Tampere University

Verneri Välilehto

DESIGNING CONTROL SYSTEM ARCHITECTURE FOR PRESSURIZATION SYSTEM

Master of Science thesis
Faculty of Engineering and Natural Sciences
Examiners: Senior Research Fellow Matti Linjama
University Lecturer Petteri Multanen
October 2023

ABSTRACT

Verneri Välilehto: Designing control system architecture for pressurization system
Master of Science thesis
Tampere University
Automation Engineering
October 2023

Pressurization and controlling pressure are one of the basic applications in hydraulic systems. In hydraulic systems significant amount of energy can be lost, so as energy efficiency is becoming more important, attention should be paid to reducing unnecessary losses. The purpose of this thesis is to design and implement control system architecture for pressurization system, which is used to study improving energy efficiency in this application. Control system is implemented as distributed system with regular programmable logic controllers, actuators and sensors, which are connected with CAN bus. Some of the code for control system is generated from Simulink model.

First step in the project was to get familiar with operation of the system to be controlled and the theory behind proposed control system solution. Operation or operation principle of the system to be controlled will not be discussed in this thesis. Designing control system architecture and upper-level controller for system in Simulink was possible after getting familiar with system and systems special requirements. System model was tested and improved in Simulink, until it was considered to meet requirements that were set for it. After this PLC code could be generated and the rest of the control system could be implemented. In the last step it was verified with HiL-simulations, that the control system and CAN bus meet their performance requirements for this application.

Measurements verified that performance of the control system is good enough for this application. CAN messages might need some optimizing if the number of actuators in system is increased. Extrapolating system performance from measurements must be considered as rough estimate, because the number of datapoints was low.

This project contains confidential information, so some parts of the system, it's operation or purpose are not discussed in this thesis.

Keywords: Programmable logic controller, model-based design, CAN bus, pressurization, code generation

TIIVISTELMÄ

Verneri Välilehto: Ohjausjärjestelmän arkkitehtuurin suunnittelu paineistusjärjestelmään
Diplomityö
Tampereen yliopisto
Automaatiotekniikka
Lokakuu 2023

Paineistaminen ja paineensäätö on yksi perussovelluksista hydrauliikassa. Hydraulijärjestelmissä huomattava osa energiasta voi mennä hukkaan varsinaisen työn sijasta, joten energiatehokkuuden merkityksen kasvaessa hukan vähentämiseen tulee kiinnittää aiempaa enemmän huomiota. Tämän työn tarkoituksena on suunnitella ja toteuttaa ohjausjärjestelmän arkkitehtuuri erääseen paineistusjärjestelmään, jolla tutkitaan energiatehokkuuden parantamista kyseisessä sovelluksessa. Ohjausjärjestelmä toteutetaan hajautetusti tavanomaisilla ohjelmoitavilla logiikoilla ja erinäisillä toimilaitteilla ja antureilla, jotka on kytketty toisiinsa CAN-väylällä. Osa ohjausjärjestelmän ohjelmakoodista on tuotettu hyödyntämällä mallipohjaista suunnittelua ja PLC-koodin generoimista Simulink-mallista.

Työn aluksi tutustuttiin ohjattavan järjestelmän toimintaan ja valitun ohjausjärjestelmäratkaisun taustalla olevaan teoriaan. Ohjattavan järjestelmän toimintaa tai toimintaperiaatetta ei käsitellä tässä työssä. Ohjausjärjestelmän arkkitehtuurin ja ylemmän tason logiikan suunnitteleminen ja toteuttaminen Simulinkissä oli mahdollista ohjattavaan järjestelmään ja sen erikoisvaatimuksiin tutustumisen jälkeen. Järjestelmän mallia testattiin ja parannettiin, kunnes sen katsottiin täyttävän sille asetetut tavoitteet. Tämän jälkeen PLC-koodin generoiminen ja ohjausjärjestelmän toteuttaminen ohjelmoitavilla logiikoilla oli mahdollista. Lopuksi HiL-simuloinnin avulla varmistettiin, että toteutetun ohjausjärjestelmän ja CAN-väylän suorituskyky on riittävä tälle sovellukselle.

Mittaustuloksien avulla varmistui, että ohjausjärjestelmän suorituskyky on riittävä tälle sovellukselle. Järjestelmän viestiliikennettä voidaan joutua optimoimaan, jos toimilaitteiden määrää kasvatetaan. Järjestelmän suorituskyvyn ekstrapolointi toimilaitteiden määrän kasvaessa on hyvin suuntaa antava, koska siihen käytettävien datapisteiden määrä oli pieni.

Tässä työssä käsitellään ohjelmoitavien logiikoiden toimintaperiaatetta, CAN-väylän toimintaa, mallipohjaista suunnittelua, ohjausjärjestelmän arkkitehtuurin suunnittelemista, käytettävää testijärjestelmää ja mittaustuloksia.

Tämä projekti sisältää luottamuksellista tietoa, joten joitain osia järjestelmästä, sen toiminnasta tai tarkoituksesta ei käsitellä tässä työssä.

Avainsanat: ohjelmoitava logiikka, mallipohjainen suunnittelu, CAN väylä, paineistaminen, koodin generoiminen

PREFACE

This Master of Science thesis was written in the research group of Digital Hydraulics in Tampere University between November 2022 and September 2023. This thesis was cofunded by EIT RawMaterials project Aufero and Tampere University.

I wish to thank my supervisor Senior Research Fellow Matti Linjama for the interesting subject for my thesis and guidance during this process. I also want to thank my other examiner University Lecturer Petteri Multanen for his input, as well Project Researcher Jyrki Tammisto for his help.

Finally, I want to thank my family, friends, and co-workers at IHA.

Tampere, 3rd October 2023

Verneri Välilehto

CONTENTS

1. Introduction 1
2. Theoretical background 3
   2.1 Programmable logic controller 3
      2.1.1 PLC hardware 3
      2.1.2 Program cycle 4
      2.1.3 PLC programming elements 5
      2.1.4 IEC 61131-3 data types 7
      2.1.5 IEC 61131-3 programming languages 8
   2.2 Controller area network 9
      2.2.1 CAN bus physical properties 9
      2.2.2 CANopen 10
      2.2.3 CAN data frame 10
   2.3 Model-based design 11
3. Designing control system architecture 14
   3.1 Designing upper-level architecture 15
   3.2 Re-usability of components 16
   3.3 Handling I/O 16
   3.4 Handling parameters 18
   3.5 Handling CAN messages between devices 18
   3.6 Handling real world constraints 19
   3.7 Designing and implementing control system architecture 23
   3.8 Designing and implementing user interface 25
4. Test system 28
   4.1 Control hardware 28
   4.2 Selected bus solution 28
   4.3 Actuator 29
   4.4 IO in test system 30
   4.5 Hydraulic valves in test system 30
      4.5.1 Servo valve 30
      4.5.2 Independent metering valves 31
   4.6 Hardware-in-Loop simulation setup 33
   4.7 Simulating system performance 35

5. Results 36
5.1 System performance, Hardware-in-Loop 36
5.1.1 250 kbit/s CAN bus 37
5.1.2 500 kbit/s CAN bus 40
5.2 Analyzing results 43
6. Conclusions 45
References 47
Appendix A: CAN network architecture 49
Appendix B: Program architecture, display 50
Appendix C: Program architecture, secondary controller 51

LIST OF FIGURES

2.1 PLC scan based on [1]. 4
2.2 PLC program structure based on [2, 4]. 5
2.3 CAN bus state based on [15]. 9
2.4 Example MBD workflow in Simulink based on [22]. 12

3.1 High level architecture for control system. 15
3.2 Controller operating permission. 21
3.3 Queuing logic. 22
3.4 Upper-level controller. 24
3.5 Graphical user interface, main view. 26

4.1 Hydraulic diagram of the actuator. 29
4.2 Servo valve symbol. 31
4.3 Independent metering valve system. 31
4.4 IMV differential control, positive direction. 32
4.5 IMV differential control, negative direction. 32
4.6 HiL test system. 33

5.1 CAN bus 1 load. 37
5.2 CAN bus 2 load. 38
5.3 Normalized actuator pressure and cylinder chamber pressures. 38
5.4 Normalized actuator pressures and cylinder chamber pressures. 39
5.5 CAN bus 1 load. 40
5.6 Normalized actuator pressure and cylinder chamber pressures. 41
5.7 Normalized actuator pressures and cylinder chamber pressures. 42

LIST OF TABLES

2.1 Simulink – CODESYS data type transformation. 7
2.2 CANopen bit timing table [18]. 10
2.3 CAN bus arbitration based on [11]. 11
2.4 Incremental model-based test integration levels based on [21]. 13

5.1 Control system performance, Hardware-in-Loop. 250 kbit/s. 37
5.2 Control system performance, Hardware-in-Loop. 500 kbit/s. 40
5.3 Control system performance analyzing, Hardware-in-Loop. 250 kbit/s bus. . 43
5.4 Maximum number of actuators, 250 kbit/s bus. 43
5.5 Control system performance analyzing, Hardware-in-Loop. 500 kbit/s bus. . 44
5.6 Maximum number of actuators, 500 kbit/s bus. 44

LIST OF SYMBOLS AND ABBREVIATIONS

CAN	Controller area network
CPU	Central processing unit
GUI	Graphical user interface
IMV	Independent metering valve
I/O	Input/output
MBD	Model-based design
PLC	Programmable logic controller
POU	Program organization unit

1. INTRODUCTION

Hydraulic is used to transfer power via pressurized fluid. Hydraulic power can be used to generate linear motion and force or rotational motion and torque with applicable actuators. In both applications, pressurization of the fluid is key to make transferring power possible. Force or torque can be controlled in said actuators with some different strategies. One method for controlling force and torque is to control hydraulic pressures in actuators chambers. There are a few possible strategies to control pressure in hydraulic line or actuators chamber. The simplest way for this is to connect a line or chamber directly to pressure- or tank line of the system. Even though this option is simple, it has multiple disadvantages. It is restricted to single pressure level(s), and pressure level in actuator can be varied only via altering pressure level in system or force caused by external load. In addition, controlling the position or velocity of the actuator can be difficult and inaccurate. This option is even worse if there are multiple actuators in the system that require different pressure levels at the same time. This method can be improved to some extent with pump control. Therefore, this control method is feasible only for most basic applications.

Bit more advanced control for pressure level(s) can be achieved with basic on/off valves, flow restriction valve(s), pressure reducing valve(s) and/or pressure relief valve(s) for each actuator to be controlled. With this solution, the pressure level for actuator can be lower than pressure level of the system. This pressure control method can be feasible for some simple systems, but it still lacks real time controllability for pressure or position, if it is implemented with valves that need to be adjusted manually.

Third and most feasible solution for better pressure control in many applications is utilizing some kind of valve solution, which opening can be adjusted proportionally. This can be achieved for example with proportional valve(s) or servo valve(s). These kinds of valve solutions can be used to control hydraulic pressure and/or fluid flow rate with electronic control. These valve solutions performance in accuracy, speed and flow capacity can vary greatly, as well their cost.

Energy efficiency is becoming more important in many applications in various fields, as well as in hydraulics. As pressurization and pressure control is one of the main functions in hydraulic systems, it is crucial to find ways to reduce losses and recover energy. In addition, reducing costs for controller solutions that can be used to improve efficiency is necessary. Key factor for achieving this can be utilizing commercial control hardware and relatively inexpensive valves instead of control- and measurement systems that are meant for laboratory conditions and relatively expensive valves.

The purpose of this thesis is to design and implement control system architecture for hydraulic pressurization system with commercial hardware. Implemented control system must handle all operations and communication in system. Control systems performance must be good enough for selected hardware.

Control system architecture and graphical user interface (GUI) will be implemented in CODESYS. Simulink and Model-Based design (MBD) will be utilized for analyzing the system, designing components for upper-level controller and actuator level controllers and generating code for programmable logic controller (PLC). The control system will be integrated and tested with model and actual PLC controlling model. Control system performance will be measured and analyzed with Hardware-in-Loop simulation.

System contains multiple subsystems for different purposes. Main controller is used to control system operation with touchscreen and GUI. All subsystems must be able to communicate with the main controller. Main controller is connected to secondary controllers with fieldbus. GUI can also be used to monitor system and some subsystems operation statuses.

The test system contains a hydraulic actuator, which is used to control pressure in hydraulic capacitance. This kind of actuator requires robust, fast, and accurate control to meet performance requirements that are set for it. The test system will be used to measure control performance with two different hydraulic valve solutions. This project contains confidential information. Therefore, exact properties of the system or purpose of the subsystems will not be discussed.

The background of the problem is described briefly in the first chapter. Theoretical background of the programmable logic controllers, CAN bus and Model-Based Design are discussed in the second chapter. Some factors that must be considered when designing control system architecture, design process of control system architecture and design process of GUI are discussed in the third chapter. The test system and some most important factors are presented in the fourth chapter. Control system architectures performance measurements with Hardware-in-Loop simulations are presented and analyzed in the fifth chapter.

2. THEORETICAL BACKGROUND

The theory presented in this thesis will focus on programmable logic controllers, CAN bus operation and model-based design. Theory of hydraulic system or test system itself will not be discussed.

2.1 Programmable logic controller

Programmable logic controller (PLC) is a microprocessor-based controller that is used to control machines and processes. It has programmable memory to store instructions and implement functions. PLC is designed to be operated and programmed by engineers, who might not be familiar with computer programming languages [1].

2.1.1 PLC hardware

PLC consists of a central processing unit (CPU), memory, power supply, and modules for inputs, outputs, and communication. Depending on PLC size, these all can be in single unit or separate components in larger modular system [1, 2].

CPU is unit, that contains microprocessor. CPU is used to perform instructions and calculations for system [1, 2]. Normally CPU, memory and communication modules are part of one physical unit [2].

There are commonly two types of memory units in PLC, which are Read-only memory (ROM) and Random-access memory (RAM). Contents of ROM cannot be deleted, so it is used to store PLC's operating system and system data. RAM is used to store programs and data, that is related to executing programs. This data can be for example I/O values, timers, and counters [2].

Power supply is used to convert 220 V AC or 120 V AC to 24 V DC for system. 24 V DC is usually used for sensors and transmitter. 24 V DC is also converted to 5 V DC for PLC's internal electronics [2].

I/O modules are used to read information from inputs and write controls for outputs. I/O can be digital or analog. Types for analog I/O can vary greatly depending on system, ranging from different voltage ranges to current signals. I/O signals can be isolated from microprocessor with optoisolators, relays, transistors, and triacs. Analog signals are digitalized for PLC, and digital outputs from PLC can be converted to analog [1, 2].

Communication unit is used for communication between devices and for programming PLC. Various protocols can be used for communication between devices. PLC can be programmed connecting PC directly to PLC, via Ethernet or via network of devices [2].

2.1.2 Program cycle

PLC program runs on a continuous loop, that contains multiple sequences. PLC program execution cycle is called program scan. PLC program scan according to [1] is presented in figure 2.1.

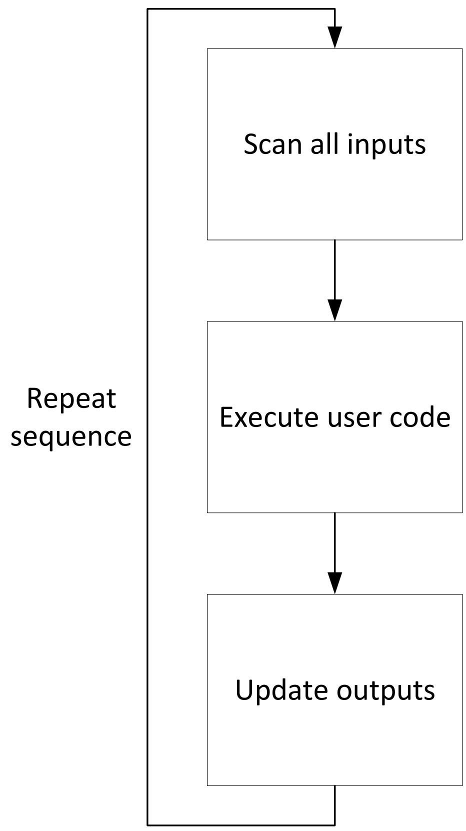


Figure 2.1. *PLC scan based on [1].*

First, PLC checks its own state in internal processing before performing any other operations. In the second step all inputs are read to input image in program scan. After reading all inputs, user code is executed with input image and internal user storage and outputs are saved to output image. When all user code has been executed, output image is written to outputs. Therefore, the program cycle operates with snapshots of system inputs and outputs instead of continuous reading and writing [1, 2, 3].

As a program runs continually, it can be advantageous to be able to select more specific operation mode for execution. There are three common modes for PLC program execution, which are cyclic, freewheeling and event based. In cyclic mode each program scan is started with fixed interval. Cyclic mode is a great choice, for example for PID-controller, so controllers operation is predictable. Freewheeling means, that program is ran continually as fast as possible. This causes cycle time to be unpredictable, so it is not suitable for use cases that contain, for example timers. In event-controlled execution program is ran only, if condition for execution is true [1, 2].

2.1.3 PLC programming elements

Standard IEC 61131-3 defines basic elements for PLC programming [1, 2]. Basic elements contain program structures, programming languages and elementary data types. PLC program structure based on [2, 4] is presented in figure 2.2.

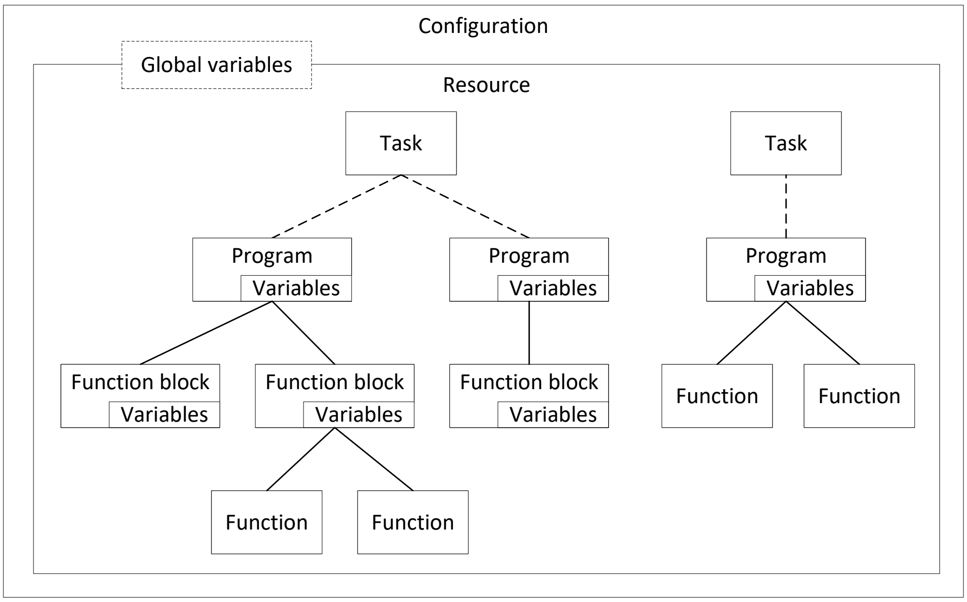


Figure 2.2. *PLC program structure based on [2, 4].*

Top level software structure is called configuration, and it is defined in standard IEC 61131-1 [5]. Software can contain multiple configurations, but one control system can contain just one configuration. One control system can be one PLC or multiple CPUs on larger rack mounted solution [2].

Each configuration contains one or more resources. Resource can be PLC or application [2].

Each resource can contain one or more tasks. Task controls, how program is executed. Each task can have different cycle time and operation mode. Each task also has priority, which determines execution order for tasks within resource [2]. If lower priority task is executed while higher priority task is activated, there are two options for PLC to decide, how to handle conflict. It depends on PLC vendor, which option is selected. First option is interrupting execution of the lower priority task immediately and execute higher priority task. This is called pre-emptive scheduling. Second option is not to interrupt lower priority task, but finish executing it before executing higher priority task. This is called non-pre-emptive scheduling [6].

Each task can contain one or more programs. Programs are constructed with program organization units (POU). There are four different types of POUs defined in standard IEC 61131-3 [2, 4], which are

- Program
- Function block
- Function
- Class.

Class is not covered in this thesis, as it is less common in PLC applications. For example, CODESYS does not recognize class [7].

Each POU is independent program unit. Using POUs makes it possible to create reusable blocks of code and construct software from more manageable pieces. POU encapsulates its data [2].

Program can contain zero or more function blocks. Function blocks can be user defined or standard blocks. Function blocks can have internal states, so output of the function block can be different with same inputs. There can be multiple instances of the same function block, which all have their own internal states. Function blocks can contain other function blocks [2].

Programs or function blocks can also contain zero or more functions. Unlike function blocks, functions don't have any internal states. Functions output will always be same with same inputs. Functions can also call other functions [2].

Variables are used to identify data objects that may change. Variable can be used for example for data in memory, inputs, and outputs. Variable can be elementary data type, user defined data type or reference to another variable. Scope of the variable varies depending on its declaration, and it can range from POU's internal variables to global variables. Qualifiers can be used to define some properties of the variables. By default, all variables are set to their initial values at system start-up. This can be adjusted with qualifier RETAIN. The variable that is declared with qualifier RETAIN will not be set to initial value at startup but will instead keep its last value. Another example of qualifiers is CONSTANT. Variables that is declared as CONSTANT value cannot be altered [4].

2.1.4 IEC 61131-3 data types

There are large number of different data types defined in standard IEC 61131-3 for different purposes. In scope of this thesis, numeric and boolean data types are most important. Data types defined in standard for time, date, characters, and character strings are not discussed in this thesis. Keywords for same data types defined in IEC 61131-3 [4] and in MATLAB and Simulink [8, 9] vary from each other. Numeric and boolean data types and their equivalents in Simulink are listed in table 2.1.

Table 2.1. *Simulink – CODESYS data type transformation.*

Simulink	CODESYS
double	LREAL
single	REAL
int8	SINT
uint8	USINT
int16	INT
uint16	UINT
int32	DINT
uint32	UDINT
boolean	BOOL

In addition to elementary data types, user can define own data types that extend some existing data type. For example, user can define allowed range for custom data type. Allowed range can also be defined when declaring variable, so creating custom data type isn't always necessary [2].

User defined data types also include three different options for multielement variables, which are enumerated data type, array data type and structured data type [4]. Multielement variables can be used to create custom data types, that contain multiple variables that are derived from elementary data types. Using multielement variables can be beneficial, if there is need to process sets of variables or structure code [2]. Enumeration can be used to specify value, that is associated with identifier. Array is collection of data elements of same type. Structure can contain various types of sub-elements, that can be accessed with names. Arrays and structures can be nested [4]. Enumerated and structured data types must be defined as data type, and they are accessible globally. Array can be defined in a similar manner to make it accessible globally, or it can be defined locally [2, 4].

2.1.5 IEC 61131-3 programming languages

There are five programming languages defined in standard IEC 61131-3. Structured text and Instruction list are textual languages, and Function block diagram, Ladder diagram and Sequential function chart are graphical languages. It is possible with some implementer dependent limitations to use programming languages defined in standard in conjunction with others. For example, a program that is programmed with function block diagram can call function, which is programmed with structured text [4].

Structured text (ST) is textual language, that is derived from Pascal. ST is constructed from expressions and statements. Expression contains operators and operands. Evaluating expression results to value. Expressions are used to perform mathematical operations or comparisons for variables. Statements are used to assign values, call functions, function blocks or methods, perform selections or perform iteration. It is worth noting, that end of textual line in ST is same as space and statements end with semicolon [4]. ST has been developed specially for more complex operations [2].

Instruction list (IL) is assembler like language. It is considered outdated, and it will be removed from the next edition of IEC 61131-3 standard. It contains a sequence of instructions. Each instruction is on the new line and shall contain operator. Operator can also contain modifiers and operands. Operand can be any of the data representations [4]. IL is used in some applications, because IL and ladder diagram are older than other languages defined in standard. With some older PLC's IL and ladder diagram are only supported programming languages [2].

Program elements in ladder diagram (LD) or function block diagram (FBD) are constructed as networks, which can contain other program elements, inputs, and outputs. Program elements are presented with blocks. Blocks can contain various number of inputs and outputs. The structure of the networks is different for LD and FBD [4].

Function block diagram is graphical language, that is used to create required functionality of program with connecting programs, function blocks and functions. Same operations can be executed with FBD and ST. FBD might give better overview of the system [2].

LD is based on relay diagram. Network in LD is constructed from left and right power rails. Network can contain one or more rungs between power rails. Program elements are connected in rungs. Rungs can also contain inputs and outputs. Inputs in LD are presented as contacts. Outputs in LD are presented as coils [4]. LD can be used for simple systems, but it might be limited for more complex systems [2].

Sequential function chart is graphical tool, that can be used to create program structure based on flow chart. Other languages defined in standard can be used with SFC. Transitions and actions in SFC must be implemented with some of them [1].

2.2 Controller area network

Controller Area Network (CAN) is bus, that was originally designed for vehicles by Bosch. Using CAN can reduce the amount of required wiring harnesses for system instead of using point-to-point communication. Standard ISO 11898 describing protocol and physical layer for CAN bus was released 1993 [10]. ISO 11898-1 describes link layer [11] and ISO 11898-2 describes some physical layer options [12]. CAN bus can also be used as a field bus in multiple applications and there are large amount of defined application profiles and specifications by CAN in Automation [13]. First applications for CAN were on other industries than passenger cars [10].

2.2.1 CAN bus physical properties

Nodes in bus are connected with two wires, CAN_H(igh) and CAN_L(ow). Differential voltage of two lines is used to represent logic state of bus. Recessive mode represents logic value of 1 and dominant mode represents logic value of 0 [14]. If the bus is in recessive mode, nominal voltage in CAN_H and CAN_L is 2,5 V. If bus is in dominant mode, nominal voltage in CAN_H is 3,5 V and nominal voltage in CAN_L is 1,5 V. Acceptable voltage in CAN_H and CAN_L in recessive mode is from 2,0 V to 3,0 V. Acceptable voltage range in CAN_H in dominant mode is from 2,75 V to 4,5 V and acceptable voltage range in CAN_L in dominant mode is from 0,5 V to 2,25 V. Acceptable differential voltage in dominant mode between CAN_H and CAN_L is from 1,5 V to 3,0 V [15]. The bus state based on [15] is visualized in figure 2.3.

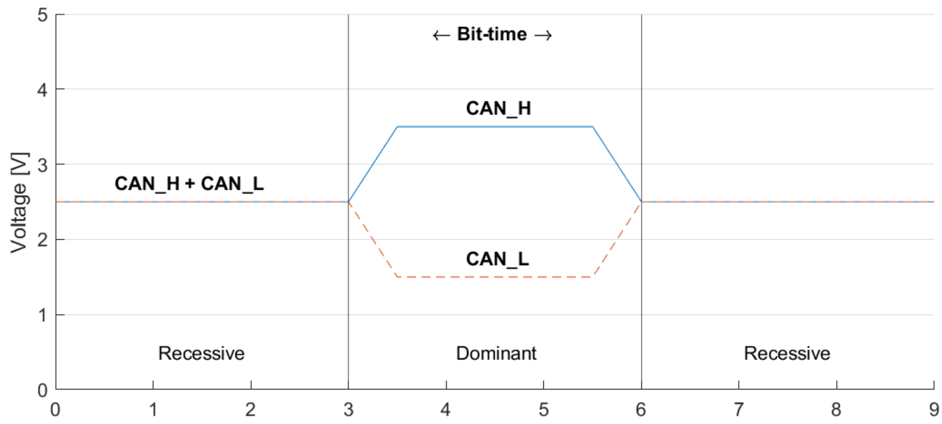


Figure 2.3. *CAN bus state based on [15].*

CAN network must have at least two nodes. Ends of the network are terminated with 120 Ω resistors. In some application termination can be done instead with two 60 Ω resistors with intermediate capacitor that is connected to ground [14].

2.2.2 CANopen

CANopen is a CAN-based communication system. Standardized profiles for devices and applications simplify integrating of system [16]. The first version of CANopen specification was released 1994. The purpose of CANopen was to develop application layer for embedded machine control that was easy to implement [17].

The greatest achievable bit-rate depends on the total bus length. The longest total bus length for each bit-rate in CANopen network is presented in table 2.2.

Table 2.2. *CANopen bit timing table [18].*

Bit-Rate	Bus length
1 Mbit/s	25 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
50 kbit/s	1000 m
20 kbit/s	2500 m
10 kbit/s	5000 m

Communication in the CAN network should be reduced as much as possible. Communication without new information should also be avoided. Bit-rate should be as low as possible, because the lower the speed of the network, the more robust the communication [19].

2.2.3 CAN data frame

Communication between devices in CAN bus is done with data frames. Data frames contain data and some additional information. The data frame can carry up to 8 bytes of data. Additional information is used to inform start of frame (SOF), frame identifier, remote transmission request (RTR), frame type, data length, correction sum, acknowledge proper transfer and end of frame [11]. There are 44+3 additional bits in the data frame, if 11-bit identifier is used. If 29-bit identifier is used, there are 64+3 additional bits of data. The +3 bits are for interframe spacing [14].

Synchronization in the bus is always performed when bus is transmitting. Hard synchronization is performed at SOF for each data frame. Re-synchronization is also performed during transmission when bus state changes. If there are more than five same bits in transmission, bit stuffing is used automatically to change bus state for duration of one bit. This is made to guarantee, that clocks of the devices should not drift too much [14].

Priority for transmitting data frames in bus is decided based on CAN data frame identifier. The identifier for each data frame in the network should be unique. The lower the identifier is, the higher the priority for transmit. If multiple data frames are trying to be transmitted at same time, priority is decided with bit-wise comparison of the objects. This allows using the same bus for time frames with different real-time requirements [11]. Bus arbitration is non-pre-emptive. Therefore, transmission of the lower priority message that has been started cannot be interrupted by higher priority message [14]. Arbitration principle for deciding node priority based on [11] is presented in table 2.3.

Table 2.3. *CAN bus arbitration based on [11].*

	SOF	10	9	8	7	6	5	4	3	2	1	0	RTR	Control	Data
Node 1	0	1	1	0	0	1	1	Listen only							
Node 2	0	1	1	0	0	1	0	1	1	0	0	1	0		
Bus	0	1	1	0	0	1	0	1	1	0	0	0	0		

When comparing data frame identifiers bit-wise between two or more nodes, node transmitting in dominant state overwrites node transmitting in recessive state. From logic point of view this can been seen as logic 0 overwriting logic 1. All lower priority nodes will switch to listen only state, until bus is free for next transmit [11].

In theory, it is impossible to overload CAN network, but excessive amount of data transmitted via network will cause delay for lower priority messages. This might cause the application to not work properly. Average busload should be less than 50 %, even temporary busloads over 50 % are acceptable [19].

2.3 Model-based design

According to The Economist, there are 10-20 bugs in 1000 lines of code in commercial software [20]. Therefore, improving software quality is necessary. One way to improve software quality is to have a way to test the system and verify its performance. Model-based design is one option for this [21].

Modeling can be used to simulate system behavior under various conditions without real world system. This is especially advantageous in early stages of development, and design can be improved by iterating between modeling and simulation. Model can also be used to generate code for real control system. [22].

There are a few steps in Model-Based Design process. Example workflow for MBD in Simulink from Mathworks is presented in figure 2.4.

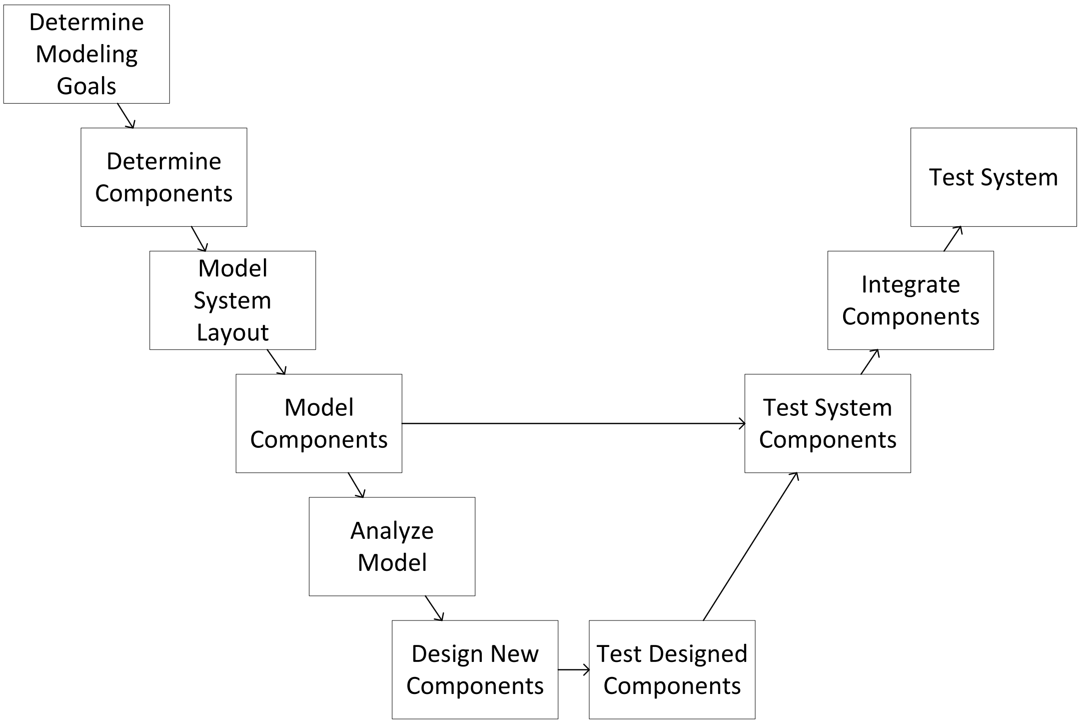


Figure 2.4. *Example MBD workflow in Simulink based on [22].*

The left branch of the V-model represents designing, modeling, and analyzing model. The right branch of the V-model represents testing and integration. The higher the step is in V-model, it is also higher level in the system. Higher level steps in modeling, testing and integration focus on system level operations, and lower-level steps will focus on smaller details and components. Higher level operations focus on what needs to be done and check if it is done as it is supposed to. Lower-level operations focus on how it is done and is it done as supposed to.

Model-based design enables using incremental level design, integration and testing for system. Testing during design can allow unconstrained prototyping and therefore demonstrating more options before implementation [21]. Integration levels in vehicle context based on [21] are represented in table 2.4.

Table 2.4. *Incremental model-based test integration levels based on [21].*

Integration level	Control method	Software environment	Hardware environment	System environment
Model-in-Loop	Simulated	N/A	N/A	Real or simulated
Software-in-Loop	Code	Simulated	Emulated	N/A
Processor-in-Loop	Code	SW Code	Evaluation Environment	N/A
Hardware-in-Loop	Code	SW Code	ECU	Simulated
Iron-in-Loop	Code	SW Code	ECU	Real Plant (e.g. Engine)
Environment-in-Loop	Code	SW Code	ECU	Vehicle in Field
Customer-in-Loop	Code	SW Code	ECU	Fielded Telematics

Test cases are defined in the beginning of the development process. Same test cases can be used through all integration levels and more defects can be found in earlier stages of system development [21].

3. DESIGNING CONTROL SYSTEM ARCHITECTURE

There are multiple factors that must be considered when designing control system architecture for system. Most important of these factors are discussed in this section.

First, upper-level architecture for the system must be designed based on requirements and physical properties of the system. In addition, in this case modifying some properties of the control system must be possible without regenerating code. Therefore, those parts of the system must be implemented in CODESYS.

Second, components must be designed to be reusable to some extent. For example, adding more actuators to the system, using different units for measurements, or changing the type of valve must be possible with reasonable amount of work. In addition, the same controller, for example basic PI-controller, must be usable for multiple peripheral actuators.

Third, handling I/O must be designed to be simple enough. There must be a way to configure inputs and controls from and to system without Simulink and regenerating code.

Fourth, handling different parameters must be designed with care. There are parameters for multiple different purposes in the system, and they are meant to be altered by different level users. Even though there are different levels of users, user access management is not implemented in scope of this thesis because test system is for research purposes. If system would be used for production purposes, implementing user access management would be necessary.

Fifth, handling CAN messaging must be designed to be efficient and robust enough. Excessive messages and using too large or too complex data types must be avoided to reduce bus load to be as low as possible.

Sixth, real world constraints must be considered during the development process. Dynamic for all hydraulic valves is modeled in simulation model, but there are some simplifications for other actuators in the system model. Those simplifications must be taken into account while designing and testing control system architecture. For example, the dynamic of some system level components is not modeled in simulation model, but they cannot be neglected when designing, implementing, and testing any states and logical operations that require control of those components.

3.1 Designing upper-level architecture

In scope of this thesis, controlling and monitoring different operations in the system are divided into two levels, which are system level control and actuator level control. In this case system level means that number of operations on this level is not dependent about the number of actuators in the system. Actuator level means control related to actuator, which control is subject of the further study. If the system is scaled up or down, the number of actuator level control systems varies. For example, controlling and monitoring hydraulic power unit is system level operation, and controlling and monitoring pressure in actuator is actuator level operation. High level architecture for control system is presented in figure 3.1.

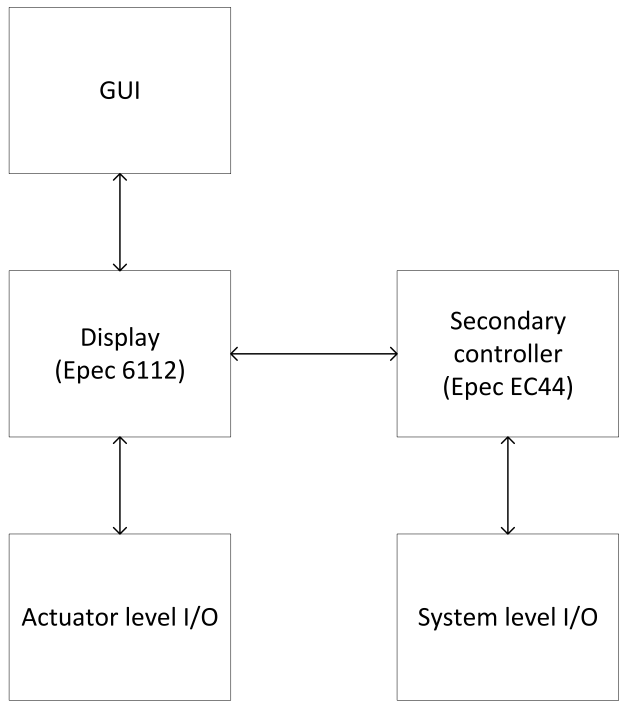


Figure 3.1. *High level architecture for control system.*

The control system is divided to multiple controllers. In this case, display is main controller for system operation, and it handles all actuator level monitoring and control. Graphical user interface also runs on display. The upper-level controller runs and is operated on display. Secondary controller is used to control system level operations, which include for example, monitoring and controlling hydraulic power unit operation. Communication between controllers is done via field bus. There are also I/O devices in field bus. I/O devices don't have any user code except configuration, and they are controlled by display or secondary controller. Some of the I/O devices communicate only with display, and some of the I/O devices communicate only with secondary controller. None of the I/O devices communicate with display and secondary controller.

3.2 Re-usability of components

Control system is constructed from multiple subsystems, that are designed to be as reusable as possible. All generated code must be reusable in this system domain without need for Simulink and regenerating code. Interfaces for subsystems must remain as similar as possible, even when, for example, a new controller is implemented with some new parameters. Altering the number of actuators must be possible without regenerating code from Simulink. Communication between subsystems and PLC's is handled in CODESYS.

All subsystems which code is generated from Simulink handle only general inputs and outputs without units. Therefore, for example, the same pressure controller can be used with any pressure unit. All controllers, that code is generated from model, outputs are also normalized for range [0,1] or [-1,1], depending on case. Any input and output handling is implemented in CODESYS in parameterized manner. For example, pressure transmitters signal is converted from 4-20 mA to pressure in given range in CODESYS function, not in any Simulink generated code.

All function blocks that are used to construct control system, are made in CODESYS. Function blocks can contain generated code, but with this approach interfaces for the system can be kept as constant, as all inputs and outputs for the rest of the system remain unedited. These function blocks also contain formatting control signals to format, that is suitable to be used as a control value for actuator. The exact format for each signal type depends on the controller or I/O-unit that is used, so formatting must be easy to edit, if there is need to use different hardware. Correct format and control signal range depends also on actuator that needs to be controlled.

3.3 Handling I/O

Display is used to store and transmit process parameters, whether parameter is used by software on display or on secondary controller. If there are parameters that are used by secondary controller, they are sent as CAN-message from display to secondary controller. There are also some digital controls from display to secondary controller, which are sent as CAN-message. For example, hydraulic power unit is not allowed to start, unless start is requested by display.

The secondary controller is also used to send system level information to display. This information can contain various types of data, including and not limiting to digital inputs, system operation statuses and analog measurements, such as hydraulic pressure. Digital inputs and system operation statuses include but are not limited to hydraulic power unit operation status, information about if hydraulic oil temperature is within allowed range and if hydraulic oil filter is not clogged.

Display controls some of the I/O devices, that are connected to CAN bus. These devices include for example analog output device, that is used to control some actuators with voltage signal.

There are also separate I/O devices for actuator level control, that can handle digital and analog inputs and digital and some analog outputs. In this case these I/O devices can output analog signals only as PWM or PWFM signal.

Secondary controller is used to control all system level I/O. Some of this I/O can be handled by secondary controller, and some of this I/O requires separate I/O devices.

All measurements from system, whether they are from system level or actuator level operations, are read and formatted to required format before upper-level controller.

There are some measurements that are not required by upper-level controller, but they are used for determining if some operation is done or not. In this kind of situation status of operation in question is also decided based on criteria before upper-level controller.

Almost all inputs for upper-level controller are constructed to structs to reduce the total number of inputs. Structs are constructed from similar type inputs, such as analog inputs, digital inputs, and user inputs from graphical user interface. All input structs are constructed with separate handlers, whether their inputs are read from controller input pins, CAN message, parameter file or graphical user interface inputs.

This same applies to outputs from upper-level controller. Outputs are constructed to similar type and domain outputs, such as actuator level digital outputs, system level digital outputs and references. All outputs are deconstructed with handler for each corresponding struct, and outputs are used to control actuator level outputs or sent with CAN message to secondary controller for system level outputs.

Most of the references received as output from upper-level controller are not suitable to be used as they are, so they need further handling. For example, changing pressure reference from low pressure to high pressure can cause instability and windup in system. Therefore, large changes in reference must be made with ramp. Due to some uncertainties in the system, dynamic reference is implemented to prevent too fast movement of actuator, that might cause pressure to drop too low in hydraulic system.

Digital controls from upper-level controller are handled with separate handlers. System level controls to CAN bus are constructed from all system level controls from all upper-level controllers. System level controls are read from structs constructed in upper-level controller, and they are deconstructed in handler. Actuator level controls are also deconstructed in the handler. Unlike handlers for inputs, handlers for digital outputs are implemented as function blocks, and not as functions. This is made to allow the use, for example, timers for handlers.

3.4 Handling parameters

System contains large number of parameters. Some of the parameter's values can be edited only from CODESYS, and some of the parameters can be edited also via GUI.

Parameters are separated into three main categories based on their purpose, which are process parameters, tuning parameters, and safety parameters. All parameters are stored as Global variable list, so they can be accessed easier from one place. This will also reduce the number of separate parameters as low as possible in programs, function blocks and functions.

Process parameters are used for controlling the process cycle. These parameters can be adjusted via graphical user interface or CODESYS. The allowed set value ranges for these parameters via user interface are limited for safety reasons. Process parameters are stored as global variables, that are declared to be remnant with keyword RETAIN. This means, that variables value remains as set, even when system is rebooted. These parameters can be overwritten via CODESYS by downloading program new for PLC.

Tuning parameters are stored as global variable list. Tuning parameters include all parameters that are related to tuning controllers or formatting measurements or control signals. These parameters can be adjusted only via CODESYS. Because the test system is used for research, it may be necessary to edit some tuning parameters online. Therefore, tuning parameters are stored in this case as variables and not as constants. Tuning parameter values will be reset to declared values, when the system is rebooted, or new program is downloaded to PLC.

Safety parameters are stored as global variable list. Safety parameters define allowed set value ranges for process parameters, that can be altered via graphical user interface. These parameters include for example allowed position reference range for hydraulic cylinder. There is not any reason to be able to adjust safety parameters while the program is running, so safety parameters are declared as constant, and they cannot be adjusted from program or online change via CODESYS. Safety parameters will remain as set when the system is rebooted. The only way to adjust safety parameters is to download program to PLC with new values, that are set in CODESYS.

3.5 Handling CAN messages between devices

As discussed earlier, the display is CAN master in this application. Therefore, display is responsible for configuring any devices in network, that cannot be configured separately. All messaging between different devices is done via CANopen buses. There are total two separate CAN buses in system. First CAN bus is used for actuator level communication, and other CAN bus is used for system level communication.

Data types for messages between controller and specific I/O devices are defined by electronic data sheet (EDS) files for each I/O device. EDS files are supplied by the manufacturer of the device. Data types for messages between controller and each actuators I/O devices are defined based on I/O device configuration in Multitool. Some I/O devices output ports can be configured to output digital output or analog output as voltage or current message.

Configuration for different CAN messages timing is done in Multitool. The timing for each message depends on the purpose of the message. For example, set value that is kept constant from display to secondary controller doesn't need to be sent frequently, but control value for valve or pressure measurement must be sent on every program cycle.

Most of the time critical CAN messages in this system are declared as event-based messages. Message is sent every time when value is changed. Therefore, message is not sent, if its content stays exactly same. To prevent excessive number of messages, inhibit time is used to set minimum time between two messages with same id. All these measures are made to keep CAN bus load as low as possible and to prevent risk of delaying lower priority messages in CAN bus with too frequent messages, that are not that critical for system operation.

CAN messages between controllers containing measurements or references are sent in as small and simple format as possible. For example, if reference is within range [0,255] and it contains only integers, there is no need to use REAL (single) that uses 32 bits for one numeric value, and USINT (uint8) can be used instead. If boolean controls or statuses are needed, byte with configured bits is used. Integers that allow using greater numeric values or negative values are used only when it is necessary for system operation. This applies also to floating point numbers.

3.6 Handling real world constraints

There are multiple real word constraints in the system, which must be considered in simulation model and control system. There is also some behavior of the real-life components, which modeling in simulation model would be out of the scope of this thesis. Therefore, they are considered in control system as delays and their expected behavior is parameterized. Acquiring proper parameter values for those actuators delays will require some tuning with real world system.

Some examples of these kinds of components are some system level on/off actuators, that are not specified in this thesis. They are controlled with digital on/off control, but their dynamic is unknown. Therefore, parameterized delay is implemented for activating those components, to make sure that previous process steps components have had enough time to deactivate.

Minimum time for each process step is implemented to make sure, that no step is skipped by accident. This is necessary especially in those kinds of steps, which criteria if step is done depends about operations, that are started during step. Because there is delay for some system level components, this is necessary to prevent scenarios, where step could be considered to be done, before operations for step are even started.

Most of the measurements from process are analog, so there is some noise in all measurements. Because of this, for example deciding if set pressure for actuator is reached, cannot be considered as easy as with digital measurements. Therefore, tolerance for each analog measurement must be implemented. For example, if the set value for pressure is 10 MPa, it is unlikely to reach exact pressure of 10 MPa due measurement noise. If 0,5 MPa tolerance is implemented, pressure can be considered to be reached, if absolute error is less than set tolerance. This allows pressure or any other analog measurements to be used as condition for state transition. Finding exact tolerances for different measurements will require some tuning with the actual test system. In addition, real world process to be controlled can set some limits for tolerances.

There are controllers for pressure- and position control for each actuator. The controller to be used is decided by upper-level controller. Therefore, for example position controller is disabled, if current step requires pressure control. Controllers are also disabled temporally, if current error doesn't require any controller or error is small enough. This is done to prevent unnecessary control of the actuator due to measurement noise.

For example, if error is less than half of set tolerance for corresponding measurement, controller is temporally disabled, even if upper-level controller allows operation of that specific controller. Controller can be set to enabled from disabled if error is greater than or equal to tolerance or error between generated dynamic reference and step reference is greater than tolerance. In addition, using controller must be allowed by upper-level controller. Therefore, permission to use the controller depends also on the current state of the controller. Permission logic for allowing pressure controller operation is presented in figure 3.2.

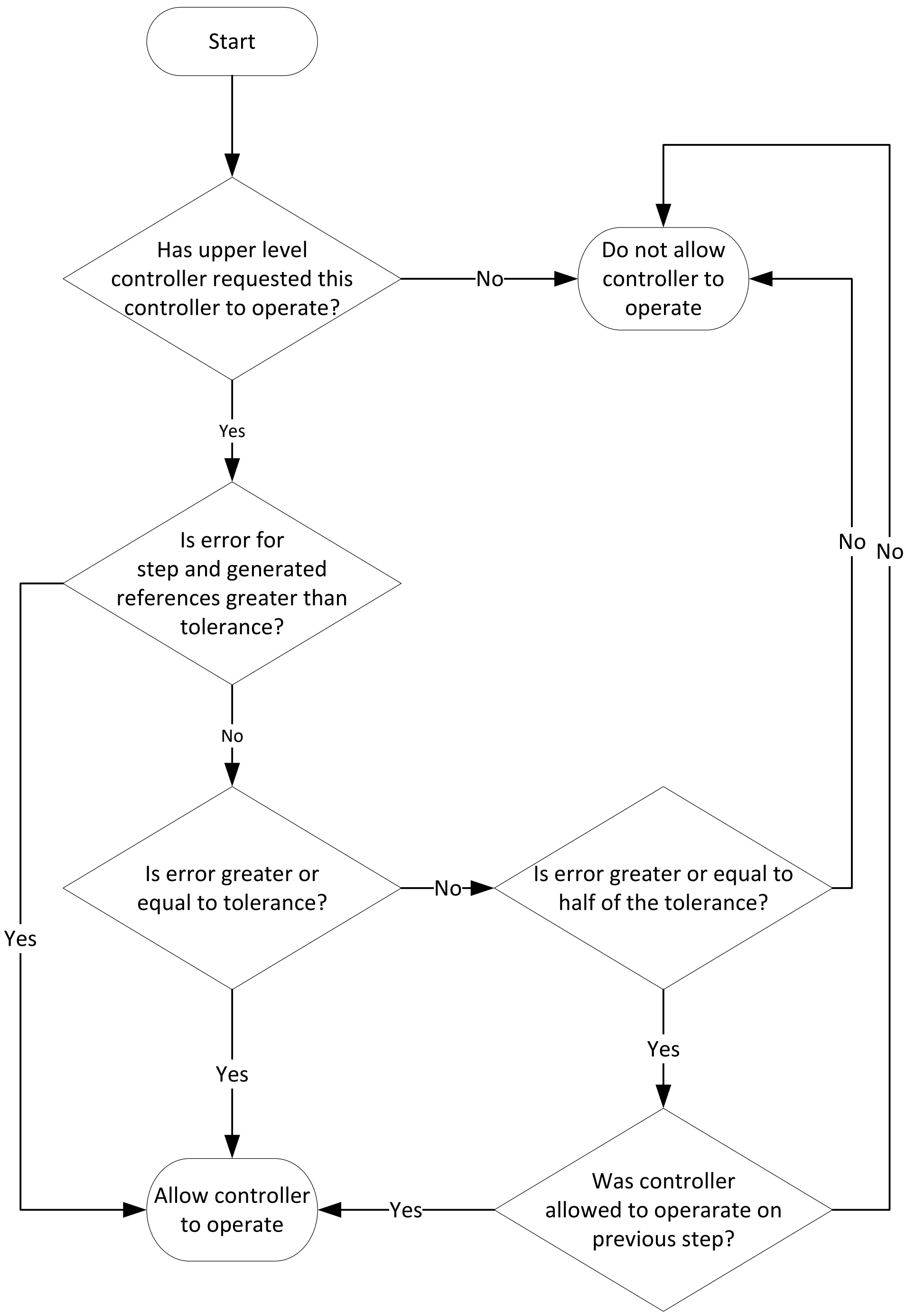


Figure 3.2. *Controller operating permission.*

Due to some constraints in the test system, there must be way to queue some operations to prevent simultaneous actions. This queuing must also be done as robustly and computationally efficiently as reasonably possible. Therefore, updating permission is done only if one or more actuators are requesting permission. Queuing logic for permission is presented in figure 3.3.

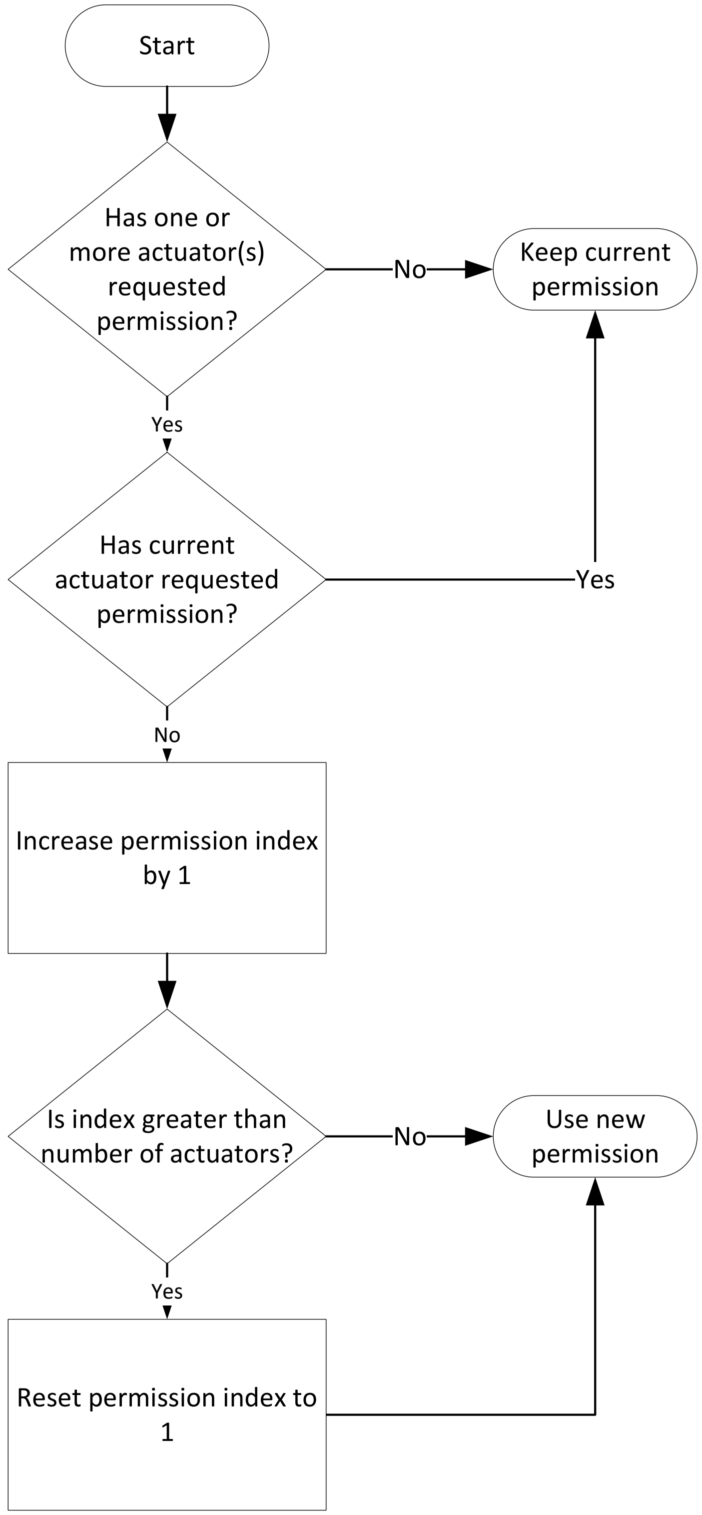


Figure 3.3. *Queuing logic.*

Queuing for permissions is run at most once in program scan. If there are no actuators requesting permission, permission is kept as it is. If one or more actuators have requested permission, permission is given to the next index. If the index is greater than the number of actuators, permission index is reset to 1. This is relatively simple solution for this kind of problem, and it is feasible if total number of actuators is relatively low. If the total number of actuators is high, this kind of permission queuing and indexing can cause excess delay and better queuing must be implemented.

3.7 Designing and implementing control system architecture

Control system design process was done with few straightforward steps. Some of the steps were iterative. Final CAN network architecture for system is presented in appendix A. Final program architecture for display is presented in appendix B and final program architecture for secondary controller is presented in appendix C. In this thesis, system integration was done in two levels, which were Model-in-Loop and Hardware-in-Loop.

First step in design process was to get familiar with process to be controlled. There are some features that require special attention while designing the control system. Flow chart and state models were created from the process. In this step the focus was to develop initial plan about different states in system and find some conditions for transitions from state to another. In this stage of design process was decided that system will need three separate control modes. Automatic mode is meant to be default operation mode for process. In automatic mode the process will perform single cycle on all active actuators, or system will run on continuous cycle. Second mode is for controlling process steps manually. In this operation mode each process step can be performed individually, without performing all prior steps in cycle. This control mode is meant especially for first tests on hardware and for some specific use cases in research. If the process is controlled in manual mode, user can control only one actuator at a time. Third operation mode is manual recovery. This operation mode is meant just to allow recovery from error. For example, pressure can be released with manual mode, if there has happened something unexpected in the process.

Second step in process was to analyze and perform some simulations with actuator model and controller for independent metering valve (IMV) system. There was existing proof of concept model from this part of the system, that required some fine tuning. In simulations it was found out that the system requires very precise control in some operation points, especially when pressure in actuator is high.

Third step was to develop initial upper-level controller for system. Upper-level controller is implemented with Stateflow chart. Developing upper-level controller was done in iterative steps implementing and testing required functions in small steps. Controller is divided into three operation modes and safe state, where any operation of any part of the system is forbidden. Automatic mode and manual process control mode use the same process steps to implement controls for outputs, so control steps are implemented in parallel Stateflow chart. Therefore, managing operation logic and controls is easier. Parallel charts states handle all I/O and logic related to steps and deciding if step is done or in progress. Operation modes are used to implement just basic cycle for process. Upper-level controller is presented in figure 3.4.

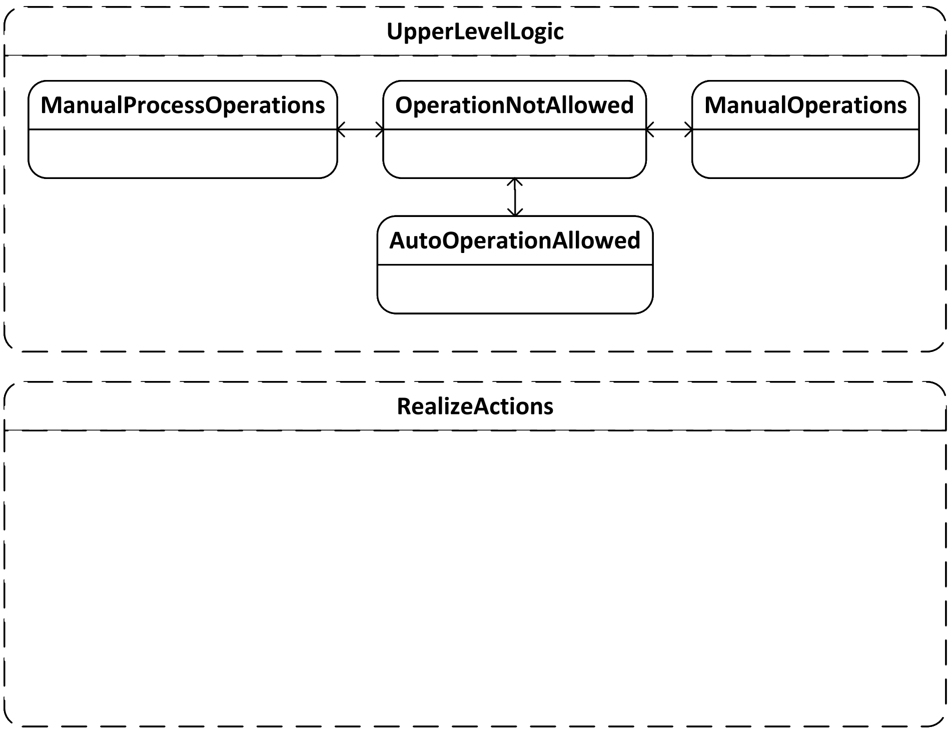


Figure 3.4. *Upper-level controller.*

Fourth step was designing and implementing handling for measurements, user inputs and parameters. Implementation for parameter handling in this step was to allow testing in Simulink, but Simulink implementation from this part will not be used for code generation. Initial design for handling outputs was also made in this step, but it will be implemented as well in CODESYS. Because there is large number of various type of measurements from multiple different equipment vendors, formatting measurements to same standard format is necessary. In addition, there are large number of inputs especially for upper-level controller. Therefore, constructing structs for system inputs is necessary.

Fifth step was designing and testing initial graphical user interface with Simulink dashboard and real time simulation. This was made to specify exact requirements for various user inputs, that are especially related to manual process control. Design process and final design for graphical user interface will be discussed later.

Sixth step was to configure subsystems for code generation and generate code for subsystems. This step included defining all data types for inputs and sample times for subsystems. Configuring input data types is necessary to ensure proper operation of subsystem, optimize performance and prevent usage of unintended inputs. For the same reasons sample time must be configured for subsystems containing any operations that require time, as sample time for generated code will be decided based on subsystems sample time. If sample time is not configured properly, subsystem doesn't operate as meant.

Seventh step was to import generated code to CODESYS and create function blocks. In this step handlers for buses were also implemented. Function blocks are constructed to contain one functionality, or in some cases one functionality group. For example, function block for controllers contains function blocks for generating references and controllers for each actuator for servo valve and IMV systems. Operation principle for handlers follows the following principle. First each handler reads corresponding inputs from CAN messages, inputs on controller, mapped variables for user inputs or from parameter file. In second step, inputs from CAN message or inputs on controller are formatted to specified format, which can be for example pressure from current message in mA to pressure in bar. In third step, all various inputs will be appended to struct for corresponding inputs.

Eight step was testing control system with hardware in loop simulation. The actual control system was running on display, but there was real-time Simulink model for all peripherals and actuator. The display was connected to the same CAN network as the computer that was running the model. The computer was connected to CAN network with CAN adapter. All relevant controls from display were read from CAN messages and used as inputs for system model. Outputs from the system model were sent as CAN messages to CAN network. All CAN messages were extracted from CAN database-file. All inputs and outputs from and to model were formatted to match signals from measurements and to actuators in real system. For example, pressure in actuator was converted from pressure to current. In addition, current measurement was scaled for same range as real I/O device would scale it. In this case, 4-20 mA current measurements from I/O device are read as microamperes. Another example is formatting proportional valve control from CAN message as current in mA to control in range [0,1].

3.8 Designing and implementing user interface

Operating system and configuring some of the parameters is done via graphical user interface. User interface is separated to multiple tabs that contain similar functionality or information. Figures representing graphical user interface in this thesis do not contain all measurements and information from actual graphical user interface.

Main tab is used to show basic information about all actuators to be controlled. User can select with checkbox, which actuators will be used for automatic operation. System safety status is shown with large blue lamp. If all safety related inputs are not as they should be, safety cannot be confirmed. Safety related inputs include, but are not limited to, hydraulic power unit operation status, hydraulic oil pressure, hydraulic oil temperature and hydraulic oil filter clogging status. System safety status can be confirmed when all safety-related inputs are within allowed range. System is not allowed to perform any operations with automatic or manual control if safety status is not confirmed. Main tab without some measurements and information is presented in figure 3.5.

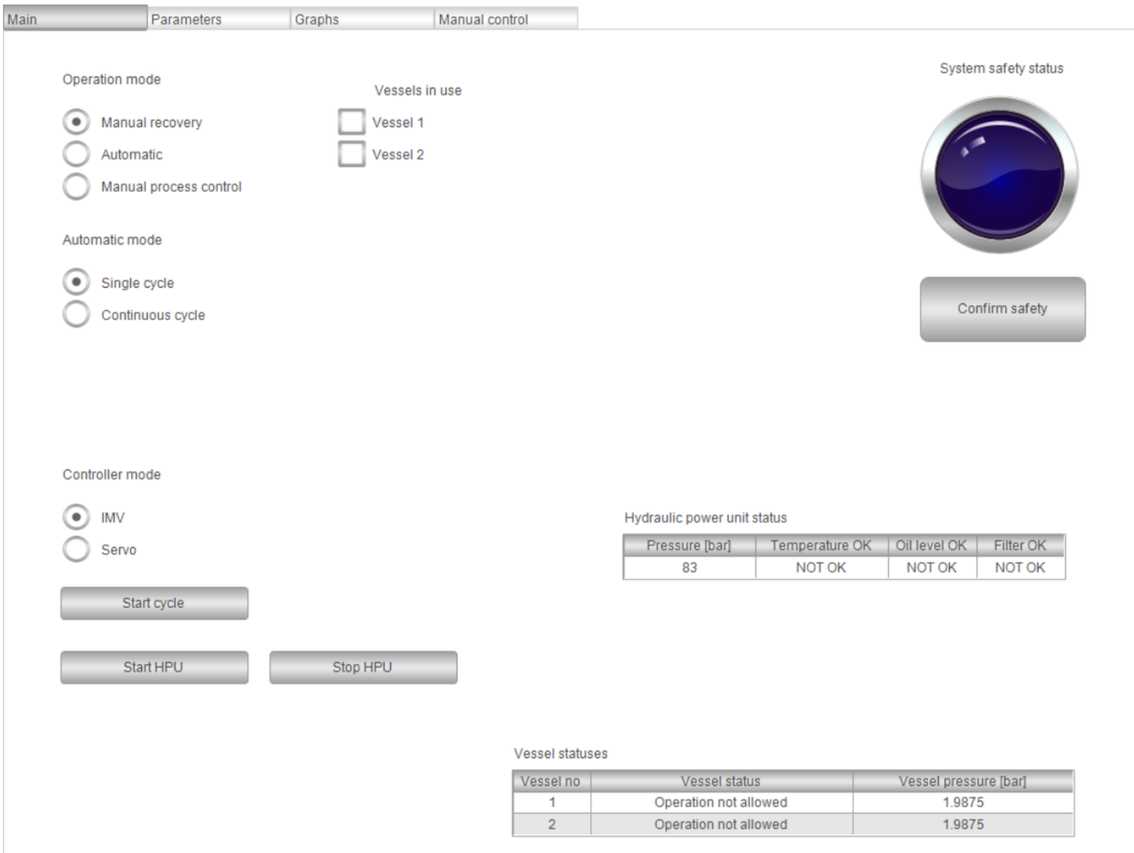


Figure 3.5. *Graphical user interface, main view.*

There is also some basic information about hydraulic power unit, like hydraulic pressure, hydraulic power unit operation status and information if pressure difference over filter and hydraulic oil temperature are within allowed range. Main tab contains also controls for starting and stopping hydraulic power unit, selecting operation mode, selecting if operation shall be single cycle or continuous cycle and selecting, if system shall use servo valve or IMV. Controls within the scope of display control system are connected to local or global variables in project. Controls that are required on some other controllers on system, are connected directly to corresponding CAN message.

Parameters tab contains all parameters regarding process, that user is allowed to edit. All these parameters are limited to specified range to ensure safe and predictable operation in all circumstances. For example, position reference for hydraulic cylinder must be always within cylinder stroke. Local parameters in scope of display are connected to global variable list in display project and parameters to other controllers are connected to CAN messages. Allowed ranges for all user configurable parameters are defined in the separate global variable list.

Graphs tab contain plotted historical data about some of the measurements in system. These measurements contain for example information about position of hydraulic cylinder and pressure in actuator to be controlled.

Manual control tab contains options to control operations, that are related to recovery from error or controlling process one step at time. This tab also contains option to select actuator to be controlled for recovery or manual process control.

4. TEST SYSTEM

Some properties of the test system are presented briefly in this chapter. Exact properties of the actuator and rest of the system are not discussed due to confidential information. The test system in this project is hydraulic pressurization system. Main purpose of the test system is to measure control performance and energy efficiency on controlling pressure of hydraulic actuator with two different valve configurations, which are servo valve and IMV.

4.1 Control hardware

The control hardware used in this project is from Epec. Epec 6112 is selected to be the display and main controller for the system. GUI and actuator level control runs on 6112. Epec 6112 has two CAN buses, ethernet and USB. There is also five IO pins, reference voltage and some other connections, that are irrelevant in this project.

Epec EC44 is used as secondary controller for some system level operations. These operations include, for example, controlling of hydraulic power unit. Epec EC44 has two CAN buses, 16 digital or analog inputs, 16 PWM capable outputs and reference voltage. Some of the inputs can read pulse inputs. Some of the outputs are capable of current measurement.

Epec GL84 responder units are used for some actuator level I/O. Responder units cannot run any custom program code. They are only used to control some outputs based on CAN messages that are sent to them. Epec GL84 has a total of 35 inputs and 34 outputs. Some of the inputs can read analog inputs and some of the inputs can read pulse input. Some of the outputs are capable of PWM control and/or current measurement.

4.2 Selected bus solution

Selected field bus for test system was CANopen. CANopen was selected, because researchers working with the project are familiar with solution, and there are required CAN based I/O devices readily available. CANopen offers good real time capabilities and priority handling for messages. When configured properly, most important messages always receive the highest priority. CANopen is also electronically relatively simple due to dual wires for signal and no need for central hub. Simplicity of solution is also good for reducing costs for system.

Using field bus makes it possible to distribute control system and I/O for different actuators and subsystems. This also reduces the amount of wiring required for system operation. Reduced amount of wiring will make system installation and maintenance easier.

4.3 Actuator

The concept of the actuator is presented in this thesis, but no further details will be discussed. The hydraulic diagram of the actuator is presented in figure 4.1.

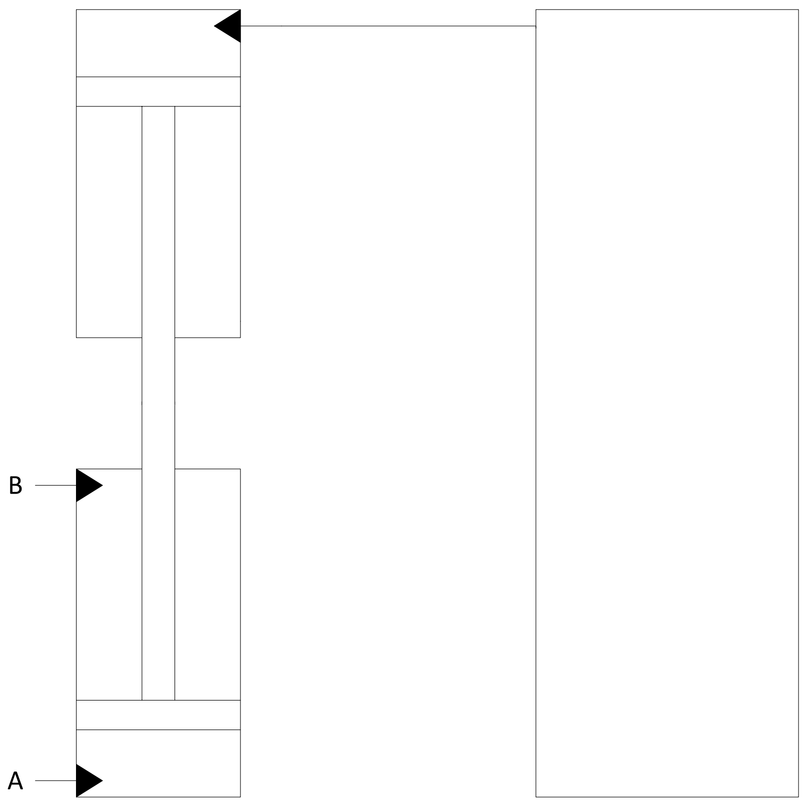


Figure 4.1. *Hydraulic diagram of the actuator.*

The actuator to be controlled contains hydraulic cylinder, that is connected mechanically to another cylinder, which is connected to volume. Total volume of the system and therefore pressure of the volume is controlled by controlling the hydraulic cylinder. Position of the cylinder is controlled based on pressure in volume. Reducing total volume leads to increasing pressure and increasing total volume of the actuator leads to reducing pressure. Total volume of the system can be decreased by directing hydraulic oil to chamber A and directing hydraulic oil out from chamber B. Total volume of the system can be increased by directing fluid out from chamber A and directing fluid to chamber B. Exact volume, pressure range and other characteristics of the system will not be discussed in this thesis due confidential information.

4.4 IO in test system

The test system contains various number of different type measurements and digital inputs. System level measurements are connected directly to the secondary controller. Actuator level measurements are connected directly to actuators I/O device, which is connected to CAN bus. There are also some measurements, that are connected directly to CAN bus via AD-converter.

There is also various number of different types of controls in system. System level outputs are connected to secondary controller, and actuator level controls are connected to actuators I/O device. There are also some outputs that are connected to CAN bus via DA-converter, such as servo valves. DA-converter is required because servo valves in this case require control value in range [-10,10 V], and secondary controller or actuators I/O-devices don't have required type of output signal.

Different measurements and controls require different amounts of formatting. The correct format for each signal depends on the signal, controller, or I/O device to be used. Signal for I/O devices depends on configurations of the devices and/or from factory settings. For example, in this application DA-controller takes control value for voltage as millivolts.

4.5 Hydraulic valves in test system

The test system contains two different types of hydraulic valves, which are servo valve and independent metering valves. Main purpose of the further research that will be made with test system is to study system performance on each valve or valve system type. Servo valves used in the system are Rexroth 4WRPE10E50SJ-3X/M/24A1. They are closed center 4/3 valves with nominal flow rate of 50 l/min at pressure difference of 0,5 MPa. The proportional flow control valves used in the system are Hydac PWS10Z-11-C-N-P40-0. They are normally closed 2/2 valves with integrated check valve with nominal flow rate of 40 l/min at pressure difference of 0,5 MPa.

4.5.1 Servo valve

Servo valve can be controlled with infinite positions. The direction of flow and opening of each port depends on the position of the spool. Relative position for each opening is constant with same spool position, and it cannot be adjusted. Servo valve contains internal controller and position feedback. Symbol for closed center servo valve used in the test system with 4/3 configuration is presented in figure 4.2.

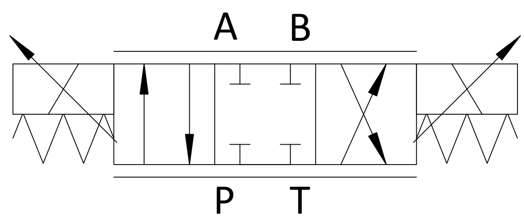


Figure 4.2. *Servo valve symbol.*

Servo valve is known to be fast and accurate. Disadvantages for servo valve are relatively high price and higher energy consumption. Servo valves are also more prone to impurities due to their relatively complex design.

4.5.2 Independent metering valves

The independent metering valve system contains separate valves for each port. Each port (PA, PB, AT, BT) can be controlled independently with infinite positions, so it is possible to use some control strategies that are impossible with traditional control valves without adding more valves to the system. IMV system for double acting hydraulic cylinder with A- and B-chambers is presented in figure 4.3.

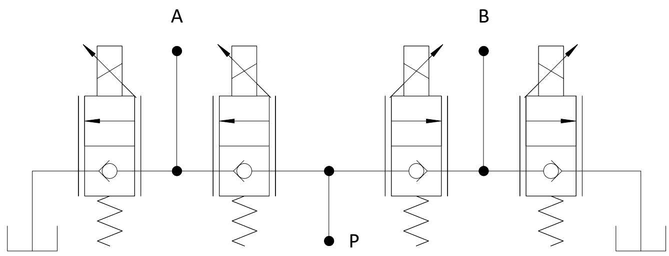


Figure 4.3. *Independent metering valve system.*

Most important of said control strategies is differential control, where hydraulic cylinders chambers A and B are connected, and therefore effective piston area equals to cylinder rod area. Differential control can be used to reduce the amount of hydraulic oil flow required for positive movement when restrictive load is low. IMV differential control to positive direction is presented in figure 4.4.

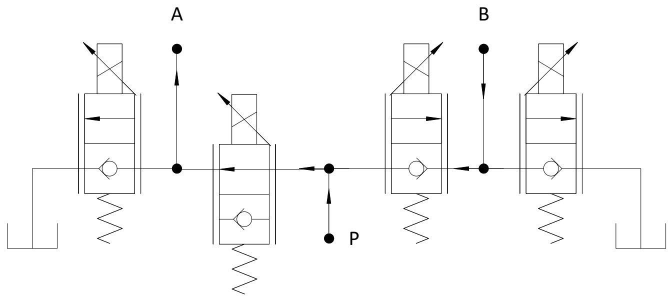


Figure 4.4. *IMV differential control, positive direction.*

Differential control can also be used to restore some hydraulic energy in negative direction, when overrunning load is high enough. IMV differential control to negative direction is presented in figure 4.5.

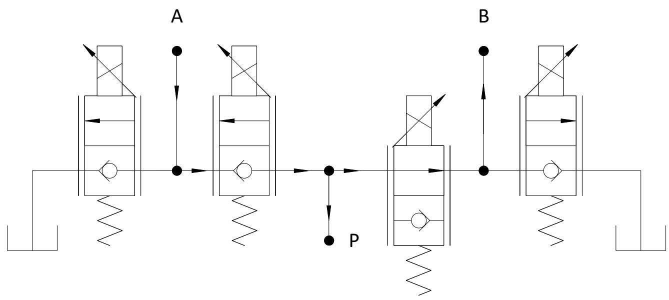


Figure 4.5. *IMV differential control, negative direction.*

Valves used in the system are also relatively cheap compared to servo valves. Disadvantages for IMV system in this case are relatively slow and inaccurate proportional valves. IMV also requires more advanced control system than traditional proportional or servo valve to achieve all advantages. Valves used in the test system for IMV contain integrated check valves. This allows using less complicated controller for IMV, because when using differential control, only one valve must be controlled instead of two. Excess pressure in other chamber will be released to pressure line in hydraulic system, and it can be used for some other actuator, or it can be restored in hydraulic accumulator. Internal check valve with configuration presented in figures 4.3, 4.4 and 4.5 also prevents hydraulic cylinder acting as pressure multiplier and causing excessive pressure in system in case of some fault in controller software or in electrical control.

4.6 Hardware-in-Loop simulation setup

Hardware-in-Loop simulation is utilized to test system operation and verify that system performance meets requirements that are set for it. As control of the system operation is done mainly with display which is connected to rest of the system only via CAN buses, Hardware-in-Loop simulation can be done with relatively simple setup and there is no need for extra equipment to handle inputs and outputs. Hydraulic power unit and its controller are simulated with Simulink model. HiL test system setup is presented in figure 4.6.

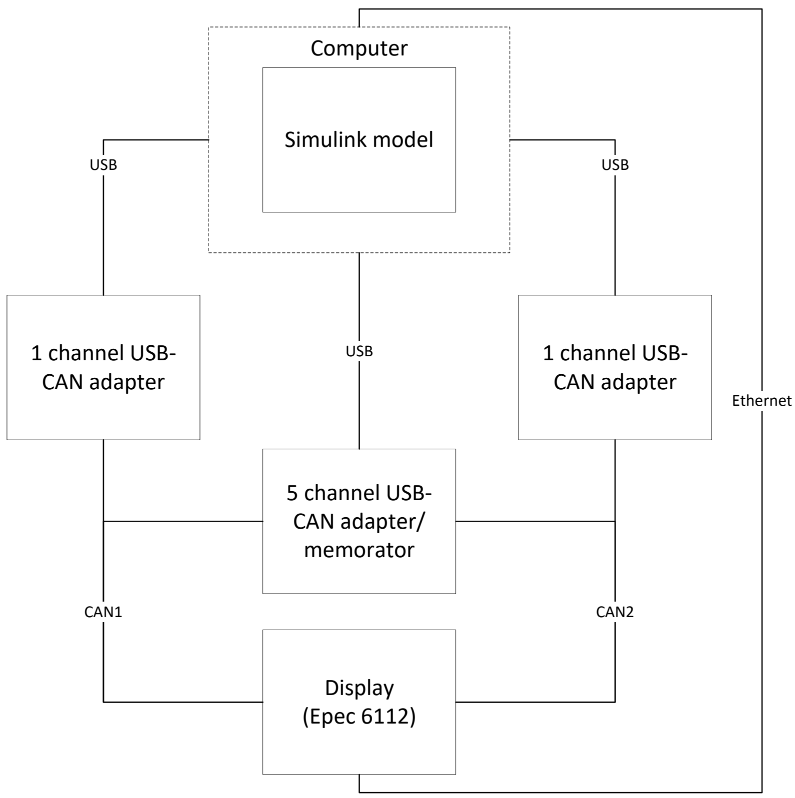


Figure 4.6. *HiL test system.*

There are a few major components in HiL-simulation setup, which are display, computer for running simulation model and USB-CAN adapters. In addition, CAN memorator/adaptor connected to computer is used to log messages in CAN buses over measurement cycle.

The computer is connected to display with ethernet for analyzing. This way monitoring of system operation is possible via CODESYS, when logged in to controller. The computer is also connected to CAN buses via USB-CAN adapters to allow communication between the display and Simulink model. Separate CAN memorator/adapter is required for logging CAN messages, because one USB-CAN adapted can only be used by one computer program at a time. Therefore, Simulink and Busmaster cannot use same USB-CAN adapter.

CAN messages are handled in Simulink model with Vehicle Network Toolbox blocks. Each USB-CAN adapters bus speed is configured with CAN Configuration block. Failing to set the correct bus speed can cause improper operation of the CAN bus communication.

Messages are received from CAN bus with CAN receive block. Adapter to be used is set at CAN receive blocks configuration. Sample time for receiving messages, filtering messages and how many messages can be received at one step are also set here. In this system, sample time is set equal to PLC program cycle time and lowest interval for CAN message. There is no limit how many messages can be received in each time step and in this application no messages are filtered by id. CAN receive block outputs received message and function call event, that can be used for subsystem containing CAN unpack message block.

Received messages are unpacked with CAN unpack message block. This block is used to select specific message and unpack data that it is carrying to usable format. CAN unpack message block must be in subsystem, that contains input for function call event. CAN message can be sent to model as raw message, or different signals in message can be separated. This separation can be configured manually or by utilizing CAN database. In this test system CAN databases are utilized.

Signals read from CAN messages with CAN unpack block are used as inputs for simulator. As these signals are meant to control actual actuators in system, for example proportional valves in IMV system, signals must be edited to format that model is expecting for them to be. For example, control for proportional valves in IMV is current, but it must be scaled to range [0,1] for model.

There are no major edits required to model for HiL-simulation, except that controller is implemented on actual control hardware instead of in model. The rest of the model is kept without changes.

Outputs from the model that are sent to controller in CAN messages, are edited to same format as real measurements from system. For example, pressure is converted from pascals to current message same way as pressure transmitter. In addition, in this setup outputs are discretized to equal time step as PLC program. Outputs from system that are meant for internal use in model are not edited.

Messages that will be sent to CAN bus are packed with CAN pack blocks. CAN databases can be utilized with this block.

Messages packed with CAN pack are transmitted to CAN bus with CAN transmit block. With this block can be set, if messages are sent when its content is changed and/or if message is sent with set interval.

Real time desktop could not be utilized for simulation due to constraints in model, as it would require using discrete solver. Pressurization system will require small step size to prevent improper operation. The computer that was used to run the model did not have enough computing performance to use time step that would be small enough. Therefore, variable step solver and simulation pacing was used to achieve acceptable compromise between required computing power and good enough simulation performance. There might be sometimes when simulation is slightly slower than it should be, but with this setup there were no major lag spikes.

CAN messages are logged with Busmaster. CAN databases were not utilized while logging. Messages were logged as raw data. Information from logged messages was extracted with CAN databases when data was analyzed with MATLAB.

4.7 Simulating system performance

System performance was simulated from multiple different perspectives during development, but the focus in this thesis is performance of the upper-level control system. Performance of the upper-level control system was analyzed from two perspectives, which were CAN network load and program cycle time. In both cases the main measurement was average performance over relatively long time period, when automatic operation was used. Performance while controlling single actuator in manual process control mode or recovery mode is not that critical, because in both manual control modes only one actuator is used at a time.

Simulations were performed as Hardware-in-Loop simulation with actual control hardware and Simulink model. Performance and proper system operation were analyzed with one active actuator and with two active actuators. Therefore, it is possible to extrapolate system performance in some extent, for example if number of actuators is varied.

Average cycle time for tasks containing program and graphical user interface were monitored on CODESYS, while program was running. Interval time for task containing program was 10 ms. Interval time for task containing graphical user interface was 100 ms. Cycle time should always be significantly lower than interval time to ensure proper operation of the control system, especially on the program containing time critical functionality.

The average bus load on CAN network was analyzed with Busmaster, while the program was running on display. Initial bus speed for system was 250 kbit/s. Average bus load should not exceed 50 %, even temporary bus load higher than 50 % can be acceptable. There was some white noise added to measurements from Simulink model to make measurements more realistic.

5. RESULTS

Control system architectures performance was measured from two aspects in two different scenarios. First aspect was PLC task cycle times for task containing program and task containing GUI. Second aspect was CAN bus average load and peak load. First scenario was only single actuator and it's I/O. Second scenario was two actuators and their I/O. All measurements were made over relatively long time when system was performing automatic process cycle.

Main purpose of these measurements was to validate, that cycle times and bus load are within allowed ranges. Second purpose of these measurements was to allow extrapolate control systems capacity in both terms if number of actuators in system is increased. Measurements were made with IMV system.

Cycle times were measured with CODESYS. Cycle times were read from Task Configuration in CODESYS, while logged in to PLC. Maximum and minimum cycle time were dismissed, because first cycle after boot will take more time than normal cycles. It is noticeable that with this configuration task containing GUI has lowest priority. As Epec controllers utilize pre-emptive scheduling, task containing control system will be always executed with set interval, even if task for GUI is not finished. Therefore, execution time for GUI can be higher than cycle interval for control system, without risking proper operation of the system. Task containing GUI must still be executed within cycle interval to ensure proper operation, even if it is interrupted for task containing control system. The bus load was measured and logged with Busmaster from both buses. The bus load was analyzed with MATLAB from log file.

5.1 System performance, Hardware-in-Loop

System performance was measured over some time with Hardware-in-Loop simulation. Measurements were made with two different CAN1 bus speeds, which were 250 kbit/s and 500 kbit/s. CAN2 bus speed was kept at 250 kbit/s in both measurements.

5.1.1 250 kbit/s CAN bus

Measurements with Hardware-in-Loop simulation and maximum values for each parameter are presented in table 5.1.

Table 5.1. *Control system performance, Hardware-in-Loop. 250 kbit/s.*

	One actuator	Two actuators	Max
Average cycle time, program [μs]	491	533	10 000
Average cycle time, GUI [μs]	3718	3968	100 000
Average bus load, CAN1 [%]	10.8	21.5	50
Average bus load, CAN2 [%]	11.1	11.1	50
Maximum bus load, CAN1 [%]	16.1	30.2	
Maximum bus load, CAN2 [%]	12.8	12.9	

From table 5.1 can be seen, that all performance measurements are well within allowed range. Average cycle time with one or two actuators for program is considerably lower than cycle interval, which is 10 ms. Same applies to average cycle time for GUI. Cycle interval for GUI is 100 ms.

It can be beneficial to see how constant bus load is over measurement cycle. Normalized bus load over measurement time for bus 1 is presented in figure 5.1.

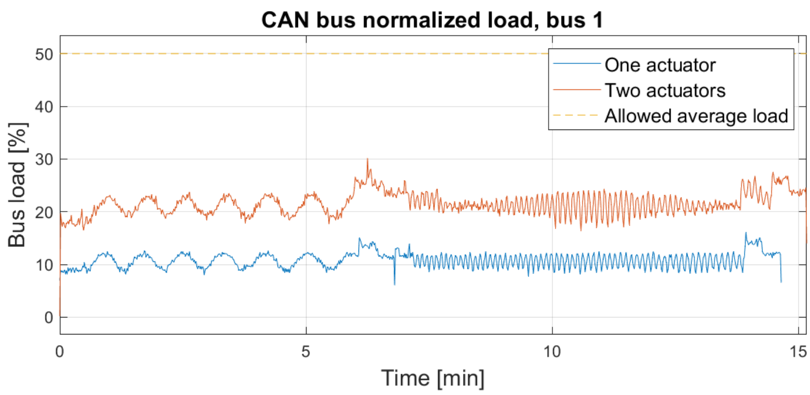


Figure 5.1. *CAN bus 1 load.*

From figure 5.1 can be seen that bus load is relatively constant over measurement period. There are no major spikes or lag spikes in the load.

Normalized bus load over measurement time for bus 2 is presented in figure 5.2.

From figure 5.2 can be seen, that number of actuators has no effect to bus load in bus 2. This is as expected. Therefore, when analyzing scalability of the system, the load of bus 2 can be dismissed.

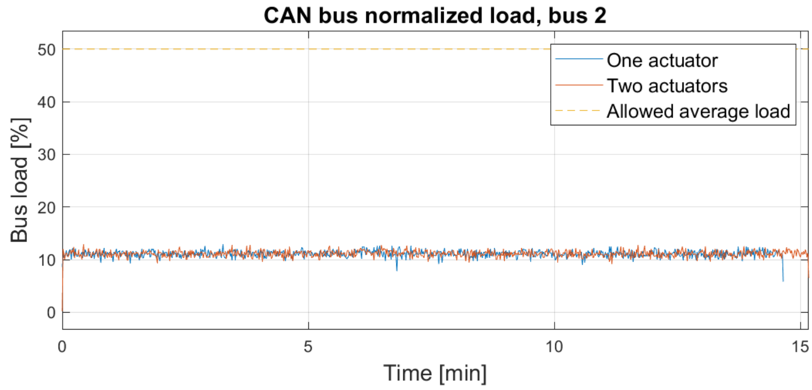


Figure 5.2. *CAN bus 2 load.*

Pressures in the system over measurement cycle were plotted for analyzing and visualization purposes. Normalized actuator pressure, hydraulic cylinder chamber pressures and supply pressure with one actuator are presented in figure 5.3.

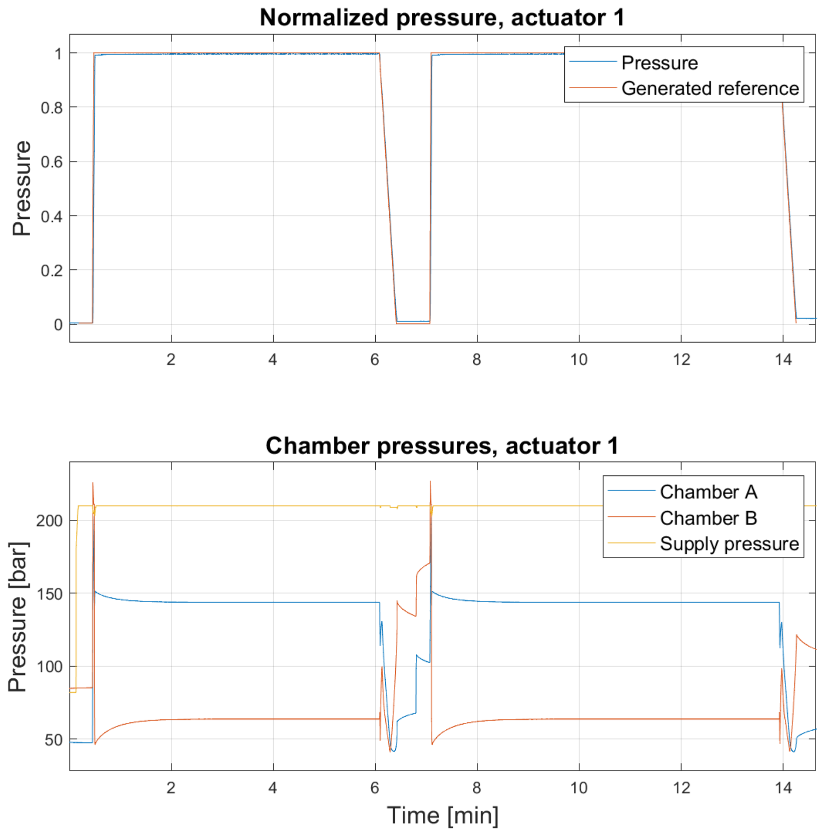


Figure 5.3. *Normalized actuator pressure and cylinder chamber pressures.*

From figure 5.3 can be seen that there are no major lag spikes in system during measurement cycle. It can also be seen, that pressures in hydraulic cylinder chambers is relatively constant when reference is reached. Chamber pressures require some time to settle, until the controller is turned off. This can be improved in future with better controller, improving controllers tuning or increasing tolerance for pressure. This will not be discussed further, as it is not in the scope of this thesis.

As measurements were performed also with two actuators, it can be beneficial to see system operation also in that case. Normalized actuator pressures, hydraulic cylinder chamber pressures and supply pressure with two actuators are presented in figure 5.4.

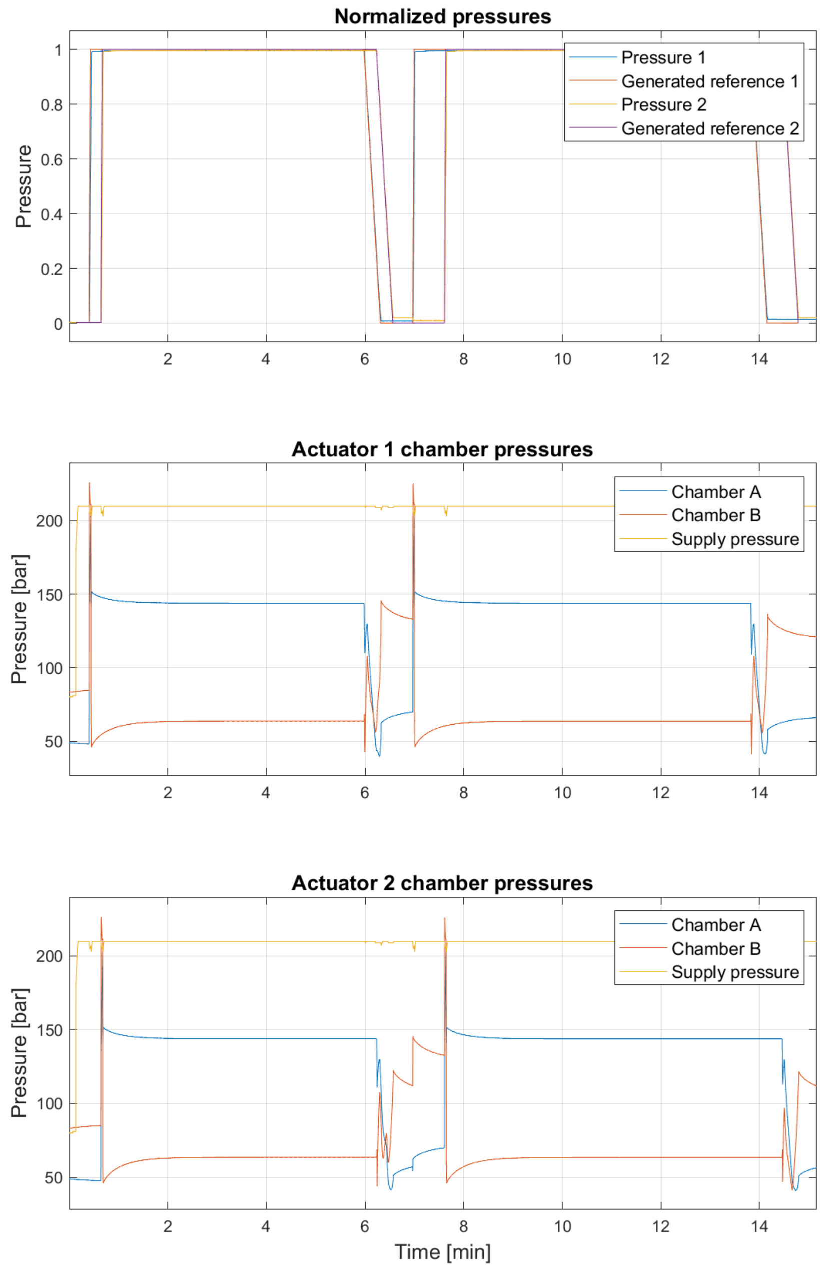


Figure 5.4. *Normalized actuator pressures and cylinder chamber pressures.*

From figure 5.4 can be seen, that system operation with two actuators is relatively similar to operation with one actuator. Main differences are few increased pressure drops in supply pressure, which are related to pressurization of the actuators. There are no other major concerns in operation, that can be seen from this simulation.

5.1.2 500 kbit/s CAN bus

Measurements with 500 kbit/s bus speed for CAN1 were performed in similar way as measurements with 250 kbit/s bus. Bus speed for CAN2 was kept at 250 kbit/s. Measurements with simulation and maximum values for average loads are presented in table 5.2.

Table 5.2. *Control system performance, Hardware-in-Loop. 500 kbit/s.*

	One actuator	Two actuators	Max
Average cycle time, program [μs]	470	574	10 000
Average cycle time, GUI [μs]	3838	4082	100 000
Average bus load, CAN1 [%]	5.4	10.8	50
Average bus load, CAN2 [%]	11.1	11.1	50
Maximum bus load, CAN1 [%]	7.6	14.6	

From table 5.2 can be seen that PLC performance for program and GUI are relatively similar to measurements done with 250 kbit/s bus. There is some variation in average cycle times, that can be considered to be random. Most important factor for measurements done with 500 kbit/s bus is that, that average cycle times are also well within allowed range. CAN1 bus performance scaling is close to linear, when compared to measurements performed with 250 kbit/s bus.

The bus load in CAN1 was plotted to visualize bus load over measurement cycle. Bus load in CAN1 with 500 kbit/s bus is presented in figure 5.5.

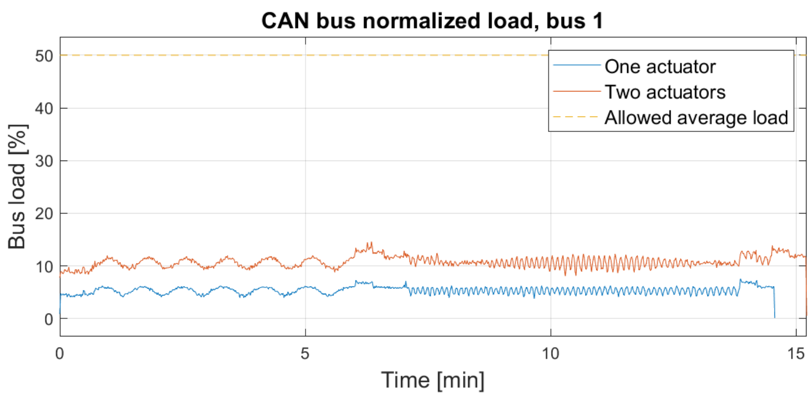


Figure 5.5. *CAN bus 1 load.*

From figure 5.5 can be seen, that bus load over measurement cycle is similar to measurements done with 250 kbit/s bus. Main difference between measurements is that normalized load is half of the normalized bus load of the measurements with 250 kbit/s bus.

Pressures in the system over measurement cycle were plotted as well with 500 kbit/s bus speed. Normalized actuator pressure, hydraulic cylinder chamber pressures and supply pressure with one actuator are presented in figure 5.6.

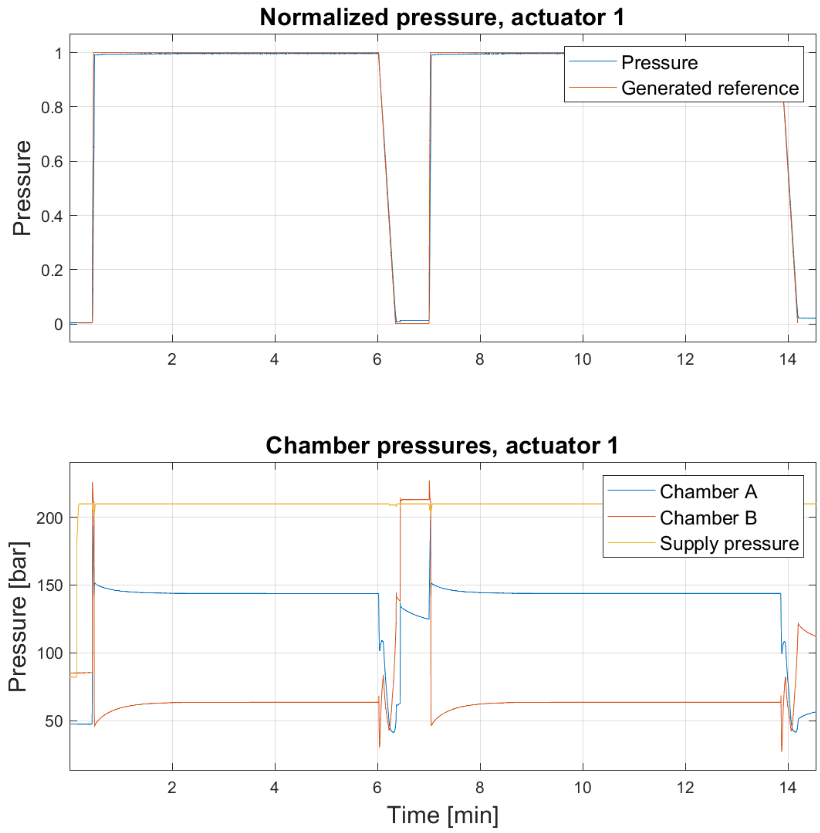


Figure 5.6. *Normalized actuator pressure and cylinder chamber pressures.*

From figure 5.6 can be seen that there are no major differences in system operation with one actuator between 250 kbit/s bus and 500 kbit/s bus. Main difference that can be seen between figures 5.3 and 5.6 is different pressures in hydraulic cylinders chambers. This can be caused by some irregularities in measurement, that are caused by measurement noise in simulation.

Normalized actuator pressures, hydraulic cylinder chamber pressures and supply pressure with two actuators are presented in figure 5.7.

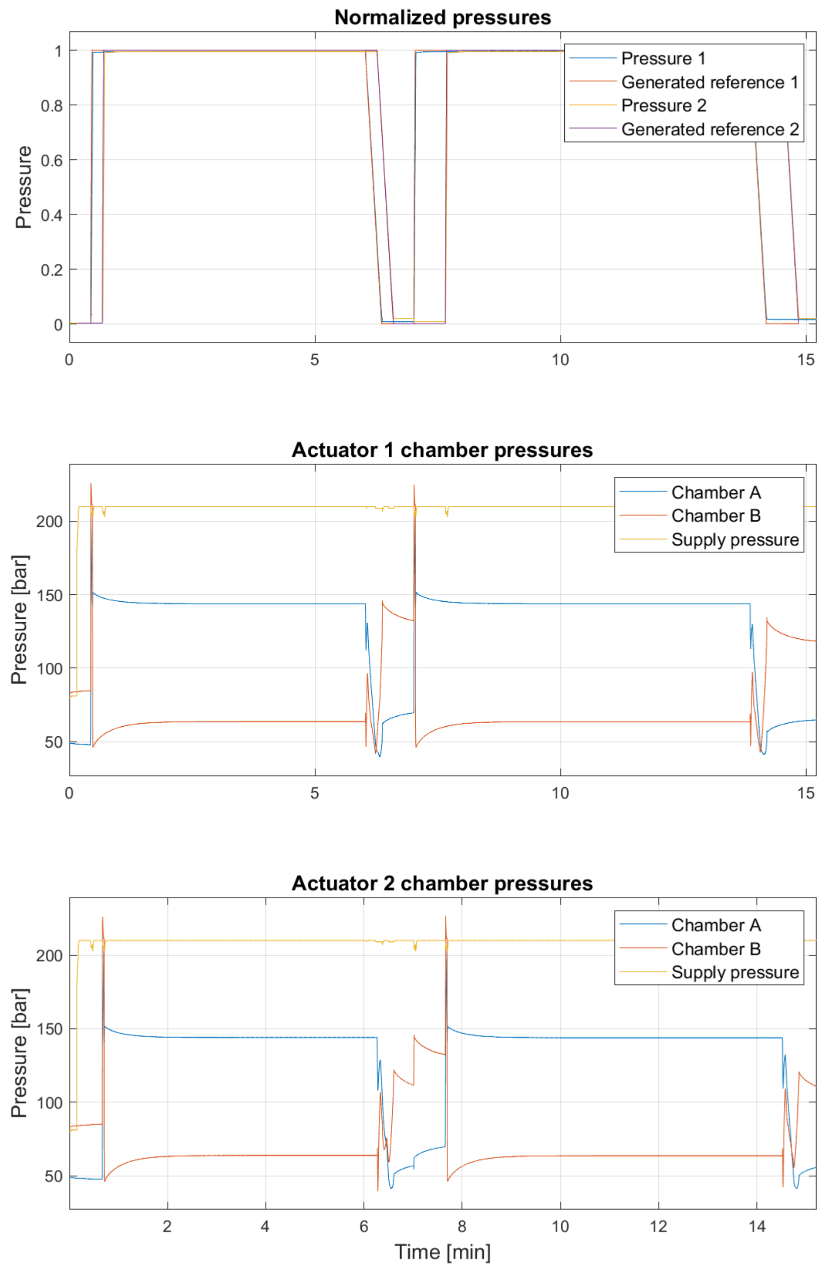


Figure 5.7. *Normalized actuator pressures and cylinder chamber pressures.*

From figure 5.7 can be seen that the operation cycle with 500 kbit/s bus and two actuators is similar to operation cycle with 250 kbit/s bus and two actuators. This is as expected, because the bus load was small enough with 250 kbit/s bus to ensure proper operation of the system at all times.

5.2 Analyzing results

As seen from tables 5.1 and 5.2 and figures 5.1, 5.2 and 5.5, all performance measures were within the allowed range. Even maximum bus load in bus 1 is under 50 %. Therefore, performance is acceptable for this application.

For future, system performance can be extrapolated in some extent from these measurements. For extrapolation it is assumed, that bus load and cycle time scale in relatively linear way if the number of actuators is increased. If this assumption is accurate enough, base load with no active actuators and increase with each actuator can be determined with first order polynomial equation.

Bus load in bus 2 in dismissed in extrapolation. Extrapolated base load with no actuators and increase with each actuator with 250 kbit/s bus are presented in table 5.3.

Table 5.3. *Control system performance analyzing, Hardware-in-Loop. 250 kbit/s bus.*

	No actuators	Increase for actuator
Average cycle time, program [μs]	449	42
Average cycle time, GUI [μs]	3468	250
Average bus load, CAN1 [%]	0.7	10.7

Based of values presented in table 5.3, maximum number of actuators in system is defined. When calculating maximum number of actuators based on GUI cycle time, it is taken to account that task containing control system is ran up to ten times in same time frame as GUI is ran once. Maximum number of actuators based on CAN bus load and cycle times are presented in table 5.4.

Table 5.4. *Maximum number of actuators, 250 kbit/s bus.*

Limiting factor	Maximum number of actuators
Average cycle time, program	227.4
Average cycle time, GUI	137.3
Average bus load, CAN1	4.6

Determining base load and increase per added actuator for each performance measurement was performed as well for 500 kbit/s bus. Results are presented in table 5.5.

Table 5.5. *Control system performance analyzing, Hardware-in-Loop. 500 kbit/s bus.*

	No actuators	Increase for actuator
Average cycle time, program [μs]	366	104
Average cycle time, GUI [μs]	3594	244
Average bus load, CAN1 [%]	0.2	5.4

Based on values presented in table 5.5 maximum number of actuators is defined. Same considerations were taken as with 250 kbit/s bus. Maximum number of actuators based on CAN bus load and cycle times with 500 kbit/s bus are presented in table 5.6.

Table 5.6. *Maximum number of actuators, 500 kbit/s bus.*

Limiting factor	Maximum number of actuators
Average cycle time, program	92.6
Average cycle time, GUI	72.2
Average bus load, CAN1	9.2

From tables 5.4 and 5.6 can be seen, that there is some variation in measurements and therefore in extrapolations. This variation can be seen especially in program and GUI cycle times between 250 kbit/s bus and 500 kbit/s bus. Despite this variation, it can be seen that limiting factor for scaling the number of actuators up is the average bus load. There are multiple ways to improve this factor.

The bus load can be reduced by further optimization of CAN messages. Most of the messages are event based, but some inhibit times might be increased. This will require further studying with test system if it is found necessary. Another option is to separate some of the actuators to second CAN bus, as bus load is second CAN bus is relatively low and steady. If some of the actuators are separated to second CAN bus, increasing its speed to match first CAN bus can be considered.

It must be considered that extrapolating this far from original measurements can be inaccurate. In addition, the number of data points is low. Due to the low number of data points, it cannot be confirmed with this test setup, that scaling is linear. Therefore, the number of actuators is only rough estimate, and further measurements and analysis about control system performance should be done if the number of actuators is increased.

6. CONCLUSIONS

Main purpose of this thesis was to design and implement control system architecture and graphical user interface for hydraulic pressurization system. There were multiple factors that had to be taken in account while designing and implementing control system architecture.

Actuator has relatively low moving mass, so control systems cycle time should be as fast enough to meet control performance requirements, that are set for the system. This is achieved via using proper control hardware and generated code, that has good performance. Proper hardware includes field bus solution, which real time performance and robustness is good enough. These are validated with Hardware-in-Loop simulations.

There are two actuators in the system. Hydraulic pressure in the system must always be high enough to ensure proper operation of the system. This is achieved with proper controller solutions, tuning controllers, and utilizing dynamic references for actuator. Pressurizing rate is dependent on hydraulic pressure level.

System operates in the real world, so some real-world constraints must be considered. Most of the measurements from the system are analog and they contain some measurement noise. If measurement and set value are used as condition for transition in state machine, it is unlikely to reach exact value. Therefore, it was necessary to implement tolerance for transition conditions. Some actuators in subsystems that are controlled as digital outputs have delay when turning them on and off, which modeling was out of this thesis scope. This was considered with parameterized delay for turning those actuators on. This was made to ensure that actuators that perform opposite actions are not operated at same time. In addition, minimum time for the steps in state machine was implemented. This was to prevent considering step to be done, even if it is not actually started.

Second purpose of this thesis was to simulate, measure and analyze control system performance in this application. This was done with Hardware-in-Loop simulation. Simulation was done with actual controller, program, and GUI. The controller was connected to the computer running simulation model with CAN adapters. Controller and computer were also connected with Ethernet for diagnostic purposes.

Control systems performance was simulated and measured with one and two actuators. The focus on the measurements was to determine if PLC program cycle time and CAN bus average load are low enough for proper operation. Measurements validated, that CAN bus load and program cycle time were in allowed ranges, and therefore control systems performance should be good enough for this application. Secondary focus on measurements was to extrapolate control system performance if number of actuators is increased. Extrapolation implicated that there is room to increase number of actuators in system, while cycle time and CAN bus load are within allowed range.

This work can be considered overall mostly as success. Designing, implementing, and testing the control system with Hardware-in-Loop was completed within schedule. Implementing and testing the control system architecture with the actual test system was not possible due to time constraints.

There are multiple factors that can be improved in this topic in future. Queuing logic implemented in this topic was relatively simple, and it is feasible only when number of items in queue is relatively low. With the current solution delay increases slightly with each actuator that is added to the system. It doesn't contain options for prioritizing some items or operations over others. Conditions for deciding if some operations in process are done might be improved. This will require further studying with the actual test system. CAN messages could be optimized further with test system to decrease bus load. As premature optimization is the root of all evil, it should not be done unless necessary in future.

REFERENCES

[1] W. Bolton. *Programmable logic controllers*. eng. 6th ed. Amsterdam, Netherlands: Newnes, 2015. ISBN: 0-12-802929-3.

[2] D. H. Hanssen and D. Lufkin. *Programmable logic controllers : a practical approach to IEC 61131-3 using CODESYS*. eng. First edition. Chichester, England: Wiley, 2015. ISBN: 1-118-94922-6.

[3] E. A. Parr. *Programmable controllers : an engineer's guide*. eng. 1st edition. Oxford, England: Newnes, 1993. ISBN: 1-4831-8377-7.

[4] *Programmable controllers - Part 3: Programming languages*. Standard. Geneva, CH: International Electrotechnical Commission, Feb. 2013.

[5] *Programmable controllers - Part 1: General information*. Standard. Geneva, CH: International Electrotechnical Commission, May 2003.

[6] K. H. John. *IEC 61131-3: Programming Industrial Automation Systems Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. eng. 2nd ed. 2010. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. ISBN: 3-642-43694-3.

[7] Codesys GmbH. *Object 'POU'*. Kemptren, Germany. URL: https://help.codesys.com/webapp/_cds_f_obj_pou;product=codesys;version=3.5.17.0 (visited on 09/19/2023).

[8] The Mathworks Inc. *Integers*. Natick, Massachusetts, United States. URL: https://se.mathworks.com/help/matlab/matlab_prog/integers.html (visited on 04/25/2023).

[9] The Mathworks Inc. *Floating-Point Numbers*. Natick, Massachusetts, United States. URL: https://se.mathworks.com/help/matlab/matlab_prog/floating-point-numbers.html (visited on 04/25/2023).

[10] CAN in Automation (CiA). *History of CAN technology*. Nuremberg, Germany. URL: https://www.can-cia.org/can-knowledge/can/can-history/ (visited on 05/08/2023).

[11] CAN in Automation (CiA). *Classical Controller Area Network (CAN)*. Nuremberg, Germany. URL: https://www.can-cia.org/can-knowledge/can/classical-can/ (visited on 05/08/2023).

[12] CAN in Automation (CiA). *Physical layer options*. Nuremberg, Germany. URL: https://www.can-cia.org/can-knowledge/can/systemdesign-can-physicallayer/ (visited on 05/08/2023).

[13] CAN in Automation (CiA). *Technical documents*. Nuremberg, Germany. URL: https://www.can-cia.org/groups/specifications/ (visited on 05/09/2023).

[14] M. D. Natale, A. Ghosal, P. Giusto and H. Zeng. *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. ger ; eng. 1. Aufl. New York, NY: Springer-Verlag, 2012. ISBN: 9781461403135.

[15] CAN in Automation (CiA). *CAN high-speed transmission*. Nuremberg, Germany. URL: https://www.can-cia.org/can-knowledge/can/high-speed-transmission/ (visited on 05/09/2023).

[16] CAN in Automation (CiA). *CANopen – The standardized embedded network*. Nuremberg, Germany. URL: https://www.can-cia.org/canopen/ (visited on 05/08/2023).

[17] CAN in Automation (CiA). *CANopen history*. Nuremberg, Germany. URL: https://www.can-cia.org/can-knowledge/canopen/canopen-history/ (visited on 05/08/2023).

[18] CAN in Automation (CiA). *CANopen lower layers*. Nuremberg, Germany. URL: https://www.can-cia.org/can-knowledge/canopen/canopen-lower-layers/ (visited on 05/08/2023).

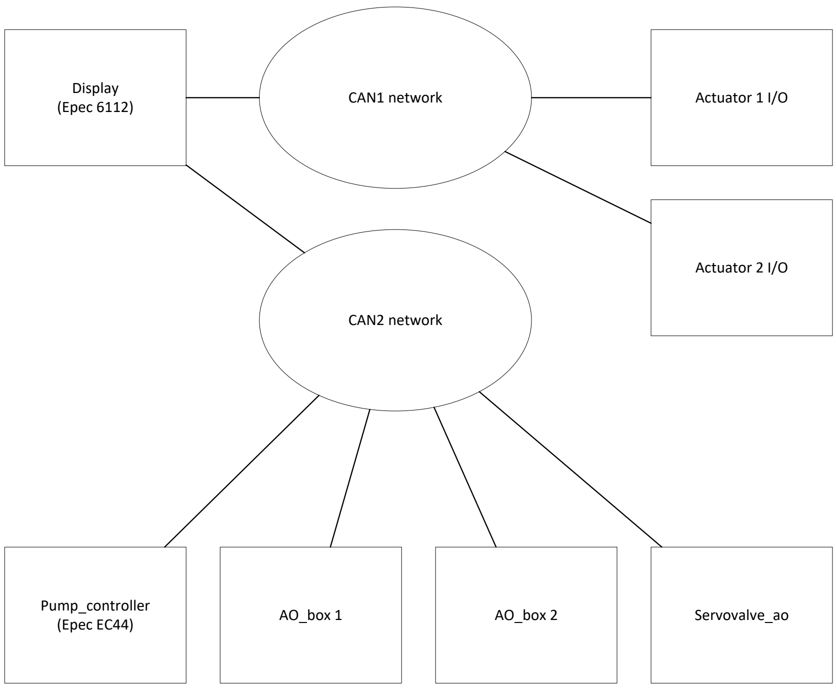
[19] CAN in Automation (CiA). *Designing a CAN network*. Nuremberg, Germany. URL: https://www.can-cia.org/can-knowledge/can/design-can-network/ (visited on 05/09/2023).

[20] The Economist. *Tech.View: Cars and software bugs*. URL: https://www.can-cia.org/can-knowledge/can/high-speed-transmission/ (visited on 05/09/2023).

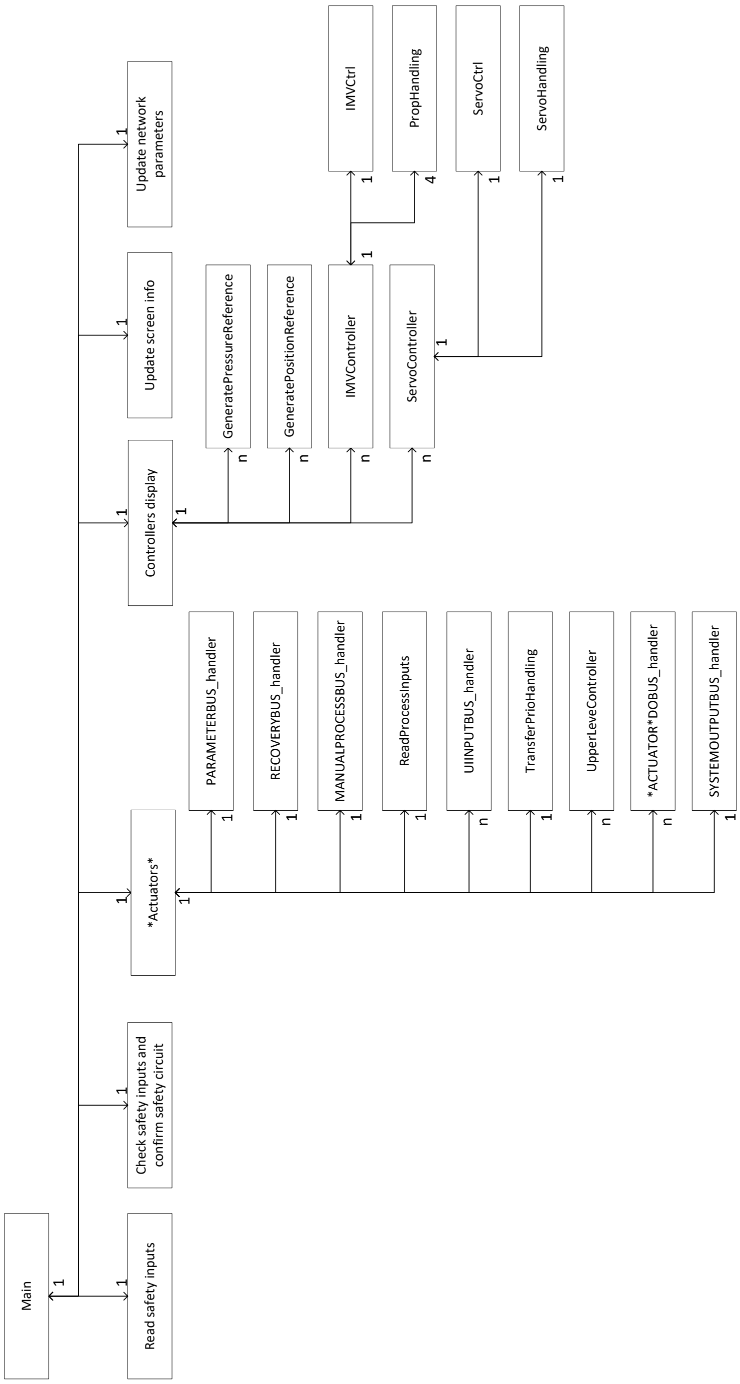
[21] V. Socci. Implementing a model-based design and test workflow. eng. *1st IEEE International Symposium on Systems Engineering, ISSE 2015 - Proceedings*. IEEE, 2015, pp. 130–134. ISBN: 9781479919208.

[22] The Mathworks Inc. *Model-Based Design with Simulink*. Natick, Massachusetts, United States. URL: https://se.mathworks.com/help/simulink/gs/model-based-design.html (visited on 04/25/2023).

APPENDIX A: CAN NETWORK ARCHITECTURE



APPENDIX B: PROGRAM ARCHITECTURE, DISPLAY



APPENDIX C: PROGRAM ARCHITECTURE, SECONDARY CONTROLLER