Content Development for “Lern-Umgebung für SPS Systeme”

Hendrik Busch

Bachelor’s thesis
January 2012
Environmental Engineering
ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Environmental Engineering

Hendrik Busch:
Content Development for “Lernumgebung für SPS Systeme”

Bachelor's thesis 39 pages, appendices 22 pages
August 2012

Germany is a big player in engineering and very attractive to a number of countries as a trade partner. A European project called “Automatisierung und Deutsch im Online-Kurs” attempts to connect the teachings of automation and the German language, to make students ready to deal with German companies on the subject.

Teaching the programming with Step7, this thesis combines theoretical teaching with practical information. The educational material produced as part of this thesis is utilized in the online course of the project. The students will gain general background information on the hardware and selected programming languages. Tests in form of multiple choice quizzes give feedback on the students understanding of the subject matter. A detailed explanation depicts the practical approach of solving a small programming task.

Key words: automation, step7
CONTENTS

1 ABBREVIATIONS .................................................. 4
2 INTRODUCTION .................................................. 5
3 THE ADOK-COURSE .............................................. 6
  3.1 General structure ............................................ 6
  3.2 Moodle courses .............................................. 7
  3.3 The ADOK Website .......................................... 8
  3.4 Modules .................................................... 9
  3.5 Thesis Material ........................................... 11
  3.6 Background Information ................................ 12
    3.6.1 PLC Hardware ........................................ 12
    3.6.1.1 Modular Setup/Rack ................................ 12
    3.6.1.2 CPU Module ........................................ 12
    3.6.1.3 Input/Output Module ............................... 13
    3.6.1.4 Digital I/Os ....................................... 13
    3.6.1.5 Analog I/O ........................................ 14
    3.6.1.6 Interface module .................................. 14
    3.6.2 Simatic Step 7 Software ............................. 14
    3.6.2.1 Ladder Logic (LAD) ................................ 14
    3.6.2.2 Function Block Diagram (FBD) .................... 19
    3.6.2.3 Graph ............................................... 21
4 PROGRAMMING TRAFFIC LIGHTS ................................ 25
  4.1 The Setup ............................................... 25
  4.2 Buttons .................................................. 28
    4.2.1 Day and Night Modes ................................ 28
    4.2.2 Fire Mode ............................................. 29
    4.2.3 Synchronized Mode ................................... 30
    4.2.4 Pedestrian Crossing .................................. 31
  4.3 The Sequencers .......................................... 31
    4.3.1 Sequencer 1/4 Pedestrian Crossing 1/2 Day ...... 31
    4.3.2 Sequencer 2 Pedestrian Crossing ................... 35
    4.3.3 Sequencer 3/5 Intersection 1/2 Night ............. 36
    4.3.4 Sequencer 6 Synchronized Mode ..................... 36
5 DISCUSSION .................................................. 38
BIBLIOGRAPHY .................................................... 39
APPENDICES .................................................... 39
  Appendix 1. Video descriptions .............................. 39
  Appendix 2. Quizzes ......................................... 39
  Appendix 3. Program Traffic Lights ......................... 39
1 ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>DP</td>
<td>decentralized periphery</td>
</tr>
<tr>
<td>FBD</td>
<td>function block diagram</td>
</tr>
<tr>
<td>I/O</td>
<td>in- and outputs</td>
</tr>
<tr>
<td>LAD</td>
<td>ladder logic</td>
</tr>
<tr>
<td>MPI</td>
<td>multi point interface</td>
</tr>
<tr>
<td>PLC</td>
<td>programmable logic controller</td>
</tr>
<tr>
<td>RAM</td>
<td>random access memory</td>
</tr>
</tbody>
</table>
2 INTRODUCTION

Providing students with understandable information is the main objective of this thesis. The information gathered here is available multiple times in books, manuals and websites. The attempt is to select the information necessary to develop simple programs and provide and present a knowledge foundation in a understandable way.

The programming described in chapter 4 shows how to solve a unique problem. From the unique example the generic approach can be learned and transferred to other tasks. The aim is to show a way to analyze a given task and then translate it into a functioning program. It complements the material for the ADOK project in order to demonstrate a practical understanding of the subject matter.
3 THE ADOK-COURSE

As a European project schools from Czech, Estonia, Finland and Germany are involved in creating a multinational learning environment, in which students simulate the order and installation of a plc-system.

Currently a pilot course is being held at three schools to test course material and get feedback on possible soft spots. A student exchange between Tampere and Tallinn serves as opportunity to collaborate and present the project work.

3.1 General structure

The course requires students to work in small international groups of maximum 4 students from different universities. Each group of students plays two roles. They lead a project as purchaser of a system and also supply a system in return. Teams work together with a team from different Universities. The official course language is German so communication between the teams as well as teaching takes place in German, however additional technical descriptions are supplied in English to improve technical understanding. Mainly all instructions and material is supplied via a moodle-website, as further explained in the next chapter. In the University of Applied Sciences Tampere (TAMK) the course is coached by a automation teacher with German language skills and a German language teacher. They are initially giving lectures on STEP 7 programming and German for project communication and later supervise the project work.
3.2 Moodle courses

Moodle (Modular Object-Oriented Dynamic Learning Environment) online learning environments are widely used in teaching with over 68,000 registered websites in 220 countries. (moodle.org, 1) In general, the emphasis is on collaborative work, to make use of the students’ abilities of teaching each other. The software is provided under a public license to the public for free.

A course organized in moodle offers an easy and clear way to arrange documentation. Beyond being used as just a download platform for students, it has the potential for creating a second environment for collaboration between students and teachers aside from the classroom. There are many ways to shape the style of this collaboration tool. A simple forum may create a space for students to help each other and learn by observing their peers.

When a course website is constructed with the moodle software, there are many available building blocks that can be included to add features. The main features are activities customized to fit the courses. Some activities as used in this course can be seen on the right in Figure 2. For example assignments from students can simply be uploaded to a return forum to receive grading. It is possible to have the assignment return forum close at a specific time to encourage punctual behavior.

Figure 2: Activities
3.3 The ADOK Website

To use the ADOK online platform, the student needs to register with username and password on the website http://moodle.adok-projekt.eu. When logging in, the front page shows the following:

![Main Window and Navigation Block]

The “Main Window” shows the complete course. The top segment consists of general information about the course and links to organisational information. The course description is available in the languages of all participants. Below it, all course content is arranged in modules as explained in the next chapter.

In the “Navigation Block” on the left, a number of organisational tools are collected in small boxes. From top to bottom the boxes are: list of participants, online users, activities, search function, links to all modules, administration (grades and profile), news, events, apps, and progress bar.
3.4 Modules

The ADOK-course is structured in modules as seen below on the left in Figure 4. Each module conveys information and asks the student to recite in form of different assignments. As an example the content of module 1 (Introductions) is pictured below. It consists of a number of assignments, mostly about German language.

![Figure 4: Modules](image)

Every module has clear learning objectives, stated at the top of the module content. The summary, all module objectives are stated below.
Introductions
In the first module the student will learn to introduce himself, understand the introduction of others and basics about PLC components.

Hardware
The student will get a basic understanding of the basic functions of a PLC and binary logic. Further the student will be introduced to learning strategies for the German terminology.

Software
Basic knowledge of STEP7 programming and use of binary logic in programs is taught along with German definitions.

Examples
Exercises explain the practical use of hard- and software while the vocabulary is trained.

Customer Communication I
The student is introduced to German business communication in form of e-mails and letters and has to practice it.

Assignment Agreement
The assignment is delivered in German and has to be properly understood and analyzed.

Customer Communication II
Details to dealing with German businesses in terms of payment, delivery and deadlines are conveyed while a deadline for the project is set.

Delivery
To better understand technical documents the specific grammar is improved while the project is finished.

Company Visit
The intercultural differences in communication are explored in a company visit and the projects are presented.
Reading Strategies
A number of exercises help understanding German texts.

3.5 Thesis Material

Background Information
The background information is a theoretical collection of the basics concerning hardware and software.

Video Descriptions
The online course also features a number of videos in module 4 as seen on the right in with instructions for the most basic tasks in programming a Step7 program. Explanations for these Videos are produced as part of this thesis in German, so the students have to try to understand basic German while they can look up individual steps. There is a small quiz in German available on the instructions. The instructions are enclosed in appendix 1.

Quizzes
The quizzes about the background information and the video descriptions are created as part of this thesis. Questions on background information are given in English language, the questions on video descriptions are German like the source material. All questions are multiple choice questions with one right answer out of three. The quizzes are enclosed as appendix 2.
3.6 Background Information

3.6.1 PLC Hardware

A programmable logic controller is an automation tool for machine control. It can execute instructions based on the installed program and information from inputs. These instructions are then related to actuators. The input information can be in binary or continuous form. (Hugh 2004, 2 p. 37) The binary signals are a simple form of information and thus used if possible. A plc centralizes the control of a machine or process, it can execute complex programs, but the programming is done in languages that are based on logic operations. (Bolton 2006, 3)

3.6.1.1. Modular Setup/Rack

The S7-300 station is a plc system produced by the Siemens company. It is a flexible system of modules which are mounted on one or multiple racks. The number of modules used varies with the application, however only in the number of input, output and interface modules. The size of a S7-300 is limited to eight modules on four racks. In the central rack the first slot is always reserved for the power supply, the second for the CPU and the third for the interface module. Remaining slots can be used for I/O or other modules.

3.6.1.2. CPU Module

Usually when using the term CPU it refers to the whole CPU module of a PLC rather than just the microprocessor. The CPU is the main unit of the system, it contains a microprocessor that executes the user program that is stored in the module. In the S7-300 station the module also communicates with the programming device and other stations via communication standards called integrated interfaces. The available integrated interfaces always include a MPI interface, but the availability of DP and PROFINET interfaces differ between models. (Siemens 2011, 4)

It also contains memory to store the user program consisting of a load memory and a work memory. The load memory is used to save the user program and the configuration data while the work memory is a fast ram which is changed while running. (Hugh 2004,
Memory cards can be used to extend the existing memory. Available are RAM cards, flash eprom cards and micro memory cards, however in the S7-300 only micro memory cards can be used to increase the load memory.

When running, the CPU scans the program in cycles continuously and processes it one instruction at a time. This means that the length of the program has great influence on the reaction time, which may be shorter than a few milliseconds. (Zhang 2008, 5) Because of the short cycling times operations are effectively happening almost simultaneously.

3.6.1.3. Input/Output Module

In- and Output modules are the connection of the CPU with the sensors and actuators. They can be divided into logical, also called digital, I/Os which have a discrete value and Analog I/Os, which have a continuous value. It depends on the sensors and actuators used what kind of I/O is necessary. (Hugh 2004, 2)

When a module is installed it gets assigned a module address by the hardware configuration automatically. (Berger 2009, 6) The module gets its address according to the rack number and the slot on the mounting rack.

3.6.1.4. Digital I/Os

In a digital module inputs and outputs are grouped to eight slots, which are called bytes. A module can contain several bytes. S7-300 I/O-Modules are available with up to 64 slots.

A logic sensor determines if a measured state is existent or not, a switch for example. The digital input transfers this information to the CPU. Digital actuators are used as switches, usually connected to a relay or transistor.
3.6.1.5. Analog I/O

Analog inputs are used to transfer the information of sensors with a continuous value. Since analog information is more complex to convey, analog modules are more expensive. There are fewer channels in a module with one to eight channels per module available. The information is provided in form of a continuously varying voltage or current and needs to be translated into a digital signal. The resolution affects the accuracy of the digital data. A 16 bit resolution means that the range of the analog signal is divided into $2^{16}$ steps. The accuracy is thus limited by the resolution. (omega.com, 7)

3.6.1.6. Interface module

This module handles the communication between mounting racks. Every expansion rack needs an interface module to be connected to the CPU.

3.6.2 Simatic Step 7 Software

Step 7 offers multiple programming languages. This overview over basic functions of the languages Ladder Logic (LAD), Function Block Diagram (FBD) and Graph shows functions and setup of LAD and FBD which are very similar and the very different Graph. When programming in LAD or FBD the view can be switched, which means all elements can be expressed in the other language.

3.6.2.1. Ladder Logic (LAD)

Ladder logic is a graphic programming language that is modeled after relay logic. (Hugh 2004, 2 p. 20) The program or program block for the control of the PLC is written by designing a network of contacts, coils and boxes. This is a very close resemblance to the actual physical electronic systems the PLC derived from, to simplify the programming process. However additional functions have been added.

To form a program a number of elements/operations are available. These can be combined to perform more complex operations. A knowledge about the operations available is necessary to find programming solutions.
The basic layout of the LAD programming window shows two vertical wires on the left and right side. Every network that is inserted creates a connection between the wires. In this network you can insert symbols such as contacts, coils and boxes with different functions. Multiple networks together create a user program. The descriptions given here are based on (Berger 2008, 8) and (Siemens 2010, 9).

Contacts

A contact serves as an input. It converts a binary input into the states open or closed. It is connected to an input address. There are two types of contacts, normally open or normally closed contacts. The normally open (NO) contact shows a state “0” as open or not connected and the state “1” as closed or connected. The normally closed (NC) contact works exact opposite. The state is connected for the input value “0” and disconnected for “1”.

![NO-contact](image)

![NC-contact](image)
Coils
The output of binary signals is realized by inserting coils, as seen in Figure 7 and Figure 8, into the network as shown above. The value of the connected address is set to “1” when the coil is under power, as soon as the coil is not under power anymore the value is set to “0”.

NOT
This element inverts the power flow.

![NOT element](image)

Figure 9: NOT element

Series Circuits AND
NO contacts that are connected in series act like a AND function. Only when all contacts are closed is the value at the end of the series set to “1”, whenever one of the contacts is not connected the value is “0”.

![AND element](image)

Figure 10: AND element

Parallel Circuits OR
NO contacts that are connected parallel act like an OR function. Whenever one of the contacts is connected the value on the right side of the contacts is set to “1”. Only when all contacts are disconnected the value is “0”.

![OR element](image)

Figure 11: OR element
XNOR

This layout with four contacts in the following way shows a true value at the output, if both inputs have the same value, either “0” or “1”.

![XNOR element](image)

Figure 12: XNOR element

Memory functions

Set and reset coils can be used to store a value in an address. If power is flowing through a set coil, the value of the address is set to “1”. The reset coil resets the value to “0” when the power is flowing. If the power is cut off, the operand is not affected. Memory Boxes such as the SR or the RS box combine set and reset coil to one element. The box has two inputs and a single output. The set (S) input is used to set the output value to “1”, the reset input resets the output value to “0”.

The difference between the two versions of this memory box is the dominance which can be on the setting or on the resetting. A SR box is dominant on the setting, meaning that when both the set and reset input signal values are “1”, the dominant setting signal will set the output signal. Thus the output value will be “1”. In a RS box the resetting is dominant, thus the output value, when both input values are “1”, will be “0”.

![Set-coil](image)

Figure 13: Set-coil

Midline Outputs

To output value in the middle of a network, a midline output coil can be inserted. It does not influence the power flow.
Edge Evaluation
Often it is necessary to detect the change of a value which is called an edge. When a false value becomes true the edge is positive or rising, when a true value becomes false it is negative or falling. The program compares the current state of a value with a stored value, if it differs, the edge is detected.

Timers
All functions that require delays in action or timed coordination require the use of a timer. There are several timer types with different characteristics. There is memory assigned for the timers, so the number is limited. This also restricts the time value to 9,990 seconds. Time values have to be formatted as S5TIME values, e.g. S5TIME#50ms (50 milliseconds) or S5T#1H20m (1 hour and 20 minutes).

Pulse Timer S_PULSE
This timer is used to produce a pulse of a set length. When the input changes to true, the pulse starts and is true for the set time, as long as the input value is still true. The timer can be stopped using the reset input, which is dominant.

Extended Pulse Timer S_PEXT
Also used to produce a pulse of set length, this timer does not require the input to be true for the time of the pulse. Changes of the input value from false to true, while the pulse is still active causes a restarting of the timer and thus the pulse to be extended.

On-Delay Timer S_ODT
To delay the transmission of a value, this timer is started when a rising edge at the input is detected. When the defined time is over the output value changes to true, however only as long as the input signal is still true.

Retentive On-Delay Timer S_ODTS
The rising edge on the input also triggers this timer, causing the output value to change to true after the defined time has passed. If there is a second rising edge during the time the timer is running, the timer is restarted. After the timer is started, the input value doesn’t have to be true for the output value to be true.
3.6.2.2. Function Block Diagram (FBD)

This graphic programming language uses a similar layout as the ladder logic language. As the name suggest the program is written by inserting and connecting boxes called function blocks. These boxes can perform simple binary logic or complex non-binary functions. (Berger 2008, 8 p. 115) The descriptions given here are based on (Berger 2008, 8) and (Siemens 2010, 10). A single network can contain a circuit with a single or a large number of function boxes that all have to be connected. The data type on both ends of a connection has to be the same. A function box may have multiple inputs and outputs. The programmer can also create new function blocks, which is especially useful, if a user-defined function is used repeatedly in the program.

To invert a logic value, the input on a function box can be inverted, by adding a negation circle. Note that “Coil 1” on the assign statement is just a variable name connected to the output address Q124.4.

![Figure 14: Inverted input](image)

AND (&)

This function compares all inputs and only gives the value “1” when all inputs are “1” at the same time. As soon as the condition is no longer fulfilled, the output value is reset to “0”.

![Figure 15: AND-box](image)

OR (>=1)

The output value of this function is “1” when at least one of the inputs is “1”. Only when none of the input values is “1”, is the output reset to “0”.

![Figure 16: OR-box](image)
XNOR

The exclusive or function compares the inputs and only shows “1” on the output if all inputs show the same state, either both “1” or both “0”.

![XNOR element](image)

Memory Functions

Set and reset boxes can be used to store a value in an address. If the input on a set box is “1”, the value of the address is set to “1”. The reset coil resets the value to “0” when the power is flowing. If the power is cut off, the operand is not affected. Memory Boxes such as the SR or the RS box combine set and reset box to one element. The box has two inputs and a single output. The set (S) input is used to set the output value to “1”, the reset input resets the output value to “0”.

![Set-box](image)

![Reset-box](image)

The difference between the two versions of this memory box is the dominance which can be on the setting or on the resetting. A SR box is dominant on the setting, meaning that when both the set and reset input signal values are “1”, the dominant setting signal will set the output signal. Thus the output value will be “1”. In a RS box the resetting is dominant, thus the output value, when both input values are “1”, will be “0”.

![SR-box](image)
Midline Outputs
To output value in the middle of a network, a midline output box can be inserted. It does not influence the output.

Edge Evaluation (N/P)
Detects change in a value. When a false value becomes true the edge is positive or rising, when a true value becomes false it is negative or falling. The program compares the current state of a value with a stored value, if it differs, the edge is detected.

Off-Delay Timer S_OFFDT
This timer allows prolonging the output signal for a defined time, after the input signal value has a negative edge. When the input signal is true, so is the output signal, with the exception of when the reset input is true.

Counters
The counting function is used in tree counter boxes, the up-down (S_CUD), down (S_CD) and up (S_CU) counter. The S_CD and S_CU boxes are simplified versions of the S_CUD box. The inputs for counting up and/or down de- or increase the count for each positive edge detected. To be detected the change in value needs to be present for at least one program cycle.

3.6.2.3. Graph

Very different, this programming language is used to create chains of events instead of focusing on logic control. The descriptions given here are based on (Siemens 2004, 11). These chains of events are called sequencers. A sequencer consists of steps, which contain the orders that are being executed, and transitions to control the flow of the se-
The function block that contains the sequencer may contain multiple sequencers that can be connected through jumps.

A step can contain a number of actions. An action is always a combination of variable and command. A transition is situated between steps and has conditions that control value of the transition. The conditions are expressed in LAD or FBD. A positive value allows switching to the next step.

![Figure 21: Sequencer with a single step](image)

In the example above you can see a sequencer with only a single step. The box to the right of the sequencer contains the actions that will be taken in step one. The top line is just the name of or commentary to this set of actions. The second line is a actual command. On the left side the letter S stands for “set”. Which means the variable in the right field, here conveniently named “Variable”, will be set to the value “1”.

![Figure 22: Actions-box](image)

The step will be active as long as the value of the transition is negative. In this example the value of “Variable” is directly connected to the transition. As long as “Variable” is positive the transition allows switching to the next step. The little arrow down after the transition depicts a jump, in this case to the top of the transition. It is marked with “S1” referring to the jump to the point before Step1, which in return is marked with “T1” referring that there is a jump from Transition1 to this point.

![Figure 23: Transition](image)
The following is an overview over basic commands that can be used to program actions.

N  Sets the address to “1” as long as the step is active.

S (Set)  Sets the address to “1” without resetting it when the step is deactivated.

R (Reset)  Resets the address to “0”.

D (On Delay)  The variable is set to “1” after a time period set with a time constant in the format T#Xt with X being the number and t the time unit. For example 14 minutes and 12 seconds are expressed as T#14M12S. Other time units are: D(days), H(hours), M(minutes) and MS(milliseconds).

More complex actions are available, for a full list see (Siemens 2004, 11 pp. 80-106).

A sequence may be structured in multiple ways.

Alternative Branch
Acting like an OR operation, multiple parallel branches allow different actions for different conditions. Branches can be inserted between the step and the transition, so that the transition of each branch is the condition for the branch to be active. Only one branch is executed at any time.

![Figure 24: Alternative branch](image)
Simultaneous Branch
This branch allows for different actions to be active at the same time. The branch is inserted between the transition and the step, thus making the transition the condition for the execution of all branches. When the simultaneous branches join together again, all branches have to be completed.

Branch Jump
As already explained above, the jump concludes a sequence and redirects to any other step within the function block.

Branch Stop
Another way to conclude a sequence is terminating the sequencer with a branch stop.
4 PROGRAMMING TRAFFIC LIGHTS

4.1 The Setup

Like in every project, the specification of objectives should be clear. This example consists of a model of two intersections fully equipped with traffic lights connected by a street. A pedestrian crossing with traffic lights is situated on the connecting road as shown in the Figure 29.

Objective specification:

- Both intersections independently have a functioning cycle for all traffic lights.
- Pedestrian lights turn green/red before the car signals.
- The mode of operation can be changed between day and night.
- The pedestrian crossing is button activated but ensures a long enough green phase for car traffic.
- The traffic lights of both intersections and the pedestrian crossing can be synchronized in both directions of traffic.
- The fire mode reacts to a fire station exit at the left intersection.

Keep in mind that in comparison to usual systems, this is a very limited task. Documentation becomes more important when the system size increases. As seen in Figure 28 below, the program structure is quite simple, it consists of just two function blocks.

Figure 28: SIMATIC Manager
The programming of the cycles will be done in Graph, since the traffic light cycle is a continuous flow of events that repeats itself. Additional elements will be programmed with FBD. The full program is included as appendix 3.

The first step is to establish a symbol table (Figure 30). All in- and output addresses have to be connected to the variables, which will be used in the programming. The traffic lights have been labelled already in the layout plan, so for the individual lights these names (e.g. CTL1 for car traffic light 1) are used with an index for the colour (e.g. CTL1Y for the yellow light in CTL1).

To ensure a smooth programming process, all functional elements should be given a distinct name to avoid any confusion. In this example creating a map of the street layout with all lights, buttons and the respective in- and output addresses is helpful.

![Figure 29: Road Layout](image)

The addresses shown here are the variables used in the program and the corresponding connectors on the module case. The connectors have to be connected to the PLC via wires. The correct in- or output address then has to be allocated to the variables. The symbol table on the following page is from a German version of the program, so input addresses are labelled E, outputs A and memories M.
<table>
<thead>
<tr>
<th>Status</th>
<th>Symbol</th>
<th>Adresse</th>
<th>Datentyp</th>
<th>Kommentar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>buttonDay</td>
<td>E 0.1</td>
<td>BOCL</td>
<td>Button Day</td>
</tr>
<tr>
<td>2</td>
<td>buttonNight</td>
<td>E 0.4</td>
<td>BOCL</td>
<td>Button Night</td>
</tr>
<tr>
<td>3</td>
<td>buttonPedX</td>
<td>E 0.0</td>
<td>BOCL</td>
<td>Button Ped. Xing</td>
</tr>
<tr>
<td>4</td>
<td>buttonSyncLeft</td>
<td>E 0.2</td>
<td>BOCL</td>
<td>Button Sync Left</td>
</tr>
<tr>
<td>5</td>
<td>buttonSyncRight</td>
<td>E 0.3</td>
<td>BOCL</td>
<td>Button Sync Right</td>
</tr>
<tr>
<td>6</td>
<td>CTL1G</td>
<td>A 5.3</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>7</td>
<td>CTL1R</td>
<td>A 5.5</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>8</td>
<td>CTL1Y</td>
<td>A 5.4</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>9</td>
<td>CTL2G</td>
<td>A 5.0</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>10</td>
<td>CTL2R</td>
<td>A 5.2</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>11</td>
<td>CTL2Y</td>
<td>A 5.1</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>12</td>
<td>CT_3R</td>
<td>A 6.6</td>
<td>BOCL</td>
<td>Ped. Xing</td>
</tr>
<tr>
<td>13</td>
<td>CTL5Y</td>
<td>A 6.5</td>
<td>BOCL</td>
<td>Ped. Xing</td>
</tr>
<tr>
<td>14</td>
<td>CTL4G</td>
<td>A 4.3</td>
<td>BOCL</td>
<td>Intersection2:13.3</td>
</tr>
<tr>
<td>15</td>
<td>CTL4R</td>
<td>A 4.5</td>
<td>BOCL</td>
<td>Intersection2:13.5</td>
</tr>
<tr>
<td>16</td>
<td>CTL4Y</td>
<td>A 4.4</td>
<td>BOCL</td>
<td>Intersection2:13.4</td>
</tr>
<tr>
<td>17</td>
<td>CTL5G</td>
<td>A 4.0</td>
<td>BOCL</td>
<td>Intersection2:13.0</td>
</tr>
<tr>
<td>18</td>
<td>CTL5R</td>
<td>A 4.2</td>
<td>BOCL</td>
<td>Intersection2:13.2</td>
</tr>
<tr>
<td>19</td>
<td>CTLSync</td>
<td>A 4.1</td>
<td>BOCL</td>
<td>Intersection2:13.1</td>
</tr>
<tr>
<td>20</td>
<td>day</td>
<td>M 0.1</td>
<td>BOCL</td>
<td>day variable</td>
</tr>
<tr>
<td>21</td>
<td>delay</td>
<td>M 0.0</td>
<td>BOCL</td>
<td>delay int1</td>
</tr>
<tr>
<td>22</td>
<td>delay2</td>
<td>M 0.2</td>
<td>BOCL</td>
<td>delay pedXing</td>
</tr>
<tr>
<td>23</td>
<td>delay3</td>
<td>M 0.6</td>
<td>BOCL</td>
<td>delay int2</td>
</tr>
<tr>
<td>24</td>
<td>delay4</td>
<td>M 0.7</td>
<td>BOCL</td>
<td>delay sync right</td>
</tr>
<tr>
<td>25</td>
<td>STSSTD_3</td>
<td>FC 72</td>
<td>FC 72</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>NT1</td>
<td>M 0.3</td>
<td>BOCL</td>
<td>cycle switch intersection1</td>
</tr>
<tr>
<td>27</td>
<td>NT2</td>
<td>M 0.4</td>
<td>BOCL</td>
<td>cycle switch intersection2</td>
</tr>
<tr>
<td>28</td>
<td>nt2sync</td>
<td>M 1.1</td>
<td>BOCL</td>
<td>sync int2</td>
</tr>
<tr>
<td>29</td>
<td>Power</td>
<td>A 6.2</td>
<td>BOCL</td>
<td>Switches</td>
</tr>
<tr>
<td>30</td>
<td>PowerButtonDay</td>
<td>A 6.7</td>
<td>BOCL</td>
<td>Button Day</td>
</tr>
<tr>
<td>31</td>
<td>PowerButtonNight</td>
<td>A 7.2</td>
<td>BOCL</td>
<td>Button Night</td>
</tr>
<tr>
<td>32</td>
<td>PowerButtonSyncLeft</td>
<td>A 7.0</td>
<td>BOCL</td>
<td>Button Sync Left</td>
</tr>
<tr>
<td>33</td>
<td>PowerButtonSyncRight</td>
<td>A 7.1</td>
<td>BOCL</td>
<td>Button Sync Right</td>
</tr>
<tr>
<td>34</td>
<td>PTL1G</td>
<td>A 5.6</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>35</td>
<td>PTL1R</td>
<td>A 5.7</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>36</td>
<td>PTL2G</td>
<td>A 6.0</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>37</td>
<td>PTL2R</td>
<td>A 6.1</td>
<td>BOCL</td>
<td>Intersection1</td>
</tr>
<tr>
<td>38</td>
<td>PTL3G</td>
<td>A 6.3</td>
<td>BOCL</td>
<td>Ped. Xing</td>
</tr>
<tr>
<td>39</td>
<td>PTL3R</td>
<td>A 6.4</td>
<td>BOCL</td>
<td>Ped. Xing</td>
</tr>
<tr>
<td>40</td>
<td>PTL4G</td>
<td>A 7.3</td>
<td>BOCL</td>
<td>Intersection2</td>
</tr>
<tr>
<td>41</td>
<td>PTL4R</td>
<td>A 7.4</td>
<td>BOCL</td>
<td>Intersection2</td>
</tr>
<tr>
<td>42</td>
<td>PTL5G</td>
<td>A 7.5</td>
<td>BOCL</td>
<td>Intersection2</td>
</tr>
<tr>
<td>43</td>
<td>PTL5R</td>
<td>A 7.6</td>
<td>BOCL</td>
<td>Intersection2</td>
</tr>
<tr>
<td>44</td>
<td>TIME_TCK</td>
<td>SFC 54</td>
<td>SFC 54</td>
<td>Read the System Time</td>
</tr>
<tr>
<td>45</td>
<td>Xing</td>
<td>M 0.5</td>
<td>BOCL</td>
<td>cycle switch xing</td>
</tr>
<tr>
<td>46</td>
<td>Xingsync</td>
<td>M 1.0</td>
<td>BOCL</td>
<td>sync Xing</td>
</tr>
</tbody>
</table>

Figure 30: Symbol table
4.2 Buttons

The buttons used for switching between day and night, fire mode, synchronized traffic lights and pedestrian crossing are five push buttons that only give a “true” signal at the input as long as they are being pushed. For this reason, a few small networks, to connect these buttons to useful variables, are necessary. All buttons have a small LED indicator connected to outputs.

The programming connected to the buttons will be done in FBD or LAD since it requires logic operations. Here all examples will be shown in FBD. The specific logic requirements that are given through the task, now have to be translated into binary logic. All buttons will be connected to a mode of operation. The variables connected to the indicator lights outputs of all buttons except the pedestrian crossings, can be used to indicate if the mode is on or off.

4.2.1 Day and Night Modes

The only requirement for the day and night modes are that only one of them can be active at any given time. This means when the button for one mode is pushed, the other has to be disabled. To implement this, each variable for the mode, here “PowerButtonDay” and “PowerButtonNight”, will be connected to the status of a RS flip flop as seen in Figure 31 below. As you can see, the input “ButtonDay” sets and the input “ButtonNight” resets the RS.

![Figure 31: Network PowerButtonDay](image-url)
The opposite is done for “PowerButtonNight”. Reset dominant boxes are used rather than SR-boxes so that when both buttons are pressed, both are disabled as seen in Figure 32 below. The assignment box connected to “PTL3R” is needed to switch off the pedestrian crossing at night.

![Figure 32: Network PowerButtonNight](image)

### 4.2.2 Fire Mode

The fire button should simply toggle the fire mode on and off. To build the toggle switch two networks with a SR-box each are needed and connected as in figure 22 and 23. The mode is only needed in the day. To implement this, the button should only work when “PowerButtonDay” is “true”.

4.2.3 Synchronized Mode

This mode needs a toggle switch similar to the fire mode. The synchronisation of the traffic lights is only necessary in the daytime, however it will still be possible to switch it on or off regardless of day or night. Note that “M1.2” and “M1.3” are memory vari-
ables for the memory boxes. “PowerButtonSyncRight” is always the same value as “M1.2”.

Figure 35: Network1 PowerButtonSyncRight

Figure 36: Network2 PowerButtonSyncRight

4.2.4 Pedestrian Crossing
Since the pedestrian crossing button should only indicate, that the traffic light should switch, a set-box will be used as shown in Figure 37 to set a variable, which will be used in the programmed sequence.

Figure 37: Network pedestrian crossing

4.3 The Sequencers

4.3.1 Sequencer 1/4 Pedestrian Crossing 1/2 Day
Since the day and night sequences are very different, they will be programmed in different sequencers. To make it possible to switch the sequencers, while not interrupting an ongoing cycle the variable “INT1” is set to “true” at the beginning of the traffic light
cycle and to “false” at the end. A transition at the beginning checks if “INT1” is false, making it impossible to start the night cycle while the day cycle is still running (Figure 38). The reason for the use of “PowerButtonFire” will be explained later on.

As seen above “INT1” is reset in the first step to make sure its value is “0” when the sequencer is started. The AND-box in transition 102 requires the day mode to be active and “INT1” to be “0”, so if the traffic light is in night mode, the cycle will stop here until the day mode is engaged again. It is important that it is not stopped at the first step, since then “INT1” would continue to be negative, which means that when the mode is switched it would not wait until the night cycle is completed.

To better understand which state the variables have to be in at any given time, a flow diagram is created (Figure 39). Since the intersections have the same layout, only one flow diagram for both intersections is necessary. The pedestrian crossing needs a distinct diagram.

The diagram shows the status of all variables throughout one complete traffic light cycle. The graphs are colour coded for a clearer picture. The status of each variable is represented by one line in the graph, the horizontal dashed line representing the status 0. The x-axis shows the time in the cycle. There is a vertical dashed line for every event, making it easier to spot corresponding events.
This way of visualizing the cycle simplifies the creation of sequencers since it can be implemented step by step following the timeline. Each vertical dashed line will be a step in the sequence.

The timing of the sequence can be controlled by using the action “delay”, as seen here in step 3 (Figure 40), which in the diagram is marked by the first vertical dashed line. The command delay requires two entries, the variable, that will get the value “1” when the time is run out (here “delay”), and the time length. As soon as the step is active, the timer starts running and the variable “delay” will be set to “0”. In this case after 8 seconds the value of the “delay” will switch from “0” to “1”.

---

Figure 39: Variables status graph intersection

Figure 40: Sequencer1 Step3
As soon as the value of “delay” is “1”, the condition of the transition is fulfilled and the next step is executed. The variable can be used for timers in every step, but multiple sequencers should use different variables to avoid interference.

Variables that are true for just one step are set with a “N” action, as “PTL2G” in Figure 40 above for example. Others are set with the “S” action and then later reset with “R”. The rest of the sequencer implements the diagram (Figure 39) and ends with a jump to the top of the sequencer as seen here in Figure 41.

The fire mode is implemented in the sequencer. The aim is to stop all traffic by switching all traffic lights to red. To not cause accidents the yellow phase should still be used. This means that the action depends on what step in the sequencer is active. An easy way to solve this is to attach OR-elements to all transitions, except the ones where a car traffic light switches to yellow, with one input being “PowerButtonFire”. As soon as the variable is “true” all steps are skipped until step 105 or 111 are reached. As you can see on the next page (Figure 42) first the yellow phase is completed and then the sequencer is stopped at step 106 until the fire mode is switched off. Note that instead of an OR, an AND function is used for the condition.
The second Intersection uses a copy of this sequencer with the difference that, to synchronize it to the first one, a variable is added to the condition of the first transition. A fire mode is also not implemented.

Figures 42: Sequencer1 step 105/106

Figures 43: Sequencer4 Step1

4.3.2 Sequencer 2 Pedestrian Crossing

The same approach is repeated for the pedestrian light between the two intersections with a small difference, since the traffic light is button operated and not used at night. As soon as the button is pressed the car traffic is stopped and pedestrians can cross. A waiting period has to be met after a cycle to make sure traffic is not disrupted. Also it has to be possible to synchronize the crossing, which will be done with sequencer 6.
4.3.3 Sequencer 3/5 Intersection 1/2 Night

The night mode of the intersection simply requires the yellow lights of the side street to flash. The fire or synchronized mode have no effect at night, so the only function that has to be implemented is the “INT1” variable that makes sure the day cycle finishes before switching to night mode.

4.3.4 Sequencer 6 Synchronized Mode

This mode will model a synchronization of the three traffic lights for the car traffic travelling to the right. It is achieved through variables connected to the condition of transitions at the beginning of the pedestrian crossing and intersection 2. Intersection 1 serves as the pulse generator.

First Step601 sets “Xingsync” and “int2sync”, the variables that control the timing of the pedestrian crossing and the second intersection, as “true” to allow sequencers 2 and 4 to be executed. The rest of the sequencer is executed as soon as “PowerButtonSyncRight” is set “true”. “Xingsync” and “int2sync” are then immediately set “false” to stop the respective sequencers at their beginning.
Since “INT1” is set to “false” for a short moment, every time sequencer 1 is executed from the beginning, it can be used as a pulse source. To know when the sequencer 2 and 4 can be started, the travel time from intersection to intersection is set to 10 seconds to simulate a distance of roughly 140m. This means “int2sync” is set to “true” 10s later to start sequencer 4 (intersection 2).

Some more thought is necessary for the pedestrian crossing. Since it is button operated, there will be a time window, in which it is possible to switch the car traffic light to red without interrupting the synchronized traffic. By looking at both status shows that the sequencer 2 (pedestrian crossing) may be started 16 to 27 seconds later than sequencer 1, giving a window of 11s. The implementation can be seen below.

The full sequencer can be viewed in the full program attached as appendix 3. This concludes the programming instructions.
5 DISCUSSION

This course is a way to teach automation and German in an online environment. The high amount of group work, especially between students from different countries and schools makes it an experience that stands out from the usual university course.

Students who are introduced to the subject while having to apprehend tasks presented in a foreign language now have another tool at hand to improve their understanding. All material is tailored for beginners in plc-programming, however advanced German skills are required.
BIBLIOGRAPHY


APPENDICES

Appendix 1. Video descriptions

This appendix contains video descriptions, of the videos attached on DVD, in German language. The videos were created by Mira Kiiski and are not part of this thesis. The video descriptions are added to the course module four to explain the content of the videos that are attached on a separate medium. Video 6 did not require a written explanation.
1.1 Video 1: Öffnen und konfigurieren eines neuen Projektes.

Opening and configuration of a new project.

Starten des Simatic Managers. Schließen des Wizard-Fensters durch drücken von Cancel. Über die Schaltfläche File –> New... wird ein neues Projekt erstellt, es muss nur der Name und Speicherort des Projektes gewählt werden. Da das Programm auf einer Simatic 300 Station läuft, wird über Insert –> Station –> Simatic 300 Station die Station eingefügt. Die Station kann durch einen Doppelklick auf Simatic 300(1) im Explorer links gewählt werden. Die Hardwarekonfiguration wird durch einen Doppelklick auf Hardware erreicht. Es öffnet sich das Fenster SIMATIC 300(1) (Configuration) – Ampelanlage. Hier kann das Modell der SPS ausgewählt werden. Erst muss im Explorer auf der rechten Seite unter Rack-300 -> Rail ausgewählt werden, da es die Grundlage für die SPS darstellt. Die Rail wird dann als Tabelle dargestellt. Die CPU wird in Spalte 2 installiert, also muss diese ausgewählt werden. Danach kann die richtige CPU im Explorer unter CPU-300 -> CPU 314-2 DP -> 6ES7 314-6CF00-0AB0 durch einen Doppelklick eingefügt werden. Das sich öffnende Fenster Properties kann durch drücken von OK bestätigt werden. Das CPU-Modul enthält genug Ein- und Ausgänge, es werden also keine weiteren Bauteile benötigt. Die Hardware ist somit konfiguriert, das Fenster Configuration kann geschlossen werden. Die Programmierung läuft in Organization Blocks ab, im Simatic Manager unter Simatic 300(1) -> CPU 314C-2 DP -> S7 Program(1) -> Blocks befindet sich bereits OB1. Durch einen Doppelklick auf OB1 öffnet sich erst das Fenster Properties, wobei zu beachten ist, dass unter Created in Language: FBD eingetragen ist. Durch bestätigen mit OK wird der Block zur Programmierung geöffnet.

1.2 Video 2: Zuweisung der Ein- und Ausgänge.

Allocation of in- and outputs.

1.3 Video 3: Programmieren mit Function Block Diagram (FBD)

Bei der Programmierung mit FBD werden logische Operationen als Blöcke dargestellt und über Ein- und Ausgänge verbunden. Das gesamte Programm ist unterteilt in einzelne Networks welche je eine logische Funktion enthalten. Für das Programm Ampelanlage müssen einige Networks programmiert werden. Ein neues Network kann über Insert -> Network oder die Schaltfläche in der Toolbar eingefügt werden. Im Network können logische Blöcke durch Auswählen der Position im Network und Klicken auf die Symbole And Box oder Or Box in der Toolbar eingefügt werden. Soll eine Variable einem Eingang zugewiesen werden, so kann über Rechtsklick auf die gewünschte Position Insert Symbol ausgewählt werden, was eine Liste aller möglichen Variablen öffnet. Ausgänge logischer Blöcke können über den Block Assignment Variablen zugewiesen werden.

Weitere Blöcke können in der Box Library auf der linken Seite des Fensters gefunden werden. Für das Programm wird ein Timer benötigt, der unter Timers -> S_ODT gefunden werden kann.

1.4 Video 4: Simulieren des Betriebes.

Simulation of Operation.

Der Programmablauf kann mit dem SimView Programm simuliert werden. Gestartet wird das Programm über die Schaltfläche Simulation On/Off im Simatic Manager. Im SimView können alle Fenster außer CPU geschlossen werden. Über Insert werden die Fenster Input und Output Variable eingefügt. Im Textfeld IB bzw QB in den Fenstern kann die Adresse der Variablen angegeben werden, in diesem Fall 125. Aktivieren der Schaltfläche Always On Top vereinfacht die Bedienung, da auch im Fenster OB gearbeitet wird.

Um das Programm auf die simulierte CPU zu übertragen, muss im Fenster OB die Schaltfläche Download aktiviert werden. Zum Starten des Programmes wird im CPU Fenster in SimView der Haken auf RUN gesetzt. Durch Aktivieren der Schaltfläche Monitor (on/off) werden die Zustände der Simulation im OB Fenster dargestellt. Die Zustände der Eingänge können durch Setzen des Hakens im IB Fenster geändert werden.
1.5 Video 5: Archivieren eines Projektes.

Archiving a Project.

Das Programm kann im Simatic Manager archiviert werden. Dazu muss im Simatic Manager Fenster File -> Archive... ausgewählt werden und in der Liste im Archiving Fenster das aktuelle Projekt ausgewählt werden. Es öffnet sich ein Explorer Fenster, in dem der Speicherort ausgewählt werden kann.

1.6 Video 7: Übertragen des Programmes auf die SPS

Copying the program onto the PLC.


Nach erneutem Download des Programmes im OB Fenster wird das Programm auf die SPS übertragen.
Appendix 2. Quizzes

This appendix contains a copy of the moodle page containing the quizzes on the German video instructions and the background information. It is separated into two sections that belong to different modules of the course. The quizzes are copied from the editor in moodle.tamk.fi where they were created.

Appendix 2.1
The first two pages contain the five questions about the video instructions in German language.

Appendix 2.2
The pages three to eight (labeled one to six) contain the 26 questions on chapter 4.2 in English.
Preview Test 1

1. Wähle die richtigen Antworten aus.

   Marks: 1

   Choose one answer.

   a. Es werden ein CPU-Modul und weitere Ein- und Ausgänge benötigt.

   b. Der Organization Block ist für die Programmierung.

   c. Es muss ein Organization Block eingefügt werden.

   d. In der Konfiguration kann die CPU auf allen Spalten der Rail installiert werden.

2. Choose one answer.

   Marks: 1

   a. Variablen müssen durch Namen beschrieben werden.

   b. Die Variablen werden in der Symbol Table Ein- und Ausgängen zugewiesen.

   c. Mehrere Variablen können nicht nacheinander eingetragen werden.

   d. Der Name der Variablen wird unter Address eingetragen.

3. Choose one answer.

   Marks: 1

   a. Im Network können nur "And Box" und "Or Box" eingefügt werden.

   b. Networks enthalten das gesamte Programm.
c. Der Befehl "Insert Symbol" fügt Sonderzeichen ein.
d. Logische Operationen sind in FBD dargestellt durch Blöcke.

4 📜
Marks: 1 Choose one answer.

- a. Weitere Elemente befinden sich in der Box Library.
- b. Eine Box ist immer einer Variable zugewiesen.
- c. Die Toolbar zeigt alle verfügbaren Elemente.
- d. Variablen werden einer Box in der Symbol Table zugewiesen.

5 📜
Marks: 1 Choose one answer.

- a. SimView zeigt alle Input und Output Variablen in einer Tabelle.
- b. Die im Programm verwendeten Variablen werden automatisch angezeigt.
- c. SimView dient zur Ansicht des Programmes.
- d. SimView kann den Programmablauf ohne Hardware simulieren.
Preview Test 1

1. Marks: 1 Choose one answer.
   a. Ladder logic is based on technical drawing.
   b. Ladder logic is based on relay logic.
   c. Ladder logic is based on graphic design.

2. Marks: 1 Choose one answer.
   a. The program contains coils, boxes and contacts.
   b. The program contains boxes, coils and sensors
   c. The program contains contacts, boxes and relays.

3. Marks: 1 Choose one answer.
   a. A contact is always connected.
   b. A contact is used as an output.
   c. A NO contact is normally closed.

4. Marks: 1 Choose one answer.
   a. A NOT element stops the power flow.
   b. A coil connects two elements.
c. A coil connects to an output.

5 Marks: 1 Choose one answer.
- a. A AND function is always made out of two contacts.
- b. A AND function can be made with one contact.
- c. A AND function is made out of contacts in a row.

6 Marks: 1 Choose one answer.
- a. The Value of a AND function with one Contact closed is "1".
- b. A OR function is always made of parallel connected contacts.
- c. A OR function is true only when both contacts are connected.

7 Marks: 1 Choose one answer.
- a. If a reset coil is under power the addressed value is set to "0"
- b. If a set coil is under power the addressed value is set to "0"
- c. If a reset coil is under power the addressed value is set to "1"

8 Marks: 1 Choose one answer.
- a. A SR box has two inputs
- b. A RS Box has two outputs
- c. A SR Box has two outputs.

9 Marks: 1 Choose one answer.
- a. Edge evaluation is used to determine the length in time of a positive value.
- b. A move funktion changes a data address.
- c. A midline output is used to connect to a output adress in the middle of a network.
10 Marks: 1 Choose one answer.
- a. A S_ODT timer always produces the same length pulse.
- b. A S_Pulse timer always produces the same length pulse.
- c. A S_PEXT timer always produces the same length pulse.

11 Marks: 1 Choose one answer.
- a. The FBD programming language uses the same elements as LAD.
- b. The FBD programming language is a part of LAD.
- c. The FBD programming language layout is similar to LAD.

12 Marks: 1 Choose one answer.
- a. Function blocks can perform binary logic only.
- b. Function blocks can perform complex non-binary functions only.
- c. Function blocks can perform simple binary logic or complex non-binary functions.

13 Marks: 1 Choose one answer.
- a. The user can define new function blocks.
- b. A function block has one input and one output.
- c. Each network contains only one function block.

14 Marks: 1 Choose one answer.
- a. A inverted input is marked with a small circle.
- b. A inverted input is marked with a small cross.
- c. A inverted input is marked with a small square.

15 Marks: 1 Choose one answer.
- a. The XOR function can be replaced by two OR and one AND box.
answer.

- b. The XOR function can be replace by three OR boxes.
- c. The OR element is labeled >=1

16 🟢
Marks: 1 Choose one answer.

- a. The R-box reads a stored value.
- b. The S-box sets a value.
- c. The S-box reads a value.

17 🟢
Marks: 1 Choose one answer.

- a. Set input is dominant in a RS-box.
- b. Reset input is dominant in a SR-box.
- c. Set input is dominant in a SR-box.

18 🟢
Marks: 1 Choose one answer.

- a. GRAPH is used to display content graphicly.
- b. GRAPH is used to program sequences.
- c. GRAPH is used to program mathematic solutions.

19 🟢 Choose the correct order.
Marks: 1

Choose one answer.

- a. Sequencer - Function Block - Action - Step
- b. Function Block - Sequencer - Step - Action
- c. Sequencer - Step - Action - Function Block

20 🟢
Marks: 1 Choose one answer.

- a. Jumps can connect different sequences.
- b. A function block contains one sequence.
- c. Transitions control single actions.
21
Marks: 1 Choose one answer.
  a. Each step contains any number of transitions.
  b. Each step contains a number of actions.
  c. Each step contains any number of conditions.

22
Marks: 1 Choose one answer.
  a. Transitions are LAD networks.
  b. Actions control the condition of a transition.
  c. The condition of a transition is a LAD or FBD network.

23
Marks: 1 Choose one answer.
  a. The action "R" reads the value of a variable.
  b. The action "N" sets and resets a variable.
  c. The action "S" starts a sequencer.

24
Choose the correct time format for a "on delay" action.
Marks: 1
Choose one answer.
  a. T#17S5MS
  b. T15MS7S
  c. T7S15MS

25
Marks: 1 Choose one answer.
  a. The alternative branch can be inserted behind step or transition.
  b. The alternative branch is inserted behind the step.
  c. The alternative branch is inserted behind the transition.

26
Marks: 1 Choose one
  a. Only one of alternative branches is executed.
answer.

- b. Only one of simultaneous branches is executed.
- c. All alternative branches can be executed at the same time.
Appendix 3. Program Traffic Lights

This appendix contains a print of the program created as explained in chapter 5.

Pages one to three contain the FBD program as shown in chapter 5.2.

Pages four to eleven (labelled one to eight) contain the GRAPH program as shown in chapter 5.3.
FB1 - <offline>
**
Name: 
Familie: 
Autor: 
Version: 0.1 
Bausteineversion: 2 
Zeitstempel Code: 
10.09.2012 15:22:40 
Interface: 
19.07.2012 15:40:07 
Längen (Baustein / Code / Daten): 00204 00098 00000

<table>
<thead>
<tr>
<th>Name</th>
<th>Datentyp</th>
<th>Adresse</th>
<th>Anfangswert</th>
<th>Kommentar</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN_OUT</td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAT</td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEMP</td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Baustein: FB1

Netzwerk: 1  Button Day

Netzwerk: 2  Button Day
Netzwerk: 3  cycle switch xing

Netzwerk: 4  Button Sync Right

Netzwerk: 5
Netzwerk: 6  Button Sync Right

Netzwerk: 7
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step301</td>
<td><strong>Intersection1 Night</strong></td>
<td>R &quot;INT1&quot;</td>
</tr>
<tr>
<td>Step302</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step303</td>
<td></td>
<td>R &quot;PTL1R&quot; R &quot;PTL2G&quot; R &quot;PTL2R&quot;</td>
</tr>
<tr>
<td>Step304</td>
<td></td>
<td>N &quot;CTL2Y&quot; D &quot;delay&quot;</td>
</tr>
</tbody>
</table>

**Delay**

- INT1
- PowerButtonOnNight
- delay
Intersection 2 Day

Step 401

T401

Trans 4 01

Step 402

T402

Trans 4 02

Step 403

T403

Trans 4 03

Step 404

T404

Trans 4 04

Step 405

T405

Trans 4 05

Step 406

T406

Trans 4 06

Step 407

T407

Trans 4 07

Step 408

T408

Trans 4 08
Intersection 1 Day
S "Xingsync"
S "Int2sync"

Step80
R "Xingsync"
R "Int2sync"

Step603
D "delay4"
T#10S

Step77
S "Int2sync"
D "delay4"
T#6S

Step605
S "Xingsync"
D "delay4"
T#11S