Jouni Laakso

Learning environment of Exertus control systems

Bachelor's Thesis

Spring 2013

School of Technology

Automation Engineering

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Engineering

Specialisation: Machine Automation

Author: Jouni Laakso

Title of thesis: Learning environment of Exertus control systems

Supervisor: Niko Ristimäki

Year: 2013	Number of pages: 49	Number of appendices: 1

The purpose of this thesis was to develop and implement a learning environment for Exertus OY at Seinäjoki University of Applied Sciences. The development of the learning environment included the selection of Exertus' products that the learning environment will contain and the aggregation of useful and sufficient instructions for the use of Exertus PC software. The development of the learning environment included also the designing and programming of the simulation software that can be used in the testing of control systems made with Exertus' products.

This report consists of a theory part, development of the learning environment and implementation of the simulation software. The subjects of the theory part are CAN bus, CANopen protocol and TCP/IP protocol suite.

As a result of this Bachelor's thesis, a learning environment was designed. The thesis work proceeded in schedule and the goals of the thesis were achieved.

Keywords: CAN bus, CANopen, control systems, distributed systems

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Koulutusohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Koneautomaatio

Tekijä: Jouni Laakso

Työn nimi: Exertus control systems -koulutusympäristö

Ohjaaja: Niko Ristimäki

Vuosi: 2013 Sivumäärä: 49 Liitteiden lukumäärä: 1

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa koulutusympäristö Exertus OY:n tuotteilla Seinäjoen ammattikorkeakoululle. Koulutusympäristön kehitys sisälsi laitteiston ja ohjelmiston valinnan sekä hyödyllisten ja riittävien käyttöohjeiden laatimisen ohjelmistolle. Koulutusympäristön kehitys sisälsi myös simulaatiosovelluksen suunnittelun ja ohjelmoinnin. Sovelluksen avulla voidaan testata Exertuksen tuotteille ohjelmoituja ohjaus toimintoja.

Tämä opinnäytetyön raportissa on kolme osiota: teoriaosio, koulutusympäristön kehitys sekä simulaatiosovelluksen toteutus. Teoriaosiossa esitellään CAN-väylää ja CANopen-protokollaa, jotka liittyvät vahvasti Exertuksen toimialaan. Lisäksi teoriaosiossa käsitellään TCP/IP-protokollaperhe, joka liittyy Exertuksen PC-sovelluksiin.

Opinnäytetyön tuloksena saatiin suunniteltua koulutusympäristö. Työ eteni aikataulussa ja sen tavoitteet saavutettiin.

Avainsanat: CAN-väylä, CANopen, hajautetut järjestelmät, ohjausjärjestelmät

TABLE OF CONTENTS

List of figures and tables

Terms and abbreviations

ACK	Acknowledge
CAN_H	CAN-high cable
CAN_L	CAN-low cable
CAN	Control Area Network
CAN bus	A vehicle bus standard that can be used in communication between different devices within a vehicle or machine
CiA	CAN in Automation, an automation field standardization organization
CRC	Cyclic Redundant Check
DLC	Data Length Code
EMCY	Emergency
EOF	End of Frame
FBD	Function Block Diagram
GUI	Graphical User Interface
I/O	Input / Output
IDE	Identifier Extension
IFS	Intermission Frame Space
ISO	International Standards Organization
LED	Light-Emitting Diode
LSS	Layer Settings Services

NMT	Network Management
Node	A device connected to a CAN bus
NRZ	No Return to Zero
OSI	Open Standards Interconnection
PDO	Process Data Object
r	Reserved bit
RTR	Remote Transmission Request
SDO	Service Data Object
SYNC	Synchronization
TCP/IP	Transmission Control Protocol / Internet Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus

1 INTRODUCTION

1.1 Backround

Exertus OY is still a rather small but growing company and it wants to increase people’s knowledge of the company. A good way to do this is to get students to learn to know the company and its products. Exertus decided to develop a learning environment in cooperation with Seinäjoki University of Applied Sciences at the School of Technology.

1.2 Objectives of the study

The objective of this Bachelor’s thesis is to develop and engineer a learning environment for Exertus OY at Seinäjoki University of Applied Sciences. The goal is that with the learning environment it is possible to practice the programming of Exertus I/O controllers, the development of graphical user interfaces (GUI) and the configuration and diagnostics of a CAN bus.

The thesis consists of the development and implementation of the equipment that the learning environment will contain and the aggregation of useful and sufficient instructions for the use of the Exertus PC software that will also be included in the learning environment. The idea is that one is able to learn to create his or her first, working control system with the help of the learning environment.

1.3 Structure of the report

This report consists of a theoretical part, the development of the learning environment and of the evaluation of the study. In the theoretical part the CAN bus and CANopen protocol are discussed. These two subjects are relevant when using the learning environment and Exertus products. Also the TCP/IP protocol suite is presented in the theory part. It is a relevant subject in the Exertus software.

1.4 Exertus OY

Exertus OY was founded in the year 2003 and it is located in Seinäjoki Finland. Exertus develops advanced and comprehensive mobile control systems for machine manufacturers. The main goal of Exertus is to help its customers to make their products smarter and to help them succeed in the global markets where competition is hard.

1.4.1 Know-how

The core competence of the company is in the application of the CAN bus based distributed control systems that are used widely in different kinds of machines, for example, forest harvesters and earthmovers. The customer can be offered a comprehensive control system or a part of it. The service concept of the company includes predesign, consulting, defining, design, implementation, training and maintenance of the control system.

1.4.2 Products

The electronic products developed by Exertus are based on optimal components, the existing standards and many years of experience in electronics design. The products are designed for generic use or they can be also designed for a specified use. Competitive prices are also possible due to the innovative solutions in design.

Specified uses are, for example, some machines that have small manufacturing quantities but need a smart control system or the control system requires some special feature. Generic uses are, for example, machines with either big or small manufacturing quantities but the control systems of the machines do not require any special features that the existing products do not already have.

2 CAN bus

The serial bus system CAN (Controller Area Network) was developed by Robert Bosch GmbH in the mid 1980s. It was created to respond to the requirements of the developing electronics of vehicles. The CAN bus was originally developed to be used in real-time data transmission in distributed control systems. Today the CAN bus is also used in the controlling of different actuators and in the transmission of information from sensors in vehicles and also in industry. Now many electronics and machine manufacturers have joined to the bus technology. (Juhala, Lehtinen, Suominen and Tammi 2005, 129.)

CAN bus is by its structure a multi-master bus. That means that every node on the bus has an equal authorization to send its messages to the bus. The messages are not addressed to a certain node but every node determines by itself what kind of information it needs. Therefore no addresses are bind to the nodes and this is a great advantage because new nodes can be added or old ones can be removed flexibly and easily. (Juhala etc. 2005, 129.)

2.1 Physical layer

The CAN bus is built of two conductors that are named CAN_H (CAN High) and CAN_L (CAN Low). The data transmission on the CAN bus is based on the voltage difference of these two conductors. Generally a protected twisted pair wire is used to form the CAN bus. The bus has typically 120 Ω terminators at both ends that prevent the sent signals from being reflected back to the bus as echoes from the ends. These echoes would disturb the traffic on the bus. The impedance of the bus should also be nominally 120 Ω. (Juhala etc. 2005, 134-135.) The principle of the CAN bus is presented in Figure 1.

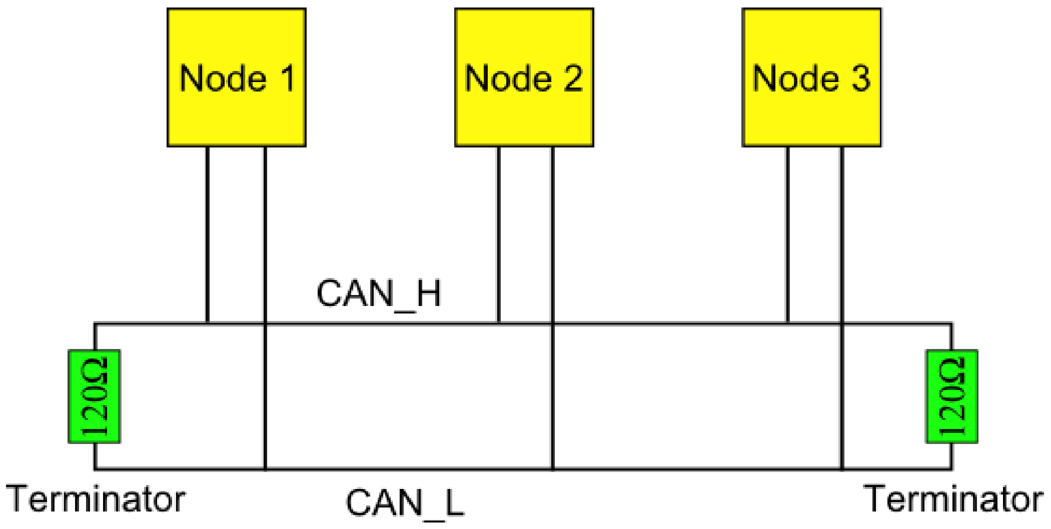


Figure 1: The principle of the CAN bus with three nodes.

There is no logical limit for how many nodes there can be on a CAN bus. Effectively the number of the nodes is limited by the used transceiver of the node. The number of the nodes on the bus varies in area of 100-200 depending on the used transceiver. The number can be increased by using reproducers but they also increase the lag of the transmission path and therefore they also decrease the maximum length of the bus. (Alanen 2000, 5.)

The maximum speed of the CAN bus is 1 Mbit/s. The maximum length of the bus with this speed is 40 m. With the speed of 50 kbit/s the length can be approximately 1 km. The lag of the transmitting path defines the maximum length of the bus. So, the maximum length depends on the travelling speed of the electromagnetic waves. The only way to increase the maximum length of the bus is to reduce the transmitting speed. This feature comes from the bus arbitration principle of the CAN protocol. The nodes have to take a sample of a single bit at the same time with certain accuracy and it requires that the lag is not too great. (Alanen 2000, 5.) The bus arbitration is presented in chapter 2.4.

The voltage difference between the CAN_H and CAN_L conductors define the state of the bus. The bus can be in two different logical states: dominant or recessive. Dominant state is the logical 0 and recessive the logical 1. ISO CAN standard defines that at the recessive state, the voltage value of both conductors is 2,5 volts and at the dominant state, the voltage value of the CAN_H is 3,5 volts and CAN_L 1,5 volts. Therefore, the voltage difference at the recessive state is 0 volts and at the dominant state 2 volts. (Juhala etc. 2005, 133-135.)

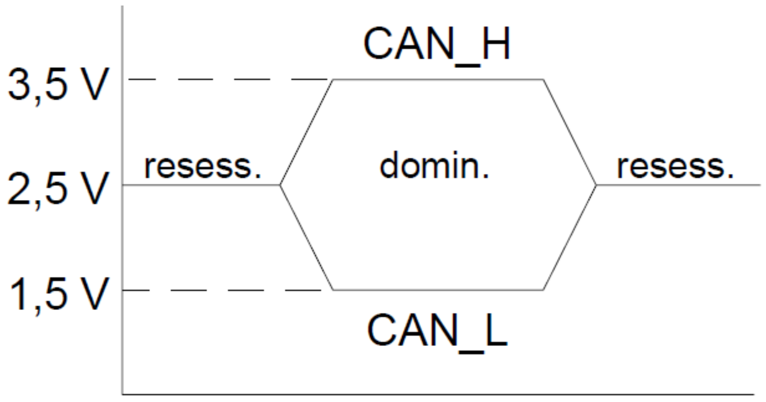


Figure 2: Voltage levels of the recessive and dominant states (Alanen 2000, 10).

2.2 Data link layer

The CAN specification includes the data link layer in both the original format and in the extended format. The data link layer is also standardized in ISO 11898 standard. (Juhala etc. 2005, 130.)

The CAN protocol defines four different types of message frames:

- Data Frame
- Remote Frame
- Error Frame
- Overload Frame. (Alanen 2000, 6.)

2.2.1 Data Frame

The total length of the original format data frame is at maximum 111 bits plus stuff bits and the extended format data frame is 131 bits plus stuff bits. The stuff bits are explained in chapter 2.3. (Juhala etc. 2005, 130-131.)

The data frame includes the following parts:

- SOF (Start of Frame): SOF is the start bit of the message. It is always of dominant state.
- Arbitration field: Arbitration field consists of the message identifier and the Remote Transmission Request (RTR) bit which is used to determine whether the frame is a data frame or a data request frame. In the original format the identifier is 11 bits and in the extended format 29 bits (Figure 4).
- Control field: Control field contains the Identifier Extension (IDE) bit, which determines whether the data frame is in the original format or in the extended format, and the Data Length Code (DLC), which is used to tell the number of bytes in the Data field. Between the IDE and DLC there is also a reserved bit which is not used, so it should be 0.
- Data field: Data field is at most 8 bytes, 0-64 bits.
- CRC (Cyclic Redundant Check): The integrity of the frame is checked with the CRC.
- ACK (Acknowledge): The ACK is formed of two bits, ACK slot and ACK delimiter. The sender of the message sends the ACK slot bit as a recessive bit and the receiver overwrites it as a dominant bit if it has received the data correctly at this time.
- EOF (End of Frame): EOF indicates the end of the message.
- IFS (Intermission Frame Space): IFS separates the consecutive messages. If there is no consecutive message, the bus remains idle after the IFS. (CAN in Automation (CiA) 2013a.)

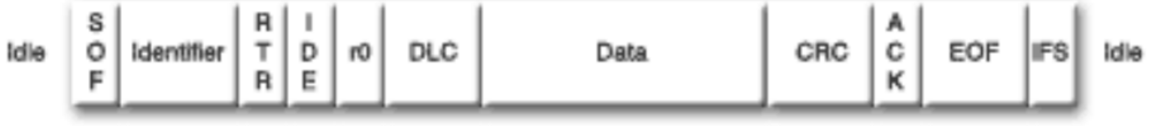


Figure 3: Original format of the data frame (CAN in Automation (CiA) 2013).

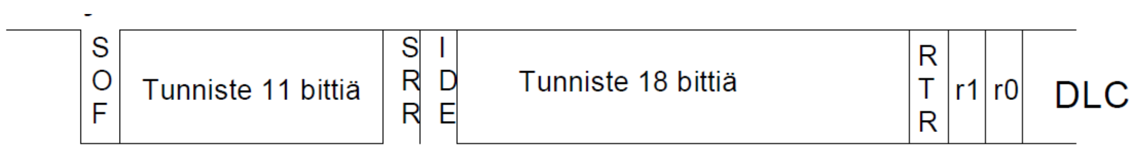


Figure 4: Extended format of the data frame (Alanen 2000, 6).

2.2.2 Remote Frame

The remote frame has almost the same structure as the data frame but the RTR bit is recessive and there is no data field. With the remote frame, desired information can be requested to the bus. (Alanen 2000, 6-7.)

2.2.3 Error Frame

The error frame is sent if an error is detected within the message. In the error frame there are six dominant bits and eight recessive bits. Therefore it violates the CAN protocols bit stuffing rule. This affects that every node on the bus states that the message is faulty and discards it. It is also possible that the node goes into a passive error state in which the structure of the error frame changes. In this state, instead of sending six dominant bits, six recessive bits are sent. This way the node will not mess up the traffic of other nodes. (Alanen 2000, 7.)

2.2.4 Overload Frame

The overload frame can be sent between messages when the bus is in an idle state. By doing this, the receiving node can get extra time to process the data message which it has received. When the overload frame is on the bus, no messages can be sent. This gives extra time. The overload frames are rarely used in practical systems, though, because the circuits today do not need extra time to process a single data message. (Alanen 2000, 7.)

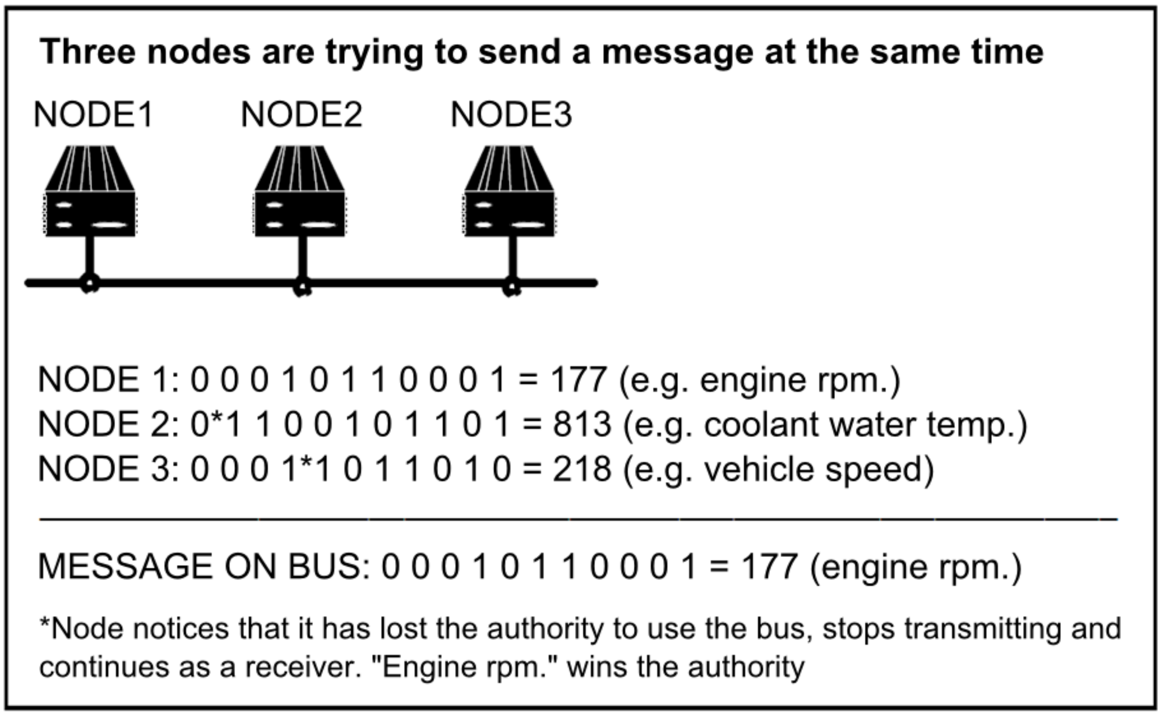
2.3 Bit stuffing

The CAN protocol has a method called NRZ (No Return to Zero) bit stuffing. If there are five bits of the same kind in a row in the bit string that the sender is sending, the sender adds a bit with a complementary value after them. This bit is called a stuff bit. The receiver of the data message removes the stuff bit from the message. The stuff bit makes it easier to detect errors and it helps the synchronization between the nodes. (Juhala etc. 2005, 131.)

2.4 Bus arbitration

Any of the nodes on the CAN bus can try sending a message, if the bus is free. If several nodes are trying data transmission at the same time, the transmitting turn is solved with the bus arbitration. This means that the identifiers of the data messages are inspected bit by bit and the turn is solved according to them. Because the zero bit is the dominant bit on the CAN bus, the data message with the smallest identifier has the biggest priority and it will be sent first to the bus. (Alanen 2000, 7.)

The bus arbitration can be seen in work in Figure 5. In the figure there is three nodes connected to the bus and they are sending a data message at the same time. Each data message has different identifier. The bit strings of the identifiers are inspected bit by bit. While inspecting the second bit, the node 2 notices that it has lost the authority to use the bus so it stops transmitting and continues as a receiver. Same thing happens to node 3 while inspecting the fifth bit. Node 1 gets the authority to send the data message first to the bus because it has the smallest identifier.



Three nodes are trying to send a message at the same time

NODE 1: 0 0 0 1 0 1 1 0 0 0 1 = 177 (e.g. engine rpm.)
NODE 2: 0*1 1 0 0 1 0 1 1 0 1 = 813 (e.g. coolant water temp.)
NODE 3: 0 0 0 1*1 0 1 1 0 1 0 = 218 (e.g. vehicle speed)

MESSAGE ON BUS: 0 0 0 1 0 1 1 0 0 0 1 = 177 (engine rpm.)

*Node notices that it has lost the authority to use the bus, stops transmitting and continues as a receiver. "Engine rpm." wins the authority

Figure 5: Bus arbitration method (Alanen 2000, 7).

3 CANopen

3.1 CAN in Automation (CiA) and CANopen

CiA is a nonprofit organization founded in the year 1992 by several international users and manufacturers. The idea of the CiA was to provide technical, product and marketing information and to promote the CAN's image and to develop and support CAN-based higher-layer protocols. The CiA group has today approximately 560 company members. (CAN in Automation (CiA) 2013b.)

CANopen protocol is standardized by the CiA. The CANopen protocol is one of the most widely used protocols of the CAN bus application layer. At first, it was defined to work only in the CAN networks but because of its popularity and flexibility, it has been taken into use also in other buses. (Saha 2006, 6.)

The most important thing that the CANopen protocol defines is the Object Dictionary, which separates the software and the bus transmission in every node. With the Object Dictionary it is possible to identify the node on the bus and to master all operations of the node via the bus. In addition to the node and data transmission models and the services of the application layer, CANopen defines also other useful things, like connector types with their pin orders to ensure the compatibility of the nodes. (Saha 2006, 6.)

There are two kinds of nodes in the CANopen protocol: CANopen-device and CANopen-manager. CANopen-device is a node that can contain any kind of software functionality. CANopen-manager is a CANopen-device that also includes the following properties: NMT-master, SDO-manager and Configuration manager. It also includes one of the following properties: SYNC-producer, TIME producer or LSS-master. (Saha 2006, 6.) The properties are explained in chapter 3.2.

3.2 Protocols of the CANopen

CANopen provides every bus system many protocols for the required services. Some functions, like starting of the network or updating of the signals between the nodes, may use several protocols to accomplish the function. (Saha 2006, 8.) The different protocols are explained below.

Layer Settings Services (LSS). LSS provides the services for setting the data transmission speed and the node identifier in a controlled way. It is typically used in editing the settings of the nodes before they are installed to the target system. The LSS-master does this. (Saha 2006, 8.)

Boot (Bootsrap). With the Boot protocol, the CANopen node informs the other nodes on the bus that it has started. This information is mostly used by the NMT-master functionality of the CANopen-manager when it is starting the bus. (Saha 2006, 8.)

Heartbeat. With the Heartbeat protocol, the node can produce information on its function state to the other nodes that are connected to the same bus. This information can be used by any node on the bus. (Saha 2006, 8.)

Service Data Object (SDO). By the SDO protocol it is possible to read values from the Object Dictionary of a CANopen node and to write values there. SDO-manager property uses this protocol in the connection management. (Saha 2006, 8.)

Network Management (NMT). By the NMT protocol, the NMT-master can start the nodes connected to the CANopen network. NMT-master is the property for starting the network. (Saha 2006, 8.)

TIME. With the TIME protocol it is possible to provide global time information to the CANopen network. TIME producer uses this protocol. On every network there can be only one TIME producer. (Saha 2006, 8.)

SYNC (Synchronization). SYNC protocol is used to synchronize the transmitting of the PDO frames. This is done by the SYNC producer property. (Saha 2006, 8.)

Process Data Object (PDO). With the PDO protocol, the process signals' states between the node Object Dictionaries are updated. A PDO can be sent both synchronously and asynchronously. (Saha 2006, 8.)

EMCY (Emergency). By the EMCY protocol, information on operations is transmitted in the CANopen network. This information typically consists of error notifications. (Saha 2006, 8.)

3.3 Network management states

Every CANopen node has three different constant network management states - pre-operational, operational and stopped - and two momentary network management states – reset node and reset communication. With a NMT command, the constant state of a CANopen node can be changed to any other state. (Saha 2006, 9.) The different states are explained below.

Reset node. Reset node state includes all initializations related to the node, except the initialization and starting of the CANopen protocol stack. (Saha 2006, 9.)

Reset communication. Reset communication state includes the initialization and starting of the CANopen protocol stack. A boot message should be sent in this state right before the node turns into the pre-operational state. (Saha 2006, 9.)

Pre-operational. Pre-operational state is a safe state into which the node turns after it has started. The node waits in the pre-operational state that the NMT-master will complete the controlled start of the bus and the possible inspections and parameter changes according to it. (Saha 2006, 9.)

Operational. In the operational state, the CANopen network is fully working. In this state, also the PDO transmitting is allowed like the SDO transmitting and notifications with the EMCY frames. (Saha 2006, 9.)

Stopped. Node is led to the stopped state typically if a serious error has occurred. In the stopped state, the node is only allowed to consume NMT commands and to produce information on its state with the heartbeat protocol. (Saha 2006, 9.)

The network management states of the CANopen node can be seen in Figure 6. The right moment for transmitting the boot message is marked into the figure with the *-mark.

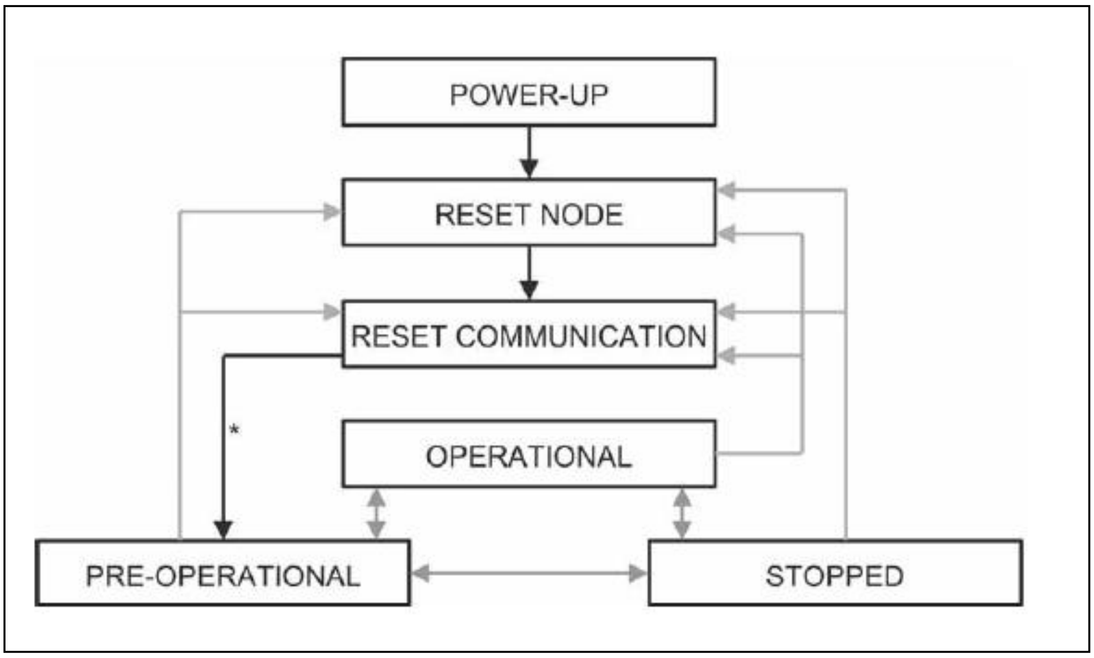


Figure 6: The states of a CANopen node (Saha 2006, 8).

3.4 Controlled start of the network

The system can work safely only if every node in the bus has the right firmware and if its parameters are correct. Because of this, a method to make a controlled start to the network is defined. (Saha 2006, 9.)

When the NMT-master starts, it restarts all other nodes CANopen protocol stacks to get a boot message from every node connected to the bus. After the NMT-master has received the boot messages, it runs a starting sequence to every node that is defined into the node list in the Object Dictionary. (Saha 2006, 9.)

The starting sequence includes the following functions

1. The existence and the correctness of the node are checked with the help of the device type, manufacturer code, product code and the version and serial number.
2. CANopen protocol stack of the node is not restarted if there is a safety risk in it. This is a so called keep-alive property.
3. The firmware version correctness is checked.
4. If the node has a wrong firmware version, the firmware will be updated.

5. The time when the parameters have been changed the last time, is checked.
6. The parameters of the node are updated if a wrong time is noticed.
7. If all the selected inspection functions have been completed without errors, the heartbeat of the node is started and the node is led to the operational state. (Saha 2006, 9-10.)

After the starting sequence has been completed, the state information on the starting sequence is sent to the application by the NMT-master's CANopen protocol stack. The application then decides if the NMT-master node may be led to the operational state. (Saha 2006, 10.) The controlled start of the CANopen network is presented in Figure 7.

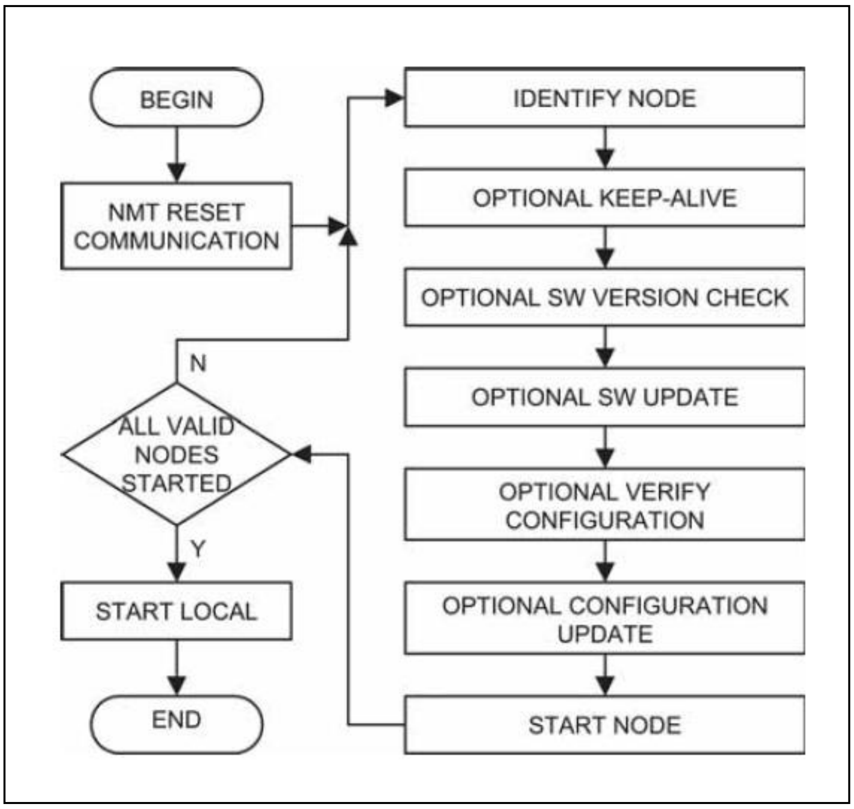


Figure 7: The controlled start of the CANopen network (Saha 2006, 8).

4 TCP/IP protocol suite

TCP/IP (Transmission Control Protocol / Internet Protocol) is commonly spoken as the TCP/IP protocol but it is not an actual protocol but it is a protocol suite consisting of protocols developed for different purposes. The TCP/IP protocol suite is originally designed for Internet use. Its function is to provide useful message transmission rules for networks that are built up of smaller networks. In a TCP/IP network, the basis of operations is the communication between the networks. (Vainio & Hakala 2005, 178-179.)

The services of modern data networks are usually based on the different software protocols of the TCP/IP protocol suite. The reasons of the popularity of the TCP/IP protocol suite are its openness and the commonly approved standards. (Vainio & Hakala 2005, 178.)

4.1 IP protocol

The IP (Internet Protocol) is the first of the most important protocols of the TCP/IP protocol suite and because of that its name is taken to the name of the protocol suite. The IP defines the packet transmission service, which is the most relevant service of the protocol suite. The service is technically defined as an unreliable, connectionless and best-effort packet transmission service. The service is called unreliable because the arrival of the packets is not ensured. Connectionless means that every packet is handled independently regardless of the others. The programs of the network try their best to transmit the packets, and this is where the word best-effort comes from. The connectionless packet transmission service of the IP protocol builds up the base for the service provided by the TCP protocol. (Comer 2002, 97.)

4.2 TCP protocol

The TCP (Transmission Control Protocol) is the second of the protocols of the TCP/IP protocol suite that has given its name to the suite. The TCP provides software a reliable message transmission service. The reliability means that the TCP can ensure the arrival of data and correct transmission failures so that the software don’t need to mind of these things. The transmission service provided by the TCP is always point-to-point style service. This means that the connection is always created between two programs and it is not possible to create broadcast or multicast connections. (Kaario 2002, 166.)

4.3 OSI model

The ISO (International Standards Organization) OSI model (Open Systems Interconnection) has had an important role in the communications protocols. The idea of the OSI model was to provide device manufacturers and network users an environment in which the systems based on the environment would be able to communicate effortlessly with each other. It would not be, therefore, needed to fit various networks created for different uses to work together. The target of the OSI model has not been reached, but the layer model has been a base for the communications. (Kaario 2002, 18.)

The OSI model has seven different layers. The layers are explained below and they can also be seen in Table 1.

Physical layer. The physical layer takes care of the physical transmission of the bit stream. Amongst other things, the physical layer must take care of the connectors, the physical properties of the transmission path and the voltage levels of the signals. (Kaario 2002, 19.)

Link layer. The task of the link layer is to reliably transmit the bit stream along the transmission path. The layer is carried out with a transfer protocol that can examine data of the signal for transmission errors and send the data in a specified packet. With the link layer it is also possible to control the flow or limit the amount of data coming to the physical layer when needed. (Kaario 2002, 20.)

Network layer. The most important task of the network layer is to route the data packets to the receiver through the network. Additionally the flow control and keeping watch of quality requirements can be done with this layer. (Kaario 2002, 20.)

Transport layer. The transport layer takes care of creating a direct connection between the transmitting systems. In other words, it provides transparent transfer of data in the network between the transmitter and the receiver. TCP protocol for example can be located on this layer. (Kaario 2002, 21.)

Session layer. The session layer creates connections between the transmitting systems and also removes them. It also superintends and supports the communication needs of the upper layers. (Kaario 2002, 21.)

Presentation layer. The task of the presentation layer is to take care of the manner of representation during the transfer and to arrange the used manner of representation. (Kaario 2002, 21.)

Application layer. The application layer defines an equal communication interface to the network for all communications software. The software can be, for example, terminal operations, email, a file transfer or a directory service. The software is outside the protocol stack. The application layer is only the interface for the software. (Kaario 2002, 21.)

Table 1: Layers of the OSI model in Finnish and in English (Kaario 2002, 18).

Sovelluskerros	Application layer
Esitystapakerros	Presentation layer
Istuntokerros	Session layer
Kuljetuskerros	Transport layer
Verkkokerros	Network layer
Siirtokerros	Link layer
Fyysinen kerros	Physical layer

4.4 TCP/IP model

The TCP/IP protocol suite is linked in some way to every OSI model layers mentioned before. It basically has five layers: a physical layer, a link layer, a network layer, a transport layer and an application layer. The physical layer's and link layer's functions are not directly defined but they can consist of many various protocol layers. Actually the TCP/IP protocols work on the network and the upper layers. This is why the physical layer and the link layer are put into the same box on Table 2, in which the relationship of the TCP/IP model and the OSI model can be seen. (Kaario 2002, 22.)

Table 2: The relationship of the TCP/IP and the OSI model (Kaario 2002, 22).

OSI	**TCP/IP**
Application layer	Application layer
Presentation layer	
Session layer	
Transport layer	Transport layer
Network layer	Network layer
Link layer	Link and physical layer
Physical layer	

For the transport layer, the TCP protocol can be used. Also the UDP (User Datagram protocol) could be used, which is similar to TCP but simpler. However, the UDP will not be discussed in this report. The software that uses the TCP protocol sees the network as logical connections. To create this type of logical connections, sockets are needed. Sockets are presented in chapter 4.5. (Kaario 2002, 22.)

4.5 Sockets

Socket is a pair formed by an IP address and a TCP port number. For example, the IP number could be 160.221.224.74 and the port number 23. With these two numbers, the software that uses the network can be recognized. Without the port number, only the computer that uses the network would be known and the TCP/IP stack could not transmit the data to the right software. The port numbers are 16-bit integer numbers, so possible port numbers are 0-65535. Some of the numbers are reserved for specific protocols. (Kaario 2002, 194.)

In order that the communication between two computers or programs would be possible, a socket pair is needed. This means IP addresses and port numbers of both ends. (Kaario 2002, 194.) The connection between two computers using sockets is expressed in Figure 8.

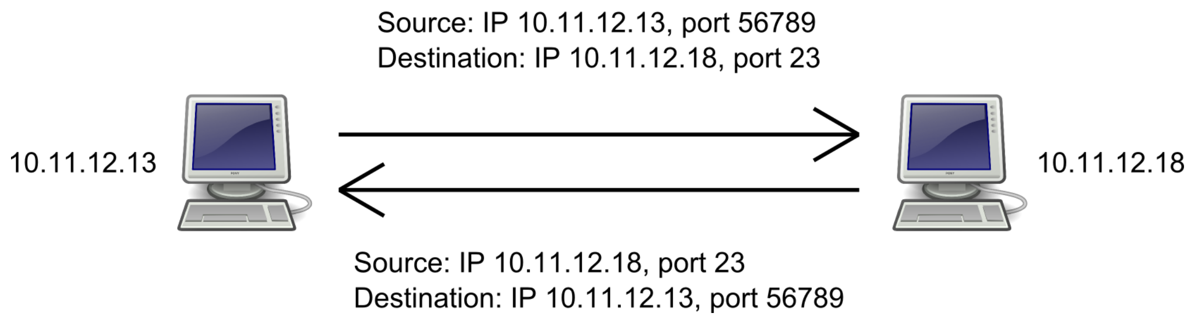

Source: IP 10.11.12.13, port 56789
Destination: IP 10.11.12.18, port 23
10.11.12.13
10.11.12.18
Source: IP 10.11.12.18, port 23
Destination: IP 10.11.12.13, port 56789

Figure 8: Connection between two computers with sockets (Kaario 2002, 194).

5 DEVELOPMENT OF THE LEARNING ENVIRONMENT

The objective of the Bachelor’s thesis was to develop and engineer a learning environment for Exertus OY at Seinäjoki University of Applied Sciences. The tasks in the thesis were the development and implementation of the equipment that the learning environment will contain and the creation of instructions for the Exertus PC software in the learning environment. In this chapter the development of the learning environment is discussed.

5.1 Steps

In the beginning it was necessary to get acquainted with the Exertus software and hardware. Getting familiarized with Guitu software was the first task. After a while of studying the Guitu, it was time to take a look into the Exertus products and the Canto 2 software. The software and the hardware are presented later in chapter 5.

A simple system was built up with a CDC1700S display module, a module testing device, a HCM4000 Hybrid Sensor and a CH8S Hub module. The other devices were connected to the CAN hub and the hub was connected to a PC with a USB CAN adapter. With this system it was possible to learn how to create control functions to the Exertus modules and graphical user interfaces to the Exertus display modules. It was also possible to study the configuration and diagnostics of a CAN bus with the system and the Canto 2 software.

After being familiarized with the Exertus control systems, it was time to think of the products and software that would be included to the learning environment. At this point also some instructions for the use of Canto 2 and Guitu were made.

There was also a PC simulation of the HCM2000 module under programming during the development of the learning environment. It was thought that some kind of a virtual version of the module testing device for the HCM2000PCSimulation would be useful. Designing of the HCM2000VirtualIOBox was started and it eventually became a big part of the development of the learning environment and this Bachelor’s thesis, for it was decided that it and the HCM2000PCSimulation would be also added to the learning environment. The implementation of the HCM2000VirtualIOBox is discussed in chapter 6.

5.2 Software

The software choices were simple. The two main programs of the software on the learning environment are the Canto 2 and Guitu software. ExMeBus Server has also an important role on the learning environment. HCM2000 PC Simulation and HCM2000VirtualIOBox were finished during the development of the learning environment and they were also included to it. In chapters 5.2.1-5.2.4 the software are briefly presented and also their functions on the learning environment are discussed.

5.2.1 ExMeBusServer

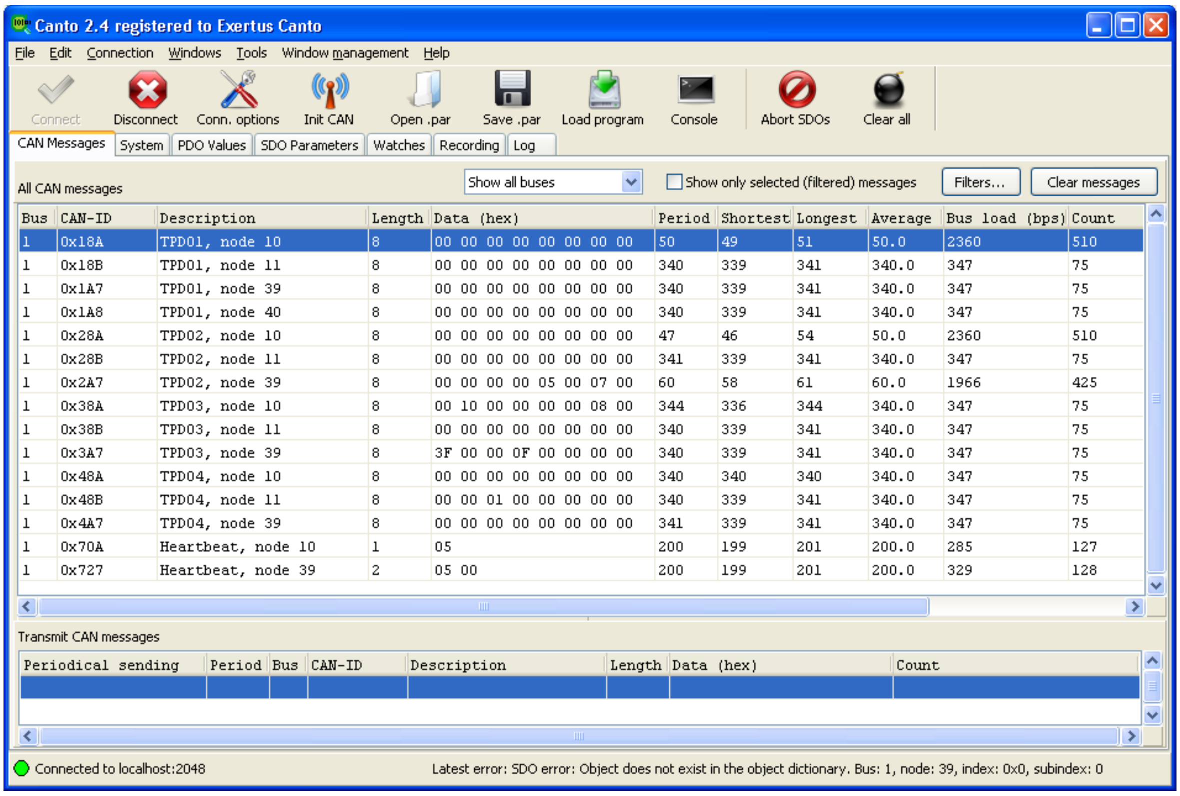
ExMeBusServer (Exertus Message Bus Server) has an important role on the learning environment. All communication between the Exertus client programs is transmitted through ExMeBusServer. The ExMeBusServer works like a hub for the messages between Exertus software. The programs send their messages to the ExMeBusServer and it distributes the messages to all programs on the server. The traffic sent via the ExMeBus is normal TCP/IP packet traffic.

5.2.2 Canto 2

Canto 2 (Controller Area Network Tool version 2) software is a CAN bus configuration and maintenance tool developed and owned by Exertus OY. Canto 2 can be used to control, analyze and record traffic on a CAN bus, configure system parameters and load programs to the Exertus modules. With Canto 2 software it is possible to monitor traffic on any CAN bus based on CAN 2.0A / 2.0B standard.

By Canto 2 it is possible to learn how to configure and analyze a CAN bus. CAN messages are led from the CAN bus to Canto 2 by connecting the CAN bus with a

USB CAN adapter to the PC, wherein the ExMeBusServer transmits the message forward to the Canto 2.



Bus	CAN-ID	Description	Length	Data (hex)	Period	Shortest	Longest	Average	Bus load (bps)	Count
1	0x18A	TPD01, node 10	8	00 00 00 00 00 00 00 00	50	49	51	50.0	2360	510
1	0x18B	TPD01, node 11	8	00 00 00 00 00 00 00 00	340	339	341	340.0	347	75
1	0x1A7	TPD01, node 39	8	00 00 00 00 00 00 00 00	340	339	341	340.0	347	75
1	0x1A8	TPD01, node 40	8	00 00 00 00 00 00 00 00	340	339	341	340.0	347	75
1	0x28A	TPD02, node 10	8	00 00 00 00 00 00 00 00	47	46	54	50.0	2360	510
1	0x28B	TPD02, node 11	8	00 00 00 00 00 00 00 00	341	339	341	340.0	347	75
1	0x2A7	TPD02, node 39	8	00 00 00 00 05 00 07 00	60	58	61	60.0	1966	425
1	0x38A	TPD03, node 10	8	00 10 00 00 00 00 08 00	344	336	344	340.0	347	75
1	0x38B	TPD03, node 11	8	00 00 00 00 00 00 00 00	340	339	341	340.0	347	75
1	0x3A7	TPD03, node 39	8	3F 00 00 0F 00 00 00 00	340	339	341	340.0	347	75
1	0x48A	TPD04, node 10	8	00 00 00 00 00 00 00 00	340	340	340	340.0	347	75
1	0x48B	TPD04, node 11	8	00 00 01 00 00 00 00 00	340	339	341	340.0	347	75
1	0x4A7	TPD04, node 39	8	00 00 00 00 00 00 00 00	341	339	341	340.0	347	75
1	0x70A	Heartbeat, node 10	1	05	200	199	201	200.0	285	127
1	0x727	Heartbeat, node 39	2	05 00	200	199	201	200.0	329	128

Periodical sending	Period	Bus	CAN-ID	Description	Length	Data (hex)	Count

Connected to localhost:2048

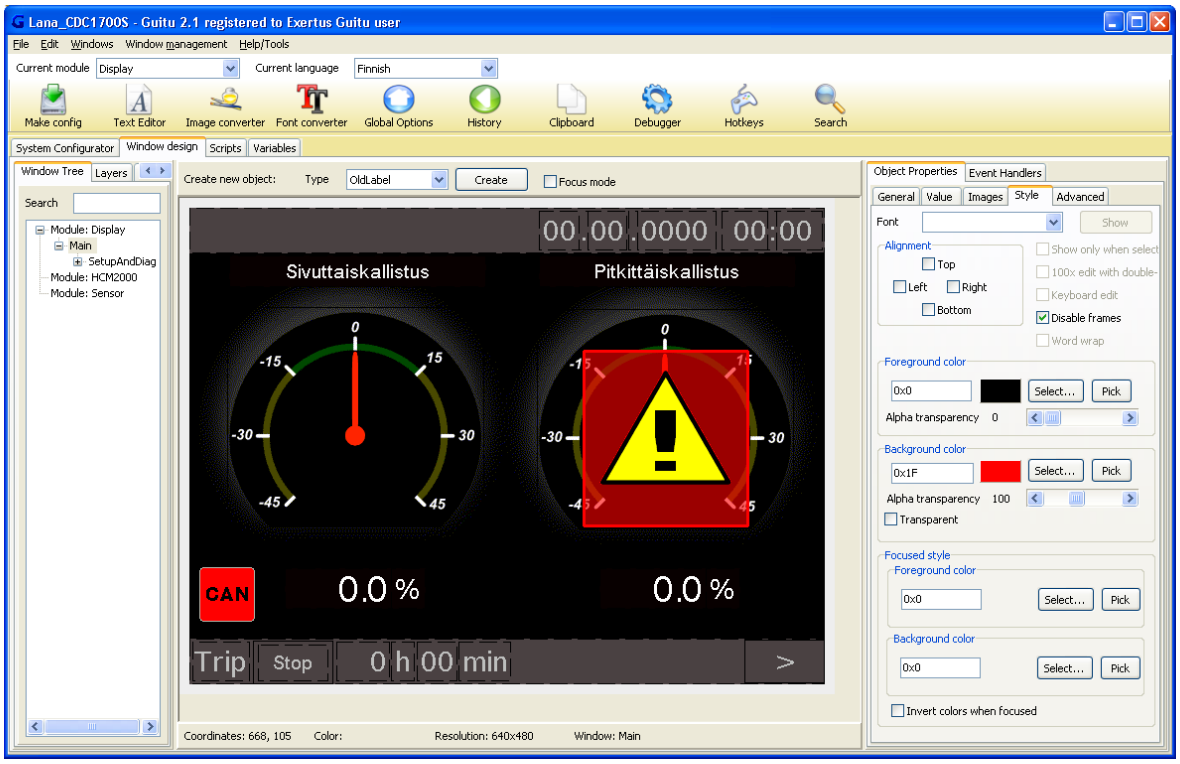
Latest error: SDO error: Object does not exist in the object dictionary. Bus: 1, node: 39, index: 0x0, subindex: 0

Figure 9: Canto 2 software (Exertus. 2013a.)

5.2.3 Guitu

Guitu software is a Graphical User Interface tool implemented and owned by Exertus OY. With Guitu, user can create GUIs for the Exertus display devices. Besides GUIs, the user can also create control functions to the Exertus modules with a programming language that is very similar to IEC 61131-3 Function Block Diagram (FBD) language.

Learning to design and produce graphical user interfaces and control functions to the Exertus modules is the main idea on the learning environment. The Guitu software is easy to learn to use. The I/O modules contain ready programmed functions. With them and the FBD programming, many kinds of control functions can be made with little coding.



Current module: Display
Current language: Finnish
Type: OldLabel
Focus mode: ☐

Alignment: ☐ Top ☐ Left ☐ Right ☐ Bottom
☐ Show only when select
☐ 100x edit with double-
☐ Keyboard edit
☑ Disable frames
☐ Word wrap

Foreground color: 0x0
Alpha transparency: 0
Background color: 0x1F
Alpha transparency: 100
☐ Transparent

Focused style
Foreground color: 0x0
Background color: 0x0
☐ Invert colors when focused

Coordinates: 668, 105 Color: Resolution: 640x480 Window: Main

Figure 10: Guitu software. (Exertus. 2013a.)

5.2.4 HCM2000PCSimulation and HCM2000VirtualIOBox

HCM2000PCSimulation. HCM2000PCSimulation is the virtual version of the HCM2000 I/O module. It has the same functionalities as the physical module. On the GUI of the program, user can set the values for the inputs of the virtual module. The virtual module is very useful for the user who is creating a control system with Guitu. With the HCM2000PCSimulation and the HCM2000VirtualIOBox, the program that the user has made, can be tested virtually.

HCM2000VirtualIOBox. HCM2000VirtualIOBox is made to be used with the HCM2000 virtual module. With HCM2000VirtualIOBox, the user can control the analog, digital and pulse inputs of the virtual module and it also simulates the outputs of the virtual module. The designing process and functionality of the program is told more precisely in chapter 6.

5.2.5 The software and the ExMeBusServer

The programs are connected to the ExMeBusServer using the socket technique presented in the theory part of this report. A program sends its messages to the ExMeBus in TCP/IP packet traffic. The ExMeBusServer then forwards the messages to all other programs on the server. This is expressed in Figure 11.

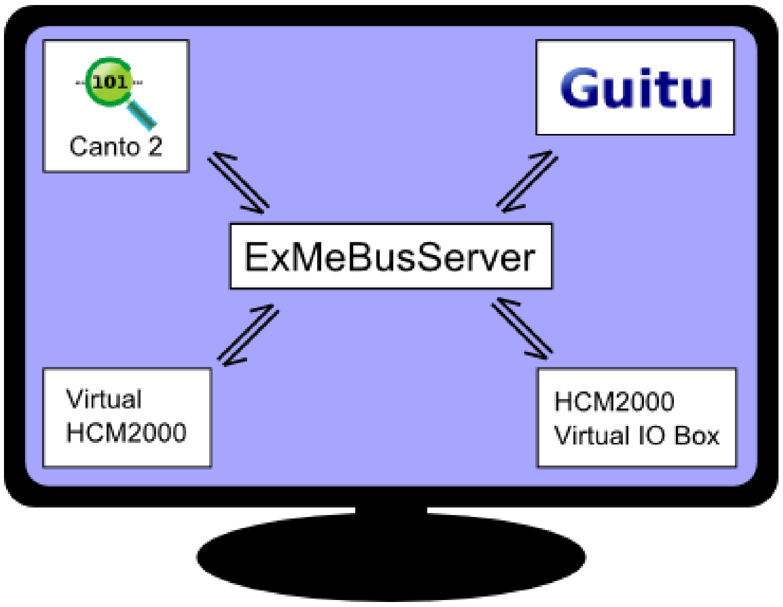


Figure 11: The software and the ExMeBusServer.

An example of the messaging on the ExMeBusServer: an input is enabled in the HCM2000VirtualIOBox. The HCM2000VirtualIOBox sends a message to the ExMeBus when the input is enabled. The ExMeBusServer receives the message and forwards it to the HCM2000 virtual module which sets the controlled input enabled. The virtual module then starts transmitting a CAN message of the state of the input to the ExMeBus. The Canto 2 receives the CAN message and the user can see that the input is enabled.

5.3 Equipment

The hardware is relevant in the learning environment. Choosing the equipment required some deliberation. There were few possibilities in products that were considered when the choices were done. It was sure that there would be a display module on the learning environment and there were two options for it. In addition an I/O module was added even though it was at first thought that it is not necessary. In chapters 5.3.1-5.3.3 the selected equipment are presented.

5.3.1 Compact Display Controller CDC1700S

CDC1700S is the display module that was selected to be used on the learning environment. It is a combined display and I/O controller module. It can, therefore, be used also alone or it can be connected to a bigger control system via a CAN interface.

Features of the CDC1700S module

- USB host port
- 5,7'' VGA color display, resolution 640 x 480
- 11 Digital inputs
- 8 Analog inputs (control with either voltage or current)
- 4 Digital outputs
- 8 Proportional PWM outputs
- Every output and analog input can also be used as a digital input meaning there are total of 31 PNP inputs (Exertus. 2013b).

There was another option for the display device with a smaller display and other different features. CDC1700S was selected because the bigger display was considered better when thinking of educational possibilities. More complex GUIs can be made for it. Also the USB host port is very useful in the learning environment because the program made by the user can be uploaded to the module by using an USB stick. It is not, therefore, necessary that the module is connected to a PC if the user wants to test his program.

The model of CDC1700S that is used has a control knob that can be used to navigate in the user interface. GUIs and control functions to the module can be created with the Guitu software.

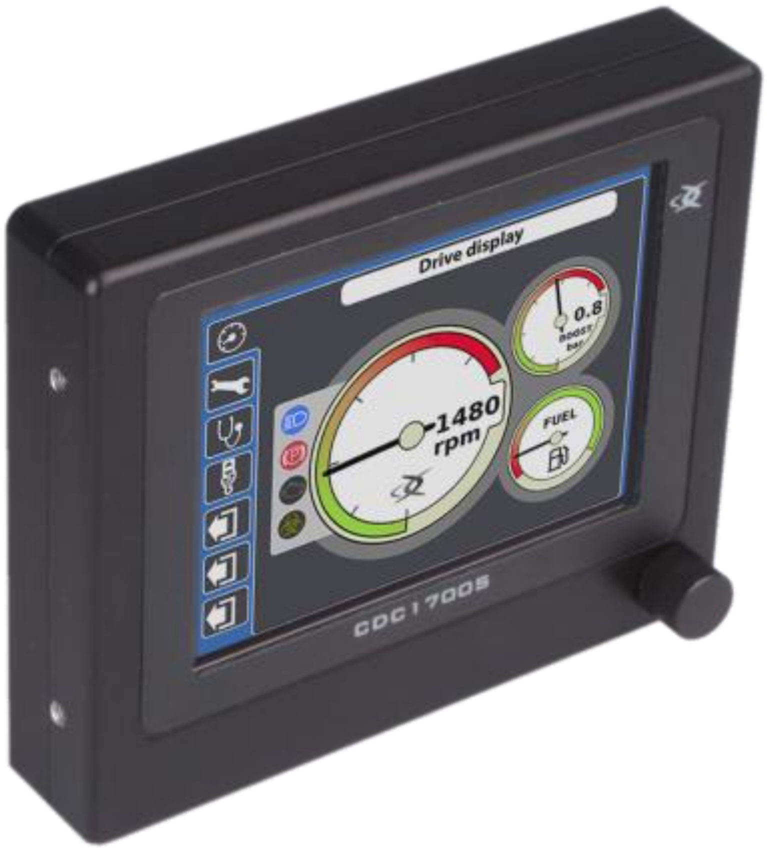


Figure 12: Compact Controller Module CDC1700S. (Exertus. 2013a.)

5.3.2 Hybrid Controller Module HCM2000S

HCM2000S is the I/O module that was chosen to be added to the learning environment. HCM2000S has water- and dustproof housing (IP67). The module has two separate ISO 11898 CAN interfaces and it is CANopen compatible. It can work as stand-alone or it can be connected to a larger CAN based control system.

Inputs and outputs of the HCM2000S module

- 8 Digital / frequency inputs
- 8 Analog inputs
- 12 Bit A/D converter
- 8 Digital outputs
- 24 Digital / proportional PWM outputs
- Every output and analog input can also be used as a digital input meaning there are total of 48 PNP inputs (Exertus. 2013c).

Figure 13: Hybrid Controller Module HCM2000S. (Exertus. 2013a.)

5.3.3 Other equipment

CAN hub. In order that all the devices can be connected to the system, a CAN hub is needed. Exertus CAN Hub CH8S is used as the hub. The CAN bus and power can be distributed up to 8 modules. The CH8S has eight M12 connectors and a power connector and 11 LEDs for diagnostic information.

Technical information of the CH8S module

- Dust and water proof IP67 housing
- Four M5 screw holes for mounting
- Eight M12/5 CAN connectors and one M12/4 connector for supply
- One LED for every CAN connector for module power indication
- Three LEDs for CAN bus traffic diagnostics (Exertus. 2013d).

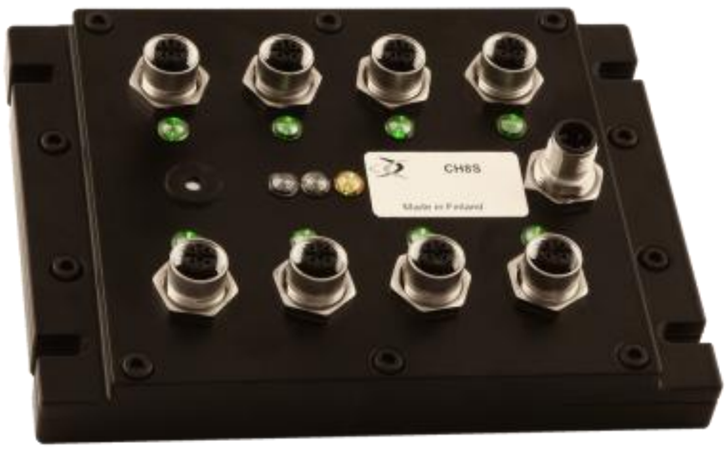


Figure 14: Can Hub CH8S. (Exertus. 2013a.)

USB CAN adapter. The system is to be connected to a PC with a USB CAN adapter. The manufacturer of the adapter that will be used is the German PEAK. The CAN bus is connected to the PC with the adapter from one of the M12 connectors of the CAN hub to the USB port in PC. The PEAK USB CAN adapter drivers must also be installed to the PC, to which the system is connected.

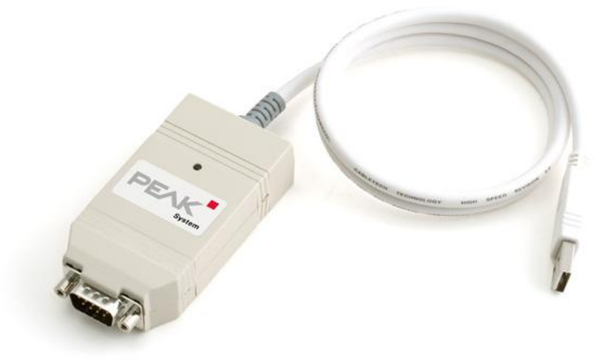


Figure 15: PEAK PCAN-USB adapter (PEAK-System. 2013.)

5.4 The system

The hardware that builds up the physical part of the system on the learning environment are the CDC1700S, HCM2000S and CAN Hub CH8S modules and the PEAK USB CAN adapter. The CAN bus, to which the modules are connected to, is connected to a PC with the USB CAN adapter. In the PC, the software that form the virtual part of the system, are the Canto 2, Guitu, ExMeBusServer, HCM2000PCSimulation and HCM2000VirtualIOBox software. The structure of the system is illustrated in Figure 16.

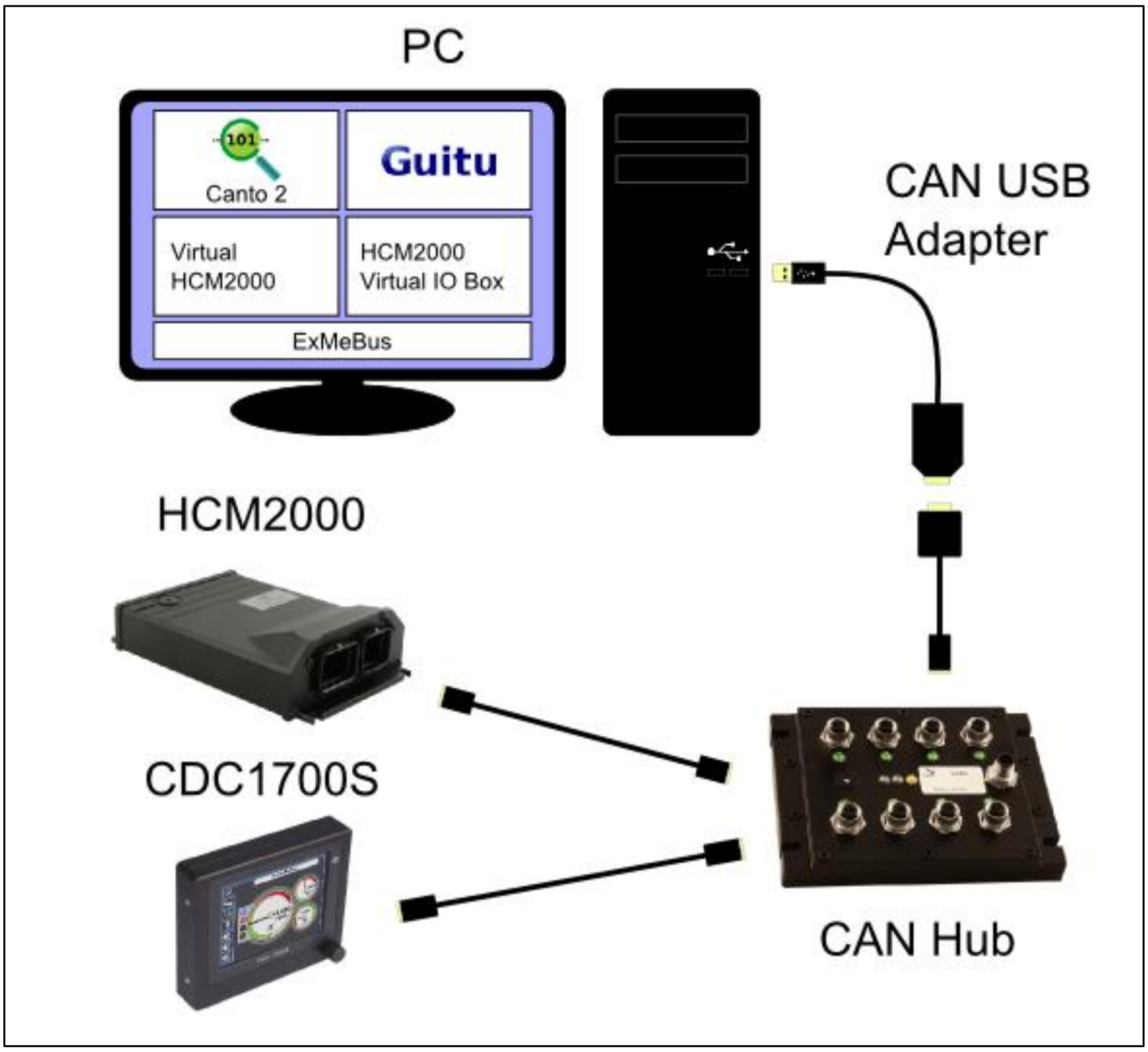


Figure 16: The planned system for the learning environment

5.5 Instructions

To create and aggregate instructions for the installation and use of the Exertus software was also part of the bachelor thesis. In this chapter it is briefly told of the making of the instructions and what kind of things they contain.

5.5.1 Instructions for the use of Canto 2 and Guitu

After some time of studying the Canto 2 and Guitu software, instructions for the use of them was made. At first, two PowerPoint presentations were made of the basic things of both Canto 2 and Guitu software, and they became more like introductions to the programs. In those two presentations it was told what kind of features the software has and how to get started with them.

After the introduction presentations were ready, a presentation with step by step instructions for creating simple GUIs and control scripts with Guitu was made. Also in the manual of Guitu, there are useful step by step instructions for different situations. The manuals for both Canto 2 and Guitu are available for the user. They are included in the installation packet of the programs.

5.5.2 Instructions for the installation of the software

The installations of the Exertus software to computers in Seinäjoki UAS will be done by the IT support of the university. The software is easy to install but it was considered that simple installation instructions would be a good thing. There are also instructions for the installation in Canto 2’s and Guitu’s manuals but it’s easier to read all instructions from the same document.

A test installation for the software was done for one computer in the university in the end of the development of the learning environment. The installations succeeded without any problems. After the installations, the software was tested and also they worked as they should. After the tests were done, the instructions for the installations were written.

6 SIMULATOR

A big part of the development of the learning environment was the programming of the simulation software. With the software, the inputs of a HCM2000 module can be controlled and also the outputs of the module are simulated. In this chapter the process of designing and programming of the simulation software is discussed.

6.1 Starting point

Exertus has produced also I/O module testing devices for different kind of module types. They have been used by the programmers for testing the modules and control systems. Now the intention is that the testing for a control system during programming could be made, more or less, only by PC. This way of testing would be faster and there would be no need for building the system with physical modules.

There had already been designed and implemented PC simulations of the HCM2000 and CDC1700 –modules. There was, therefore, also a need for some kind of virtual version for the I/O module testing device. It was decided to start designing a virtual I/O testing device for the HCM2000 module. With the help of the PC simulations of HCM2000 and CDC1700 –modules and the virtual I/O testing device, programmers could test their creations also on the learning environment.

There had been made a simple example program that had functions for connecting to the ExMeBusServer and for initialization of a CAN adapter. It was decided to use this program as a base for the new program. The new program was named HCM2000VirtualIOBox. In this report HCM2000VirtualIOBox is referred to also with the name “simulator”.

6.2 Programming language and programming tool

The example program that was used as a base for the simulator was programmed in C++ programming language as many other Exertus software. It was, therefore, logical to use C++ as a programming language in the simulator as well. It could have also been possible to use some other programming language but it was reasonable to use C++ because of the possibility to re-use already existing codes which provided easier and faster programming. The programming tool used for programming the simulator was Embarcadero C++ Builder XE3. This tool had been already in use in the company.

6.3 Design of the layout

Few possibilities were considered while designing the appearance of the simulator. The layout of the physical I/O module testing device was assessed good and clear. The users of the simulator would also be already familiar with the layout if it was the same. It was, therefore, decided to design the appearance of the simulator using the testing device as a model. The finished simulator looks almost the same as the physical version. See the appendix 1 for the layout of the simulator.

The switches, LEDs, potentiometers, labels and the black background are designed using the Inkscape software which is an open-source vector graphics editor. The sketching was quite simple using that editor and also the use of the models was a big advantage.

6.4 Features

The simulator was intended to be easy to use. That was paid attention to while designing the simulator. In this chapter the features of the simulator are shortly presented.

6.4.1 IO Box tab

There are four different tabs on the simulator. The virtual I/O box is on the first tab which is named IO Box. The objects and features of the I/O box are presented below.

Inputs. The switches on the I/O box can be switched simply by clicking them with a mouse. Every switch works as an input for the virtual module. The visual layout of the switches can be seen in Figure 17.

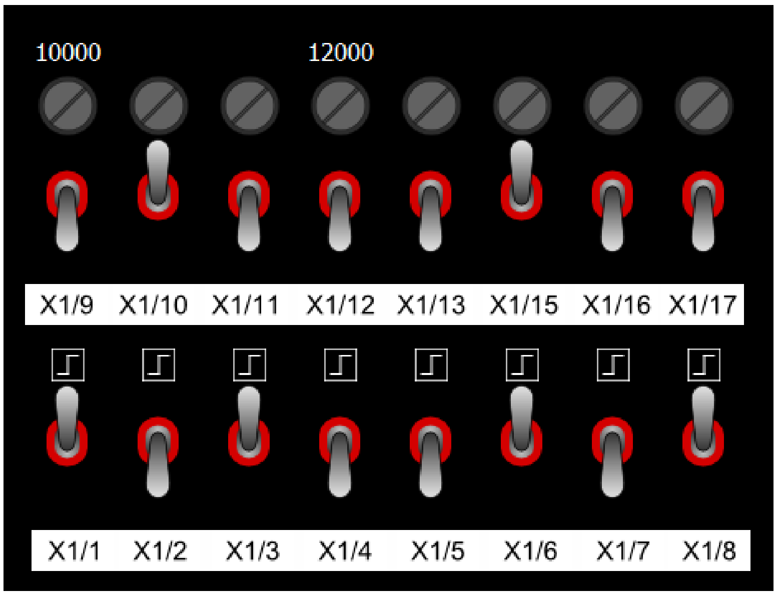


Figure 17: Switches on the IO Box.

Outputs. The outputs of the virtual module are also simulated on the simulator. Above in the X2-connector’s switches there are LEDs that work as outputs. The outputs are enabled when the switches are in the lower position as they are also in the physical testing device. The percent values below the LEDs tell the PWM ratios of the outputs. Over the LEDs there are shown the pair pin current measurement values that come from the virtual module. In Figure 18, there are a few outputs enabled.

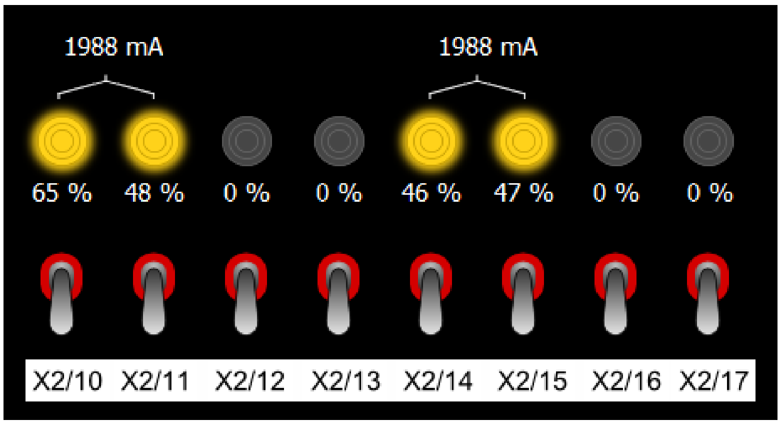


Figure 18: Outputs on the IO Box.

Analog channels. The screw-looking round objects up on the box image potentiometers. Clicking them opens a small panel with a slide bar. This slide bar can be used to set the value for an analog channel of the virtual module. A Different analog channel can be adjusted with each potentiometer. If the value of the channel is not zero, the value is shown over the potentiometers. For core temperature and the supply voltage of the virtual module there are separate buttons on the box. Clicking them opens a small panel with an edit box to which the user can input a value for the analog channel. The analog channel adjustment slide bar is shown in Figure 19.

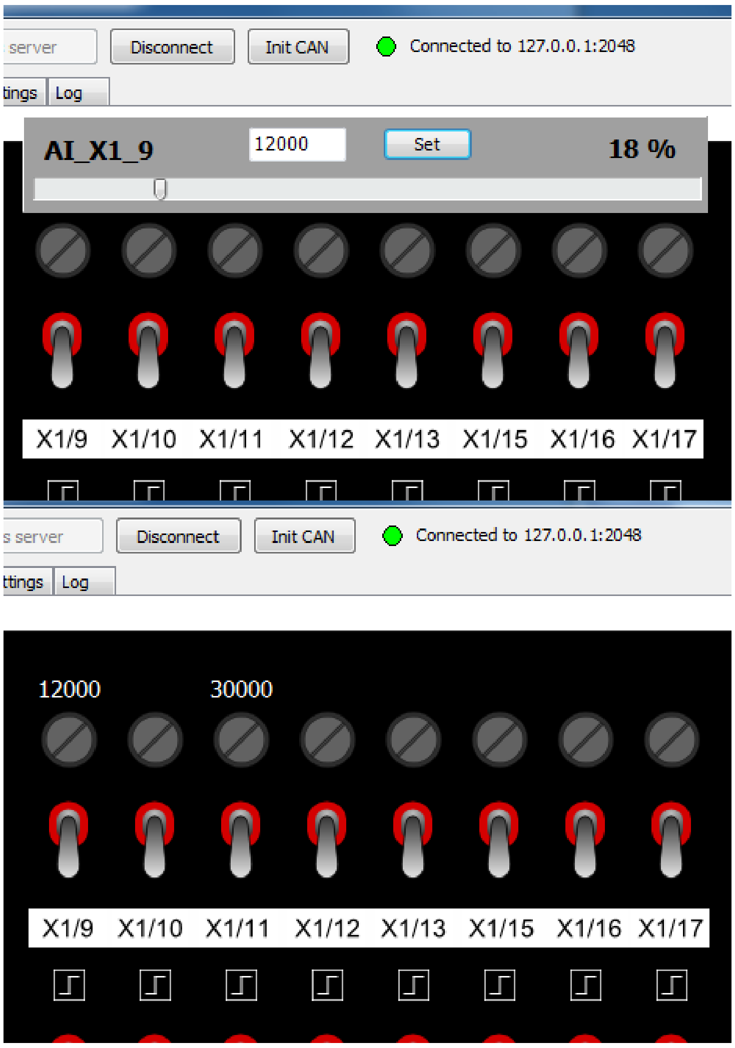


Figure 19: Analog channel adjustment.

Pulse inputs. The pulse inputs of the virtual module can be controlled from the panel which is opened when the pulse buttons on the I/O box are pressed. The panel is shown in Figure 20.

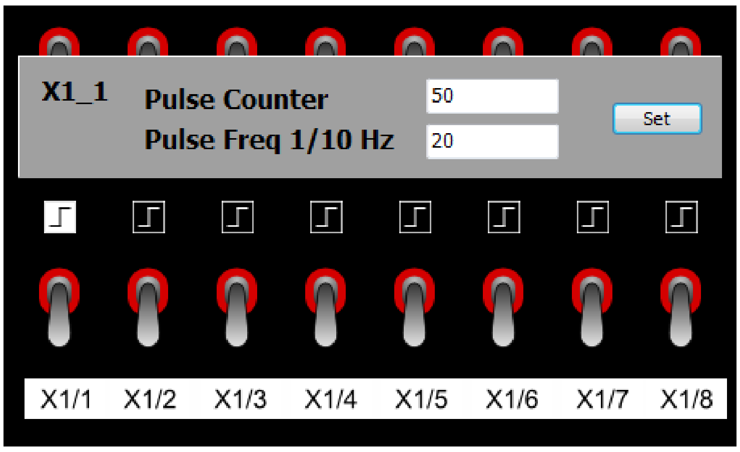
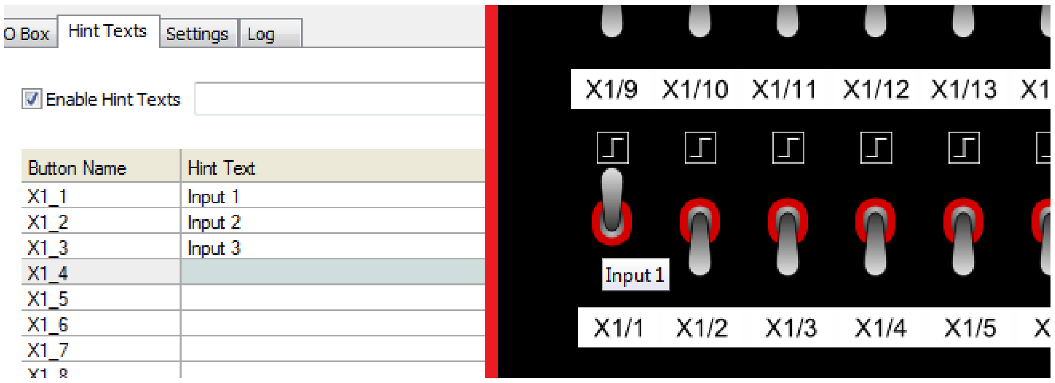


Figure 20: The panel for controlling the pulse inputs of the virtual module.

6.4.2 Hint Texts Tab

On the second tab there is a grid to which the user can write hint texts for the switches on the I/O box. Once the hint texts are enabled by a checkbox and the mouse cursor is moved on a switch, the hint text is shown. This is a useful feature for the user for it may be hard to remember what functions each input has. The hint texts can also be saved into a text file and loaded from there. The structure is clear in the text file. On each row there is one pin name and hint text. This allows the user to write and edit the hint texts also from the text file. The hint text feature can be seen in use in Figure 21.



[✓ Enable Hint Texts]

Button Name	Hint Text
X1_1	Input 1
X1_2	Input 2
X1_3	Input 3
X1_4	
X1_5	
X1_6	
X1_7	

Figure 21: Hint texts written to the grid and shown on the I/O box.

6.4.3 Settings and Log Tabs

On the third tab there are a few settings concerning the simulator. On the last tab there is a log. Information on the events that occur in the simulator is printed to the log. The log can also be disabled.

6.5 Programming

The programming language was already familiar from a previous project done in the company so it did not cause so many difficulties anymore. Some functions required some searching for information on how they should be used. Some new features came also up on the programming tool during programming but they were cleared with the help of Embarcadero online Documentation Wiki.

Some ideas were taken from the previous project and they were refined. Like in the previous project, the programming in this project was also done in the way that the code could be efficiently re-used and this was shown in many versatile and efficient functions. Another version of the simulator has actually already been made for other virtual module type.

6.6 Usage

The HCM2000VirtualIOBox can be used by the user to test the control functions he has done with Guitu software. Using the virtual I/O box speeds up the testing because there is no more need to build up the system with physical devices. Everything cannot yet be tested virtually but during the programming process the HCM2000VirtualIOBox is a great aid for the programmer. The HCM2000VirtualIOBox may be developed further in the future to provide more features to the user.

7 SUMMARY

The goal of this thesis was to develop and implement a learning environment for Exertus OY at Seinäjoki University of Applied Sciences. The idea of the learning environment was that within it, the user can practice the programming of Exertus I/O controllers, the development of graphical user interfaces (GUI) and the configuration and diagnostics of a CAN bus. The learning environment was supposed to include Exertus I/O controller modules as hardware and Exertus software.

This goal was achieved quite well. The hardware was selected and it will be brought to Seinäjoki UAS during the summer 2013. The software selections were also clear and they will be installed to the computers at the university also during the summer 2013, because the university will get some new computers during the summer.

During the development of the learning environment an idea of a new simulator software came up. This software was intended to help the programmer at the testing of made control systems. It was decided that the simulator will also be part of the learning environment and eventually it became a big part of the development of the learning environment. The simulator was finished in time. A test installation for the software was made to one computer at the university and the functionality was tested. The software was working properly.

One task was also to gather useful and sufficient instructions for the use of the Exertus PC software. The instructions for the use of the software were created in an early stage. They could be better but there was not enough time to improve them during the thesis work. Instructions to the installation of the software were made after the test installation. They are simple step-by-step instructions and easy to follow.

BIBLIOGRAPHY

Alanen, J. 2000. CAN – ajoneuvojen ja koneiden sisäinen paikallliväylä. [PDF file]. Tampere: VTT Automaatio. [Ref. 24 April 2013]. Available at: http://www.oamk.fi/~eeroko/Opetus/Ohjausjarjestelmat/CAN/CAN-perusteet_AlasenMateriaalia.pdf

CAN in Automation (CiA). 2013a. CAN protocol. [Web page]. CAN in Automation (CiA). [Ref. 25 April 2013]. Available at: http://www.can-cia.org/index.php?id=systemdesign-can-protocol

CAN in Automation (CiA). 2013b. About CAN in Automation (CiA). [Web page]. CAN in Automation (CiA). [Ref. 25 April 2013]. Available at: http://www.can-cia.org/index.php?id=aboutcia

Comer, D. E. 2002. TCP/IP. Translator Erkki Suominen. Jyväskylä: IT Press.

Exertus OY. 2013a. Company official homepage. [Website]. Exertus OY. [Ref. 9 April 2013]. Available at: http://www.exertus.fi/

Exertus OY. 2013b. CDC1700S datasheet. [Online publication]. Exertus OY. [Ref. 21 April 2013]. Available at: http://www.exertus.fi/files/Tiedostot/CDC1700S_flyer.pdf

Exertus OY. 2013c. HCM2000S datasheet. [Online publication]. Exertus OY. [Ref. 21 April 2013]. Available at: http://www.exertus.fi/files/Tiedostot/HCM2000S_flyer.pdf

Exertus OY. 2013d. CH8S datasheet. [Online publication]. Exertus OY. [Ref. 23 April 2013]. Available at: http://www.exertus.fi/files/Tiedostot/CH8S_DataSheet.pdf

Juhala, M., Lehtinen, A., Suominen, M. & Tammi, K. 2005. Moottorialan sähköoppi (8th renewed edition). Jyväskylä: Autoalan koulutuskeskus OY.

Kaario, K. 2002. TCP/IP-verkot. Porvoo: Docendo Finland Oy.

PEAK-System. 2013. Products, PCAN-USB. [Web page]. PEAK-System. [Ref. 22 April 2013]. Available at: http://www.peak-system.com/PCAN-USB.199.0.html?&L=1

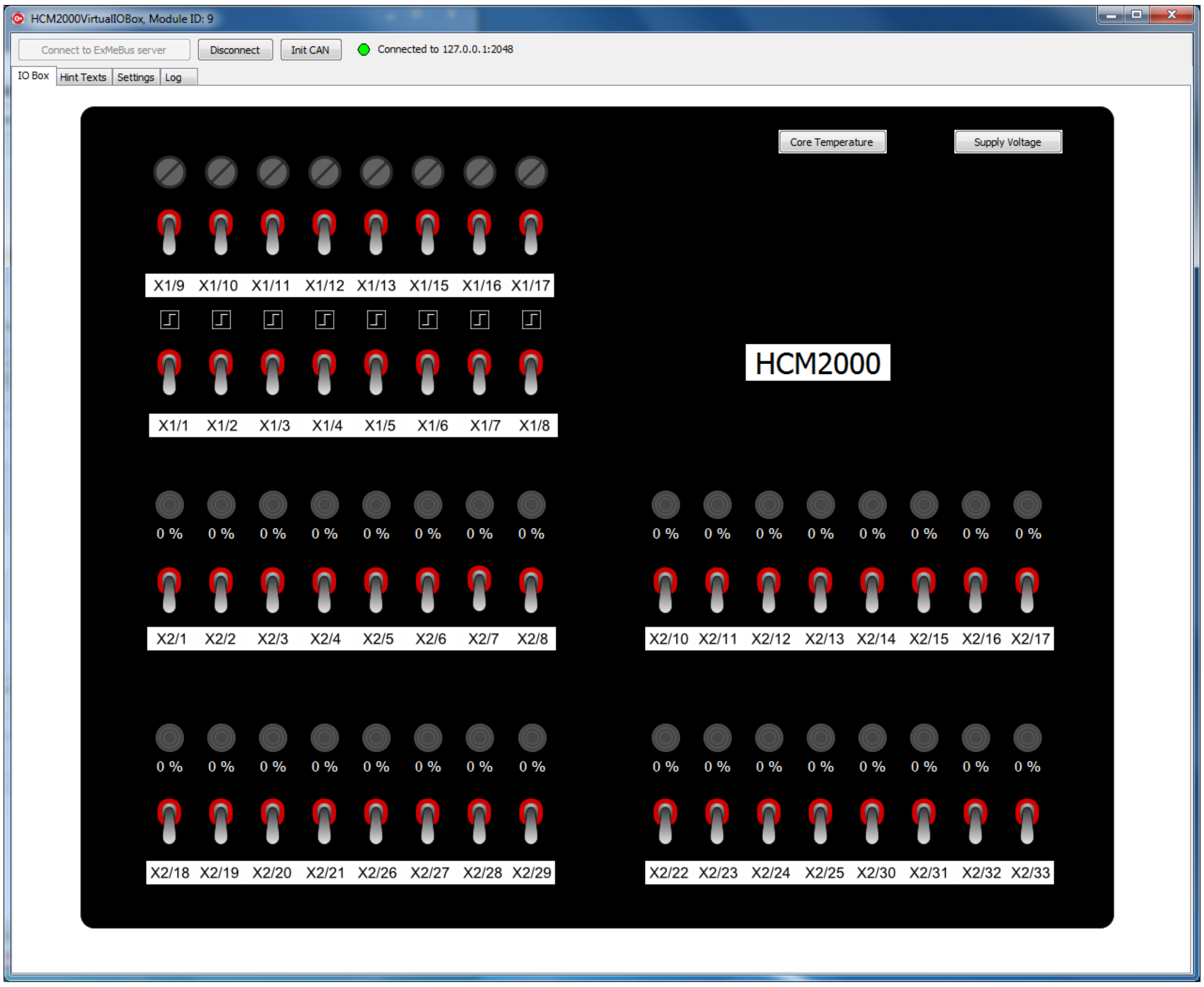
Saha, H. 2006. CANopen perusteet. [Online publication]. FLUID Finland 1 - 2006. [Ref. 25 April 2013]. Available at: http://www.canopen.fi/artikkelit/CANopen.pdf

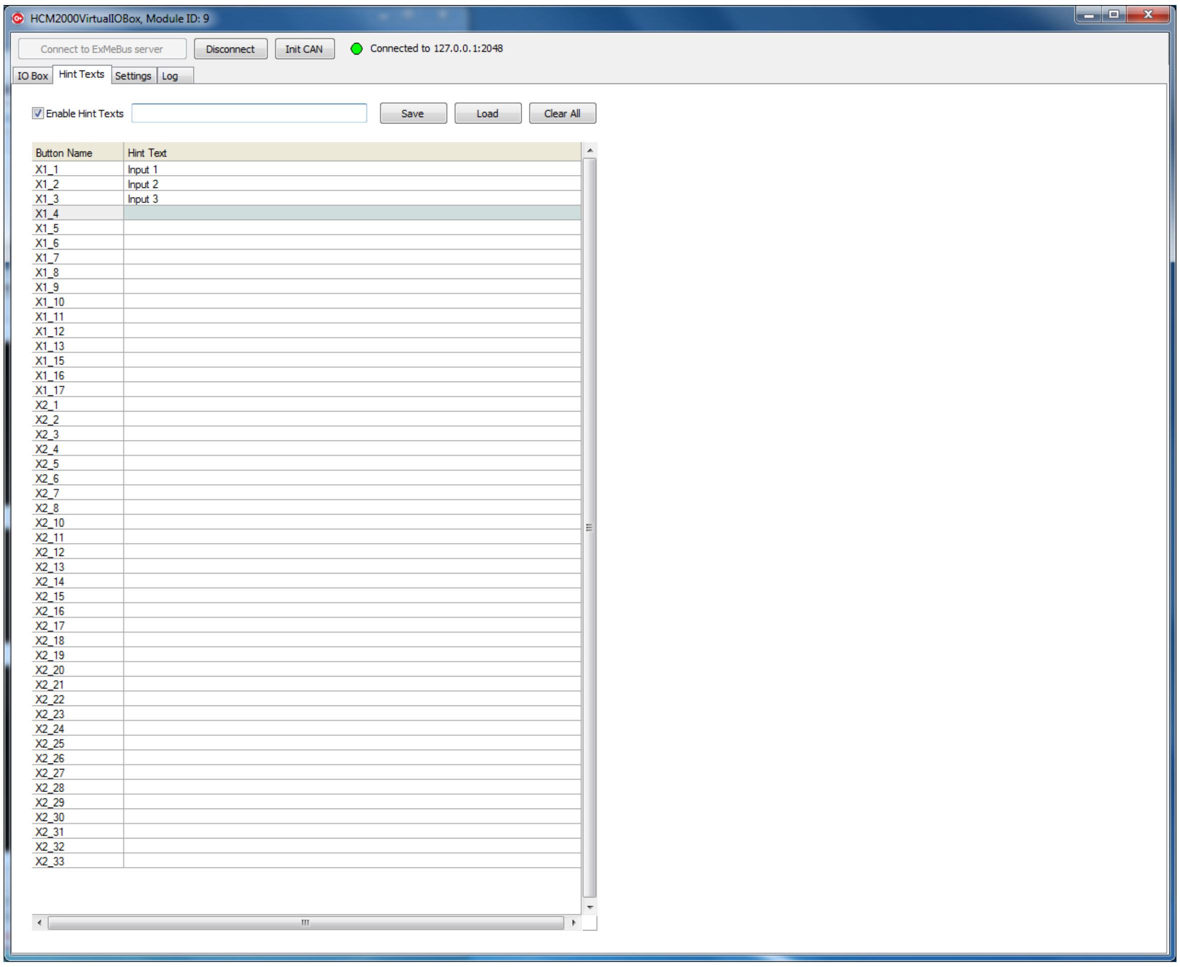
Vainio, M. & Hakala, M. 2005. Tietoverkon rakentaminen. Porvoo: Docendo Finland Oy.

APPENDIXES

APPENDIX 1. The layout of the HCM2000VirtualIOBox

APPENDIX 1

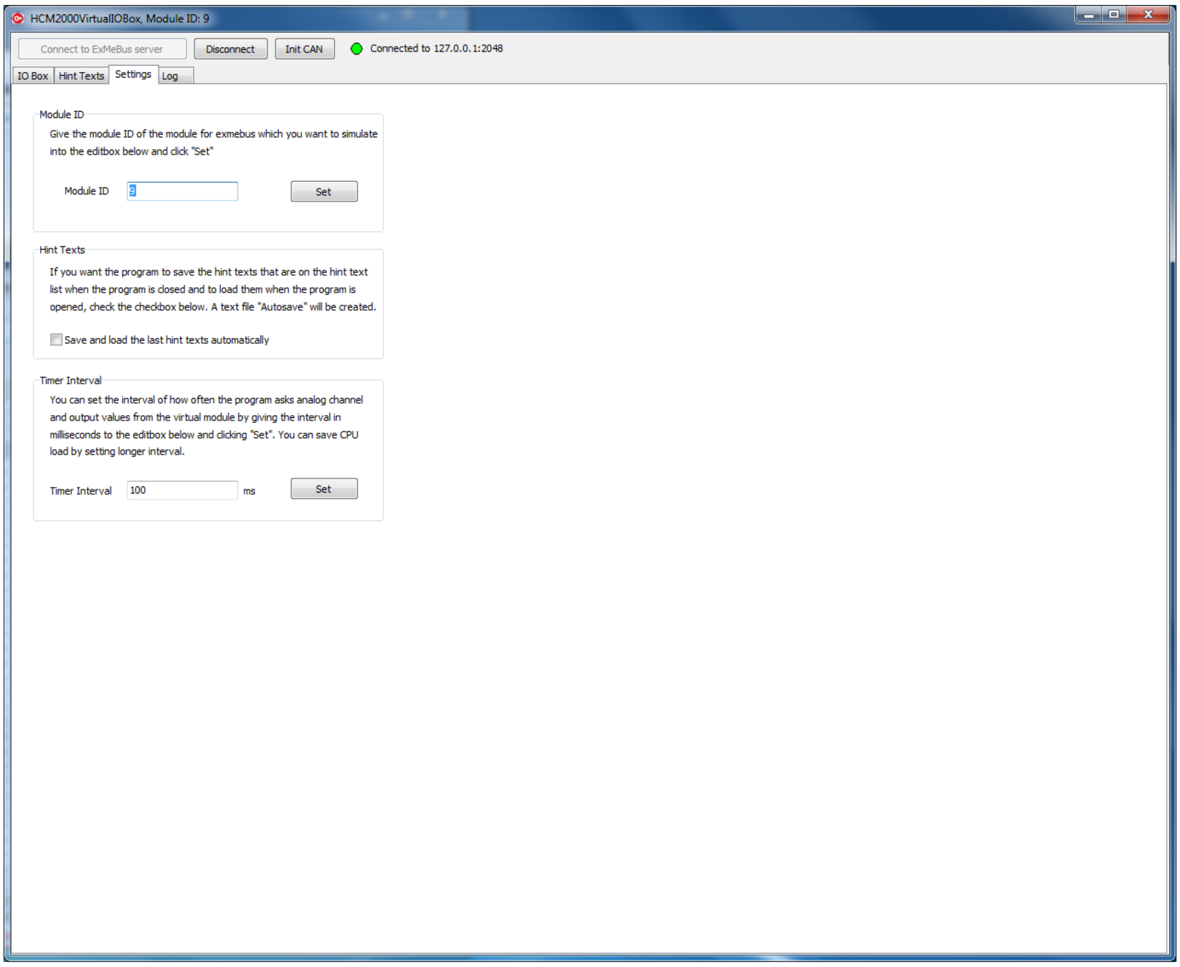

HCM2000VirtualIOBox, Module ID: 9
Connect to ExMeBus server
Disconnect
Init CAN
Connected to 127.0.0.1:2048
IO Box
Hint Texts
Settings
Log
Core Temperature
Supply Voltage
HCM2000
X1/9 X1/10 X1/11 X1/12 X1/13 X1/15 X1/16 X1/17
X1/1 X1/2 X1/3 X1/4 X1/5 X1/6 X1/7 X1/8
0 % 0 % 0 % 0 % 0 % 0 % 0 % 0 %
X2/1 X2/2 X2/3 X2/4 X2/5 X2/6 X2/7 X2/8
0 % 0 % 0 % 0 % 0 % 0 % 0 % 0 %
X2/10 X2/11 X2/12 X2/13 X2/14 X2/15 X2/16 X2/17
0 % 0 % 0 % 0 % 0 % 0 % 0 % 0 %
X2/18 X2/19 X2/20 X2/21 X2/26 X2/27 X2/28 X2/29
0 % 0 % 0 % 0 % 0 % 0 % 0 % 0 %
X2/22 X2/23 X2/24 X2/25 X2/30 X2/31 X2/32 X2/33



Connected to 127.0.0.1:2048

☑ Enable Hint Texts

Button Name	Hint Text
X1_1	Input 1
X1_2	Input 2
X1_3	Input 3
X1_4	
X1_5	
X1_6	
X1_7	
X1_8	
X1_9	
X1_10	
X1_11	
X1_12	
X1_13	
X1_15	
X1_16	
X1_17	
X2_1	
X2_2	
X2_3	
X2_4	
X2_5	
X2_6	
X2_7	
X2_8	
X2_10	
X2_11	
X2_12	
X2_13	
X2_14	
X2_15	
X2_16	
X2_17	
X2_18	
X2_19	
X2_20	
X2_21	
X2_22	
X2_23	
X2_24	
X2_25	
X2_26	
X2_27	
X2_28	
X2_29	
X2_30	
X2_31	
X2_32	
X2_33	



Connected to 127.0.0.1:2048

Module ID

Give the module ID of the module for exmebus which you want to simulate into the editbox below and click "Set"

Module ID 9

Hint Texts

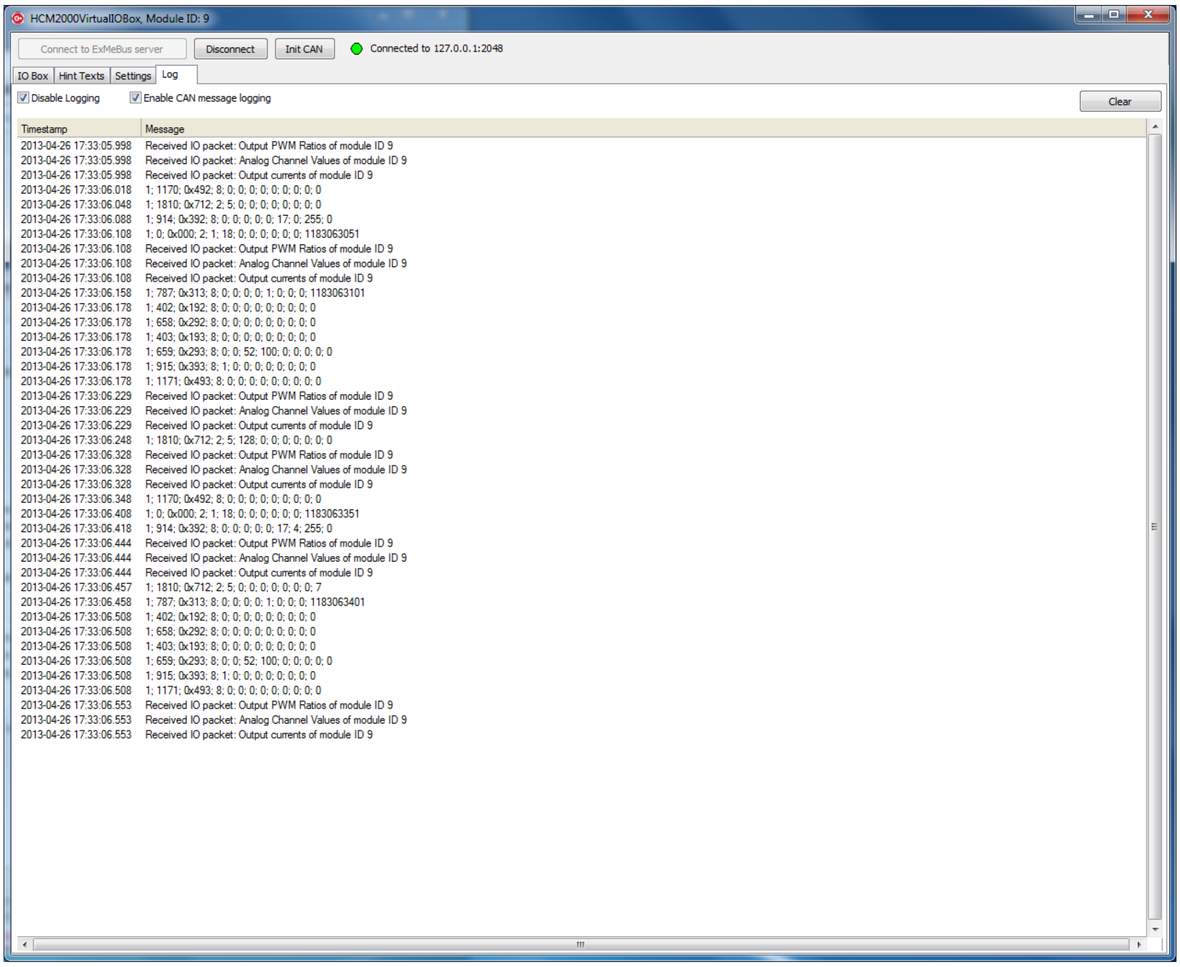
If you want the program to save the hint texts that are on the hint text list when the program is closed and to load them when the program is opened, check the checkbox below. A text file "Autosave" will be created.

☐ Save and load the last hint texts automatically

Timer Interval

You can set the interval of how often the program asks analog channel and output values from the virtual module by giving the interval in milliseconds to the editbox below and clicking "Set". You can save CPU load by setting longer interval.

Timer Interval 100 ms


HCM2000VirtualIOBox, Module ID: 9
Connect to ExMeBus server
Disconnect
Init CAN
Connected to 127.0.0.1:2048
IO Box
Hint Texts
Settings
Log
Clear

[✓] Disable Logging   [✓] Enable CAN message logging

```
Timestamp                Message
2013-04-26 17:33:05.998  Received IO packet: Output PWM Ratios of module ID 9
2013-04-26 17:33:05.998  Received IO packet: Analog Channel Values of module ID 9
2013-04-26 17:33:05.998  Received IO packet: Output currents of module ID 9
2013-04-26 17:33:06.018  1; 1170; 0x492; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.048  1; 1810; 0x712; 2; 5; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.088  1; 914; 0x392; 8; 0; 0; 0; 0; 0; 17; 0; 255; 0
2013-04-26 17:33:06.108  1; 0; 0x000; 2; 1; 18; 0; 0; 0; 0; 0; 0; 1183063051
2013-04-26 17:33:06.108  Received IO packet: Output PWM Ratios of module ID 9
2013-04-26 17:33:06.108  Received IO packet: Analog Channel Values of module ID 9
2013-04-26 17:33:06.108  Received IO packet: Output currents of module ID 9
2013-04-26 17:33:06.158  1; 787; 0x313; 8; 0; 0; 0; 0; 1; 0; 0; 0; 1183063101
2013-04-26 17:33:06.178  1; 402; 0x192; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.178  1; 658; 0x292; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.178  1; 403; 0x193; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.178  1; 659; 0x293; 8; 0; 0; 52; 100; 0; 0; 0; 0; 0
2013-04-26 17:33:06.178  1; 915; 0x393; 8; 1; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.178  1; 1171; 0x493; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.229  Received IO packet: Output PWM Ratios of module ID 9
2013-04-26 17:33:06.229  Received IO packet: Analog Channel Values of module ID 9
2013-04-26 17:33:06.229  Received IO packet: Output currents of module ID 9
2013-04-26 17:33:06.248  1; 1810; 0x712; 2; 5; 128; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.328  Received IO packet: Output PWM Ratios of module ID 9
2013-04-26 17:33:06.328  Received IO packet: Analog Channel Values of module ID 9
2013-04-26 17:33:06.328  Received IO packet: Output currents of module ID 9
2013-04-26 17:33:06.348  1; 1170; 0x492; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.408  1; 0; 0x000; 2; 1; 18; 0; 0; 0; 0; 0; 0; 1183063351
2013-04-26 17:33:06.418  1; 914; 0x392; 8; 0; 0; 0; 0; 0; 17; 4; 255; 0
2013-04-26 17:33:06.444  Received IO packet: Output PWM Ratios of module ID 9
2013-04-26 17:33:06.444  Received IO packet: Analog Channel Values of module ID 9
2013-04-26 17:33:06.444  Received IO packet: Output currents of module ID 9
2013-04-26 17:33:06.457  1; 1810; 0x712; 2; 5; 0; 0; 0; 0; 0; 0; 7
2013-04-26 17:33:06.458  1; 787; 0x313; 8; 0; 0; 0; 0; 1; 0; 0; 0; 1183063401
2013-04-26 17:33:06.508  1; 402; 0x192; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.508  1; 658; 0x292; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.508  1; 403; 0x193; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.508  1; 659; 0x293; 8; 0; 0; 52; 100; 0; 0; 0; 0; 0
2013-04-26 17:33:06.508  1; 915; 0x393; 8; 1; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.508  1; 1171; 0x493; 8; 0; 0; 0; 0; 0; 0; 0; 0
2013-04-26 17:33:06.553  Received IO packet: Output PWM Ratios of module ID 9
2013-04-26 17:33:06.553  Received IO packet: Analog Channel Values of module ID 9
2013-04-26 17:33:06.553  Received IO packet: Output currents of module ID 9
```