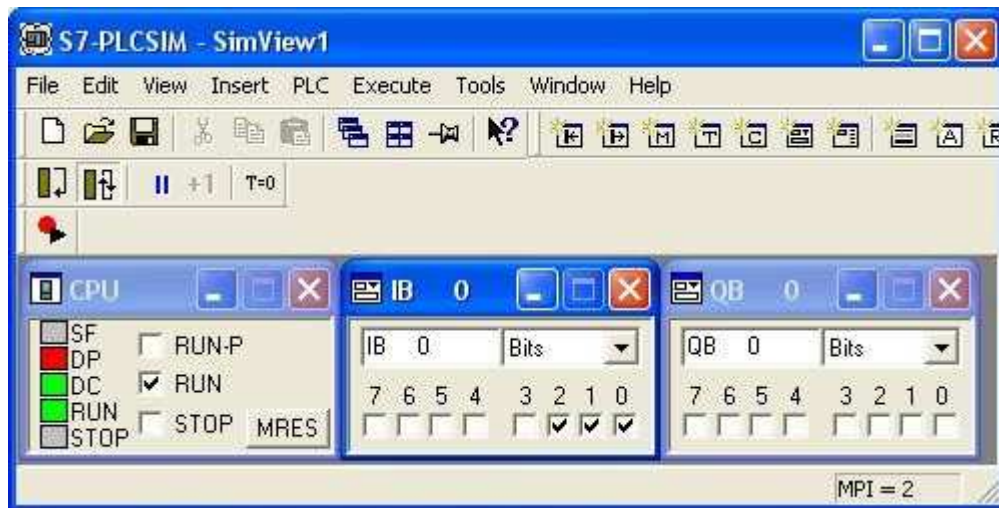


Figure 8.12 Simulation Rotating Shaft 3

Then sensor A 's ($I\ 0.0$) signal value changes “1” to “0” (negative edge detection) and the output *clockwise* ($Q\ 0.1$) is automatically set to “1”. Output keeps this signal value, until shaft stops rotating or starts to rotate counter-clockwise.



Figure 8.13 Simulation Rotating Shaft 4

8.4. Sorting and Packing of Pipes (Wellenreuther and Zastrow, 2002)

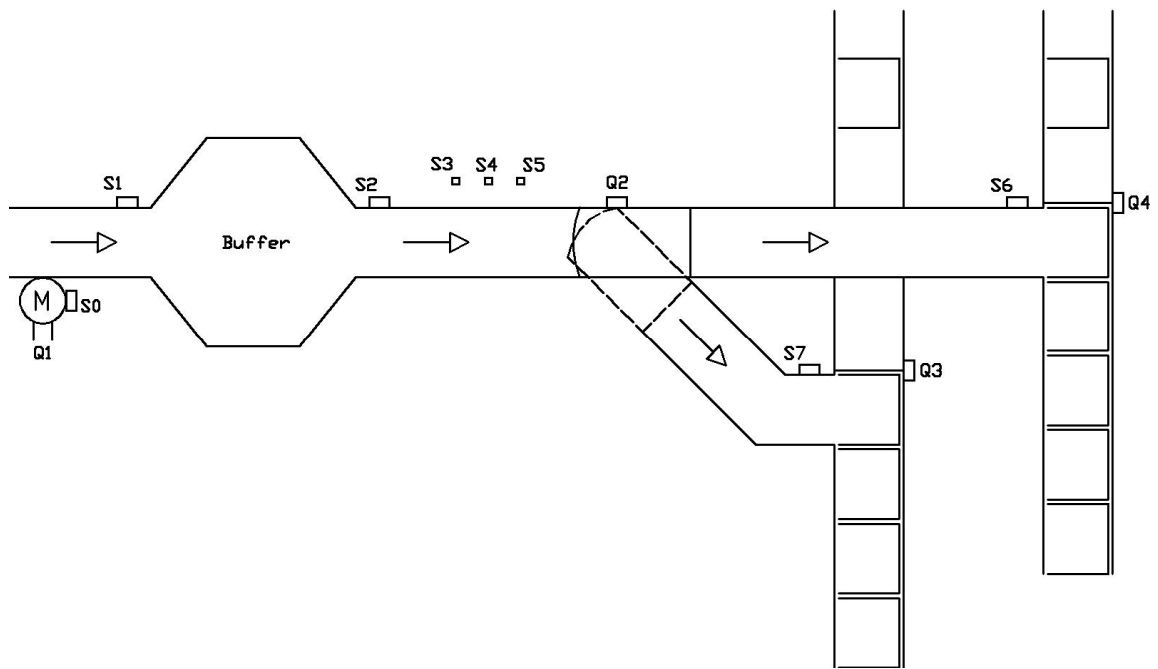


Figure 8.14 Sorting and Packing of Pipes

Pipes are carried by a conveyor belt and are stored in the buffer in order to keep the machine working, if any maintenance or repairment is required. The capacity of the buffer is 200 pipes and if the number of pipes reaches 200, the first conveyor belt is turned off, which delivers the pipes into the buffer. If the number of the pipes in the buffer is less than 30, the warning lamp is turned on. There are two types of pipe, which are to differ by their lengths. If the longer one comes, all three sensors $S3$, $S4$, $S5$ are set to “1” at the same time. For the longer pipes $Q2$ is set to “1” and the conveyor belt runs forward, for the short pipes $Q2$ is set to “0” and pipes will be transported through the right conveyor belt. After the pipes are sorted according to their lengths, they will be packed. Each package should contain 10 pipes for the longer pipes and 15 pipes for the short pipes.

Inputs and outputs of the system are shown in the table 8.10

I	0.0	S0	motor sensor
I	0.1	S1	buffer sensor-input
I	0.2	S2	buffer sensor-output
I	0.3	S3	length determining sensor
I	0.4	S4	length determining sensor
I	0.5	S5	length determining sensor
I	0.6	S6	counter for the long pipes
I	0.7	S7	counter for the short pipes
I	1.0	S8	long package blocking
I	1.1	S9	short package blocking
Q	0.0	Q1	motor ON/OFF
Q	0.1	Q2	conveyor belt changing
Q	0.2	Q3	long package release
Q	0.3	Q4	short package release
Q	0.4	Warning lamp	warning lamp

Table 8.10 Input and Output Table of Pipe Packing

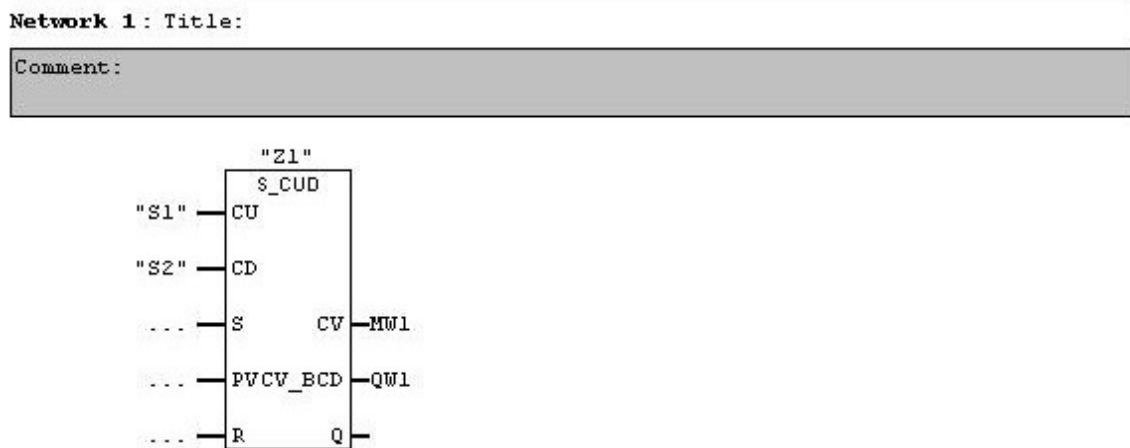
All of the inputs, outputs and counter variables are defined in the symbol editor as shown below in the table 8.11:

	Status	Symbol	Address	Data type	Comment
1		Q1	Q 0.0	BOOL	
2		Q2	Q 0.1	BOOL	
3		Q3	Q 0.2	BOOL	
4		Q4	Q 0.3	BOOL	
5		warning lamp	Q 0.4	BOOL	
6		S0	I 0.0	BOOL	
7		S1	I 0.1	BOOL	
8		S2	I 0.2	BOOL	
9		S3	I 0.3	BOOL	
10		S4	I 0.4	BOOL	
11		S5	I 0.5	BOOL	
12		S6	I 0.6	BOOL	
13		S7	I 0.7	BOOL	
14		S8	I 1.0	BOOL	
15		S9	I 1.1	BOOL	
16		Z1	C 1	COUNTER	
17		Z2	C 2	COUNTER	
18		Z3	C 3	COUNTER	
19					

Table 8.11 Symbol Table of Pipe Packing

In this example we need totally eight networks to define the process.

First network defines the counter-function that counts how many pipes are in the buffer:



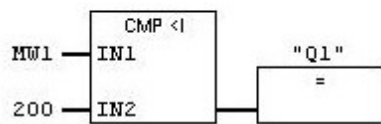
By means of counter-function, we can control all the time how many pipes are stored in

the buffer. As a result of incoming pipes, sensor S1 values changes “0” to “1” and with every change of sensor S1 value, Z1 will be counted up by 1 and Z1 will be counted down according to the change of sensor S2 value (outcoming pipes).

Second network defines when the conveyor belt will be stopped in order to prevent exceeding the capacity of the buffer:

Network 2 : Title:

Comment:



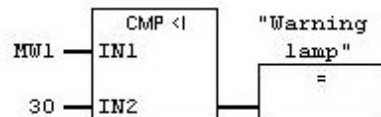
This function compares two input values and if MW1 is smaller than 200, *Q1* is set to “1”. It means if 200 pipes are in the buffer, the first conveyor belt (*Q1* is set to “0”) is turned off.

Third network is required to define when the warning lamp will be turned on:

If the number of pipes in the buffer is smaller than 30, warning lamp will be turned on and shows that few pipes are left in the buffer.

Network 3 : Title:

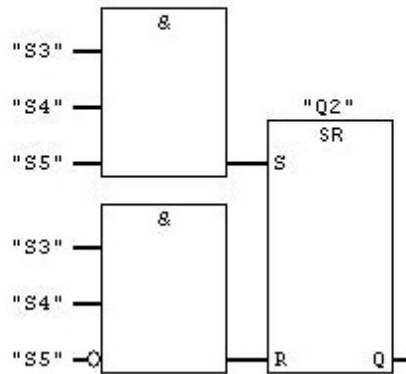
Comment:



Fourth network defines that with which conveyor belt the pipes should be transported according to their lengths:

Network 4 : Title:

Comment:

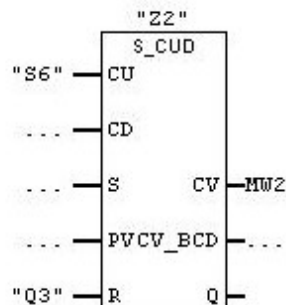


There are two length types of pipes and if the longer one comes all three sensors $S3$, $S4$ and $S5$ are set to “1” and conveyor belt transports the goods straight forward. But if the short pipe comes, then just $S3$ and $S4$ will be set to “1” and $S5$ will be set to “0”. As a result $Q2$ is set to “0” and pipes will be transported by the right conveyor belt.

Fifth network defines the counter-function which is needed to count the incoming long pipes:

Network 5 : Title:

Comment:

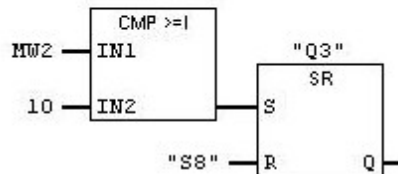


With every incoming long pipe, sensor $S6$ value changes “0” to “1” and each change is counted up by 1. The number of the long pipes in the package can be read by the counter function $Z2$ which is reset by $Q3$.

Sixth network defines when the package for long pipes is fulfilled and should be transported:

Network 6 : Title:

Comment:

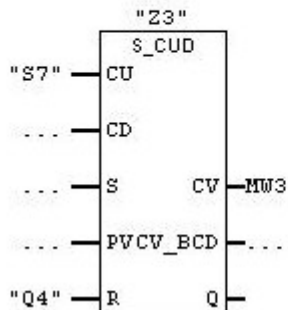


As long as the number of long pipes in the package is smaller than 10, $Q3$ is set to “0”. If totally ten pipes are in the package, the package blocker release the package and waits for another one.

Seventh network defines the counter-function which is needed to count the incoming short pipes:

Network 7 : Title:

Comment:



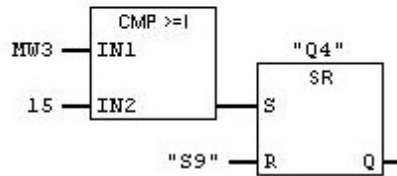
With every incoming short pipe, sensor $S7$ value changes “1” to “0” and each change is counted up by 1 and the number of the short pipes in the package can be read by $Z3$. $Z3$ is reset by $Q4$.

Eighth network defines when the package for short pipes is fulfilled and should be

transported:

Network 8 : Title:

Comment:



As long as the number of long pipes in the package is smaller than 15, $Q4$ is set to “0”. If totally fifteen pipes are in the package, the package blocker releases the package and waits for another one.

8.4.1. Simulation of Pipe Sorting and Packing

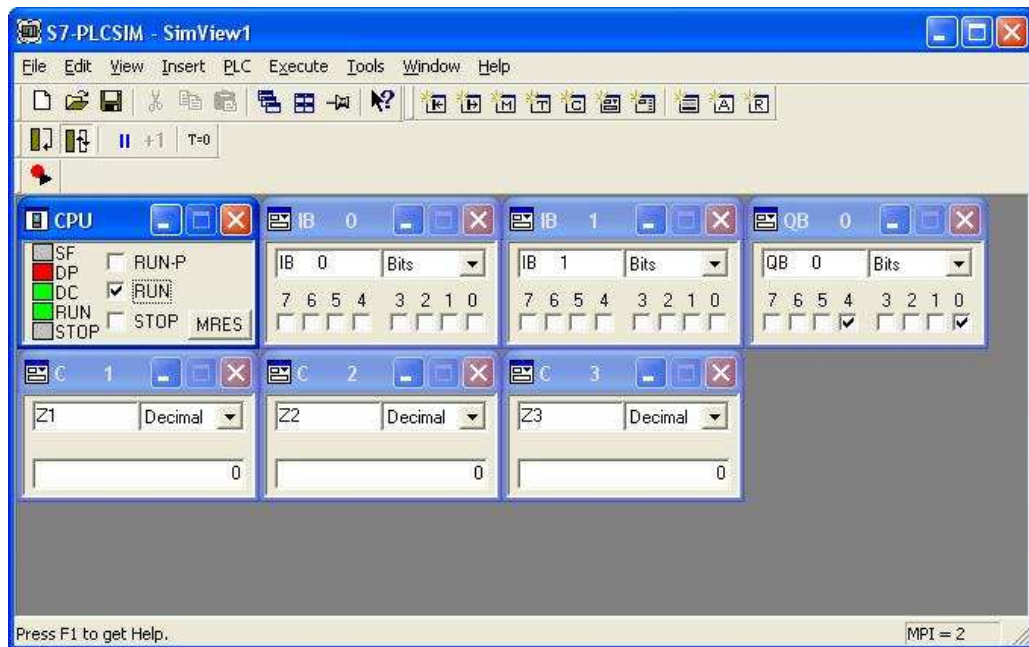


Figure 8.15 Simulation Pipe Packing 1

As we start the simulation, $Q\ 0.0$ (motor) and $Q\ 0.1$ (warning lamp) are set to “1”. If $Q\ 0.0$ is set to “1”, it means that the conveyor belt runs, which transports the pipes into the buffer. $Q\ 0.1$ is set to “1”, because the number of the pipes in the buffer is less than 30. (Figure 7.15)

The number of the pipes in the buffer can be read from the counter function $C1$. As it is shown above, if there are 200 ($C1=200$) pipes in the buffer, then motor will be stopped ($Q\ 0.0$ is “0”), which runs the conveyor belt, and remains so until minimum one pipe leaves the buffer. (Figure 8.16)

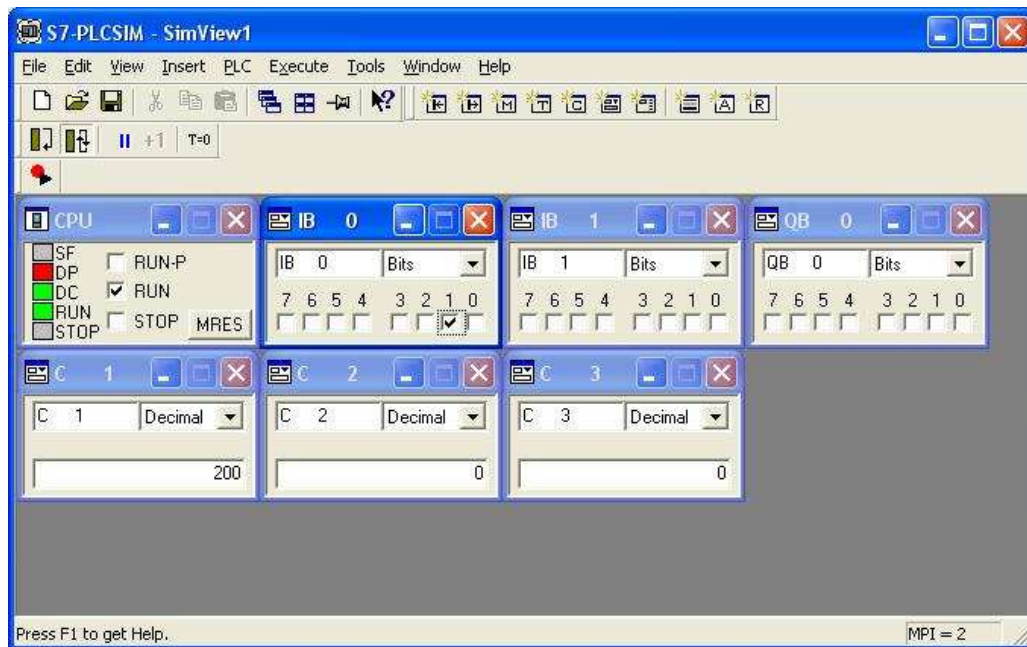


Figure 8.16 Simulation Pipe Packing 2

If the incoming pipe is a longer one, then all the sensors $S3$, $S4$ and $S5$ are set to “1” and $Q\ 0.1$ is set to “1”, which makes conveyor belt transport the pipe straight forward. (Figure 8.17)

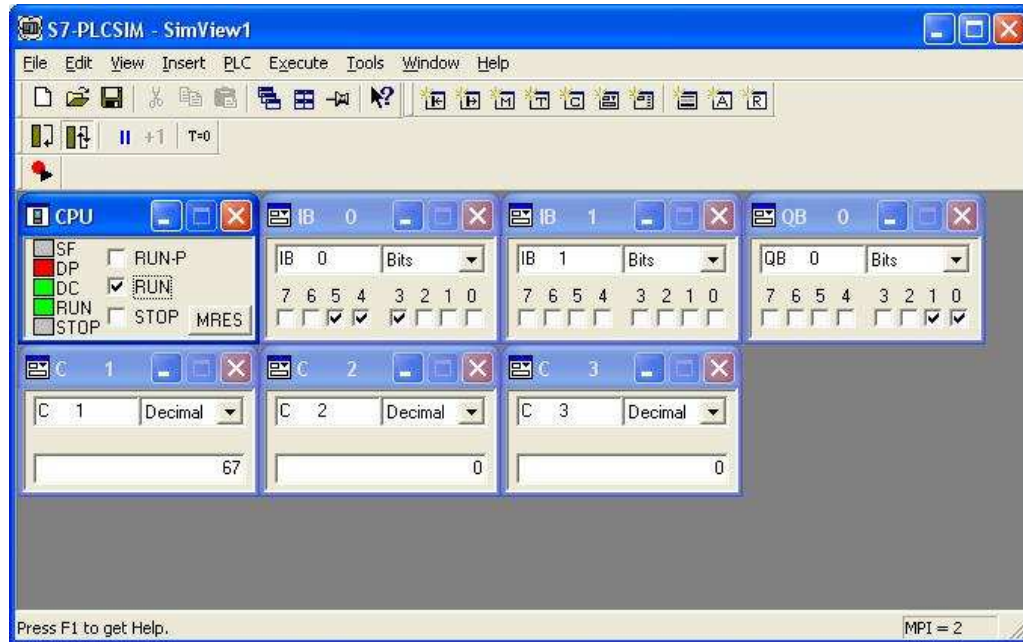


Figure 8.17 Simulation Pipe Packing 3

If the packages of the long and short pipes are full, they will be released by activating the outputs *Q 0.2* (*long package release*) and *Q 0.3* (*short package release*). (Figure 8.18)

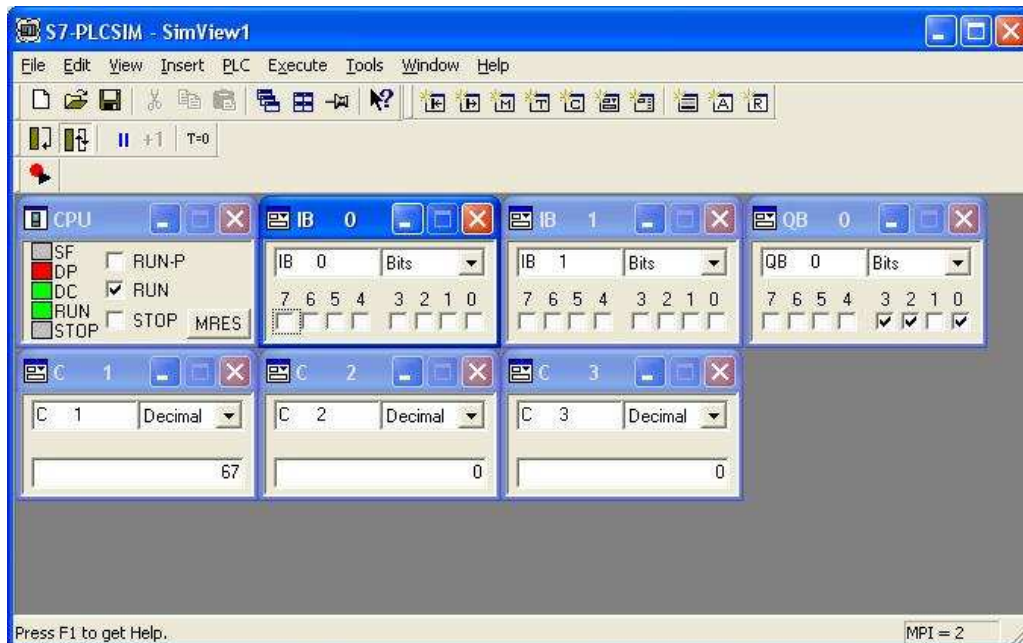


Figure 8.18 Simulation Pipe Packing 4

8.5. Garage Door Controller

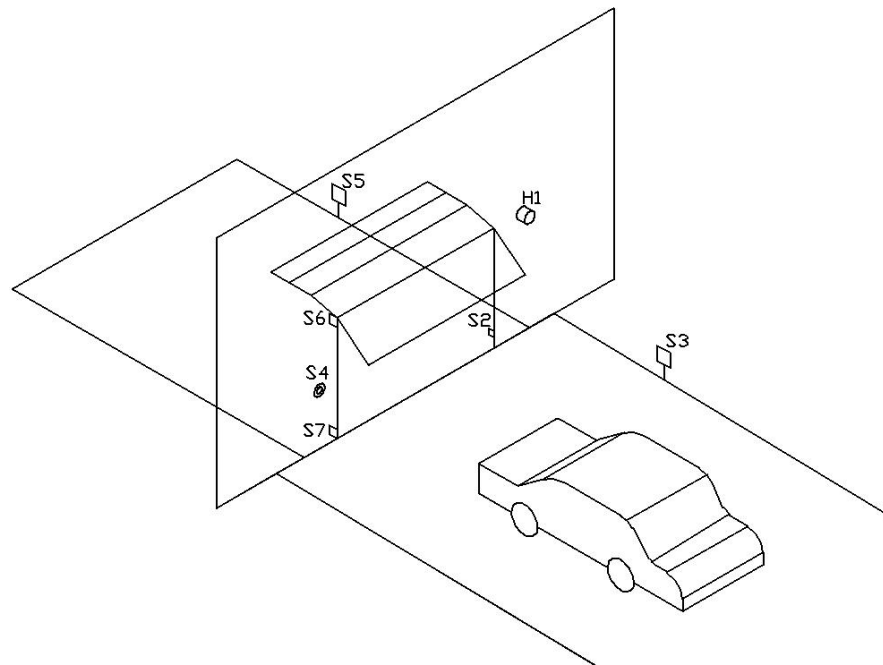


Figure 8.19 Garage Door Controlling

The garage door is opened by activating the sensor $S0$ or the door is opened automatically (e.g. On/Off switch or remote switch), if a car which is close enough to the garage door, is noticed by the sensor $S3$. If there is already a car in the garage and the sensor $S3$ or $S0$ is activated by another car then the warning lamp $H1$ is turned on for five seconds and shows that the garage is already full (the sensor $S5$ controls whether the garage is full or not). The garage door is closed by activating the sensor $S1$ but if any obstacle which prevents closing the garage door is noticed by the sensor $S2$ and stops the operation immediately. If the door remains opened, then it closes itself automatically after two minutes. By activating the sensor $S4$ all the operations can be stopped (Emergency Stop). If the door is opened completely, the sensor $S6$ stops opening the door just as the sensor $S7$ stops closing the door.

Firstly we need to define all the bit memories, timers, inputs and outputs in the symbol editor:

	Status	Symbol	Address	Data type	Comment
1		Cycle Execution	OB 1	OB 1	
2		H1-warning lamp	M 10.7	BOOL	warning lamp
3		M1	M 10.1	BOOL	
4		M2	M 10.2	BOOL	
5		O1-opening the door	Q 4.1	BOOL	opening the door
6		O2-closing the door	Q 4.2	BOOL	closing the door
7		S0-ON switch	I 0.0	BOOL	the door ON switch
8		S1-OFF switch	I 0.1	BOOL	the door OFF switch
9		S2-obstacle control	I 0.2	BOOL	obstacle control
10		S3-automatically ope...	I 0.3	BOOL	automatically opening the door
11		S4-emergency stop	I 0.4	BOOL	emergency stop
12		S5-garage full or not	I 0.5	BOOL	garage full or not
13		S6-the door is opened	I 0.6	BOOL	the door is completely opened
14		S7-the door is closed	I 0.7	BOOL	the door is completely closed
15		T1-timer closing door	T 1	TIMER	after 2 minutes close the door automatically
16		T2-timer warning lamp	T 2	TIMER	warning lamp for 5 seconds
17					

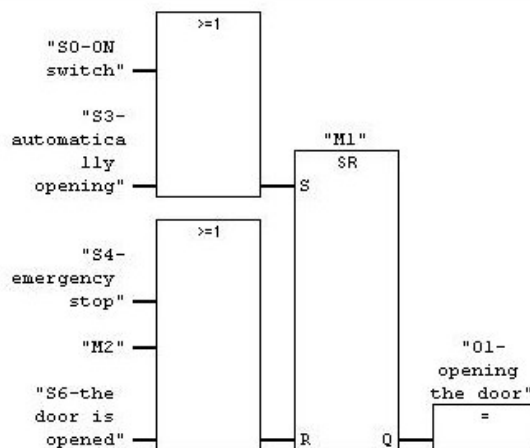
Table 8.12 Symbol table of Garage Door Controller

For programming the garage door we need four networks. The networks can be inserted by mouse right-clicking and then clicking on *Insert Network*.

The first network is for opening the garage door:

Network 1: opening the door

Comment:



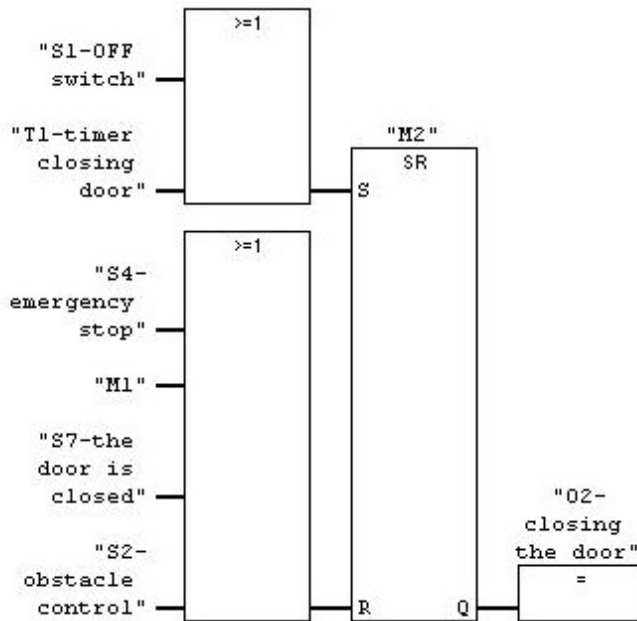
The conditions for *opening the door* are to activate *S0* or *S3* sensors. On the other hand

the values of $S4$ (emergency stop), $M2$ (it means OFF- and ON-Switch can not be activated simultaneously) and $S6$ (door is opened) have to be “0”. If one of these values is 1, it resets $M1$ and $O2$ (opening the door) remains “0”.

The second network for closing the garage door:

Network 2 : closing the door

Comment:



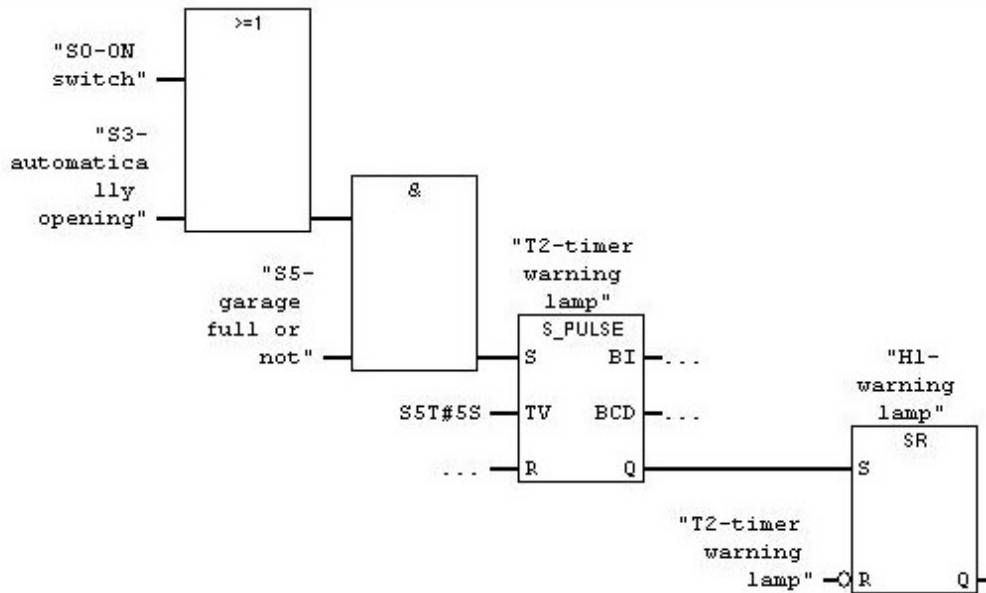
The conditions for *closing the door* are to activate $S1$ (Off-switch) or waiting for 2 minutes that the $T1$ (timer closing door) closes the door automatically. The values of $S4$, $M1$, $S7$ and $S2$ (obstacle control) have to be “0”, otherwise they reset $M2$ and $O2$ (closing the door) remains “0”.

The third network for warning lamp:

If there is already a car in the garage ($S5$ is “1”) and $S0$ or $S3$ is activated, then warning lamp will be *ON* for five seconds. But once the timer $T2$ is “0” (it means $S0$, $S3$ and $S5$ are “0”), then the warning lamp will be immediately *OFF*.

Network 3 : warning lamp

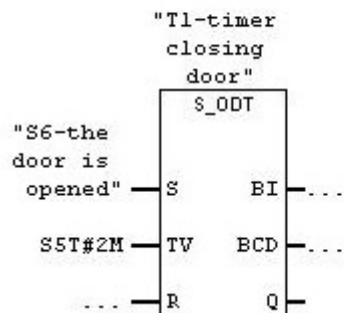
Comment:



The fourth network for automatically closing the door after 2 minutes:

Network 4 : after 2 minutes close the door automatically

Comment:



The garage door closes itself, if it remains opened for two minutes. By this way the user does not have to close the door after he left the garage.

8.5.1. Simulation of Garage Door Controller

S0 is activated and as a result *O1* and *M1* are set to “1”. After the door is completely opened *S6* is set to “1”. (Figure 8.20)

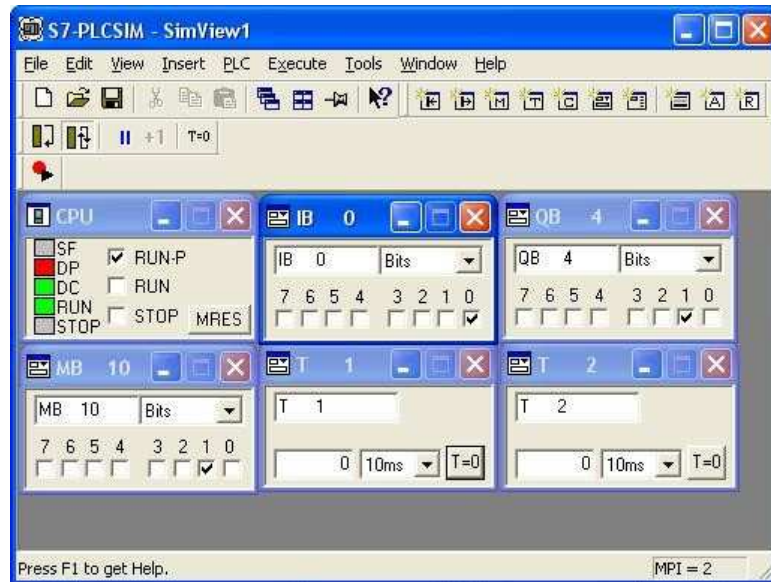


Figure 8.20 Simulation Garage Door 1

After a change from 0 to 1 at the *S6*, *O1* and *M1* are set to “0” and the timer T1 is set to 120 seconds. (Figure 8.21)

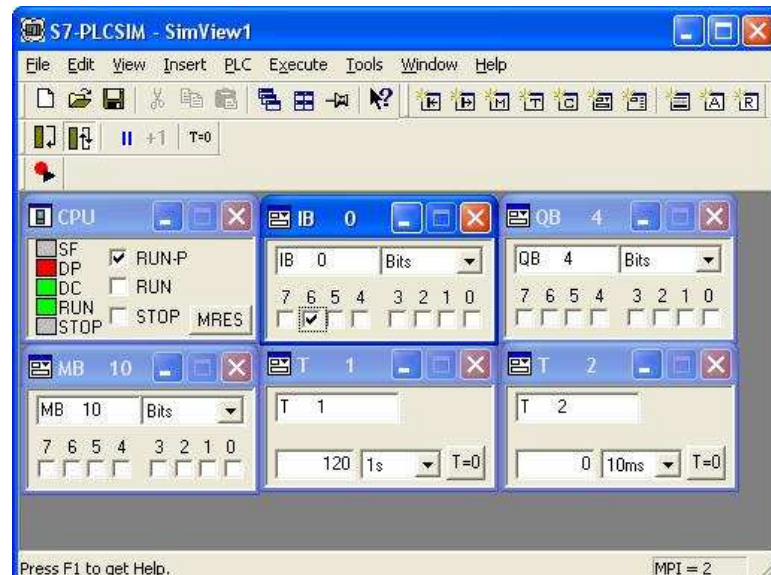


Figure 8.21 Simulation Garage Door 2

After 2 minutes elapsed, *O2* and *M2* are set to “1”. (Figure 8.22)

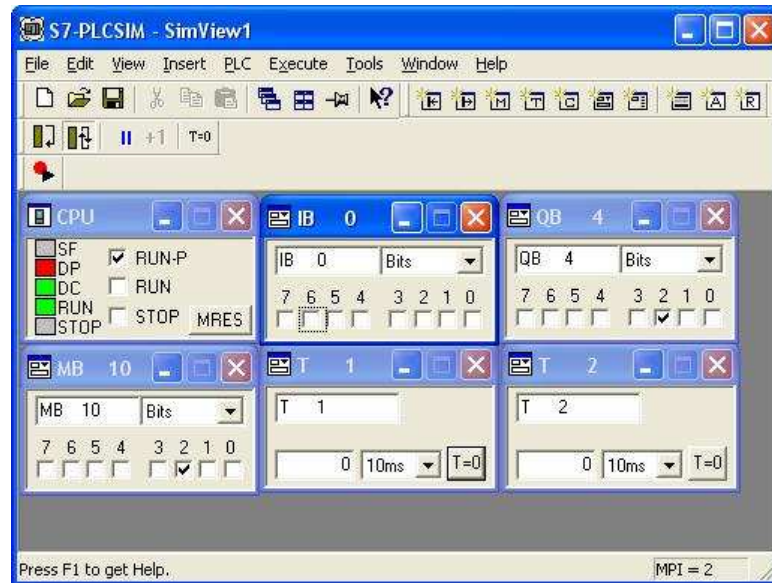


Figure 8.22 Simulation Garage Door 3

If any obstacle is noticed by *S2* while the door is closing, *O2* and *M2* are set to “0”. (Figure 8.23)



Figure 8.23 Simulation Garage Door 4

The warning lamp (*S7*) will be ON, if there is already a car in the garage (*S5* is set to “1”) and *S3* or *O1* is set to “1”. *T2* is set to 5 seconds. (Figure 8.24)



Figure 8.24 Simulation Garage Door 5

After 5 seconds elapsed, the warning lamp is OFF (*S7*). (Figure 8.25)

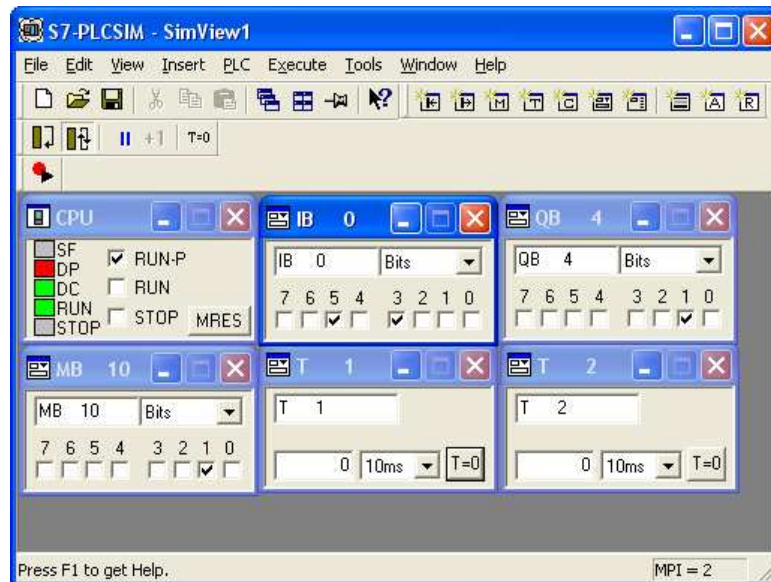


Figure 8.25 Simulation Garage Door 6

9. CONCLUSION

This work presented some of the most important issues in the field of Programmable Logic Controller. A whole process was described, how a S7-300 station could be taken into operation. In each chapter, different steps of the process were explained and illustrated with simple examples.

In the first chapter, general informations about PLCs and working principles were provided. The main advantages and disadvantages were mentioned with starting the use of PLCs in automation processes. Also according the demands of industrial manufacturers, the evolution of the PLC was described with mentioning the most important milestones in its history.

In the next chapter, an overview of Simatic controllers was given and Simatic S7 controller series and their specifications were described. This chapter focused especially on S7-300 station and its components.

Step7 is the software to program the Simatic controllers. Thus in the chapter 3 main features of Step7 were mentioned. The programming languages offered by Step7 are Ladder Logic Diagrams, Statement Lists, Function block Diagram and Structural Control Language. Advantages, disadvantages and the main differences between these languages are given in this chapter. The most used functions were explained with simple examples such as AND, OR, XOR, SR Flip Flop, On Delay Timer etc.

In the chapter Installation, it was described how the power supply, CPU and signal modules can get wired. Then the station has to be configured in the software program Step7. It has been shown, how to find station components in the program catalog and add them to their slots appropriately. After addressing the signal modules, it was shown, how the station can be connected to a programming device.

S7-PLCSIM is used to simulate user programs without additional hardware. Chapter 6

was dedicated to introduce the simulation program S7-PLCSIM. This program is especially important for us, because all examples given in the last chapter were simulated with S7-PLCSIM.

Future will show how the PLCs will evolve. The most important issues and challenges in the field of automation controllers were mentioned in the chapter 7. The developments and trends in automation systems were explained in order to understand what the future may bring and whether the PLCs will behave like a modern computer or like a usual PLC.

In the last chapter, different PLC applications were chosen, programmed and simulated. First example is a pump motor that works with different speed according to the fluid status of the containers. Since only AND and OR functions were used to program this application, it was easy to understand basic principles of programming and how to use the simulator.

Conveyor belts are used in many automation processes to transport goods or machine parts. Thus conveyor belts were programmed and simulated in the second application. In this example only AND and OR functions were used like the first one, but in different way.

In the third example a rotating shaft was simulated. To determine the direction of rotation, SR (Set-Reset) Flip Flop function, Positive RLO Edge Detection, Negative RLO Edge Detection and memories were used additionally. SR Flip Flop function is required often at programming where the signal values have to be saved. This example was also suitable to show, how Positive and Negative RLO Edge detection function and can be used.

The example “sorting and packing pipe” was chosen, thus it was possible and easy to explain, how counter functions and compare function can be used in the program. Since we have to deal with the numbers of the pipes, this time another data type has to be used

too. Simulator program counts the pipes by using WORD data type.

The last application of PLC was garage door opener. In this example timer functions S_ODT (On Delay Timer) and S_Pulse (Pulse Timer) were used and shown how the timers are set. By using timers automatic loops can be created and the system can work by itself.

REFERENCES

Berger, Hans (2005). *Automating with STEP 7 in LAD and FBD*. Publicis Corporate Publishing, Germany.

Berger, Hans (2006). *Automating with Simatic*. Publicis Corporate Publishing, Germany.

Wellenreuther, G. and Zastrow, D. (2002). “*Automatisieren mit SPS Theorie und Praxis*.” Der Verlag Vieweg, Germany.

Kopacek, Peter (1993). “*Einführung in die Automatisierungstechnik*.” R. Oldenbourg Verlag, Vienna/Austria.

SIEMENS AG, “Function Block Diagram (FBD) for S7-300 and S7-400 Programming – Reference Manual”, (Edition 01/2004)

Wardzel, Filomena (2006). “Long Live the PLC”. *Siemens Automation Summit – Users Conference*.

Melore, Phil. “PLC History.” (2001). Retrieved 19.July.2008 <<http://www.plcs.net/chapters/history2.htm>>.

SIEMENS AG, “Simatic S7-300 Introduction.” (2007). *Siemens Simatic Catalog ST70*. 128

SIEMENS AG, “Software PLC - SIMATIC WinAC RTX 2008.” (2008). Retrieved 28.July.2008 <http://www.automation.siemens.com/simatic/controller/html_76/produkte/software-plc-winac-rtx.htm>

SIEMENS AG, “Technical Specifications of the S7-300 digital input modules.”(2007). Retrieved 23.10.2008 <<http://www.automation.siemens.com/download/internet/cache/3/1438553/pub/de/tech-data-s7-300-io-en.pdf>>

“Totally Integrated Automation.” (2008). Retrieved 23.July.2008
<http://en.wikipedia.org/w/index.php?title=Totally_Integrated_Automation&oldid=185826305>

LIST OF FIGURES

Figure 1.1.....	3
Figure 1.2 (http://www.angelcontrols.com/Graphics/PLC_Group_Shot.jpg , 23.Sept.08).....	4
Figure 2.1 (Siemens Simatic S7-300 Programmable Controller Hardware and Installation Manual Edition 1, 1998, p.1-2).....	10
Figure 3.1.....	16
Figure 3.2 (Function Block Diagram (FBD) for S7-300 and S7-400 Programming – Reference Manual, 2004, p.13-6).....	17
Figure 3.3 (Function Block Diagram (FBD) for S7-300 and S7-400 Programming – Reference Manual, 2004, p.13-10).....	18
Figure 3.4 (Function Block Diagram (FBD) for S7-300 and S7-400 Programming – Reference Manual, 2004, p.13-14).....	19
Figure 4.1 (SIEMENS AG, S7-300 CPU 31xC und CPU 31x: Aufbauen, 2006, p. 4-3)	20
Figure 4.2. (SIEMENS AG; S7-300 Programmable Controller Hardware and Installation Edition 1, 2008, p, 2-8).....	20
Figure 4.3 (SIEMENS AG, S7-300 CPU 31xC und CPU 31x: Aufbauen, 2006).	21
Figure 4.4 (SIEMENS AG, S7-300 CPU 31xC und CPU 31x: Aufbauen, 2006, p. 6-7)	21
Figure 4.5	22
Figure 4.6	23
Figure 4.7	24
Figure 4.8	25
Figure 4.9 (SIEMENS AG, S7-300 Programmable Controller Hardware and Installation, 1998, p. 3-6)	27
Figure 4.10 (SIEMENS AG, S7-300 Programmable Controller Hardware and Installation, 1998, p. 3-7)	27
Figure 4.11	28
Figure 5.1	30

Figure 5.2	30
Figure 5.3	34
Figure 6.1	36
Figure 6.2	38
Figure 8.1 (Kopacek, Peter, 1993, <i>Einführung in die Automatisierungstechnik</i> , p.67).	43
Figure 8.2	48
Figure 8.3	49
Figure 8.4	49
Figure 8.5	50
Figure 8.6 (Kopacek, Peter, 1993, <i>Einführung in die Automatisierungstechnik</i> , p.61) ...	51
Figure 8.7	54
Figure 8.8	54
Figure 8.9 (Kopacek, Peter, 1993, <i>Einführung in die Automatisierungstechnik</i> , p.98) ...	55
Figure 8.10	59
Figure 8.11	59
Figure 8.12	60
Figure 8.13	60
Figure 8.14 (Wellenreuther, G. and Zastrow, D., 2002, “ <i>Automatisieren mit SPS Theorie und Praxis</i> , p.82) ...	61
Figure 8.15	67
Figure 8.16	68
Figure 8.17	69
Figure 8.18	69
Figure 8.19	70
Figure 8.20	74
Figure 8.21	74
Figure 8.22	75
Figure 8.23	75
Figure 8.24	76
Figure 8.25	76

LIST OF TABLES

Table 3.1	14
Table 3.2	14
Table 3.3	15
Table 3.4	15
Table 3.5	15
Table 4.1	26
Table 5.1	31
Table 8.1	44
Table 8.2	44
Table 8.3	45
Table 8.4	45
Table 8.5	51
Table 8.6	52
Table 8.7	52
Table 8.8	55
Table 8.9	56
Table 8.10	62
Table 8.11	63
Table 8.12	71