Symphony Plus as application for power plants

AC500 Subproject

Daniel Hummel

Bachelor’s thesis
Electrical Engineering
Vaasa 2014
ABSTRACT

This thesis work is in its entirety a project consisting of two parts made to study and evaluate the functionality of the S+ Operations in combination with the AC500 PLC. This thesis covers the part of the AC500 PLC, developed by ABB.

It provides information on programming of a demo process and the applications associated with the AC500 that are used to achieve a functioning demo process. The demo process is used to test the functionality in combination with S+ Operations while evaluating the features and characteristics of the PLC. The Water & Wastewater library is used to provide extended programming content for the demo solution. The actual programming is done in PS501 Engineering Tool.

A comparison with another system is made to weigh characteristics and solutions against each other. The comparison is made on the basis of a system overview from an engine power plant. A solution with AC500 as an application in a power plant is presented and discussed based on the comparison.

The end result is a working demo process that is controlled using the AC500 in combination with S+ Operations, which was also considered to be an overall flexible solution. From the demo process it is possible to continue to develop and test the functionality with AC500.

Language: english  Keywords: AC500, PLC
EXAMENSARBETE

Författare: Daniel Hummel
Utbildningsprogram och ort: Elektroteknik, Vasa
Inriktning: Automationsteknik
Handledare: Ronnie Sundsten

Titel: Symphony Plus som applikation för kraftverk - AC500 delprojekt

10.04.2014 41 sidor 28 Bilagor

ABSTRAKT

Detta examensarbete är i sin helhet ett projekt bestående av två delar, utförda för att studera och evaluera funktionaliteten vid S+ Operations i kombination med AC500. Detta examensarbete omfattar delen för PLC:n AC500, utvecklad av ABB.


Slutresultatet är en fungerande demoprocess som styrs med hjälp av AC500 i kombination med S+ Operations med god funktionalitet. Från demoprocessen är det möjligt att fortsätta vidareutveckla och testa funktionaliteten med AC500.

Språk: engelska  Nyckelord: AC500, PLC
## Table of contents

1 Introduction ................................. 1
  1.1 Target ................................... 1

2 ABB ........................................... 2
  2.1 In Finland ................................. 2
  2.2 Power Generation ......................... 2

3 Programmable Logic Controllers .......... 3
  3.1 History .................................. 3
  3.2 Functionality ............................ 3

4 CS31 ......................................... 4
  4.1 Technical data ............................ 4
  4.2 In practice ................................ 4
  4.3 Restrictions ................................ 5

5 AC500 ......................................... 6
  5.1 Introduction .............................. 6
  5.2 Overview ................................ 7
  5.3 Technical data and features ............ 8
  5.4 References ................................ 9
    5.4.1 Sewage Treatment Plant, China ..... 10
    5.4.2 Desalination Plant, Israel .......... 10
    5.4.3 Water Reuse Treatment Plant, China 11
  5.5 Summary .................................. 11

6 PS501 Engineering Tool .................... 12
  6.1 Control Builder Plus ...................... 12
    6.1.1 Hardware configuration ............. 12
    6.1.2 Parametrization ..................... 13
    6.1.3 Diagnostics ......................... 13
  6.2 CoDeSys .................................. 13


# Table of Contents

## 6.2 Languages
- Languages .................................................. 13

## 6.2 Programming
- Programming ..................................................... 13

## 6.3 CoDeSys with other brands
- CoDeSys with other brands .......................................... 14

## 7 Water & Wastewater
- Functionality .................................................. 15
- Function blocks ................................................. 16
  - Hardware interface ........................................... 17
  - Application interface ...................................... 18
  - HMI interface ................................................ 18
  - FB_Motor1_1 .................................................. 18
  - FB_Motor2_1 .................................................. 18
  - FB_Valve1_1 .................................................. 19
  - FB_Valve2_1 .................................................. 19
  - FB_Transmitter1_1 .......................................... 19
  - FB_Alarm1_1 .................................................. 20
  - FB_AlternationTime1 ........................................ 20
  - FB_AlterationPri2_1 ........................................ 20
  - FB_LimitControl1_1 ......................................... 21
  - FB_LimitControl2_1 ......................................... 21
  - FB_Actuator1_1 .............................................. 21
  - FB_TimeData1 ................................................ 21
  - FB_OperatingData1_1 ....................................... 22
  - FB_Accumulator1_1 .......................................... 22
  - FB_Weir1_1 .................................................. 22
  - FB_Inflow1_1 ................................................ 23

## 7.3 Implementation ............................................. 23

## 8 Project X - Using AC500
- Introduction ................................................... 24
- Communication ................................................. 24
  - Siemens ..................................................... 24
  - ABB .......................................................... 24
- Common Control Panel ......................................... 25
  - Siemens ..................................................... 25
  - ABB .......................................................... 25
- Generator Control Panel 1-5 ................................... 26
  - Siemens ..................................................... 26
  - ABB .......................................................... 26
- HMI .............................................................. 27
  - Siemens ..................................................... 27


<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5.2</td>
<td>ABB</td>
<td>27</td>
</tr>
<tr>
<td>8.6</td>
<td>Summary</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>Creating a Project</td>
<td>28</td>
</tr>
<tr>
<td>9.1</td>
<td>Project</td>
<td>28</td>
</tr>
<tr>
<td>9.2</td>
<td>Overview</td>
<td>28</td>
</tr>
<tr>
<td>9.3</td>
<td>Hardware installation</td>
<td>29</td>
</tr>
<tr>
<td>9.4</td>
<td>Communication protocols</td>
<td>29</td>
</tr>
<tr>
<td>9.5</td>
<td>Hardware configuration</td>
<td>30</td>
</tr>
<tr>
<td>9.6</td>
<td>Software Configuration</td>
<td>32</td>
</tr>
<tr>
<td>9.7</td>
<td>Summary</td>
<td>35</td>
</tr>
<tr>
<td>10</td>
<td>Results</td>
<td>36</td>
</tr>
<tr>
<td>11</td>
<td>Discussion</td>
<td>38</td>
</tr>
<tr>
<td>12</td>
<td>Bibliography</td>
<td>40</td>
</tr>
</tbody>
</table>

Appendices
List of Figures

3.1 PLC functionality ................................................. 3
4.1 Overview of bus topology in practice [5] ........................ 4
4.2 Connecting bus on AC500 terminal base [5] ...................... 5
4.3 CS31 module connection [5] ..................................... 5
5.1 AC500 [8] .......................................................... 6
5.2 AC500 Centralized system [7] .................................... 7
5.3 AC500 Terminal base and CPU .................................. 8
5.4 CPU processing time comparison [4, 15, 14] ..................... 9
5.5 Load and work memory comparison. (observe different symbols of measure) [4, 15, 14] ......................... 9
6.1 PS501 Engineering Tool ........................................... 12
7.1 Structure [3] ....................................................... 15
7.2 Functionality of the hardware interface. [3] ...................... 17
7.3 FB_Motor1_1 [3] ............................................... 18
7.4 FB_Motor2_1 [3] ............................................... 18
7.5 FB_Valve1_1 [3] ................................................ 19
7.6 FB_Valve2_1 [3] ................................................ 19
7.7 FB_Transmitter1_1 [3] .......................................... 19
7.8 FB_Alarm1_1 [3] ................................................ 20
7.9 FB_AlternationTime1 [3] ....................................... 20
7.10 FB_AlternationPri2_1 [3] .................................... 20
7.11 FB_LimitControl1_1 [3] ...................................... 21
7.12 FB_LimitControl2_1 [3] ...................................... 21
7.13 FB_Actuator1_1 [3] ............................................ 21
7.15 FB_OperatingData1_1 [3] .................................... 22
7.16 FB_Accumulator1_1 [3] ...................................... 22
7.17 FB_Weir_1 [3] ................................................ 22
7.18 FB_Inflow1_1 [3] .............................................. 23
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Communication</td>
<td>24</td>
</tr>
<tr>
<td>8.2</td>
<td>Common Control Panel</td>
<td>25</td>
</tr>
<tr>
<td>8.3</td>
<td>Engine panels 1-5</td>
<td>26</td>
</tr>
<tr>
<td>8.4</td>
<td>Engine power plant operator stations</td>
<td>27</td>
</tr>
<tr>
<td>9.1</td>
<td>Watertank process</td>
<td>29</td>
</tr>
<tr>
<td>9.2</td>
<td>Hardware installation</td>
<td>29</td>
</tr>
<tr>
<td>9.3</td>
<td>Opening CoDeSys</td>
<td>31</td>
</tr>
<tr>
<td>9.4</td>
<td>I/O setting in HW_PRG</td>
<td>32</td>
</tr>
<tr>
<td>9.5</td>
<td>PLC_PRG</td>
<td>32</td>
</tr>
<tr>
<td>9.6</td>
<td>Calling CALLBACK_STOP using system task configuration</td>
<td>32</td>
</tr>
<tr>
<td>9.7</td>
<td>High availability function blocks</td>
<td>33</td>
</tr>
<tr>
<td>9.8</td>
<td>Modbus function blocks</td>
<td>33</td>
</tr>
<tr>
<td>9.9</td>
<td>Analog signal scale configuration values</td>
<td>33</td>
</tr>
<tr>
<td>9.10</td>
<td>Scale configuration to signal</td>
<td>34</td>
</tr>
<tr>
<td>9.11</td>
<td>Transmitter block</td>
<td>34</td>
</tr>
<tr>
<td>9.12</td>
<td>Web visualization of CPU load</td>
<td>34</td>
</tr>
</tbody>
</table>
LIST OF ABBREVIATIONS

PLC  Programmable Logic Controller
CPU  Central Processing Unit
CRC  Cyclic Redundancy check
CBP  Control Builder Plus
ST   Structured Text
IL   Instruction List
LD   Ladder Diagram
FBD  Function Block Diagram
SCL  Structure Control Language
CPU  Central Processing Unit
PMU  Power Monitoring Unit
AVR  Automatic Voltage Regulator
I would like to thank Mr. Ronnie Sundsten at Novia University of Applied Sciences and Mr. Frank Redlig at ABB, who have been my supervisors during this thesis work. They have provided me with guidance and support throughout this project.
1. Introduction

This Bachelor’s thesis is conducted on behalf of the ABB Power Generation department in Vaasa. I have been given the task to participate in research on Symphony+ operations in combination with AC500 as a solution for engine power plants. This thesis focuses on AC500, and describes all essential parts necessary to know for readers to understand the outcome and discussion of this thesis. Mr. Anton Wargh is responsible for the S+ operations part of this investigation.

1.1 Target

The purpose of this thesis is to investigate Symphony Plus Operations in combination with AC500, using the Water & Wastewater library, as a solution for power plants. This thesis focuses on AC500 and the goal has been to develop a demo solution of a watertank process, where the functionality of the AC500 can be tested and further developed. This has been done through research, consultation with experts and tests done with the Demo solution. This solution will be an example of how to configure AC500 for establishing a connection between S+ Operations and AC500 using OPC. It is also used to determine the possible area of use for the AC500 in power plants and compare it to existing solutions and thereby determine advantages and disadvantages as well as potential improvement areas regarding S+ Operations in combination with AC500, and with AC500 in general.
2. ABB

ABB is globally known for being leaders in power and automation technologies. Currently ABB is based in Zurich, Switzerland, but the company is active in approximately 100 countries, with about 150,000 employees.

The company’s current form was created in 1988 and comprises five divisions that are in turn organized in relation to the customers and industries that are being served. This form was established through a merger between ASEA and BBC, which are both electrical companies established before the 20th century and are responsible for innovative solutions in areas like turbines, transformers, switchgears, robots etc. [6]

2.1 In Finland

ABB in Finland can be found in over 30 locations with about 5,500 employees. ABB is also one of Finland’s biggest employers in the industry sector with a revenue of 2.3 billion euro of which 184 million is used on research and development.

The ABB organization in Finland consists of Discrete Automation and Motion, Low Power Products, Process Automation, Power Systems and Power Products, and their respective sub-units. [2]

2.2 Power Generation

Power Generation is a part of the ABB Power Systems division, which consists of Power Generation, Substations and Network Management. The Power Generation unit focuses on planning and delivering power plants as turnkey solutions. The Power Generation unit in Finland is located in Vaasa and specializes in gas, gas turbine, hydro, thermal and nuclear power plants. The hydropower unit has its biggest focus in the Nordic countries. [1]
3. Programmable Logic Controllers

The AC500 is a PLC (Programmable logic controller) and therefore it is necessary to know the basics of the functionality of the PLC to be able to understand later chapters of this thesis. The features and functions of newer PLCs have changed since the literature references for this chapter were written, but the principle of the PLC is generally the same.

3.1 History

Programmable logic controllers, or PLCs as they are referred to in the industry, have since 1969 become the most popular mean of controlling machinery and plant operations. These small logics would replace long cabinets of relays and wiring that were used earlier. Microprocessors have been used as the brain of the PLC since around 1974 and from that evolved together with the advances in the electronics industry to provide powerful and reliable PLCs.[13]

3.2 Functionality

The PLC system consists of a CPU which contains the microprocessor that interprets the input signals and, according to programs stored in its memory it carries out control actions that communicate decisions as signals to the outputs. To have an interface between the system and the outside world the PLC needs an input and output section, where it receives information from external devices and also communicates it back to external devices. For the PLC to have an understanding of what to do it needs a programming device and a memory unit. The programming device is used for inserting the project specific program into the memory unit of the PLC. The memory unit is then used by the microprocessor for control actions, also input and output data are stored in this area. Lastly, a power supply is needed for the processor and interface modules. [9]

![Figure 3.1: PLC functionality](image)
4. CS31

This chapter describes the features of the communication protocol CS31 with AC500 using RS458 as a transmission medium. This bus protocol was developed by ABB in 1989 and is widely used in AC500 solutions. It is provided as an onboard interface with most AC500 CPUs. This chapter should give the reader an input in bus characteristics that will be necessary to know to fully understand later chapters.

4.1 Technical data

The CS31 bus uses mostly RS485 (twisted pair, with shield) for communication, but can be used with fiber optic cables (requiring a converter) or contact lines and slip rings. It should be noted that bus characteristics may differ from RS485 when using other types of transmission mediums. There is a limit of 31 modules that can function as slaves on the bus. The master handles communication with slaves using polling, which means that it sends a request to the slave and receives a response. The maximal length of a busline is 500m, or with repeaters it can extend to 2000m. The baudrate used is 187,5 kB/s with an 8-bit CRC appended to each telegram. This enables process input/output data to be written and read. [10]

4.2 In practice

In practice when the AC500 PLC is used as a CS31 Master, the busline is connected on COM1 interface of the PLC terminal base as seen in the following pictures. These configurations may differ from module to module, but these are taken from those used in chapter 9 and are very general. [5]
4.3 Restrictions

One mentionable restriction when using the CS31 protocol is in decentralized systems with the slave module. The Slave module can be viewed as a cluster with multiple attached I/O modules. The maximum amount of I/O modules connected to a slave module depends on the used CS31 bus module. The different bus modules may be specified for a given number of I/O module extensions, and a cluster with maximal configuration, which is a CS31 module with the maximum amount of I/O modules attached, can occupy two addresses. This means that if the first CS31 bus module is located at address 2, then the following has to be set at address 4.

However, getting maximal configuration on a CS31 cluster does not necessarily mean that all I/O module slots are used. This occurs when I/O modules exceed the amount of digital or analog subscribers. This can either be manually counted or viewed using a tool like Control Builder Plus. These restrictions may differ from module to module. [5]
This chapter describes the features and benefits of the ABB AC500 PLC. To prove the goal of this thesis it is crucial to be familiar with the AC500 PLC, and therefore it is a necessary part to include. The chapter describes the basics of the AC500 that are needed to fully understand later chapters. Some references where AC500 have been used are also shown and discussed to give a better grasp of an example area of use. The AC500 series can be found with different special features such as:

- AC500 - With or without internal Ethernet coupler.
- AC500 eCo - Budget PLC.
- AC500 S - Offers safety features for critical applications.
- AC500 XC - Offers state of the art technology for extreme conditions.

This chapter focuses on the AC500 with internal Ethernet coupler.[4]

5.1 Introduction

The AC500 PLC and the whole AC500 family consist of modules that are easily combined and scaled to fit descriptions given by customers. There are always expansions made throughout the lifetime of power plants and other plant industries. Most AC500 CPU modules are installed on the same terminal base, so when upgrading, only the CPU module has to be replaced for a more powerful version. This leads to minimal maintenance and downtime.

AC500 offers high availability, which can be described as warm standby redundancy without cycle synchronization (Software-layer redundancy). This is discussed and described further in chapters 10 and 11. All AC500 CPU modules offer a display for setting mostly communication addresses but also different start-up modes. Errors and diagnostics are shown on the display allowing fast troubleshooting.[4]
5.2 Overview

![AC500 Centralized system](image)

AC500 as a centralized system offers very efficient scalability of projects and ease of use. As seen in figure 5.2, the green highlighted module refers to the CPU unit. The CPU unit itself as seen in figure 5.1 has to be mounted on a terminal base and supplied with 24VDC (figure 5.3 Power). The terminal bases for CPU units can have one, two or four slots for communication modules to be mounted on. All modules have to be mounted on terminal bases. Centralized I/O modules are connected with the CPU module through connectors on the terminal base.

Communication interfaces for the AC500 is possible using common open industrial networks via Ethernet, PROFINET, EtherCAT, ARCNET, Profibus, CANopen, DeviceNet, Modbus and CS31.

Using a centralized I/O rack, the maximum amount of modules supported by AC500 is ten. The amount of decentralized I/O racks is restricted by the used fieldbus type. The AC500 does not support hot swap when replacing I/O modules.

A lithium battery and SD-memory card are not supplied with the CPU and does not need to be used. It is however a recommended solution to use both a lithium battery and an SD-memory card. The lithium battery is used for saving RAM content and as a back-up for the real-time clock. The SD-memory card is used for updating CPU firmware, storing user programs and as a back-up of user data.

AC500 modules offer Ethernet communication with TCP/IP and UDP/IP protocols through the internal Ethernet coupler and the RJ45 connector located on the terminal base. These can be used simultaneously.[4]
5.3 Technical data and features

Figure 5.3: AC500 Terminal base and CPU

- 1. Battery Slot
  - Save RAM content
  - Back-up real-time clock
- 2. SD-Card Slot
  - Back-up user data
  - Store user programs
- 3. FieldBusPlug
  - Profibus DP (slave)
  - CanOpen (slave)
  - DeviceNET (slave)
- 5. Ethernet RJ45
  - Programming
  - Internet Protocols (webserver, FTP, e-mail, time sync, etc.),
  - IEC 60870-5-104
- 6. COM 2 - serial
  - Programming
  - ASCII protocol
  - MODBUS-RTU (master or slave)
- 7. COM 1 - Spring terminal
  - Programming
  - CS31 Bus (master)
  - ASCII protocol
  - MODBUS-RTU (master or slave)

All CPUs of the AC500 family are equipped with the same features, but with an exception for some models like PM572 and PM582 that don’t offer Web server’s data for user RAM disc. PM592-ETH, which is the most powerful module, is the only CPU offering User flashdisc, stated in technical documents as 4GB Flash nonremovable used for Data-storage, program access or FTP functions. The amounts of integrated memory and process time are features that improve when changing from an inferior to superior CPU.[4]
The following comparison is done between AC500 and Siemens S7 CPUs with values gathered from their respective data sheets. An "average" model from each vendor was chosen as well as one of the more powerful versions.

As seen in the figure above, the processing times for PM583-ETH and CPU3125-2 show minimal differences when compared, whereas the PM592-ETH from ABB is multiple times faster than the others. The CPU416-5H, which is a powerful Siemens CPU, does not reach the same processing time as the PM592-ETH, but exceeds the other ABB and Siemens CPU multiple times.

A memory comparison is made using the same devices. The memory type is stated differently from both vendors. In ABB data sheets stated as Integrated User Data Memory is compared to in Siemens data sheets stated as Integrated (for data) for work memory size. Load memory is stated in ABB data sheets as Memory Size User Program and in Siemens integrated (for program).

As seen in the figure above, again the same proportions of difference can be noted. It should be noted that the memory size used for CPU416-5H is the integrated value and can be expanded using a RAM memory card up to 64 Mbyte.

5.4 References

The AC500 family is a fairly "new" line of PLCs on the market and has therefore not been used in such a large scale as PLCs that have a longer product history, which due to their time on the market have established bigger "communities" for knowledge and support. Taking this into consideration it can be assumed that ABB customers should be introduced to a solution using AC500 to prove their features and benefits. This section of references is included to show one area of use where PLCs from the AC500 family have been used by ABB as a new solution or as a replacement for existing solutions. All these references comprise different water and wastewater solutions.
5.4.1 Sewage Treatment Plant, China

- **Design Request**
  - Full open system, run steadily
  - High control precision
  - Integrate & maintain easily based on module solutions
  - Programmed easily, fit on sequence flow control

- **Project Introduction**
  - Occupies < 26 km$^3$
  - First step 20 km$^3$/day, in future 40 km$^3$/day

This was a new solution and based on the design request and the project introduction two AC500 PM581–ETH are used, both with four DI524-32DI, two DC532-24DC and one AI523-16AI. Both PLCs were installed in the power distribution room and communicate with SCADA using Modbus TCP/IP. The AC500 PM581–ETH fulfills high-speed computing requirements of the sewage treatment and the system that is fitted on a sequence flow chart can be developed by engineers using PS501 open programmable environment. It also fulfills the communication equipment and network requirements. Moreover it has passed a variety of international standard certifications considering sewage treatment plant environment.

5.4.2 Desalination Plant, Israel

- **Design Request**
  - Energy saving solution.
  - Multicontrol PLCs for reverse osmosis desalination process.

- **Project Introduction**
  - Older Plant controlled by Modicon PLCs with redundant CPUs, bus lines
  - Desalination facility utilizes Reverse Osmosis
  - It will produce 100 million m$^3$ fresh water per year

In this project AC500 was used as a replacement for an old design and based on the new design request and project introduction, the project realization involve 120 × AC500 PLCs, AC drives and a solution based on Microsoft Dot Net and XML technology. One of the challenges was to use relatively independent PLC groups with separate PLCs for each task instead of using a small amount of PLCs with redundant CPU and bus lines. Competitive pricing position, energy savings and reduced maintenance costs are beneficial to the customer as well as improved membrane life thanks to pressure regulation.
5.4.3 Water Reuse Treatment Plant, China

- Design Request
  - Communication
  - Stability
  - Compatibility

- Project Introduction
  - Wastewater treatment for economic development area
  - Former existing solution: Siemens S7-300

Based on the design request and project introduction, the project configuration consists of two different stations. The PLC main station with a PM581 CPU and several I/O modules, which is used to coordinate to remote station controllers and the Chemical Dosing Station. These are linked by industrial switches for high speed data exchange on site. There are totally 6 remote stations equipped with a PM581 CPU and several I/O modules. The benefits for the customer are flexible options for communication integration with Phase I Siemens system, as well as a common programming environment - PS501. The use of CS31 bus for decentralization is another of the technical benefits.

5.5 Summary

The previous section is gathered from ABB AC500 success stories and these three were picked to display that AC500 has been successfully used as a "new" solution in projects, as well as a replacement for an existing control system and that it has been implemented to coexist with a former existing control system. They are shown as descriptions of those projects, but the whole project consisting of all details can be found at http://www150.abb.com/spaces/PLC-and-automation-Marketing-Team-Space/SitePages/References-and-SuccessStories-INTERNAL.aspx. This is an internal ABB database for information that cannot be accessed without access permission.
6. **PS501 Engineering Tool**

PS501 Engineering Tool is the ABB AC500 vendor-specific configuration tool with a range of different functions. This chapter gives an explanation of the two main parts of the PS501 Engineering Tool, which are Control Builder Plus and CoDeSys. They will to some extent be compared to other programming and configuration tools. Since the PS501 Engineering Tool is the programming environment for the AC500, it is necessary to know the overall basic parts of it to be able to fully understand later chapters. The current version of PS501 Engineering tool is version 2.3.0.

![Figure 6.1: PS501 Engineering Tool](image)

6.1 **Control Builder Plus**

Control Builder Plus is the vendor-specific part of the PS501 Engineering tool suite, and can be referred to as CBP. CBP is the starting point when configuring a new project. CBP handles all hardware configuration and to some point also the bus and Ethernet configuration of the project. These hardware configurations of used modules are sent to CoDeSys for finalization in the program.

6.1.1 **Hardware configuration**

All CPUs, I/O modules, interface and fieldbus modules are added into CBP to provide the user with an overview from the device tree. From the device tree the user can add or delete modules, but there is no graphical overview of the communication nor the system.
6.1.2 Parametrization

Configuration of hardware modules is done through parametrization. From here the user can choose specific configuration outcomes and solutions. Here all hardware settings are set for the different modules, such as mappings of I/O channels that are sent to CoDeSys and are a crucial part of the system functionality. Also IP addresses and module addresses have to be set according to hardware setup for the correct functionality. All parameters and settings for respective module are accessible by doubleclicking on them.

6.1.3 Diagnostics

When using diagnostics in CBP the following can be monitored: CPU Diagnostics, CPU statistics, Version information and PLC Browser. Also input values can be viewed live from CBP as well as communication module and fieldbus diagnostics.

From the different diagnostics it is easy to monitor cycle times, load on buses and CPUs which makes maintenance and troubleshooting much easier.

6.2 CoDeSys

CoDeSys itself is free to download programming environment developed by a German company called 3S-Smart Software Solutions. CoDeSys licenses are free of charge. This chapter only discusses CoDeSys version 2.3, as it is the version included in PS501.

6.2.1 Languages

CoDeSys offers programming in all five IEC 61131-3 languages, which include Instruction List, Structured Text, Ladder diagram, Function block diagram and Sequential function chart. Beyond that it also offers programming in a language called Continuous function chart, which is not defined as an IEC standard.

Structured Control Language, which is used by Siemens, cannot be used when programming in CoDeSys, but SCL is based on Structured Text so the similarities when programming are noticeable. Using IEC 61131-3 languages when programming is beneficial because it allows third party tools and module access as well as an easier integration with other systems.

[11]

6.2.2 Programming

CoDeSys itself is hardware independent so the capabilities when creating programs are endless. Learning the programming environment in CoDeSys may have a steep learn-
ing curve since it differs very much from e.g. Simatic manager, which is a widely used programming environment from Siemens.

Since CoDeSys by itself is hardware independent it needs to know the configuration of the project-specific hardware and its attributes, especially for the PLC. When using AC500 it is gathered from Control builder plus into CoDeSys as a Target file. This is needed for the software when using hardware specific blocks and different hardware diagnostics.

The main components when programming in CoDeSys is Program Organization Unit (POU), Data types, Visualizations and resources. POU consists of all functions, function blocks and programs. The data types section makes it possible to create your own data types such as structs and references etc. along the standard data types. Visualization is for making HMI visualizations, or if the PLC supports web visualization (visualization on webserver) or target visualization (visualization directly on PLC display). In resources, the configuration and organization of the project are found, e.g. global variables, task configuration, Library manager etc.

Program execution in CoDeSys is performed through task configuration, which can be found under resources tab. From the task configuration different forms of task types such as cyclic with interval time, freewheeling and triggered by event or external event can be made and have to be set with a priority. Those programs made under POU can then be appended in the task. There are restrictions on PLCs regarding how many tasks can be used. Both PM583-ETH and PM592-ETH can have a maximum of 16 tasks, whereas for example PM573-ETH can only handle a maximum of three tasks.

6.3 CoDeSys with other brands

There is a large number of PLC manufacturers that offer programming with IEC 61131, but the cross compatibility is lost because there is to my knowledge no standards for import and export of programs or projects. There are only “guidelines” written as standards for the program-code itself. This means that when making a program in for example FBD it follows IEC 61131-3 standards and is compatible to some extent with other platforms and programming environment, but since there are no standards for exporting and importing files, they cannot be shared between vendors. [12]

Some manufacturers that offer CoDeSys for their modular PLCs are Berghof, Eaton, Festo, Hitachi, Mitsubishi, Schneider, WAGO etc. The whole list can be found at CoDeSys.com. Just like AC500 has a specific target file, most of these also have a vendor-specific part creating a Target file specific to their own platform. If the project only consists of basic functions and function blocks that don’t need the information about the PLC, that program can be used on multiple platforms. [16]

When using CoDeSys for programming an extension of function blocks with OSCAT library is very useful (http://www.oscat.de/). This is a hardware independent IEC 61131-1 library provided license free. Since it is open source it offers great flexibility and includes over 800 library modules.
7. Water & Wastewater

This chapter describes the functionality of ABB Water & Wastewater library and the library in combination with S+ operations. The library has been developed by ABB for the purpose of giving standard solutions for waste and wastewater applications. A license must be ordered and requested via ABB when included and used in CoDeSys. The Water & Wastewater library is used in this thesis because of its current content of function blocks. This is explained in the Water & Wastewater chapter and also used in later chapters.

The current library version is V1.0, but the Water & Wastewater library V1.1 is planned to launch in 2014.

7.1 Functionality

When trying to understand the functionality of the Water & Wastewater library one must first understand the basic PLC project structure. Many PLC systems as well as AC500 systems provide diagnostic information of the hardware layer that is nearly impossible to make use of in user applications. When using ABB AC500 systems, projects are separated in two main parts, which are hardware configuration and user application. With the Water & Wastewater library it is possible to create a program called HW_PRG between hardware and user application. HW_PRG is where pre-made function blocks for S500 I/O modules with their individual configuration and mappings are called, as seen in figure 7.1.

![Figure 7.1: Structure [3]](image)

This program is not included in Task configuration but instead called through the main program with a function called HW_Diag, which is included in the library. Calling HW_Diag is necessary for the functionality of the library and should be included in a cyclic manner.

HW_diag is protected in the library with a nowrite flag and can therefore not be changed by the user. Instead HW_PRG must be made as a program containing those hardware configurations that are specific to the project. HW_Diag uses HW_PRG as a reference.
when refreshing IO data and passing diagnostic messages. For example: When using the default program PLC_PRG and appending it to a cyclic task it is possible to call HW_Diag simply by calling HW_Diag() in PLC_PRG. This means that the diagnostic function HW_Diag is called every PLC cycle through PLC_PRG, and HW_PRG is therefore not necessary to append in task configuration. Following the guidelines given for HW_Diag, input/output processing and diagnostic reset is executed in HW_PRG as:

```plaintext
IF HW_Diag.HWCallTask=1 OR HW_Diag.HWCallTask=2 OR 
    (HW_Diag.HWCallTask=3 AND (HW_Diag.HWDiagStart OR HW_Diag.HWDiagEnd)) THEN
    (* Processing of Input/Output or global diagnostic buffer shift *)
    (* For update of I/O, modules have to be listed here *)
    Module1();
    Module2();
END_IF;
```

The second part of HW_Diag is diagnostic processing. Function blocks for I/O modules have no information of where they are located on the I/O bus or fieldbus, so the program has to be made so that it passes the diagnostic message to the right function block. Diagnostic messages can then be read from the outputs of HW_Diag.

```plaintext
IF HW_Diag.HWCallTask=3 THEN
    (* Modules are called periodically to process errors. This is based on their position *)
    IF HW_Diag.HWDiagComp=14 THEN (* I/O bus *)
        CASE HW_Diag.HWDiagDev OF
            1: (* Module number one on I/O rack *)
                Module1();
            2: (* Module number two on I/O rack *)
                Module2();
        ELSE
            (* Non-recognizable module *)
            END_CASE;
    ELSE
        (* Non-recognizable module *)
        END_CASE;
    END_IF

7.2 Function blocks

The Function blocks in the Water & Wastewater library can be sorted into four different groups depending on the functionality.

- Device control blocks
  
  `FB_Motor1_1, FB_Motor2_1, FB_Valve1_1 FB_Valve2_1, FB_Transmitter1_1`
• Application logic blocks
  
  \textit{FB\_Alarm1\_1, FB\_AlternationTime1, FB\_AlterationPri2\_1, FB\_LimitControl1\_1, FB\_LimitControl2\_1, FB\_Actuator1\_1}

• Real time function blocks
  
  \textit{FB\_TimeData1, FB\_OperatingData1\_1, FB\_Accumulator1\_1}

• Calculation blocks
  
  \textit{FB\_Weir1\_1, FB\_Inflow1\_1}

All these blocks have an application interface and an HMI interface. Device control blocks have an additional hardware interface since they are connected to the hardware layer. The function blocks under device control blocks also have a first scan behavior because of this. This ensures a smooth transition when the program is updated by giving the hardware layer time to read the hardware inputs and outputs and update correct IO variables. This delay occurs because block outputs are not updated and I/O signals are ignored for the first cycle.[3]

![Figure 7.2: Functionality of the hardware interface. [3]](image)

### 7.2.1 Hardware interface

As mentioned before, hardware interface is only present in those function blocks that are meant to be connected to the hardware layer. This interface interacts between the function blocks and the I/O modules of the PLC as seen in figure 7.2. The data types used for this interface are marked with an \textit{IO\_} prefix followed by the name.[3]
7.2.2 Application interface

Application interface is present in all Water & Wastewater function blocks. Application interface variables are used to connect to other parts of the program. Input variables of the application interface can be changed from both inside and outside the block. Output variables of the application interface can be used directly as inputs to other blocks or copied to other variables.

Output variables cannot be changed from outside the block, nor can they be accessed from HMI or SCADA.[3]

7.2.3 HMI interface

HMI interface is present in all Water & Wastewater library function blocks that have to be accessed from S+ operations. HMI variables are divided into read-only and writable variables. Read-only variables are in general marked with an SO_ prefix and writable with an IP_ prefix, e.g. in FB_Transmitter1_1, the HMI.SO_value which is the output value from the block can only be read and displayed on the HMI, but HMI.CONF.IP_LimAHH which is the level of the HH-alarm can be set from S+ operations.

Device control blocks can also be accessed from S+ operations in such a way that protection is bypassed allowing writing directly to PLC outputs. This has to be done with careful consideration.[3]

7.2.4 FB_Motor1_1

This function block can be used when a motor or similar device has to be controlled using only one activation signal and one feedback signal. [3]

7.2.5 FB_Motor2_1

This function block can be used when a motor or similar device has to be controlled using only one activation signal and one feedback signal. [3]
This function block can be used when a motor or similar device has to be controlled using analog speed control. [3]

7.2.6 FB_Valve1_1

This function block can be used when a valve has to be controlled using activation signals in both directions and feedback from both directions. This valve block can be set from e.g. alarm outputs or other BOOL type variables. The operating time of the valve can be set and valve position can be monitored from that. [3]

7.2.7 FB_Valve2_1

This function block can be used when a valve has to be controlled using analog value as setpoint for the valve, and analog feedback from the valve. This block can e.g. be operated with a controller output ordering the position of the valve.[3]

7.2.8 FB_Transmitter1_1

This function block is used when evaluating an analog signal. It has a hardware interface where it handles both analog and digital input. The type of variable into the hardware interface of the block is REALIO or BOOLIO. This means that it has to be connected to the hardware layer and cannot be used with another type of variable as input signal.

When the block is in normal operation and has a value from e.g. an analog input, the value is passed to both application and HMI interfaces. Scaling of the signal is done inside the block. [3]
7.2.9 FB_Alarm1_1

This function block monitors a BOOL variable and if a TRUE edge occurs for a longer time than the configurated DelayOn, an output alarm signal is set to TRUE. This block can be used to monitor fire alarms, level switches, door alarms etc. [3]

7.2.10 FB_AlternationTime1

This function block can handle four objects. The functionality of the block is to alternate between the four objects. The longest running one is stopped first and the one being stopped for the longest time is next to start. [3]

7.2.11 FB_AlterationPri2_1

This function block can handle four objects. The functionality of the block is to alternate between the four objects. Unlike FB_AlternationTime1 this function block uses a priority list when alternating between objects. The possibilities are:

- Choice 2: LineB – LineC – LineD – LineA

[3]
7.2.12 FB_LimitControl1_1

![FB_LimitControl1_1](image)

Figure 7.11: FB_LimitControl1_1 [3]

This function block monitors a reference value of type REAL. This value is measured against a maximum of four different limits to which output is set, which makes it suitable for different types of reservoirs, tanks, etc.

7.2.13 FB_LimitControl2_1

![FB_LimitControl2_1](image)

Figure 7.12: FB_LimitControl2_1 [3]

This function block monitors a reference value of type REAL and, when the configured limit value IP_LimStart is exceeded, the output BOOL is set to TRUE. This function block can be connected directly to start/stop of motors and valves.

7.2.14 FB_Actuator1_1

![FB_Actuator1_1](image)

Figure 7.13: FB_Actuator1_1 [3]

This function block works as a counter, monitoring the rising edge of a BOOL type signal. When exceeding the pre-configured number of pulses, the output signal is set to TRUE.

7.2.15 FB_TimeData1

![FB_TimeData1](image)

Figure 7.14: FB_TimeData1 [3]

This function block handles system time and provides time information for the other blocks of the real time clock function block group. The internal clock of the PLC is used for calculating the local time and daylight savings.
7.2.16 FB_OperatingData1_1

![Figure 7.15: FB_OperatingData1_1 [3]](image)

This function block is used for calculating operating data for certain devices. It monitors a BOOL type signal and while the signal is TRUE it uses FB_Timedata1 to calculate operating data. Runtime data, time to service etc. are obtained with this function block.[3]

7.2.17 FB_Accumulator1_1

![Figure 7.16: FB_Accumulator1_1 [3]](image)

This function block is used for calculating different types of quantities. For example energy or water consumption can be calculated on timebase using FB_Timedata1. The pulse mode or analog mode can be used based on preference.[3]

7.2.18 FB_Weir1_1

![Figure 7.17: FB_Weir1_1 [3]](image)

This function block calculates the flow in rectangular and triangular weirs. The weir is divided into ten segments. Data on volume and height of the individual segments has to be configured for the right functionality of the block. [3]
7.2.19 FB_Inflow1_1

This function block calculates the flow into or out from a tank. The functionality is similar to FB_Weir1_1 and the tank has to be divided into ten segments. Data on volume and height of the individual segments has to be configured for the right functionality of the block. \[3\]

7.3 Implementation

As of today the Water & Wastewater library consists of earlier listed function blocks, but for this thesis I was also able to try the PID controller function block, which is planned to be launched in Water & Wastewater V1.1 later this year. It had the necessary interfaces to function with S+ operations. These function blocks cover almost all critical blocks used for Project X, which is discussed in Chapter 8. An exception is the function block for breakers which cannot be found in Water & Wastewater, but can be developed based on a requirement specification and included in the library in the future. Symphony Plus Water Automation department in Sweden is responsible for updates of the library.

The function blocks are easily implemented in CoDeSys from where they can be utilized from S+ operations after they are downloaded on the PLC. This is a part that is described by Mr. Anton Wargh in his thesis - Symphony plus as application for power plants - S+ operations subproject.
8. Project X - Using AC500

This chapter is an essential part of the thesis, as it shows how an existing system solution could be solved using AC500. The layout is from an engine power plant in Liberia using a Siemens solution. This chapter gives a clear view of the advantages and disadvantages of the different system solutions. PLC program sizes and programming environment advantages and disadvantages are not addressed in this chapter. The full system overview can be seen in Appendix 1.

8.1 Introduction

The engine power plant is a 20MW Power Plant with the configuration of 5 Diesel engines at 4,164 MW a piece, located in Liberia (Personal communication 26.03.2014). I believe there have been some changes in the system overview since the revision used for this thesis work. This should not be of any concern, since the overall solutions are more relevant than the specific solution for this project.

8.2 Communication

This section describes different types of communication protocols used, based on the layout. They are compared to those available with the AC500.

8.2.1 Siemens

For the Siemens solution the different communication types used are listed in Figure 8.1. On panel-level, Ethernet is used to communicate both inside and outside the panels via switches. Profibus is used to communicate to remote stations and it is also used to achieve redundancy to remote stations, and via Y-link to singular devices. Communication to relays is done with Modbus TCP in this case, instead of the widely used IEC 61850 standard.

Figure 8.1: Communication
8.2.2 ABB

For the ABB solution, Ethernet can be used in the same way as in the Siemens solution, but instead of Profibus a solution using CS31 is mandatory for achieving high availability. Redundancy on device level is not manageable because of the lack of an Y-link. Both TCP and UDP can be utilized at the same time, offering both fast transmission and a more reliable transmission. Communication to relays can be achieved with Modbus TCP or using IEC 60870-5-104 depending on device support.

8.3 Common Control Panel

The common control panel is the "main" panel of the plant. It is also the most critical panel and has to be kept running for the functionality of the power plant.

8.3.1 Siemens

For the CCP (central control panel) two S7-400 H type PLCs are used to handle all communication and program operations. Synchronization between the two redundant PLCs is achieved through Siemens communication. The redundancy is used for remote I/Os and motor control devices, e.g. Simocode, which is used to give more control over different motors located on the plant. The redundancy is synchronized continuously using Siemens’ own internal communication via an optical fiber.

8.3.2 ABB

AC500 cannot be used in this case, as the high availability solution using CS31-buses will cause problems because of its restrictions on the ratio between nodes and bandwidth (about 60 nodes in this case). Instead, using the AC800 PLC for the common control panel in this case could be a more efficient solution to maintain redundancy in the system. Communication to AC500 in other panels can be achieved using Modbus TCP or Profibus, as well as a working communication with S+ operations using Ethernet. The AC800 is a PLC developed by ABB, but this will not be discussed any further in this thesis.
8.4 Generator Control Panel 1-5

There is one Generator Control Panel per engine in the power plant. They are not as critical as the Common Control Panel because the plant can even function with one engine faulting.

8.4.1 Siemens

For the GCP (Generator Control Panel) an S7-400 PLC is used to handle communication and program operations. GCPs in this case are installed with PMUs and protection relays and communication to these are achieved using standard Modbus TCP. The generator protection relay is connected directly to operator stations. There is no redundancy in the GCP panels, and communication to Remote I/O stations is done with Profibus. The AVR (Automatic Voltage Regulator) is connected only to other AVRs in this case. A viscosimeter used to measure the viscosity in fluids is connected via a switch and converter (Modbus TCP/RS485).

8.4.2 ABB

The AC500 is a suitable solution for the Generator Control Panels thanks to the lack of redundancy in those panels. Profibus can be used in the same way as for the Siemens solution. For communication to PMU and relays, AC500 supports IEC standard 60870-5-104 and Modbus TCP. This could also be solved using another fieldbus, e.g. Profinet which would support a ring type topology if necessary. Using the AC500 as a Profibus or Profinet master is achieved by using separate communication modules. The viscosimeter could be used via the same type of solution via a switch and a converter.
8.5 HMI

Figure 8.4: Engine power plant operator stations

This part is explained in Mr. Anton Wargh’s subproject on S+ operations, but some aspects are worth mentioning in this part as well.

8.5.1 Siemens

The solution shown in Figure 8.4 was in this case changed to a Siemens solution, using WinCC SCADA.

8.5.2 ABB

This could be solved by using OPC server and S+ operations. The benefits of this solution are presented in the thesis conducted by Mr. Anton Wargh and are discussed more thoroughly there, so readers should refer to this document.

8.6 Summary

This chapter gives a direct input for the reader to see the current capabilities of the AC500. There are some aspects of the AC500 that could be improved to make it more suitable for this area of use. These aspects are discussed in chapters 10 and 11 where it is directly weighed against the Siemens PLCs, using information from experience gathered during this thesis work and inputs from specialists and co-workers.
9. Creating a Project

This chapter describes the overall making of a project using PS501. This project is going to be connected with S+ operations. It is based on the demo setup made by me and Mr. Anton Wargh to test the usability and functionality of AC500 in combination with S+ operations. The steps are explained in an "overall" type of way instead of a step by step way, because the overall configuration is more relevant than program-specific settings and configurations. The project can be seen in Appendix 2.

9.1 Project

The target with the demo solution is to determine the functionality of the AC500 PLC. It was therefore necessary to get a hardware setup providing those functions that are mostly used in e.g. power plant projects. The focus was on performance, redundancy, communication and I/O modules consisting of both digital and analog inputs and outputs, with support for thermocouple. The hardware modules chosen for this were:

- 2 X PM590-ETH CPU
- AX522 - Analog input/output module, 8AI/8AO, PT100.
- AI531 - Analog input module, 8AI, thermocouple.
- CI590 - High availability, CS31, 16DC
- Ethernet switch

9.2 Overview

Mr. Anton Wargh and I chose to make the demo solution based on a watertank process. The watertank process itself is a simple type of process consisting of mainly tanks and valves, but can be expanded to use almost all function blocks included in the Water & Wastewater library. Since it is a demo solution intended to be shown and displayed, the watertank solution is a good solution based on its simplicity.

The functionality of the watertank process is based on two different water tanks. These tanks have a continuous flow between them from tank 1 to tank 2. The level in Tank 1 is regulated by its inflow through a valve that is operated by a PID controller. The level in tank 2 is only regulated if the level reaches a high limit, leading to a drain-valve opening and triggering an outflow from tank 2. If the limit reaches a high-high limit, not only does the drain-valve of tank 2 open, but a pump starts to run which then increases the outflow of the tank. The operating data for the pump is calculated as well as the outflow quantity.
9.3 Hardware installation

It is easiest to start with mounting of all modules on their respective terminal bases. The terminal bases are easily locked on a DIN-rail, but the order in which the I/O modules are mounted has to be noted if the hardware is not present during the hardware configuration in Control Builder Plus. This has to be noted because it has to comply with the configuration in Control Builder Plus. Since this is a decentralized system, power supply is needed for both "clusters".

9.4 Communication protocols

The communication protocols used in this project are:

- CS31 bus for high availability
- Ethernet for programming and for communication to S+ workstation
- Modbus TCP for testing functionality
UDP/IP communication between CPUs

The CS31 bus that is used in this project to achieve high availability has to be connected from the COM1 port on both CPU modules to the CI590 high availability module. End terminators are default for the CI590 module, but on the COM1 port of both CPUs an 120Ω resistor has to be added for the bus to function properly. This can be verified by measuring the resistance over the bus with a multimeter. The multimeter should show a resistance of 60Ω. Two rotary switches are located on the CI590 module for setting the address of the module. This address has to comply with the configuration in both Control Builder Plus and CoDeSys.

Ethernet communication is used for programming the PLCs and for being able to go online and monitor the functionality in CoDeSys. Connection to S+ operations is also achieved using Ethernet. When having multiple computers and PLCs connected to the same network, it is crucial to avoid IP-conflicts by choosing different static IP-addresses. The IP-addresses of the two PLCs can be set either from the display or through Control Builder Plus. Communication parameters are then set to either of those addresses depending on the desired PLC to connect to.

Modbus TCP was used in this project to test its functionality and load on CPU. There are hardware independent Modbus function blocks in CoDeSys that were used for this. In Control Builder Plus the onboard Ethernet coupler has to be configured for this by adding Modbus TCP/IP server/client. Read and write of registers was tested by using a secondary laptop running a Modbus server simulator. The Modbus program was appended to a 10ms cyclic task and recorded a polling interval of 20ms in the Modbus server simulator.

UDP/IP is used for data transfer between PLCs. It is used with high availability for the communication between CPUs. Modbus TCP can also be used for this and is a more reliable way, because TCP has checksums and uses resending to assure stable data transaction. With UDP this cannot be achieved on the same level, but it is a faster type of communication.

9.5 Hardware configuration

Hardware configuration is done in Control Builder Plus. To be able to start the hardware configuration, specifications should be known. There are ways to change the hardware configuration if modules are replaced after the configuration is done, but then those changes have to be downloaded into CoDeSys from Control Builder Plus. The hardware configuration begins with naming of a new project and saving it. One can choose to start a new AC500 project or just new project, but by choosing AC500 project the CPU can be chosen directly from the AC500 project window. Though two CPU modules of the same type are used in this project, only one has to be added to the project tree. This is because high availability needs to have the same project configuration to function properly. Therefore the same project configuration is downloaded into both CPUs with the exception of individual IP-address configuration. The IP-address
is set to 192.168.0.10 as default, but can be changed from the display of the CPU module, in Control Builder Plus under tools/IP-Configuration or from IP_Settings located in the Project tree. When using the last mentioned it should be known that it overwrites both display and IP-Configuration tool data. In this project the AC500 web server is tested, so under IP_setting in tab Extended settings the option Web server active is checked.

Since this project uses a decentralized I/O rack, the I/O modules should not be added directly to the CPU module like in centralized extension, but under the used communication interface module. Interfaces, located as one of the main "branches" of the project tree displays onboard interfaces that can be used. These have to be configured for the bus protocol used, and for this project, the CS31 - bus is chosen on COM1. The parameter operating mode is default set to master and therefore it does not need to be changed. When the bus protocol is chosen for respective interface, modules that are connected to the interfaces should be added as devices under that interface. For this project the CI590 module is added under the earlier configured COM1 interface, and I/O modules are then added onto the CI590 module. The address chosen from the rotary switch that is physically located on the CI590 module has to comply with the parameter module address on the CI590 module in Control Builder Plus.

As mentioned before, I/O modules are added to the CI590 module in this project. Like all other modules in hardware configuration they are added from an ABB vendor specific device list. I/O modules have to be added in Control Builder Plus in the same order that they are physically installed on the DIN-rail, for the correct functionality. The I/O module configuration is critical for the functionality of the project, because the mappings, which are the variables names, addresses, channels and types, are exported to CoDeSys to achieve the use of I/O modules in the program.

Only those variables that are configured with names are exported to CoDeSys. The configuration of the channels also has to be done in Control Builder Plus. This is dependent on the module used, but for this project 0..20mA was chosen for all active channels, since two analog inputs and two analog outputs are used. Under branch Onboard_Ethernet in the project tree, UDP data exchange, Modbus server and Modbus client are added. When these configurations are done, configuration data and mapping data are exported to CoDeSys by doubleclicking on source code file.
9.6 Software Configuration

CoDeSys opens with an empty program called PLC_PRG located in the POU tab. This project uses multiple Water & Wastewater function blocks, thus the library has to be added. For the library to function properly the option "Replace constants" has to be checked. This option can be found under Project/Option/Build.

The mapping variables are imported to CoDeSys and can be seen in Resources tab under Global variables/Interfaces/COM_CS31_Bus/CI590_CS31. To make use of these variables a program called HW_PRG is made. Here both I/O modules are implemented using their specific function blocks from the Water & Wastewater library. HW_PRG for this project is structured based on guidelines in document 2VAA002998 with a few modifications. Structured text is used because it allows for easier and faster coding.

```plaintext
module pRINput.*AR jeunes;
module pRINput.*AR out0:

(* I/O bus module 01 application connections *)

module AI0.*AR(level1 IO_0 Voltage);
module A00.*AR(Vatertask value outWord1);
module A1T.*AR(level2 IO_0 Value);
module A0L.*AR(Vatertask value outWord2);

(* Analog outputs unused *)
module A11:-
module A12:-
module A13:-
module A14:-
module A15:-
module A16:-
module A17:-
```

**Figure 9.4:** I/O setting in HW_PRG

PLC_PRG is used in this project as a “main” program where first scan behavior and HW_Diag both are executed. Going to Resources tab and by opening Task configuration a cyclic 50 ms task is created and PLC_PRG is appended to that task.

The high availability configuration requires an additional library called HA_CS31_AC500_V23. HA_PRG and CALLBACK_STOP are programs added to POU for high availability. Callback stop, which is the mandatory name for the program, calls function HA_CS31_CALLBACK_STOP intended to detect CPU stop event.

**Figure 9.5:** PLC_PRG

```plaintext
/* System event */

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Call POU</th>
</tr>
</thead>
<tbody>
<tr>
<td>init</td>
<td>Called when program runs</td>
<td>pH_Init</td>
</tr>
<tr>
<td>stop</td>
<td>Called when program stops</td>
<td>callback_stop</td>
</tr>
<tr>
<td>When end</td>
<td>Called before next task runs</td>
<td>callback_stop</td>
</tr>
<tr>
<td>start</td>
<td>Called before next task runs</td>
<td>callback_stop</td>
</tr>
</tbody>
</table>
```

**Figure 9.6:** Calling CALLBACK_STOP using system task configuration

This program is called from Task configuration under System events by checking stop and typing the program name under the column called POU. Another program called
HA_PRG is made containing HA_Diag and HAcontrol which are the diagnostic block and the block handling the high availability switchover. This program is appended in a cyclic task of 20ms called HAtask.

The Modbus communication was solved by using function blocks called ETH_MOD_MAST in a program called MODBUS. This program was also appended in a Task of 10ms called MODBUS. Modbus was tested using a second laptop running Modbus server simulator, where the polling time and register set and read were monitored.

The rest of the programs implement mostly function blocks from Water & Wastewater. These were made to control the process tank level using values from the program called watertank, consisting of calculations that simulate inflow or outflow of a watertank. Since no real process tank was used, the simulated level calculated in the watertank was set to the first analog output of module AX522. This had to be done because the signal used in FB_Transmitter1 has to be RealIO. That is accomplished by wiring the first analog output to the first analog input, thus making it possible to use FB_Transmitter1. Parameters have to be set for the outgoing signal or else the module function block will get an error because of the lack of maximum and minimum values.

The analog input is then used in FB_Transmitter1 and scaled as 0..100. This signal is used for the PID controller block in the program CONTROLLERS. The valves and motor blocks are located in the program called MOVING_OBJECTS.

```
0220 AnalogSignalReal = SignalRealReal - (MinVal -0. MaxVal;=100. Unit;=’%’);
```

Figure 9.7: High availability function blocks

Figure 9.8: Modbus function blocks

Figure 9.9: Analog signal scale configuration values
The Water & Wastewater function blocks need to have an IN_OUT type variable containing a specific configuration for certain constants e.g. scaling values for the transmitter block. These values are saved in a Global variable file under resources tab. Since these values are configuration values and therefore need to be persistent, they are saved as VAR_GLOBAL RETAIN PERSISTENT. These values are then loaded into the struct CONF_var under Data types tab, from where they can be called to function blocks with configuration values.

As the web server option was checked already in Control Builder Plus, the only thing that needs to be done is to open target settings under resources tab, under which Visualization "Use 8.3 file format" and "Web visualization" have to be checked. In this project the CPU load is visualized as a web server and is accessible by IPAddressofCPU/webvisu in Internet explorer.
9.7 Summary

This chapter has given an understanding of the overall configuration of the demo solution from where the results shown in chapter 10 are determined. To establish a working communication with S+ operations symbol files had to be loaded into the PLC as well as loaded into S+ operations. These had to be up-to-date for the communication to work.
10. Results

The demo solution for investigating S+ operations in combination with AC500 was made according to Chapter 9. As a result of the demo solution Mr. Anton Wargh and I were able to determine the functionality of S+ operations in combination with AC500, and AC500 in general.

Despite the fact that the whole program size is relatively small it was interesting to see that the cycle time was not reduced even though I was running Demo, Modbus and HA programs at the same time on minimal task cycle times, trying to stress the PLC. This was observed from the Modbus server simulation software polling time and Control Builder Plus diagnostics.

High availability was tested with functional switchover between the two CPUs. However, since the demo process and its values are calculated, instead of using "real" inputs some values tend to keep counting beyond their limits. This is because both programs run simultaneously and calculate tank volumes and levels instead of using "real" process values. It was easy to work with and configure the CS31 bus, but unfortunately it was the only bus tested with AC500. However, since AC500 supports multiple bus-types and these hold international standards, using them should not be harder than with the CS31 bus. CS31 was the only bus supported with high availability.

At first it was difficult to understand the functionality of the Water & Wastewater library with all the different layers. After understanding the connection between these layers and the other attributes of the library, I came to the conclusion that it is a very efficient way of handling function blocks, especially when they have to be connected to some form of HMI. The current selection of function blocks worked as they were intended. The Preliminary PID-function block felt a bit unfinished but could be used in S+ operations, and also the application functionality worked well for the purpose in the demo solution.

There was no standard Timestamping solution for the AC500, but Mr. Mika Kuukasjärvi had programmed a function block for timestamping earlier, which I was able to try. It was only tested on an application level, since no form of function to link the timestamps further was found in the function block. My supervisor Mr. Frank Redlig and I decided that programming such a link fell outside the target of this thesis.

Overall we made a working demo solution from where the earlier mentioned aspects could be determined. Using Control Builder Plus and CoDeSys has a steep learning curve, and since CoDeSys is basically a "freeware" from the beginning it didn’t feel like it had the same standard of usability as proprietary branded programs. Function blocks were used in S+ operations from where Mr. Anton Wargh could operate Water
& Wastewater function blocks, but variables that I made to determine which CPU was master for the high availability switchover in the OPC server was something we did not get to function properly. Timestamping was the only critical feature that we didn’t get to function between AC500 and S+ Operations. Mr. Anton Wargh describes some aspects of the timestamping from an OPC and S+ operations point of view in his thesis.

As a result of the comparison in Project X to determine advantages and disadvantages I was able to determine that the current features of the AC500 could not be used for a common control panel with the specifications of Project X. This is because high availability is only supported using the CS31 bus, and the amount of nodes in Project X (≈ 60) exceeds the amount of nodes that can be used with CS31 (31). The other features of AC500 should suffice based on the specification from the layout of Project X.

On the other hand, when analyzing the generator control panels that don’t need redundancy, AC500 could be a possible solution. This is based on technical specifications that have been compared as well as supported protocols and network types. As a conclusion the AC800 could be used for the common control panels until a solution for high availability with AC500 that meets all the specifications is developed. On generator control panels AC500 could be a possible solution for this specification.

As a conclusion, assuming a realization of power plant projects using AC500 in the future, areas of improvement should include AC500, the Water & Wastewater library and CoDeSys. AC500 high availability should have a comprehensive and practical solution to achieve communication to third party modules and devices. Timestamping should be developed to a fully functional solution that can be used in engine plant projects. The Water & Wastewater library could be updated according to different plant specifications to improve the extent of the library. CoDeSys usability should be improved by investigating the abilities of the SFC and CFC languages for sequential programs and also the use of project templates in CoDeSys.
11. Discussion

This thesis work covers a wide area, including PLCs, networks, Programming, implementation, power plant and water treatment plant configurations. Since it is also part of a two-part project investigating S+ operations in combination with AC500, me and Mr. Anton Wargh who was in charge of the second part, had to have meetings at regular intervals to determine where we were and how we would continue from that. We had both made similar timetables so we would be able to keep the same pace and be able to achieve the target more easily.

I also had regular follow-ups with Mr. Frank Redlig about the progress and he explained aspects of power plants and their PLC system layouts in general. He also helped with providing information and useful contacts for the AC500. Because I was conducting this thesis work on behalf of ABB Power Generation I had good and modern equipment available as well as large databases of information. But regarding examples and guides on AC500 it was hard to find useful information. The most difficult thing was to find good information and guides on PS501, which really slowed down the progress.

The time spent on the project during autumn/winter of 2013 was very limited because of a hectic school schedule, but I was able to intensify the time at the beginning of 2014. In the beginning when I was learning to use both CoDeSys and Control Builder I had to use evenings and weekends to do this, which slowed down the process. If I was to conduct this project work again, I would try to order components at an earlier stage and also try to have more structure in the way we tested the demo solution. Narrowing down the written part was difficult since it involved a wide area. I tried to include all those parts that are necessary to get an overall picture of both the demo solution and Project X to be able to understand the results.

It is going to be interesting to see the outcome of this work. I think the AC500 PLC has great potential for power plant implementation when some of the features such as eg. high availability has been improved. With the shown interest from domestic sales and R&D looking for a pilot project, I think improvements and support could be accelerated if it were to be realized.

Mr. Frank Redlig and I discussed the solution using Siemens, to where he pointed out that synchronization between CPUs put a huge load on them, because of all the communication running and the heavy synchronization communication, which leads to slower cycle times. Using AC500 with faster process time and high availability could be a better solution. This depends on requirements on the redundancy from customers, since high availability is a software-layer redundancy and has a switchover time of a multiple number of cycles. I have been in contact with Mr. Mika Kuukasjärvi from
domestic sales, stated as a PLC specialist. He told me about a PROFINET solution that is going to be launched at the end of next year or beginning of 2016. This solution will support High availability to redundant PROFINET IO-devices. Currently the only supported solution is CS31, to which only AC500 I/Os can be connected. The Profinet solution should have a wider device support compared to CS31.

Communication to relays was in this case using Modbus TCP, which is supported by AC500, but according to Mr. Frank Redlig communication to relays is often achieved using IEC standard 61850. This again, according to Mr. Mika Kuukasjärvi, cannot be used with AC500, but instead it can be achieved using supported IEC standard 60870-5-104. Devices that support this standard are unknown.

When I was in contact with Mr. Mika Kuukasjärvi to verify the solution of chapter 8, he pointed out an interest from domestic sales when I attached a system layout from Project X. He mentioned that if the high availability is the only thing preventing this from realization, it could probably be accelerated to a solution from R&D because they are looking for pilot projects. Considering the improvement areas mentioned in chapter 10, a form of workgroup consisting of Water & Wastewater library personnel and e.g. Mr. Mika Kuukasjärvi as an ABB PLC specialist as well as personnel from Power generation in Vasa would give a wider area of inputs for a more complete project solution.

A solution for the timestamping could also possibly be achieved by this workgroup. Possibly by using Mr. Mika Kuukasjärvi’s timestamp block linked to an OPC server and if necessary, interfaces to S+ operations. Using it in CoDeSys, appended to a triggered-by-event type of task, it could possibly be triggered on I/O-level in combination with Water & Wastewater library.

There was no complete comparison made on price differences when using the AC500. My own opinion is that a solution using AC500 would reduce costs, both regarding hardware and software, considering that the program environment is based on CoDeSys and also that the AC500 PLC has a lower cost in general.

This is based on a rough comparison on prices between a Siemens bundle set consisting of two pieces of CPU416-5H with racks, synch-modules and backup batteries (a listing price of about 21200 €) and the ABB PM592-ETH module as a high availability setup consisting of two pieces of PM592-ETH with terminal bases (a listing price of about 8000 €). Even on a more basic level when comparing Siemens CPU315-2 and ABB PM583-ETH, the result is the same, as the Siemens CPU315-2 has a listing price of about 1950 € and the PM583-ETH about 1250 €. It would be interesting to see such a comparison on a project-level, comparing the costs of program environments and their licenses as well as hardware modules.

I have learned a lot during the time I have spent on this thesis. Both theoretically and practically in areas such as PLCs, programming, hardware testing and implementation and power plant technology overall. Considering this I feel like I now have a better understanding of power plants in general and their critical parts. I believe I have improved my skills when working in a group and also when consulting others for getting other points of view, and maybe a better solution in the end.
12. Bibliography


Demo process
Daniel Hummel
V1.0
Watertank process with AC500 in combination with S+ Operations
ANALOG (PRG-FBD)

0001 PROGRAM ANALOG
0002 VAR_EXTERNAL
0003 level:FB_Transmitter1_1; (*Level In Tank*)
0004 level2:FB_Transmitter1_1; (*Level In Tank2*)
0005
0006 END_VAR

0008

Analog in 1, Tank 1

FB_Transmitter1_1
level
FBMode SO_Value
2400000000 SO_SignErr
SO_GlobalAlarmBlock ExtReset SO_AHH
DisHighAlarm SO_AH
DisLowAlarm SO_AL
conf.level CONF SO_ALL

0009

Analog in 2, Tank 2

FB_Transmitter1_1
level2
FBMode SO_Value
2400000000 SO_SignErr
SO_GlobalAlarmBlock ExtReset SO_AHH
DisHighAlarm SO_AH
DisLowAlarm SO_AL
conf.level CONF SO_ALL
PROGRAM CALLBACK_STOP

VAR_EXTERNAL

  dwEvent: DINT;
  dwFilter: DINT;
  dwOwner: DINT;

END_VAR

("Mandatory name for Program")

HA_CS31_CALLBACK_STOP(dwEvent, dwFilter, dwOwner);
CONTROLLERS (PRG-FBD)

0001 PROGRAM CONTROLLERS
0002
0003 VAR_EXTERNAL
0004 controller:FB_PID1_1;
0005 END_VAR

0001

PID controller Tank 1

controller

FB_PID1_1

Level SO_VALUE
- SO_PV SO_Out
- SO_SP SO_OutHigh
- SO_MinOut SO_OutLow
- 100 SO_MaxOut SO_SPMagnitude
- SO_Maxincr SO_SumAlarm
- SO_MaxDecr
- SO_TrackVal
- SO_Track
- SO_FeedForward
- InitModelReset
- InitModeReset
- conf.controller

CONF

0002

FALSE AND
FALSE

CONF.controller.IP_CmdManOut
%0001 PROGRAM HA_PRG
%0002
%0003 VAR
%0004 HA_diag: HA_CS31_DIAG;
%0005 HAcontrol: HA_CS31_CONTROL;
%0006 hacontrol_man_change:BOOL:=FALSE;
%0007 hacontrol_ackn:BOOL:=FALSE;
%0008 hacs31_sync_EN: BOOL:=TRUE;
%0009 HA_sync: HA_CS31_DATA_SYNC;
%0010 END_VAR

High availability diagnostics

```
TRUE
1

EN
ERR
DONE
COM
ERRNO
NUM_SLV_CFG
NUM_SLV_ACT
ACTIVE_SLV
ERR_MIX_WIRING
```

High availability synchronization

```
AT1V
ADR
hacs31_sync_EN
HA_CS31_DATA_SYNC
EN DONE
DATA ERR
LEN ERN
SizeOF
```

```
AT2V
ADR
hacs31_sync_EN
HA_CS31_DATA_SYNC
EN DONE
DATA ERR
LEN ERN
SizeOF
```

```
controller.so_sp
ADR
hacs31_sync_EN
HA_CS31_DATA_SYNC
EN DONE
DATA ERR
LEN ERN
SIZEOF
```

High availability switchover data handling

```
TRUE
0

EN
DONE
ETH_SLOT
ERR
192.168.0.10
IP_ADR_CPU_A
IP_ADR_CPU_B
192.168.0.11
FALSE
NOX_CHG_OVER
FALSE
MANUAL_CHG_OVER
```

APPENDIX 2
Page 5 of 27
PROGRAM HW_PRG

VAR
  HWOnlineChangeCount_old:USINT :=0;

module1:HW_MOD_AX522; (*S500 I/O module*)
module2:HW_MOD_AI531; (*S500 I/O module*)
test:REAL:=100;

END_VAR

VAR_EXTERNAL
  (*Value_outWord: INT ;*)
  HWOnlineChangeCount:USINT :=10;  (* Increased +10 with each online change *)

END_VAR

IF HW_Diag.HWCallTask=1 OR HW_Diag.HWCallTask=2 OR (HW_Diag.HWCallTask=3 AND ( HW_Diag.HWCallStart OR HW_Diag.HWCallEnd)) THEN
  (* Input/Output processing or global diagnostic buffer shift *)
  (* All modules have to be listed here, if they want to update their I/O and diagnostics *)
  module1();
  module2(); (*END_IF;*)
ELSIF HW_Diag.HWCallTask=3 THEN
  (* Modules are periodically called to process errors, based on their position in the rack *)
  CASE HW_Diag.HWDiagComp OF
    11: (* CS31 *)
      IF HW_Diag.HWDiagMod=31 THEN (* Whole slave station failed *)
        IF HW_Diag.HWDiagDev=1 THEN (*CS31*)
          module1(ExtAlarm:=TRUE); (* First module on I/O rack *)
          module2(ExtAlarm:=TRUE); (* Second module on I/O rack *)
        ELSE; (*CS31*)(* Unrecognized station on bus *)
          END_IF;
        END_IF;
      END_CASE;
    ELSE; (* Wrong call *)
      END_CASE;
  ELSE
  END_CASE;
END_IF;

IF HWOnlineChangeCount<>HWOnlineChangeCount_old THEN
  HWOnlineChangeCount_old:=HWOnlineChangeCount;
(* Only variable connections to HW and Application here *)
(* Calling modules should be done further down the program *)
(* I/O bus module 01 HW configuration *)
(module1.InvertIn:=2#00000000;)
(module1.InvertOut:=2#00000000;)
(module1.phInput:=ADR(in0));
(module1.phInput:=ADR(in0));
(module1.phOutput:=ADR(out1));
(module1.phOutput:=ADR(out1));
(* I/O bus module 01 application connections *)
(module1.AI0:=ADR(level.IO.IO_Value ); (*Input to Transmitter block*)
(module1.AO0:=ADR(Water tank.value_outWord1); (*Output to Analog I/O module*)

0053 module1.AI1:=ADR(level2.IO.IO_Value); (*Input to Transmitter block*)
0054 module1.AO1:=ADR(Watertank.value_outWord2); (*Output to analog I/O module*)
0055 (* Analog inputs unused *)
0056 (* module1.AI1:=; module1.AI2:=; module1.AI3:=; module1.AI4:=; module1.AI5:=; module1.AI6:=; module1.AI7:=;*)
0064
0065 (* Digital outputs *)
0066 (* Unused I/O *)
0067 MOD01.DO0:=ADR();
0068 MOD01.DO1:=;
0069 MOD01.DO2:=;
0070 MOD01.DO3:=;
0071 MOD01.DO4:=;
0072 MOD01.DO5:=;
0073 MOD01.DO6:=;
0074 MOD01.DO7:=;
0076 MOD01.DO8:=;
0077*)
0078 (* I/O bus module 02 HW configuration*)
0082 (*module2.pHWInput:=ADR(%%%%%%);*)
0083 (* I/O bus module 02 application connections *)
0083 (* Analog inputs *)
0084 (* Unused I/O *)
0085 module2.AI0:=ADR();
0087 module2.AI1:=;
0088 module2.AI2:=;
0089 module2.AI3:=;
0090 module2.AI4:=;
0091 module2.AI5:=;
0092 module2.AI6:=;
0093 module2.AI7:=;
0094 module2.AI8:=;
0095 module2.AI9:=;
0096 module2.AI10:=;
0097 module2.AI11:=;
0098 module2.AI12:=;
0099 module2.AI13:=;
0100 module2.AI14:=;
0101 module2.AI15:=;
0102*)
0103 END_IF;
MODBUS (PRG-FBD)

0001 PROGRAM MODBUS
0002 VAR
0003 write_data:ETH_MOD_MAST;
0004 read_data:ETH_MOD_MAST;
0005 edge:R_TRIG; (*Temporary solution, doesn't need to be used*)
0006 errnumber AT %MD0.0 : WORD;
0007 DATA AT %MW0.0 : DWORD;
0008 test:BOOL:=TRUE;
0009 END_VAR

Enables block with pulse signal BLINK with a set time of 1 sec. Function block is dedicated to the CPU internal Ethernet port with command 0 on SLOT. Write data with Function 16 on FCT.

Enables block with pulse signal BLINK with a set time of 1 sec. Function block is dedicated to the CPU internal Ethernet port with command 0 on SLOT. Read data with Function 3 on FCT.
The pump used in the demo solution to flush Tank 2 when high-high-alarm level is reached.

Digital valve to flush Tank 2

Analog valve to Tank 1 controlled by PID controller
OPERATINGDATA (PRG-FBD)

0001 PROGRAM OPERATINGDATA

0002

0003 VAR_EXTERNAL

0004 motor_data:FB_OperatingData1_1;

0005 timedata:FB_TimeData1;

0006 Waterout: FB_Accumulator1_1;

0007 END_VAR

0008 VAR

0009 END_VAR

0010

Time data that is used

conf.timedata.CONF + SO_DSTStatus

Runtime data and time to service for pump

pump.SO_answeronCONF + SO_AIService

conf.motor_data.CONF + SO_Time

Used to measure quantity of water flushed from Tank 2

water Tank.VT2OCONF + TRUE

conf.waterout.CONF + SO_Time

conf.waterout.CONF + TRUE

AccType

PulseInputUnits

FactorHour

FactorDay

FactorWeek

FactorMonth

FactorYear

FactorTotal

FactorTotal

conf.waterout.CONF + SO_Time

conf.waterout.CONF + TRUE
PROGRAM PLC_PRG

VAR_EXTERNAL

SystemClock:FB_TimeData1; (* System time handling *)
controller:FB_PID1_1;
ATIV: REAL;
END_VAR

VAR

{flag noread,nowrite on}
FirstScan:BOOL:=TRUE;
PLC_Master: BOOL;
{flag off}

(* Obligatory FUNCTION call FOR HW_Diag *)

(* Determine Master PLC for OPC server *)
IF fg_HA_PRIMARY THEN
HW_Diag();
PLC_Master:= TRUE;
END_IF

(*Calling CPU diagnostics used in visualization*)
CPU(EN:=TRUE , DONE=> , ERR=> , ERNO=> );

(* Handling configuration variables *)

(*First scan extra behaviours*)
IF FirstScan THEN
controller.InitmodeReset:=FALSE;
ATIV:=3000;
CONF.controller.IP_CmdManOut:=FALSE;)
FirstScan:=FALSE;
END_IF
TIMESTAMPING (PRG-FBD)
0001 PROGRAM TIMESTAMPING
0002 VAR
0003 TS_OUT_ARRAY: ARRAY [0..99] OF TS_DATA_STR;
0004 END_VAR
0005 VAR_EXTERNAL
0006 timestamp: TS_EVENT_STR;
0007 END_VAR

Timestamp block, only used to test functionality in application.

```
TS_EVENT_STR
  timestamp
    ev0
    ev1
    ev2
    ev3
    ev4
    ev5
    ev6
    ev7
    ev8
    ev9
    ev10
    ev11
    ev12
    ev13
    ev14
    ev15
  level tank1
    msg0
    msg1
    msg2
    msg3
    msg4
    msg5
    msg6
    msg7
    msg8
    msg9
    msg10
    msg11
    msg12
    msg13
    msg14
    msg15
  level tank2
    msg0
    msg1
    msg2
    msg3
    msg4
    msg5
    msg6
    msg7
    msg8
    msg9
    msg10
    msg11
    msg12
    msg13
    msg14
    msg15
  Clear
```

TS_OUT_ARRAY

---

APPENDIX 2
Page 13 of 27
PROGRAM water tank
VAR_EXTERNAL
AT1V: REAL; (*Actual Tank 1 Value*) (*Is set 3000 litre volume on first scan*)
AT2V: REAL; (*Actual Tank 2 Value*)
END_VAR
VAR
AnalogSignal_par: SignalPar Real:= (MinVal:=0, MaxVal:=100, Unit:='%'); (*parameter settings of analog signal*)
value_outWord1: REALIO; (*Analog signal to output 1*)
value_outWord2: REALIO; (*Analog signal to output 2*)
CIF: REAL:= 150; (*Constant IN Flow Tank 1, 100 litre*)
CV1: REAL:= -500; (*correction value if over 10000*)
CV2: REAL:= 500; (*correction value if under 0*)
MT1V: REAL:= 10000; (*Max value in Tank*)
VT1I: REAL; (*Value Tank 1 IN*)
VT2I: REAL; (*Value Tank 2 IN*)
VT1O: REAL; (*Value Tank 1 OUT*)
MVT1O: REAL:= 150; (*MAX Value Tank 1 OUT*)
VT2O: REAL:= 10; (*Value Tank 2 OUT*)
PO: REAL:= 0; (*PUMP ON = Flow increase*)
VR: REAL; (*Relative tank volume*)
IVR: REAL;
END_VAR

(*-----------------------------------INFLOW CALCULATIONS FOR TANK 1 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --*)

IF AT1V < 10000 AND AT1V > 0 THEN
VT1I := (Valve so_position/100) * CIF; (*Regulated inflow based on constant inflow of 150 litres*)
ELSEIF AT1V > 10000 THEN REPEAT AT1V:=AT1V+CV1; UNTIL AT1V < 5000 END_REPEAT; (*correct value if it floats past limits. IN CASE SYNC IS DISRUPTED*)
ELSEIF AT1V < 0 THEN REPEAT AT1V:=AT1V+CV2; UNTIL AT1V > 5000 END_REPEAT; (*correct value if it floats past limits. IN CASE SYNC IS DISRUPTED*)
ELSE;
END_IF;
VR:= (AT1V/MT1V);
IVR := (1-VR);
VT1O := MVT1O * (1-(IVR*IVR));
AT1V := AT1V + VT1I - VT1O;
value_outword1.parameters:=ADR(AnalogSignal_par);
value_outword1.Value:=AT1V*0.01;
(*-----------------------------------INFLOW CALCULATIONS FOR TANK 2 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --*)

IF AT2V <= 10000 AND AT2V >= 0 THEN
VT2I := VT1O;
ELSEIF AT2V > 10000 THEN REPEAT AT2V:=AT2V+CV1; UNTIL AT2V < 5000 END_REPEAT; (*correct value if it floats past limits. IN CASE SYNC IS DISRUPTED*)
ELSEIF AT2V < 0 THEN REPEAT AT2V:=AT2V+CV2; UNTIL AT2V > 5000 END_REPEAT; (*correct value if it floats past limits. IN CASE SYNC IS DISRUPTED*)
ELSE;
END_IF;
IF pump.so_answeron = TRUE AND digvalve.so_answeropen = TRUE THEN
PO:= 100;
ELSEIF pump.so_answeron = FALSE AND digvalve.so_answeropen = TRUE THEN
VT2O:=100;
ELSEIF pump.so_answeron = TRUE AND digvalve.so_answeropen = FALSE THEN
VT2O:=100;
ELSE;
END_IF;
0045 PO:=0;
0046 ELSE
0047 VT2O:=0;
0048 PO:=0;
0049 END_IF;
0050
0051 AT2V := AT2V + VT2I - VT2O - PO;
0052 value_outword2.parameters:=ADR(AnalogSignal_par);
0053 value_outword2.Value:=AT2V * 0.01;
0054
0055 (*--------------------------------LEVEL MANIPULATION------------------------------------------------------------*)
0056 (* These are set to manipulate values and keep the level moving*)
0057
0058 (*Limitcontroller on Tank 1*)
0059 tankcritical(  
0060  SO_Level:=AT1V/100,
0061  ControlType:=FALSE,
0062  CONF:= conf.tankcritical,
0063  SO_ActivateOrder=> ) ;
0064
0065 (*Limitcontroller on Tank 2*)
0066 tankcritical2(  
0067  SO_Level:=AT2V/100,
0068  ControlType:=FALSE,
0069  CONF:= conf.tankcritical,
0070  SO_ActivateOrder=> ) ;
Conf_app

0001 TYPE Conf_app :
0002 STRUCT
0003
0004
0005
0006 GlobalAlarmBlockResetHour: UINT := 24; (* Hour for resetting GlobalAlarmBlock; <0 AND >23 turns off automatic disabling *)
0007 END_STRUCT
0008 END_TYPE

CONF_var

0001 TYPE CONF_var :
0002 STRUCT
0003
0004
0005
0006 App: CONF_App;
0007 pump: CONF_Motor1_1;
0008 level: CONF_Transmitter1_1;
0009 level2: CONF_Transmitter1_1;
0010 controller: CONF_PID1_1;
0011 digvalve: CONF_Valve1_1;
0012 SystemClock: CONF_TimeData1;
0013 tankcritical: CONF_LimitControl2_1;
0014 Waterout: CONF_Accumulator1_1;
0015 END_STRUCT
0016 END_TYPE

HMI_Var

0001 TYPE HMI_Var :
0002 STRUCT
0003
0004
0005
0006 App: CONF_App;
0007 IP_CmdGlobalAlarmBlock: BOOL; (* Global blocking of alarms *)
0008 IP_CmdGlobalReset: BOOL; (* Global reset of latched alarms *)
0009 IP_CmdAckPresence: BOOL; (* Acknowledge station presence *)
0010 END_STRUCT
0011 END_TYPE

IO_Var

0001 TYPE IO_Var :
0002 STRUCT
0003
0004
0005
0006 GlobalResetButton: BoolIO; (* Global reset of latched alarms *)
0007 END_STRUCT
0008 END_TYPE

REC_buffer

0001 TYPE REC_buffer :
0002 STRUCT
0003
0004
0005
0006 DATAreceived: INT;
0007 END_STRUCT
0008 END_TYPE

SEND_buffer

0001 TYPE SEND_buffer :
0002 STRUCT
0003
0004
0005
0006 Value: INT;
0007 END_STRUCT
0008 END_TYPE

SO_var

0001 TYPE SO_var :
0002 STRUCT
0003
0004
0005
0006 GlobalReset: BOOL;
0007 GlobalAlarmBlock: BOOL;
0008 END_STRUCT
0009 END_TYPE
module1_Module_Mapping
VAR_GLOBAL
in0 AT %IW502 : INT ; (* Analog input 0 *)
in1 AT %IW503 : INT ; (* Analog input 1 *)
in2 AT %IW504 : INT ; (* Analog input 2 *)
in3 AT %IW505 : INT ; (* Analog input 3 *)
in4 AT %IW506 : INT ; (* Analog input 4 *)
in5 AT %IW507 : INT ; (* Analog input 5 *)
in6 AT %IW508 : INT ; (* Analog input 6 *)
in7 AT %IW509 : INT ; (* Analog input 7 *)
out0 AT %QW502 : INT; (* Analog output 0 *)
out1 AT %QW503 : INT; (* Analog output 1 *)
END_VAR

module1_Module_Mapping_1
VAR_GLOBAL
in2_1 AT %IW510 : INT; (* Analog input 0 *)
END_VAR

module2_Module_Mapping
VAR_GLOBAL
in2_1 AT %IW510 : INT; (* Analog input 0 *)
END_VAR

Global_Variables
VAR_GLOBAL
* MOVING_OBJECTS *)
pump:FB_Motor1_1; (*Starts pump in Tank 2*)
digvalve:FB_Valve1_1; (*outflow valve tank2*)
valve:FB_V alve2_1; (*inflow valve Tank 1*)

*CONTROLLER*)
ccontroller:FB_PID1_1; (*PID, valve*)

*OPERATINGDATA*)
motor_data:FB_OperatingData1_1; (*run time counter for pump*)
timedata:FB_TimeData1; (*TIME*)
Waterout: FB_Accumulator1_1; (*waterquantity*)

* ANALOG *)
level:FB_Transmitter1_1; (*Level In Tank*)
level2:FB_Transmitter1_1; (*Level In Tank2*)

* PLC_PRG *)
SystemClock:FB_TimeData1; (*Timedata*)

*TIMESTAMPING*)
timestamp:TS_EVENT_STR; (*Timestamping*)

*water_tank*)
tankcritical:FB_LimitControl2_1; (**)
tankcritical2:FB_LimitControl2_1; (**)
Value_outWord: INT; (* ***)
Global_Variables_CONF

VAR_GLOBAL

SO:SO_var;

END_VAR

VAR_GLOBAL RETAIN PERSISTENT

CONF:CONF_var := (Level := ( (* Level in the tank *)
    IP_LimAH:=80, (* Limit HH-Alarm *)
    IP_LimAH:=60, (* Limit HA-Alarm *)
    IP_LimAL:=40, (* Limit L-Alarm *)
    IP_LimALL:=20, (* Limit LL-Alarm *)
    IP_Hysteresis:=0.2, (* Hysteresis in measuring unit *)
)

IO_Value_par := ( (* Signal parameters for analog input *)
    MaxVal:=100.0, (*Max value*)
    MinVal:=0.0, (*Min Value*)
    Unit:='m'), (*Unit of the signal*)

tankcritical := (IP_LimStart:=65.0,
    IP_Hysteresis:=0),

motor_data := ( (*IP_CmdManResRnt:= FALSE, Reset accumulated runtime *)
    (*IP_CmdManResCnt:=FALSE, Reset accumulated number of activations *)
    (*IP_CmdManResSrvc:=FALSE, Reset service required and time before service*)
    IP_AccRntTot:=1000, (*Acc. time total *)
    IP_AccCntTot:=1000), (*Acc. activations total *)

controller := (IP_Out:=20, (*REAL: PID output, can be changed in manual out mode*)
    IP_SP:=50, (*SetPoint, sent to the PID*)
    IP_Gain:=0.75, (*Gain coefficient for the PID*)
    IP_TI:=1, (*Integral coefficient for the PID [sec]*)
    IP_TD:=0, (*Differential coefficient for the PID [sec]*)
    IP_MaxDeviation:=0, (*Alarm if SP and PV differ more than this value; 0=disable alarm*)
    IP_MinDeviation:=0, (*Alarm if PV derivative is less than this value; 0=ignore this condition*)
    IP_AFilterTime:=5, (*Timeout before alarm is raised*)
    IP_CmdManOut:=FALSE, (*Activate output manual override*)
    IP_CmdManSP:=FALSE, (*Activate SP manual override*)
    IP_CmdOutLim:=FALSE), (*Limit Out change rate in ManOut and Track modes*)

valve := (IP_TravelTime:=10,
    IP_CmdManMode:=FALSE), (*Command Auto (0) / Manual (1)*)

Global_Variables_CONF
digvalve:=(
  IP_AcofTime:=10, (* Limit switch timeout *)
  IP_TravelTime := 10, (* Open<>Close travel time *)
  IP_CmdManMode:=FALSE, (* Command Auto (0) / Manual (1) *)
  IP_CmdManBlock:=FALSE (* Command manual blocking valve *)
);

END_VAR

Variable_Configuration
VAR_CONFIG
END_VAR

SPC_Bit_Enum
VAR_GLOBAL CONSTANT
("Configuration bit enumeration")
Force:INT := 0; (* Stop copy form Value to ValueIO *)
Fault:INT := 1; (* Module fault, copy stopped *)
Value INT := 2; (* Value to application *)
ValueIO INT := 3; (* Value in hardware *)
Overflow:INT := 2; (* ValueIO above max *)
Underflow:INT := 3; (* ValueIO below max *)
isBoolIO INT := 5; (true=This variable is BoolIO type)
isRealIO INT := 6; (true=This variable is RealIO type)
isIntIO INT := 7; (true=This variable is IntIO type)
("HW_Diag bit enumerations")
IOBusFail: INT := 0; (* General I/O bus error *)
IOBusWarn: INT := 1; (* I/O bus subunit error *)
IntEthFail: INT := 2; (* General Internal Ethernet error *)
IntEthWarn: INT := 3; (* Internal Ethernet subunit error *)
IntCOM1Fail: INT := 4; (* General Internal COM port 1 error *)
IntCOM1Warn: INT := 5; (* Internal COM port 1 subunit error *)
IntCOM2Fail: INT := 6; (* General Internal COM port 2 error *)
IntCOM2Warn: INT := 7; (* Internal COM port 2 subunit error *)
ExtCI1Fail: INT := 8; (* General External Communications Interface 1 error *)
ExtCI1Warn: INT := 9; (* External Communications Interface 1 subunit error *)
ExtCI2Fail: INT := 10; (* General External Communications Interface 2 error *)
ExtCI2Warn: INT := 11; (* External Communications Interface 2 subunit error *)
ExtCI3Fail: INT := 12; (* General External Communications Interface 3 error *)
ExtCI3Warn: INT := 13; (* External Communications Interface 3 subunit error *)
ExtCI4Fail: INT := 14; (* General External Communications Interface 4 error *)
ExtCI4Warn: INT := 15; (* External Communications Interface 4 subunit error *)
SO_BatteryAlarm: INT := 16; (* PLC CPU Battery is not OK *)
("flag off")
END_VAR

SPC_Global_Constant
VAR_GLOBAL CONSTANT
("Visualization constants")
SecondToHourFactor: REAL := 0.000277777777777778;
("Visualization color constants")
VisuColorBackground: DINT := 16#FFFFFF;
VisuColorAlarmMain: DINT := 16#0000C8;
VisuColorAlarmAux: DINT := 16#80FFFF;
VisuColorActiveMain: DINT := 16#004000;
VisuColorActiveAux: DINT := 16#008000;
VisuColorInactive: DINT := 16#C0C0C0;
VisuColorForce: DINT := 16#00FFFF;
("flag off")
END_VAR
HWOnlineChangeCount:USINT := 10; (* Increased +10 with each online change *)

VAR_GLOBAL

VAR_GLOBAL PERSISTENT

UnderflowExt: INT := -31795; (* Underflow warning limit for analog signals *)
UnderflowNorm: INT := -3456; (* Underflow warning limit for analog signals *)
OverflowLimit: INT := 31795; (* Overflow warning limit for analog signals *)

{library public}

TimeConfDSTNoEntries: USINT := 16; (* Number of entries in DST table *)
TimeConfDSTEnd15: DT := DT#2028-10-29-03:00:00; (* Custom Daylight saving period end, local time *)
TimeConfDSTStart15: DT := DT#2028-03-26-02:00:00; (* Custom Daylight saving period start, local time *)

(***Signal limit constants***)

Value 4.20mA 0.20mA 0.10V -10..10V
OverflowLimit 31795 22.5mA 23mA 11.5V 11.5V
UnderflowNorm -3456 2mA -1.25V -1.25V
UnderflowExt -31795 - - -11.5V

Since signal limits are digital, they correspond to different actual measured values, depending on the selected signal range. The following table is for reference only, consult specific I/O module documentation for precise range data.
HWOnlineChangeCallbackRunning:BOOL:=FALSE; (*Online change callback has been started*)
HWInputCallbackWatchdog:USINT:=0; (*0:Stopped, 1:Set by Callback, 2:Set by SPCControl*)
HWOutputCallbackWatchdog:USINT:=0; (*0:Stopped, 1:Set by Callback, 2:Set by SPCControl*)

ZeroHundredNormFixedPar:SignalPar Real:=(MaxVal:=100, MinVal:=0, Unit:='%', ExtendedRange:=FALSE);
ZeroHundredExtFixedPar:SignalPar Real:=(MaxVal:=100, MinVal:=0, Unit:='%', ExtendedRange:=TRUE);

END_VAR

Global_Variables
VAR_GLOBAL
END_VAR

GL_AC500_Diagnosis
VAR_GLOBAL
CPU : CPU_LOAD; (* structure of CPU load variables *)
diagCPU : CPU_DIAG; (* structure of CPU diagnosis variables *)
diagCS31 : CS31_DIAG; (* structure of CS31 diagnosis variables *)
diagFBP : FBP_DIAG; (* structure of FBP diagnosis variables *)
END_VAR

GL_Diag_Constant
VAR_GLOBAL CONSTANT
wERNO_SIMULATION_MODE : WORD := 16#50FF;
END_VAR

GL_Diag_Constant
VAR_GLOBAL CONSTANT
wERNO_SIMULATION_MODE_EXT : WORD := 16#50FF;
END_VAR

LIBRARY_VERSION_INFORMATION
VAR_GLOBAL
LIBRARY: Ethernet_AC500_V10
DESCRIPTION:
FBs to use Ethernet on AC500
Global_Variables

VAR_GLOBAL

END_VAR

HA_Global

VAR_GLOBAL CONSTANT

(* HA specif ic constants*)

wHA_ER_WRONG_COM: WORD := 16#2001; (* Wrong COM Number at the COM Input *)

wHA_ER_WRONG_NO_CS31_PROTOCOL: WORD := 16#3003; (* No CS31 Protocol on COM *)

wHA_ER_CS590_CFG_NOT_COMPLETE: WORD := 16#3029; (* CI590 Slave configuration not complete *)

wHA_ER_CS590_Cross_Wiring: WORD := 16#2029; (* CI590 Slaves in Bus1 and Bus2 are Mix wired *)

wHA_ER_CS31_CI590_CFG_ERROR: WORD := 16#201C; (* The number CI590 Configured on both CPUs are not same *)

wHA_ER_REMOTE_CI590_FAILURE: WORD := 16#1029; (* Remote CI590 Failure *)

wHA_ER_NO_ETHERNET_LINK: WORD := 16#2013; (* No Ethernet Link *)

wHA_ER_CS31_MASTER_CROSS_WIRE: WORD := 16#2029; (* CS31 Master Cross Wired *)

wHA_ER_Remote_CPU_Failure: WORD := 16#101B; (* Remote CPU Failure *)

wHA_ER_Remote_CS31_BUS_Failure: WORD := 16#201B; (* Remote CS31 Bus (Master) Failure *)

wHA_ERNO_COUPLER_CONFI: WORD := 16#6076; (* Coupler configuration for the sync connection is invalid *)

wHA_ERNO_TBL_OVERFLOW: WORD := 16#2022; (* HA data reference table is full *)

bHA_FRAME_TYPE_STATUS: BYTE := 16#42; (* HA status frame *)

bHA_FRAME_TYPE_STATUS_DATA: BYTE := 16#DD; (* HA status and data frame *)

uHA_MadBufferSizes: UINT := 1400; (* HA Sync max Frame size *)

uHA_MadSyncEntries: UINT := 256; (* Total size of the sync entry array NtdSyncFBPins* MaxSyncFBs *)

HA_MAX_DATA_IN_ETH_FRAME: UINT := 1336; (* Ethernet Frame Length - Size of Header *)

uDelay_CI590_err: UINT := 25; (* Number of cycle delay before declaring the CI590 Failure *)

uDelay_Data: BYTE := 5; (* Max number of cycles for data update*)

uSyncCycle: UINT := 6; (* Number of cycle delay before acknowledge signal is sent *)

uFilterTimeHASyncArrayInit: UINT := 200; (*Number of cycles to wait before calculating HA_SyncArrayInit*)

END_VAR
HA_Global_Variables

VAR_GLOBAL

fG_HA_PRIMARY: BOOL := 0; (* State of the AC500 CPU (FALSE -> PM acts as Secondary, TRUE -> PM acts as Primary *)

fG_HA_PM1_PRIMARY: BOOL := FALSE; (* Indication of primary PM, TRUE -> PM1 / IP1 acts as Primary *)

fG_HA_CPU_STOP: BOOL := FALSE; (* If TRUE -> Indicates the CPU in STOP MODE *)

(* HA error information *)

fG_HA_Err: BOOL := FALSE; (* HA error state *)

wG_HA_ErNo: WORD := 0; (* HA error code *)

bitG_Data_ERR: BOOL := FALSE; (* HA data sync error state *)

wG_Data_ERNO: WORD := 0; (* HA data error sync code *)

(* HA synchronization link configuration *)

dwG_HA_OwnIP: DWORD := 0; (* own IP address on sync link connection *)

dwG_HA_OtherIP: DWORD := 0; (* other PMs IP address on sync link connection *)

bG_HA_Slot: BYTE := 0; (* slot of interface to sync link connection *)

(* OPC Server connection check *)

dwG_HA_ServerAlive: DWORD := 0; (* Life counter incremented by OPC server *)

byLastDataDelay: BYTE := 0;

byRefreshDataDelay: BYTE := FALSE;

byCntDataDelay: BYTE := 0;

wETH_Life: WORD; (* Ethernet Life Count *)

dwHATimersBaseTime : DWORD;

END_VAR
DESCRIPTION:

Common library for the AC500 system

RESTRICTIONS: Just be used in ABB AC500-PLCs

-----------------------------------------------------------------------------------------------

DATE: 2012-03-20

MODIFIED:  

2012-03-20 V1.0.0  Added function block IO_PROD_ENTRY_READ 

2012-04-04 V1.1.0  Added function block PM_INFO, IO_MODULE_INFO_EXT, a datatype for internal use only zRTS_VERSION_INFO

2012-06-22 V1.2.0  Added function blocks DPRAM_SM3XX_REC and DPRAM_SM3XX_SEND 

2013-04-09 V1.3.0  Added function block BOOTPRG_HASH_INFO

-----------------------------------------------------------------------------------------------

END_VAR
Task configuration

- System events
  - Main (PRIORITY := 10, INTERVAL := T#500ms)
  - PLC_PRG();
  - CONTROLLERS();
  - ANALOG();
  - OPERATINGDATA();
  - MOVING_OBJECTS();
  - watertank();
- UDP (PRIORITY := 10, INTERVAL := T#1s0ms)
- MODBUS (PRIORITY := 10, INTERVAL := T#500ms)
- Timestamping (PRIORITY := 10, INTERVAL := T#500ms)
- HA(task (PRIORITY := 10, INTERVAL := T#20ms)
- HA_PRG();

Notepad

Watch- and Recipe Manager

Standard
Watch0
0001 MODBUS.errnumber
0002

Workspace

Parameter Manager

0001 Parameter-Manager
0002 ===============