CAN-bus project

Using the CANbus Toolset™ software and the SELECONTROL® MAS automation system

Student: Dipl-Ing. (FH) Joerg Rett
Supervisor: Dr. Ian Fletcher
Graduation: July 2001
Acknowledgements

This report represents the work at the Control Systems Centre, School of Computing, Engineering and Technology, University of Sunderland done between 1. March and the 22. of June of the year 2001.

I want to thank my supervisor Dr. Ian Fletcher for his valuable guidance and advice. I also want to thank Prof. Chris Cox for giving me the possibility to work on this project.
Table of contents

1 INTRODUCTION........................................................................................................................1

2 CAN AND CANOPEN..................................................................................................................2
  2.1 Frame Format ..........................................................................................................................3
  2.2 CANopen objects and the object dictionary ............................................................................4
  2.3 Service Data Objects ..............................................................................................................4
      2.3.1 Download .......................................................................................................................5
      2.3.2 Upload ............................................................................................................................6
      2.3.3 SDO configuration through the Object Dictionary .......................................................6
  2.4 Process Data Objects ............................................................................................................6
      2.4.1 PDO mapping ..................................................................................................................6
      2.4.2 PDO communication parameters ..................................................................................7
  2.5 Synchronisation object (SYNC) ...........................................................................................8
  2.6 NMT Module Control Services ...........................................................................................8
  2.7 Pre-defined Connection Set ..................................................................................................10

3 SELECONTROL MAS SYSTEM .............................................................................................11
  3.1 I/O device DDC 711 ..............................................................................................................12
      3.1.1 Default-Mapping ..........................................................................................................13
      3.1.2 Parameterisation via SDO ............................................................................................14
  3.2 Analogue input module AIT 701 ............................................................................................14
      3.2.1 Default-Mapping ..........................................................................................................15
      3.2.2 Parameterisation via SDO ............................................................................................15
  3.3 Analogue output module AOT 701 ........................................................................................17
      3.3.1 Default-Mapping ..........................................................................................................17
      3.3.2 Parameterisation via SDO ............................................................................................18

4 CANBUS TOOLSET ..................................................................................................................19
  4.1 Canmenu ...............................................................................................................................19
  4.2 Simulink blocks ....................................................................................................................21
      4.2.1 Model Control Block .....................................................................................................22
      4.2.2 From CAN_1 block .......................................................................................................23
      4.2.3 To CAN_1 block ............................................................................................................23
| 4.3 | Simulation parameters | 24 |
| 5 | I/O TESTING | 25 |
| 5.1 | Canmenu configuration | 26 |
| 5.1.1 | NMT Module Control | 26 |
| 5.1.2 | Synchronisation Object | 27 |
| 5.1.3 | PDO1 (receive) | 28 |
| 5.1.4 | PDO2 (receive) | 29 |
| 5.1.5 | SDO (receive) | 30 |
| 5.2 | Simulink models | 31 |
| 5.2.1 | DDC711_BootUp model | 31 |
| 5.2.2 | DDC711_SDO model | 32 |
| 5.2.3 | DDC711 model | 33 |
| 5.3 | MATLAB procedures | 35 |
| 5.4 | Message timing | 36 |
| 6 | AIR HEATING PROCESS | 38 |
| 6.1 | Process parameters | 39 |
| 6.1.1 | Static curve of the sensor | 39 |
| 6.1.2 | Step response | 40 |
| 6.2 | Process model | 40 |
| 6.2.1 | Continuous time model | 40 |
| 6.2.2 | Hybrid model | 42 |
| 6.3 | PI controlled process | 43 |
| 6.3.1 | Control structure | 43 |
| 6.3.2 | System behaviour | 44 |
| 6.4 | Fuzzy controlled process | 45 |
| 6.4.1 | Control structure | 45 |
| 6.4.2 | Fuzzy Interference System (FIS) | 46 |
| 6.4.3 | System behaviour | 47 |
| 7 | COUPLED TANKS PROCESS | 48 |
| 7.1 | Process parameters | 49 |
| 7.1.1 | Static curves of the sensors | 49 |
| 7.1.2 | Modified Astrom test | 52 |
| 7.2 | SISO control of a MIMO process | 53 |
| 7.2.1 | Control system structure | 53 |
| 7.2.2 | Step 1 | 54 |
| 7.2.3 | Step 2 | 54 |
8 CONCLUSIONS

A MATLAB M-FILES

A.1 Project 1: I/O Testing
A.1.1 Start.m
A.1.2 Nmtstart.m
A.1.3 Rxpdo2on.m
A.1.4 CheckRxpdo2on.m
A.1.5 Txpdo2on.m
A.1.6 CheckTxpdo2on.m
A.1.7 TxpdoSync.m
A.1.8 CheckTxpdoSync.m

B MAPPING TABLES

C LITERATURE
1 Introduction

Recently the School of Computing, Engineering and Technology, University of Sunderland investigated Local Area Networks (Fieldbuses) to select one for establishing a network for the benefit of research, undergraduate, postgraduate studies and industrial projects [7]. After the CANbus had been selected the vendors and suppliers of devices were investigated and chosen.

The aim of this project is to run the CAN-bus network with the CANbus Toolset software. The devices that were selected are modules of the SELECONTROL® MAS automation system from SIG Positec Systems GmbH. So the main task is to implement a system based on the CANbus Toolset and the SELECONTROL® MAS modules.

The following chapter will explain the fundamentals of the Controller Area Network (CAN) which are significant for this project. Chapter 3 will deal with the SELECONTROL® MAS modules. Showing how process data and parameters are transferred to and from the modules. Chapter 4 explains the CANbus Toolset software that runs under the MATLAB environment. It shows how data can be written to and read from the CAN-bus.

Chapter 5 will set up and test a system consisting of the a PC, CANbus and I/O modules. Its main purpose will be to test input and output of the signals. The following chapter 6 will show the connection of a real process to the modules and its control through Simulink. The basic structure of the system can be seen in Figure 1-1.

![Figure 1-1: Parts of the system](image-url)
2 CAN and CANopen

Since 1994/95, CAN, has been the most accepted protocol for automobile applications. When the machinery automation industries identified CAN as a powerful communication protocol for machine control, a number of supplementary specifications were soon added to make CAN an “Open System” communication protocol.

Some important CAN related system features include:

- Multimaster node hierarchy
- CSMA/CD+CR media access technique
- Event driven communication
- Broadcast
- Each message generically designates the information, not the node
- Remote request response
- Acknowledge
- Bit synchronisation
- Inter node synchronisation
- Number of Nodes/System: 127
- Amount of Data/Message: 0 to 8 Byte
- Gross Maximum Length/Message (Standard): 117 bits
- Bit rate: 5kBit/s to 1Mbit/s
- Bit rate/Bus Length Relation: e.g. 40m at 1 Mbit/s
2.1 Frame Format

The CAN message consists of a certain number of bits that are divided into fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>SOF</th>
<th>Arbitration</th>
<th>Control</th>
<th>Data</th>
<th>CRC</th>
<th>ACK</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0...64</td>
<td>15</td>
</tr>
<tr>
<td>Value</td>
<td>0</td>
<td>0...2031</td>
<td>0</td>
<td>0</td>
<td>0...8</td>
<td>x</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2-1: Frame Format of CAN 2.0-Part A

**SOF:** (Start Of Frame): This field determines the beginning of the frame. Its length is 1 bit, and its value is always zero (dominant bit).

**Arbitration:** The length of the arbitration field is 12 bits and consist of the fields IDENTIFIER (11 bits) and RTR (1 bit). The IDENTIFIER determines the priority of the data frame to be sent. The RTR (Remote Transmission Request) determines which service shall be used. Its length is 1 bit and its value is always to be dominant ("0"-bit) if the frame contains data and recessive ("1"-bit) for a Remote frame. Remote frames are used to request a data transmission from a node.

**IDE:** The one bit IDE (Identifier Extension) determines the CAN type. Its value is dominant for Standard CAN and recessive for Extended CAN.

**r0:** r0 is a reserved bit for future developments. It is always to be sent dominant. But receivers will accept a recessive value too.

**DLC:** The four bit DLC (Data Length Code) determines the length of the data field in bytes. So its value is normally in the range of (0...8). Values ranging from 9 to 15 may be used for application-specific purposes.

**Data:** This field consists of the main data. Its length is in the range of (0...64) bits determined by the DLC content.

**CRC:** This field includes the information necessary for data security.

**ACK:** This field (Acknowledgement) is used to check if any other nodes except for the transmitter have received the message without any errors. The A_S is the so-called Acknowledge-Slot. The transmitter sends a recessive bit at this point and the receiver, which has detected the CRC, sequence as correct sends a dominant bit. So the transmitter knows that at least one receiver has detected the message as correct. The A_D is the so-called ACK-Delimiter. It determines the end of the Acknowledge field by sending a recessive bit.

**EOF:** This (End of Frame) field consists of seven recessive bits.
2.2 CANopen objects and the object dictionary

CANopen uses an object-oriented approach to the definition of standard devices: each device is represented as a set of objects that can be accessed through the network. Every aspect of the operation of a device is mapped onto one or more of these objects. It is therefore possible to change the configuration or the status of a device simply by using the network to alter the attributes of a particular object within it.

There are two basic types of objects:

- Communication objects
- Application objects

All objects in a device can be accessed through an Object Dictionary where they are represented. The important type for this project is the Communication Object type.

The Communication Objects can be divided into two general categories:

- Communication Objects for Network Management
- Communication Objects for Application Data Transfer

In the first category you can find the Communication objects for the boot up. The second category is further divided into two categories: Service Data Objects (SDOs) and Process Data Objects (PDOs). Both of these can be used to transfer data between the devices that support them.

Additionally another Communication Object should be mentioned which is the Synchronisation Object (SYNC) to synchronise the operation of CANopen devices.

2.3 Service Data Objects

Service Data Objects are used to establish Client/Server relationships between two CANopen devices whereby the Client device has read and write access to the Object dictionary of the Server device.

SDOs carry Index and Sub-index information, to allow object addressing in the Object Dictionary. Index and Sub-index together are called Multiplexor. Two CAN message identifiers must be allocated to an SDO. One of them is used for messages sent by the Client to the Server and the other one for messages sent in the opposite direction. The two CANopen services used for SDO transfers are called SDO Download and SDO Upload and they can be used for writing and reading from the Object dictionary respectively.

Depending on the amount of data that has to be transferred, the transfer can follow different procedures. For this project a procedure for small pieces of data (Expedited Transfer) is the most suitable. This transfer is used for data not greater than 4 bytes in length. When the Client wishes to access the Server’s Object Dictionary, it uses either an Initiate SDO Download or an Initiate SDO Upload message.
2.3.1 Download

The SDO Download service can be used by the Client to write to the Server’s Object Dictionary.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data</td>
<td>Sub-index</td>
<td>Index</td>
<td>S</td>
</tr>
<tr>
<td>Bit</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td>1h</td>
</tr>
</tbody>
</table>

Table 2-2: Client request

<table>
<thead>
<tr>
<th>Field</th>
<th>Byte 0</th>
<th>Bytes 1 ... 2</th>
<th>Byte 3</th>
<th>Bytes 4 ... 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCS</td>
<td>X</td>
<td>Index</td>
<td>Sub-index</td>
</tr>
<tr>
<td>Bit</td>
<td>7...5</td>
<td>4...0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>3h</td>
<td>0h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-3: Server response

**CCS:** The CCs (Client Command Specifier) bit field, stored in bits 5 to 7 of byte 0, is equal to 1 indicating the Initiate SDO Download command.

**X:** The X bits are unused bits and should always be set to zero.

**N:** The N field determines the size of the data held in bytes 4 to 7 of the CAN data field. In fact, this field indicates the number of bytes in the telegram that do not contain data. Its value can range from 0 to 3, and will indicate that bytes 8-N to 7 will be empty. Note that a null value N means that all bytes contain data. Additionally, since the maximum value of N is 3, at least one byte will always have to be transferred.

**E:** The E bit (bit 1 in data byte 0) indicates whether the transfer is Expedited (E = 1) or Segmented (E = 0).

**S:** The S bit indicates whether or not the size of the data is indicated in bytes 4 to 7 of the CAN message. If S is set to 0, then the size is not indicated.

**Index / Sub-index:** The Multiplexor is stored in byte 1 to 3 of the data field. The Index is coded as an UNSIGNED16 and the Sub-index is coded as an UNSIGNED8.

**Data:** The data is carried in byte positions 4 to 7.

**SCS:** The SCS (Server Command Specifier) contains the value 3, indicating a response to an Initiate SDO Download command (bits 5 to 7 of byte 0 in the response telegram).
2.3.2 Upload
The SDO Upload service can be used by the Client to read from the Server’s Object Dictionary.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reserved</td>
<td>Sub-index</td>
<td>Index</td>
<td>X</td>
</tr>
<tr>
<td>Bit</td>
<td></td>
<td></td>
<td></td>
<td>4 ... 0</td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td>0h</td>
</tr>
</tbody>
</table>

*Table 2.4: Client request*

<table>
<thead>
<tr>
<th>Field</th>
<th>Byte 0</th>
<th>Bytes 1 ... 2</th>
<th>Byte 3</th>
<th>Bytes 4 ... 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SCS</td>
<td>X</td>
<td>N</td>
<td>E</td>
</tr>
<tr>
<td>Bit</td>
<td>7 ... 5</td>
<td>4</td>
<td>3 ... 2</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td>2h</td>
<td>0h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 2.5: Sever response*

**CCS:** The CCS is equal to 2 indicating the Initiate SDO Upload request.

**SCS:** The SCS (Server Command Specifier) contains the value 2, indicating a response to an Initiate SDO Upload command.

2.3.3 SDO configuration through the Object Dictionary
Every Service Data Object that a device implements is represented in its Object Dictionary by an entry of type ‘SDO Communication Parameter’. By accessing these entries in a device’s Object Dictionary these Service Data Objects can be configured.

2.4 Process Data Objects
Process Data Objects provide direct access to Application Objects within a device. They are used to perform real-time transfers of short blocks of high priority data. The data transferred must be less than or equal 8 bytes in length. PDOs can be transmitted either synchronised or event controlled. PDOs are designed as ‘Stored events’.

2.4.1 PDO mapping
Via the mapping tables (index 1600h ... 1603h and 1A00h ... 1A05h) the objects are copied into the process data objects with real time characteristics. After switch on, the default mapping is available. Digital inputs/outputs are copied bitwise according to their sequence in the PDO. Analogue inputs/outputs are copied as 16 bit value according to their sequence.
2.4.2 PDO communication parameters

How the PDOs behave is described by the communication parameter (Index 1400 … 1403 and 1800 … 1805). The parameters can be set separately for each PDO.

<table>
<thead>
<tr>
<th>PDO identifier:</th>
<th>After switching on, the PDOs can be accessed by their default identifier that is deviated from the node address. The entry of PDO identifier in the object list allows the activation or the deactivation of the PDOs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission inhibit time:</td>
<td>The ‘Inhibit-time’ prevents an overload of the CAN bus by a module with a high priority message. It defines the minimal period time between two telegrams with equal identifiers.</td>
</tr>
<tr>
<td>Transmission mode:</td>
<td>The transmission mode defines the sending behaviour the corresponding PDO. It can be distinguished between event controlled and synchronous transmission. For event controlled transmissions, an event on the module triggers the transmission of a PDO, in synchronous mode, the PDO is transmitted after receipt of a synchronisation telegram (SYNC).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmission mode (decimal)</th>
<th>PDO transmissions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cyclic</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>1-24053</td>
<td>x</td>
</tr>
<tr>
<td>241-252</td>
<td>reserved</td>
</tr>
<tr>
<td>253</td>
<td></td>
</tr>
<tr>
<td>254</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td></td>
</tr>
</tbody>
</table>

Table 2-6: PDO transmission mode

- **Mode 0**: The output data of a received PDO are written to the output after the following synchronisation telegram. A telegram with input data is only sent if the input image is modified by a receiving SYNC signal.
- **Mode n=1 … 240**: The PDOs are transmitted at each n cycle.
- **Mode 253**: Despite an event no PDO will be transmitted.
- **Mode 254**: The transmission of a PDO is triggered via a manufacturer specific event on the module.
- **Mode 255**: The transmission of a PDO is triggered by an event indicated in the device profile (e.g. change of digital input).
2.5 Synchronisation object (SYNC)

The Synchronisation object is broadcast periodically on the network by a synchronisation device. This object is used to implement a global clock-tick type event in the network. Each device may or may not use this event to synchronise itself with other devices in the network. CANopen reserves message identifier 80h for the synchronisation Object which contains no data.

2.6 NMT Module Control Services

This object belongs to the group of Network Management (NMT) Communication Objects who are used to ensure that all devices in the network are able to communicate in the correct way. The NMT Communication Objects operate according to the Master/Slave principle (model). They are divided in the Module Control group and Error Control group.

The Module Control can be used for the initialisation of the NMT Slaves. The CAN message identifiers have to be assigned to the devices to be able to perform the communication. This assignment is normally done statically.

A CANopen NMT Slave operates according to a state diagram in which the state transitions are caused by invocations. The state diagram is shown in Figure 2-1.

![State diagram of a CANopen device](image)

<table>
<thead>
<tr>
<th>Transition</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Automatic</td>
</tr>
<tr>
<td>(2)</td>
<td>Start Remote Node</td>
</tr>
<tr>
<td>(3)</td>
<td>Enter Pre-operational State</td>
</tr>
<tr>
<td>(4)</td>
<td>Stop Pre-operational State</td>
</tr>
<tr>
<td>(5)</td>
<td>Reset Node</td>
</tr>
<tr>
<td>(6)</td>
<td>Reset Communication</td>
</tr>
</tbody>
</table>

On ‘Power-on’ a CANopen node enters an Initialisation phase (transition (1)). This Initialisation phase is conceptually divided into two states: Reset Application and Reset Communication. Passing through these states automatically, the node brings itself to its default status i.e. all the internal parameters of the device, such as communication parameters and all other Object Dictionary entries will be given their default values.
When the Initialisation phase is over, the node enters the Pre-operational state automatically (transition (1)). In Pre-operational state, communication via SDOs is allowed and can be used since at this stage each device in the network has its own set of default CAN message identifiers to communicate. These identifiers are derived from the Node ID parameter. PDOs are not allowed in Pre-operational state.

Once the device has been configured to handle all communication tasks, its normal operation can be started. The normal operation of a node is represented in the state diagram by the Operational state. In this state all the communication functionality of the node can be used e.g. PDOs, synchronisation using the Synchronisation Object, etc. The transition to the Operational state is represented in the state diagram by transition (2). This is triggered using the Start Remote Node service.

From Operational state, a device may be brought back to Pre-operational state, e.g. for additional configuration, by using the Enter Pre-operational State service. This transition is labelled (3) in the state diagram.

The Stopped state is included in the state diagram of a CANopen device to accommodate situations in which it is necessary to switch off all the communication functionality in a device. The transitions to the Stopped state, labelled (4) in the state diagram, are triggered through the Stop Remote Node service.

At any time, from any state, a node can be forced to reset itself. This reset procedure can be global i.e. the device will go through the same Initialisation process as on ‘Power on’, or it can apply only to the communication parameters of the node. The first type of reset is represented in the state diagram by the transition labelled (5). It can be triggered using the Reset Node service. The second type of reset is represented on the state diagram by the transition labelled (6) and can be triggered using the Reset Communication service.

The protocol for the Module Control NMT services is shown in Table 2-8.

<table>
<thead>
<tr>
<th>Field</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td></td>
<td>Node ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command Specifier (dec)</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start Remote Node</td>
</tr>
<tr>
<td>2</td>
<td>Stop Remote Node</td>
</tr>
<tr>
<td>128</td>
<td>Enter Pre-operational State</td>
</tr>
<tr>
<td>129</td>
<td>Reset Node</td>
</tr>
<tr>
<td>130</td>
<td>Reset Communication</td>
</tr>
</tbody>
</table>

Table 2-8: Protocol for NMT Module Control services

The CS field indicates a NMT Command Specifier that is used to choose between the different services. The Node ID field carries the Node ID of the node to which the message is intended. If all nodes are being simultaneously addressed then this field will contain zero. This is the reason why a device never can have a Node ID of 0.
2.7 Pre-defined Connection Set

To spare the systems integrator some of the work involved in building CANopen applications, particularly simple ones, a default scheme for the allocation of identifiers to devices is defined in the CANopen Communications Profile.

The CANopen default identifier allocation scheme can be thought of as a predefined Master/Slave connection set allowing the implementation of peer-to-peer communication between an Application Master device and the remaining Slave nodes without the need for an identifier distribution process.

This default configuration is available in each device after the Initialisation phase, when the device enters the Pre-operational state. It can then be partially changed using SDOs.

The default message identifiers that each Slave device will use for data exchange with the Application Master are composed of two functional parts (see Table 2-9).

<table>
<thead>
<tr>
<th>Field</th>
<th>IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Function Code</td>
</tr>
<tr>
<td>Length</td>
<td>4</td>
</tr>
</tbody>
</table>

Module (Node) ID range
CANopen: 1d – 127d

PDO1 (transmit) at node 1: 1h + 180h = 181h

PDO1 (transmit) at node 127: 7Fh + 180h = 1FFh

Table 2-9: Default message identifier allocation

<table>
<thead>
<tr>
<th>Object</th>
<th>Function Code (bin)</th>
<th>Resulting Identifiers (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMT Module Control</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>Synchronisation Object</td>
<td>0010</td>
<td>80</td>
</tr>
<tr>
<td>PDO1 (transmit)</td>
<td>0011</td>
<td>181 – 1FF</td>
</tr>
<tr>
<td>PDO1 (receive)</td>
<td>0100</td>
<td>201 – 27F</td>
</tr>
<tr>
<td>PDO2 (transmit)</td>
<td>0101</td>
<td>281 – 2FF</td>
</tr>
<tr>
<td>PDO2 (receive)</td>
<td>0110</td>
<td>301 – 37F</td>
</tr>
<tr>
<td>PDO3 (transmit)</td>
<td>0111</td>
<td>381 – 3FF</td>
</tr>
<tr>
<td>PDO3 (receive)</td>
<td>1000</td>
<td>401 – 47F</td>
</tr>
<tr>
<td>PDO4 (transmit)</td>
<td>1001</td>
<td>481 – 4FF</td>
</tr>
<tr>
<td>PDO4 (receive)</td>
<td>1010</td>
<td>501 – 57F</td>
</tr>
<tr>
<td>SDO (transmit)</td>
<td>1011</td>
<td>581 – 5FF</td>
</tr>
<tr>
<td>SDO (receive)</td>
<td>1100</td>
<td>601 – 67F</td>
</tr>
</tbody>
</table>

The Functional Code part of the identifier will determine its priority, according to the function it will be used for. The Module ID part of the identifier will permit different nodes using identifiers from the same functional group without causing conflicts. Each CANopen device is uniquely identified on the network by its Module ID.

The set of default identifiers allocates message identifiers to each device for eight Process Data Objects (four receive and four transmit PDOs) and one Service Data Object (two identifiers for receive and transmit) as also shown in Table 2-9. These objects are all for a peer-to-peer communication between Application Master and Slaves. In addition the set also reserves message identifiers for messages broadcasted on the bus like the NMT Module Control or the Synchronisation Object.
3 SELECONTROL MAS System

In the new automation system SELECONTROL®MAS control, measurement, regulation, optimisation, positioning, communication and networking are integrated within the system. The whole system can be built in a distributed manner. The input and output modules are designed to be mounted directly at the sensor/actuator work place. The decentralised input and output modules can be divided into two main categories:

- **Nodal modules**
  - DDC / DIOC: Digital input and output modules
  - DIC: Digital input modules
  - DOC: Digital output modules
  - AIC: Analogue input modules
  - AOC: Analogue Output modules

- **Extension modules**
  - DIT: Digital input modules
  - DOT: Digital output modules
  - AIT: Analogue input modules
  - AOT: Analogue output modules

Each digital nodal module can be equipped with up to 7 extension modules. The digital extension modules are provided with either 4 or 8 inputs or outputs and hence a maximum of 64 digital inputs or outputs can be provided at each network node.

As seen in Figure 3-1 the CAN bus is used as standard to link the nodal modules in the SELECONTROL®MAS automation system to the processor (PC, VME or PLC).

![Figure 3-1: CAN bus network](image)

Up to 63 modules can be connected to a central processor.
3.1 I/O device DDC 711
The DDC 711 is a bus node with digital I/Os and is extendable with digital and/or analogue I/Os. The node module itself is provided with 8 digital inputs rated at 24 Vdc and 8 digital outputs 0,5 A / 24 Vdc. It can be extended to max. 64 digital I/Os or from 12 to 32 analogue I/Os, with varying electrical specifications.

![DDC711 diagram](image)

**Figure 3-2: DDC 711**

The UC LED is lit as long as the Supply Voltage is greater than 17 V. The RUN LED oscillates with a frequency of two Hertz after entering the Pre-operational state. By entering the Operational state this LED is lit constantly. The CAN LED is lit after initialisation of the node. If the CAN-Error is reached the LED oscillates with a frequency of 0.5 Hz. If the CAN-Error drops under the Warning-limit this LED is lit constantly again. By entering the state Stopped the CAN LED is turned off.

The inputs and outputs are wired to a 10-pole terminal block (see Figure 3-3).

![Terminal block assignment](image)

**Figure 3-3: Terminal block assignment**
The DIP switch S1, mounted on the front of the nodal module, enables the interface to the CAN bus to be configured. Not only the CAN node address for the module but also the transmission rate is set up using this DIP switch (see Table 3-1).

### Table 3-1 : DIP switch S1

<table>
<thead>
<tr>
<th>Node-ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>02</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>03</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>31</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>32</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>33</td>
<td>ON</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>63</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
</tbody>
</table>

When DIP switches 1 to 5 and 8 are set OFF an internal test software will be started.

#### 3.1.1 Default-Mapping

The Default-Mapping can be seen in Figure 3-4.

*Figure 3-4: Default-Mapping of the digital I/Os*
3.1.2 Parameterisation via SDO
A parameterisation is not necessary due to the fact that the modules are in event driven mode and the Global Interrupt Digital is enabled by default.

3.2 Analogue input module AIT 701
The AIT 701 expansion module has 8 different Analogue inputs. The Signals range from 0 to +10V or 0 to 20 mA respectively. The representation of the input signal has a resolution of 10 bits (1000 units) and the value of the LSB is 10 mV (20 µA respectively).

![AIT 701 with terminal block assignment](image)

The VS connector that can be seen in Figure 3-5 works as a reference voltage source with an constant output of +10 Vdc. With this output connected it is possible to also use passive sensors for the input rather than voltage or current sources.

Every channel can be switched individually as current or voltage input by the DIP switches S1 and S2. The connection between the channels and the DIP switches is shown in Table 3-2. The DIP switches are placed on the rear side of the modules.

<table>
<thead>
<tr>
<th>DIP switch S1</th>
<th>DIP switch S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>OFF = voltage input</td>
<td>ON = current input</td>
</tr>
</tbody>
</table>

Table 3-2 :DIP switches S1 and S2
3.2.1 Default-Mapping

![Diagram of AIT 701]

<table>
<thead>
<tr>
<th>Input ID</th>
<th>Sign</th>
<th>MSB</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bit 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 3-6: Default Mapping of the Analogue Input

3.2.2 Parameterisation via SDO

Four SDO transfers should be done while in Pre-operational mode. All concern Object 1801 (Communication-Parameter TxPDO2). By default the TxPDO2 is switched off. The information about that can be found in Subindex 1 which is called the COB-ID TxPDO2. The structure of the COB-ID PDO, which is valid for every PDO, is shown in Table 3-3. PDO4 can be initialised similarly using Object 1803.
PDO: PDO active = 0; PDO not active = 1
RTR: RTR allowed for this PDO = 0; RTR not allowed for this PDO = 1
ID-format: 11-bit ID = 0; 29 bit ID = 1
ID: 11-Bit identifier

The download message to enable TxPDO2 can be seen in Table 3-4.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Data</td>
<td>Sub-index</td>
<td>Index</td>
<td>S</td>
</tr>
<tr>
<td>Value(hex)</td>
<td>COB-ID TxPDO2</td>
<td>1</td>
<td>1801</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-4: SDO message: Enable TxPDO2

To check the correct entry a SDO upload is recommended (see Table 3-5).

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Reserved</td>
<td>Sub-index</td>
<td>Index</td>
<td>X</td>
</tr>
<tr>
<td>Value(hex)</td>
<td>0</td>
<td>1</td>
<td>1801</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-5: SDO message: Check PDO setting

By default the transmission mode is set to asynchronous transmission (mode 255) while the global interrupt is disabled (Object 6423). The transmission mode will be set to synchronous transmission by writing to Subindex 2 (see Table 3-6).

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Data</td>
<td>Sub-index</td>
<td>Index</td>
<td>S</td>
</tr>
<tr>
<td>Value(hex)</td>
<td>1</td>
<td>2</td>
<td>1801</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-6: SDO message: Set synchronous transmission mode

To check the correct entry, again a SDO upload can be performed (see Table 3-7).

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Reserved</td>
<td>Sub-index</td>
<td>Index</td>
<td>X</td>
</tr>
<tr>
<td>Value(hex)</td>
<td>0</td>
<td>2</td>
<td>1801</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-7: SDO message: Check transmission mode setting
3.3 Analogue output module AOT 701

The AOT 701 expansion module has 4 different Analogue inputs (2 current inputs and 2 voltage inputs). The Signals range from 0 to +10V and 0 to 20 mA respectively. The representation of the input signal has a resolution of 10 bits (1024 units) and the value of the LSB is 9.77mV and 19.55µA respectively.

![Figure 3-7: AOT 701 with terminal block assignment](image)

3.3.1 Default-Mapping

![Figure 3-8: Default Mapping of the Analogue Output](image)
3.3.2 Parameterisation via SDO

Two SDO transfers should be done while in Pre-operational mode. All concern Object 1401 (Communication-Parameter RxPDO2). By default the RxPDO2 is switched off. The information about that can be found in Subindex 1 (COB-ID RxPDO2). The structure of the COB-ID PDO was mentioned before.

The download message to enable RxPDO2 can be see in Table 3-8.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data</td>
<td>Sub-index</td>
<td>Index</td>
<td>S</td>
</tr>
<tr>
<td>Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value(hex)</td>
<td>COB-ID RxPDO2</td>
<td>1</td>
<td>1401</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-8: SDO message: Enable RxPDO2

To check the correct entry a SDO upload is recommended (Table 3-9).

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes 4 ... 7</th>
<th>Byte 3</th>
<th>Bytes 1 ... 2</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reserved</td>
<td>Sub-index</td>
<td>Index</td>
<td>X</td>
</tr>
<tr>
<td>Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value(hex)</td>
<td>0</td>
<td>1</td>
<td>1401</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-9: SDO message: Check PDO setting

By default the transmission mode is set to asynchronous transmission (mode 255) while the global interrupt is enabled (Object 6423). So no changes have to be done because the event driven mode is suitable for the analogue outputs.
4 CANbus Toolset

The CANbus toolset constitutes an extension for the technical computing environment MATLAB. More specifically, it extends the graphical simulation system Simulink with respect to easy-to-use and direct CANbus access capabilities in realtime. Simulink’s Realtime Workshop is not necessary. For installations instructions [4] is recommended.

4.1 Canmenu

The Canmenu is a graphical user interface (GUI) to build and manipulate the database for writing to the CANbus. It provides the following features:

- Define all signals by name that shall be written to the CANbus
- Perform scaling operations on the signals
- Define the CANbus identifiers (message identifiers) which carry the signals.
- Set defined bits within the messages

The signals are named with the so called Simulink access name (SAN) defined by the user. In the Canmenu the signals are assigned to a specific CANbus identifier including its position in the data field (first bit and last bit). The following example shows the set up for the NMT Module Control services discussed in 2.6 for Node- ID=2. The input field CH selects the channel of the CANbus card installed in the PC. As shown in Figure 4-1 it is applied to channel 1.

![Figure 4-1: Canmenu](image)
The message identifier is specified in the Identifier field. New identifiers can inserted by simply pressing the CopyID button and specify the number in hex code. The signals which were defined in the old message are now also assigned to the new message. It is recommended to use the dummy message FFFFFFFFF for this operation. By highlighting an Identifier the lower square shows the assignment of the signals and the pre set bits in the datafield. Figure 4-1 shows that a signal occupies bit 0 to 7 of the datafield. From 2.6 it is already know that this field holds the command specifier. Switching to the Signals menu by the button Switch Menu reveals that in fact a signal named CS has been assigned to Bit 0 to 7 of the identifiers datafield (see Figure 4-2). It also can be seen that no scaling is performed to the signal. The menu can be left via its Switch Menu button.

![Figure 4-2: Signals menu](image)

<table>
<thead>
<tr>
<th>Signal Number</th>
<th>Signal Name</th>
<th>Signal Description</th>
<th>Bit List (from low to high)</th>
<th>Scaling coefficients y = a + b/u</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CS</td>
<td>describe</td>
<td>0:7</td>
<td>0-0, 1-0</td>
</tr>
</tbody>
</table>

Back in the main menu Figure 4-1 shows that Bit 9 had been set to 1 via a radio button. Knowing that Byte 1 is bearing the Node ID this means that the NMT Module Control service is only applied to Node 2. For other demands this datafield could also have been applied to a second signal maybe named Node-ID to have the possibility to change the receiver of the message. For this the Number of Signals field would have been set to 2 and by pressing the Define Signal(s) button the signal would have been described in the Signals menu.

Additionally on the bottom of the main window the number of transferred bytes can be specified individually for each identifier. In this case a size of 2 bytes was assigned to the datafield although in this release of the software the GUI was not able to show it properly.

By pressing the OK button the window is closed and the data is saved in the CAN_menu.dat.mat file at the current MATLAB directory.
4.2 Simulink blocks

After successful installation the CANbus Toolset is available in the Simulink Library Browser. Three blocks are of further interests for our purposes:

- From CAN_1 To read from the CANbus
- ModelControlBlock To set some communication parameters
- To Can_1 To write to the CANbus

By opening a new model in the Simulink environment the blocks can placed in the usual drag and drop manner (see Figure 4-3).

![Figure 4-3: CANbus Toolset blocks](image-url)
4.2.1 Model Control Block

The Model Control Block may only be used once within a simulation model. The parameters are set within the CAN options window by double-click on the block (see Figure 4-4).

![Figure 4-4: Options menu](image)

- **Channel:** Allows to specify the channel of the CANbus PC card (normally 1)
- **Inputs from:** The From CAN_1 blocks have the possibility to be fed from a simulink block rather than reading from the CANbus. This field allows a global initialisation for these blocks. Due to the fact that this parameter is overwritten by each From CAN_1 block parameter it should be set to CANbus by default.
- **#bytes to output:** If the already mentioned Number of Bytes in Canmenu is set to Default then the number from this field is taken into account. It should normally be set to 8 byte to be sure all data is delivered. It is recommended to set the real size of the datafield in the Canmenu.
- **CAN Outputs active ?:** This field gives the possibility to globally switch off the To CAN_1 blocks (normally YES).
- **Interface Tune:** Concerns the CANbus Pc card (normally Highspeed).
- **Baud Rate:** In this field the Baud Rate of the system is set (normally 500 k)
- **Identifier:** This Check box allows the to specify the size of the Identifier (11 bit in our case)
- **Realtime:** Enables the Realtime behaviour and should always be checked (YES).
4.2.2 From CAN_1 block

Double-click on the block opens the parameterisation window. Most of the field entries are the same like in the Canmenu and Signals menu. Only the position of the data is represented by the Bit Address field which is the first bit and the Number of bits field which is the length. The field on the top right side is the in 4.2.1 mentioned possibility to feed the CAN input form a Simulink block rather than via network. The Default entry would take the entry specified in the options menu otherwise it is overwritten by the settings of this window. The values shown are valid for an SDO upload discussed in 5.2.2.

![Parameterisation window for CAN_1 block](image)

Figure 4-5: From CAN Parameterisation window

4.2.3 To CAN_1 block

Double-click on the block opens the parameterisation window. In this case the field entries are automatically taken from the Canmenu data. The Select button reveals a Subwindow were all the signals defined in the Canmenu database are listed. By selecting one of those signals the parameters are overtaken to the Parameterisation window. In this case the CS signal was taken with the values discussed in 4.1.

![Parameterisation window for CAN_1 block](image)

Figure 4-6: To CAN Parameterisation window
4.3 Simulation parameters

The CANbus Toolset is designed for realtime simulation (without using the Realtime Workshop). Therefore only fixed integration steps are allowed. The step size needs to be entered in the standard Simulink simulation parameters menu.

![Simulation Parameters: DDC711_BootUp](image)

Figure 4-7: Simulation Parameters window

Realtime synchronisation is essential. Minimum sample time is 10 msec. However, the actual value depends on CPU speed, model size, model dynamics, etc. Good results have been achieved if the sampling time is 15 msec and more [4].
5 I/O Testing

The first set up was done to check the input and output of the modules. It consisted of a DDC 711 digital input/output nodal module, one AIT 711 analogue input expansion module and one AOT analogue output expansion module (see Figure 5-1).

![Figure 5-1: I/O test set up](image1)

As seen in Figure 5-2 the digital outputs of the system were simply fed in reverse manner to the digital inputs. The analogue outputs 00 and 10 which are voltage outputs were connected to the analogue inputs 00 and 01 which were also in voltage configuration (see Table 3-2 :DIP switches S1 and S2).

![Figure 5-2: I/O test connections](image2)

The system was configured as node 2 with a Baud rate of 500k.(see Table 3-1 :DIP switch S1).
5.1 Canmenu configuration

The identifiers that were used are shown in Figure 5-3. The identifiers were calculated as shown in Table 2-9 using Module ID number 2.

<table>
<thead>
<tr>
<th>Object</th>
<th>Identifiers (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMT Module Control</td>
<td>0</td>
</tr>
<tr>
<td>Synchronisation Object</td>
<td>80</td>
</tr>
<tr>
<td>PDO1 (transmit)</td>
<td>182</td>
</tr>
<tr>
<td>PDO1 (receive)</td>
<td>202</td>
</tr>
<tr>
<td>PDO2 (transmit)</td>
<td>282</td>
</tr>
<tr>
<td>PDO2 (receive)</td>
<td>302</td>
</tr>
<tr>
<td>SDO (transmit)</td>
<td>582</td>
</tr>
<tr>
<td>SDO (receive)</td>
<td>602</td>
</tr>
</tbody>
</table>

Table 5-1: Used communication objects

Only those objects that were written to the CANbus are specified in the Canmenu. From the viewpoint of the modules they receive the data which is written from the CANbus Toolbox to the CANbus. So all receive objects have to be specified in the Canmenu.

The object FFFFFFFF is dummy object which is placed in every Canmenu be default. Its only used to create new objects via the CopyID button.

5.1.1 NMT Module Control

This object was used as the example in 4.1 so all settings and explanations can be found there.
5.1.2 Synchronisation Object

Figure 5-4 reveals that a signal is applied to the Synchronisation object located at bit 0. This seems contrary to 2.5 were it is said that this object doesn’t contain data. In fact the signal just serves as a dummy because the To CAN_1 block needs a signal name which is applied to it. Therefore the Number of Bytes field should also be set to its minimum which is 1 Byte.

![Canmenu for Synchronisation object](image)

Figure 5-4: Canmenu for Synchronisation object

<table>
<thead>
<tr>
<th>Signal Number</th>
<th>Signal Name</th>
<th>Signal Description</th>
<th>Bit List (from low to high)</th>
<th>Scaling coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sync</td>
<td>describe</td>
<td>0</td>
<td>a = 0, b = 0</td>
</tr>
</tbody>
</table>

![Signals menu for Synchronisation object](image)

Figure 5-5: Signals menu for Synchronisation object
5.1.3 PDO1 (receive)
This object addresses the data which is written to the digital outputs of the DDC 711. Derived from 3.1.1 the first Byte has to be used for the data. Therefor the Number of Bytes field should be set to 1 Byte.

![Figure 5-6: Canmenu for PDO1 (receive)](image)

![Figure 5-7: Signals menu for PDO1 (receive)](image)

Only the receive objects have to be specified in the Canmenu. The transmit objects must be parameterised in the Simulink model.
5.1.4 PDO2 (receive)

This object addresses the data that is written to the analogue outputs of the AOT 701. Derived from 3.3.1 two Bytes are used for every channel starting with the 5th Bit. With four channels the complete datafield is used. Therefor the Number of Bytes field should be set to 8 Bytes.

![Diagram]

*Figure 5-8: Canmenu for PDO2 (receive)*

<table>
<thead>
<tr>
<th>Signal Number</th>
<th>Signal Name</th>
<th>Signal Description</th>
<th>Bit List (from low to high)</th>
<th>Scaling coefficients of x ± D*U</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AnalogOut1</td>
<td>describe</td>
<td>000010</td>
<td>0 0 7 0</td>
</tr>
<tr>
<td>2</td>
<td>AnalogOut2</td>
<td>describe</td>
<td>000011</td>
<td>0 0 3 0</td>
</tr>
<tr>
<td>3</td>
<td>AnalogOut3</td>
<td>describe</td>
<td>000011</td>
<td>0 0 3 0</td>
</tr>
<tr>
<td>4</td>
<td>AnalogOut4</td>
<td>describe</td>
<td>000011</td>
<td>0 0 3 0</td>
</tr>
</tbody>
</table>

*Figure 5-9: Signals menu for PDO2 (receive)*
5.1.5 SDO (receive)

This object is used for all SDO transfers to the module. From 2.3 it is clear that if it is not necessary to declare the size of the data. Bit 0, 2, 3, and 4 can always be set to zero. It is also derived that 5 fields must be specified to have the possibility to address every object and to differ between Upload and Download procedure.

Figure 5-10: Canmenu for SDO (receive)

Figure 5-11 show that all needed fields are defined as Signals. The SANs are directly derived from the names which are used in 2.3.1. Only the name for the data field has changed to data_down indicating that this field only contains data for the Download process. The E bit could have been also implemented as a static 1 in the Canmenu field rather than defining a signal for it.

Figure 5-11: Signals menu for SDO (receive)
5.2 Simulink models
If you think about the process of starting up a network the whole process can be split into three main parts.

1. Initialise the modules via SDO transfer
2. Start Remote Node with the NMT Module Control Services
3. Transfer the process data

So the decision was to create three Simulink models which are called subsequently.

1. DDC711_SDO.mdl
2. DDC711_BootUp.mdl
3. DDC711.mdl

5.2.1 DDC711_BootUp model
This model contains some of the blocks which will be the same for all following models. These are five blocks for three purposes. The previously mentioned Model Control Block which is parameterised as in 4.2.1. The From block receives signals from the CAN_time tag. This block is of no further use here but its implementation prevents MATLAB from displaying error messages. The Clock block together with the Display1 block simply shows the simulation time.

![ModelControlBlock (use only once!): Double-click for opening the Can Options menu](image)

Figure 5-12: DDC711_BootUp model

The Module boot up block which is of the type To CAN_1 uses the signal CS and is therefore connected to the 000 identifier. The value is delivered by the Control block which is of the type Constant by the variable cs. The value for cs is according to Table 2-8. It should be mentioned that the value of the signal must be in decimal representation.
5.2.2 DDC711_SDO model

The model consists of 5 blocks of the type To CAN_1 which are using the signals Index, Subindex Data_down, CCS and E and therefore connected to the 602 Identifier. The values are delivered by the blocks sharing the signal’s names of the type Constant. The names of the variables are also derived from the signal’s names but indicating their decimal representation if needed.

For Upload operations the From CAN SDO block is implemented which is of the type From CAN_1. Its data is fed to the To Workspace block using the variable data_up_dec. Its parameters are shown in Figure 5-14 indicating that it connected to Identifier 582 which is the SDO (transmit) object. The Bit position and length is derived from 2.3.2. Additionally displays are added for each signal.

Figure 5-14: Parameters of the From CAN SDO block
5.2.3 DDC711 model

This model performs the control of the process data. The *To DDC 711-Output* block writes via the CANbus to the digital outputs. For testing purposes it is fed by two decimal values that change with a frequency of 1 Hertz.

The counterpart is the *From DDC 711-Input* block which reads the digital inputs. Its parameters are shown in Figure 5-16.

---

**Figure 5-15: DDC711 model**

**Figure 5-16: Parameters of the From DDC 711-Input block**
The two `Constant` blocks `Voltage 1` and `Voltage 2` hold the value of the desired output voltage. The two `Gain` blocks `Resolution` calculate the decimal representation of the digital value. The values are then converted to Integers by two `Rounding Function` blocks. The blocks `To AOT 701-00` and `To AOT 701-10` sends the values via the CANbus to the analog outputs.

The blocks `From AIT 701-00` and `From AIT 701-01` receives the values of the analogue inputs via the CANbus. The two `Gain` blocks `Resolution` (4 and 5) calculate the value of the voltage. Another possibility would be to use the “Scaling coeff.” field shown in Figure 5-17.

![Diagram](image.png)

Figure 5-17: Parameters of the From AIT 701-00 and 01 block

Close related to the analogue inputs is the `To DDC 711-Sync` block which sends the Synchronisation signal. Due to the fact that this object is not supposed to carry data this block’s input is just the ground signal.

The simulation parameters are set as in 4.3.
5.3 MATLAB procedures
To start the sequence mentioned in 5.2 MATLAB M-files (see Appendix A) have been written (see Figure 5-18). The Start.m file calls 7 M-files in the desired sequence.

The first one is the RxPdo2on.m file which accesses Object 1401 to set RxPDO2 active. The required variables are set and fed to the DDC 711_SDO model. The second file (Check RxPdo2on.m) also accesses Object 1401 and reads the COB-ID PDO2 using the same model. The COB-ID PDO2 is displayed in hex and binary Bit 31 should be zero if RxPDO2 is active. The third (TxPdo2on.m) file is similar to file two but accesses Object 1801 to set TxPDO2 active. Next, file four (Check TxPdo2on.m) performs the same check as file two to show if TxPDO2 is active. With
the fifth file (TxPDO2Sync.m) the transmission mode of TxPDO2 is set to synchronous and the sixth file (CheckTxPDO2Sync.m) checks. This check should display a one.

It should be mentioned that for some unknown reasons the checking parts deliver a wrong value if the program is started for the first time. The remedy is to open the DDC711_SDO model first as it is done in start.m.

The seventh is the NmtStart.m file which defines the Command Specifier variable for the Start Remote Node (cs=1). This variable is fed to the Simulink DDC711_BootUp model which is started.

The last command in the start.m file is to open the current project model which is the DDC711.mdl for the I/O Testing set up.

It is further recommended that all the files should be placed in one project directory including the Canmenu file, the models and all M-files. The first step after starting MATLAB should be to open the Path Browser and set this directory as the Current Directory.

5.4 Message timing

With a notebook equipped with a CAN-card\(^1\) and analyser software\(^2\) the occurrence of the messages at the CANbus were observed. The Simulink step size was set to 20 ms. In Figure 5-19 the sequence starts with the message to set the analogue outputs (RxPDO2). 200 µs later the message to set the digital outputs (RxPDO1) is sent.

Figure 5-19: Message timing diagram

\(^1\) CANcard2 by softing GmbH
\(^2\) Xanalyser by Warwick Control Technologies Ltd.
Another 200 µs later the synchronisation (Sync) message for the analogue inputs is sent. With a bit rate of 1 bit per 2 µs and maximum length of 117 bits per message this is the smallest distance between two messages. After the synchronisation message has been sent it takes 5.3 ms until the analogue inputs answer with a message (Tx PDO2). This corresponds with the transition time mentioned in [3], that is 4.1 ms. 2.1 ms after that event the message of the digital inputs is sent. The next time the RxPDO2 occurs is 20 ms after the first time. So the distance between those two messages yield the Simulink step size. The distance between the first and the second TxPDO1 message is due to the frequency of the Signal Generator block and takes 500 ms. The messages that occurred between 25.5 ms and 500 ms are simply left out of Figure 5-19. The bus load which was measured with the analyser was between 2.5 and 4.3%. Figure 5-20 shows a screenshot of the Xanalyser software.

![Figure 5-20: Screenshot of the analyser software](image)
6 Air heating process

Next a real process was connected to the modules. The system’s task was to control the temperature of an air heating process. Figure 6-1 shows the principle of this experimental rig. A fan blows air through a tube from the left side. The speed of the Fan can be controlled only manually. This variation is done by a knob with a scale ranging from 0 to 10 were 0 is the lowest and 10 the highest speed. Near the fan a heating device is installed which is fed by the Power Supply. The Power Supply is controlled by the Manipulated Input signal. On the outlet of the tube a temperature sensor is placed and connected to the Bridge Circuit. The Bridge Circuit delivers the Process Output signal.

![Diagram of the air heating process](image)

*Figure 6-1: Air heating process*

Because this is a single input/single output process only one analogue input and one analogue output of the SELECONTROL® MAS system is needed (see Figure 6-2).
6.1 Process parameters

6.1.1 Static curve of the sensor

First the relationship between the measured temperature and the Process Output voltage was observed shown in Table 6-1. The power of the heating device was increased in linear steps and the temperature was measured by a manual thermometer. The output voltage of the bridge circuit was measured by a digital multimeter.

<table>
<thead>
<tr>
<th>Temperature [°C]</th>
<th>Process Output [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.0</td>
<td>11.9</td>
</tr>
<tr>
<td>46.5</td>
<td>11.0</td>
</tr>
<tr>
<td>45.0</td>
<td>10.0</td>
</tr>
<tr>
<td>43.0</td>
<td>9.0</td>
</tr>
<tr>
<td>41.5</td>
<td>7.9</td>
</tr>
<tr>
<td>40.0</td>
<td>6.8</td>
</tr>
<tr>
<td>35.8</td>
<td>5.2</td>
</tr>
<tr>
<td>34.0</td>
<td>3.8</td>
</tr>
<tr>
<td>32.0</td>
<td>2.4</td>
</tr>
<tr>
<td>28.7</td>
<td>0.3</td>
</tr>
<tr>
<td>26.7</td>
<td>-1.0</td>
</tr>
</tbody>
</table>

Table 6-1: Temperature / Process Output relationship

Figure 6-3 shows that the measured data can be approximated by a linear function (volts = 0.6013*temp - 16.9). The parameters of the first order polynomial were calculated by MATLAB’s polyfit function. It can be seen that the point of zero voltage is above the normal ambient temperature (at 28°C). This is due to the fact that the process produces heat even if the Manipulated Input is set to zero. The realization of the static function can be seen in Figure 6-4.
6.1.2 Step response
To obtain the dynamic parameters of the system a step from 0 to 9 volts (0 to 90%) and back to 0 volts was performed. The step height was chosen to get a rise time long enough to detect the parameters in a graphical way and also to keep some distance to the limit of the temperature sensor (below 45°C). The CANbus system was used to obtain the step response.

![Figure 6-5: Step response of the process](image1)

![Figure 6-6: Heating up of the tube](image2)

Figure 6-5 shows the measured step response with its discrete time behaviour. The step width of the signals was found to be 200 msec. With the analyser mentioned in 5.4 it was checked that although the analogue inputs sent their data every 20 ms the content turned out to change only every tenth message. The reason for this was investigated and Selectron contacted but as yet no reason for this behaviour has been found. Figure 6-6 shows the heating up of the tube.

6.2 Process model
6.2.1 Continuous time model
The step response of Figure 6-5 can be modelled as a first order plus delay time (FOPDT) shown in Figure 6-7.

![Figure 6-7: Continuous process model](image3)
The Transport Delay block and the Air block realize the FOPDT, the parameters were detected in a graphical way and adapted manually by comparing the step responses.

The parameters are:

\[ K_m e^{-s\tau_m} \over 1 + sT_m = 1.68e^{-0.7} \over 1 + 0.5s \]

As see in Table 6-1 the lowest possible temperature which can be detected by this system is 28.11 °C and the highest is 45 °C. This is behaviour is modelled by a Saturation block. As mentioned in 6.1.1 the measured temperature is higher than the ambient temperature even if the Manipulated Input is zero. The summation of the Ambient Temp block and the Minimum Outlet Temp block realize this circumstance. Finally the heating up of the tube (see Figure 6-6) is implemented as a first order lag.

The parameters are:

\[ K_m \over 1 + sT_m = 0.311 \over 1 + 300s \]

Figure 6-8 and compares the real process and the process model. The blue curves are the model the green curves the real process responses.

![Figure 6-8: Step response comparison](image1)

![Figure 6-9: Heating up comparison](image2)
6.2.2 Hybrid model

Due to the fact that the message delay can’t be neglected a model with a Unit Delay block may represent the process better than the continuous Transport Delay. As seen in Figure 6-10 the Transport delay block is replaced by two Unit Delay blocks (Output Message Delay and Input Message Delay), both having a sample time of 0.2 sec. The ambient temperature was higher for this test procedure so the parameter of the Ambient Temp. block has been changed. Because of the physical relationship the proportional factor of the Air block has also be changed.

![Figure 6-10: Hybrid process model](image)

Figure 6-11 shows that the model and the process behave similarly.

![Figure 6-11: Step response comparison](image)

Besides the discrete behaviour another benefit is the model will now always deliver a zero value to the Sensor Saturation block for the first sample period. This is like the real system behaves with a small difference. The real system always delivers a zero value for the first 0.1 sec then, for the next 0.1 sec, it delivers the last measured value from the previous simulation. This means that after 0.2 sec for the first time a valid value for the current simulation is available.
6.3 PI controlled process
For the first attempt to control the process the simple PI controller was chosen.

6.3.1 Control structure
The Ziegler and Nichols (1942) Tuning Rule is taken from [5]. For a FOPDT the following controller parameters are valid:

\[
G_c(s) = K_c \left(1 + \frac{1}{T_1 s}\right) \quad \text{with} \quad K_c = \frac{0.9 T_m}{K_m \tau_m} = 0.383 \quad \text{and} \quad T_1 = 3.33 \tau_m = 2.33
\]

\[
G_c(s) = 0.383 \left(1 + \frac{1}{2.33 s}\right)
\]

Figure 6-12 reveals the structure of the PI controller basically it represents the above described formula. For better performance to set point changes the \( T_1 \) parameter has been changed to \( 2.33^{-1} \) sec instead of 2.33 sec. Furthermore a Anti Reset Windup loop has been implemented and the controller saturation limits the output signal from 0 to 10 volts.

![Figure 6-12: PI controller structure](image-url)
6.3.2 System behaviour

Figure 6-13 shows the model of the whole system. The PI controller can be found twice to compare online the behaviour of the model and the real process. The blocks have all been described in the previous sections.

![Control system structure diagram]

Figure 6-13: Control system structure

The closed loop response for a set step to 40 C is shown in Figure 6-14. The green curve shows the real process and the blue curve the continuous model response.

![Closed loop response graph]

Figure 6-14: Closed loop response
6.4 Fuzzy controlled process

Next a more complex control structure was implemented and the system’s behaviour was observed. A fuzzy control structure was chosen to control the air heating process. There is a big variety of fuzzy control structures and a much bigger number of strategies to design and set the parameters of the controller. At the beginning of a fuzzy control implementation the easiest possible structure should be used. It should be easy in a sense that the structure allows the setting of the control parameters in a manner like tuning a PI controller. Having just two or three parameters ensures the tuning process converges to a good and stable performance of the system.

6.4.1 Control structure

A control structure that meets the requirements is shown in Figure 6-15. The so called Fuzzy-PI controller is derived from [6].

![Figure 6-15:](image)

The controller has three external gain parameters (ke, kde and g) that can be used to adjust its performance. In [6] the effects of changing these parameters were investigated especially with respect to the conventional PI controller’s parameters. The result is shown in Table 6-2.

<table>
<thead>
<tr>
<th>Fuzzy Parameter</th>
<th>Relation to PI parameter</th>
<th>Calculated value</th>
<th>Used value</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Kp</td>
<td>0.383</td>
<td>0.1</td>
</tr>
<tr>
<td>Ke</td>
<td>Ki/10</td>
<td>0.089</td>
<td>0.05</td>
</tr>
<tr>
<td>Kde</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 6-2: Fuzzy-PI parameters relationship

Due to the fact that it was pointed out in [6] that this can be seen as a rule of thumb, the used values were changed to get a smooth set point change behaviour.
6.4.2 Fuzzy Interference System (FIS)

With the Fuzzy Interference System from Matlab all designs necessary for the Fuzzy controller can be realized. Starting with the fuzzification of the two inputs which can be seen in Figure 6-16 and Figure 6-17. The interference done with 9 rules is shown in Figure 6-18. And last but not least the defuzzification represented by the singleton membership functions shown in Figure 6-19.

The use of concentrated singleton output sets (crisp values) like shown in Figure 6-19 instead of fuzzy output sets is simply to make the process of defuzzification more efficient in terms of computational effort.

Since the input fuzzy sets are triangular and linearly spaced and the defuzzification is via the centre of gravity method, the control surface of the fuzzy logic controller is linear (see Figure 6-20). For ke error and kde*error/dt outside of the interval [-1,1] the controller shows saturation. At this point it gets clear that this parameterisation used in [6] can be used generally as a basic fuzzy controller set up.
6.4.3 System behaviour

Like in 6.3.2 the behaviour of the system to a set point change was observed. Again the result was compared to the model of the real process. The structure of the controlled system doesn’t differ to the structure in 6.3.2 (see Figure 6-21).

![Control system structure](image.png)

*Figure 6-21: Control system structure*

The response of the system to a set point change from 30°C to 40°C and back to 30°C is shown in Figure 6-22. The behaviour is slower with less overshoot compared to the PI behaviour.

![Closed loop](image.png)

*Figure 6-22: Closed loop*
7 Coupled tanks process

The second process that was connected to the modules was the “Coupled Tanks” process. The level of two tanks had to be controlled. A pipe connects the two tanks and both tanks are connected by two pipes with the reservoir tank on the bottom (see Figure 7-1). The flow rate in the pipes is depending on the levels of the tanks and the adjustment of three manual valves. The water is fed to the tanks from the reservoir with two pumps. The pump rate can be controlled from outside and therefore these are the two inputs of the process. The flow rates the pumps produce are measured with two flow sensors. The two flow rates together with the two levels are the four outputs of the process. So it is a multi input / multi output (MIMO) system.

For this process four analogue inputs and two analogue outputs had to be connected as shown in Figure 7-2.
7.1 Process parameters

7.1.1 Static curves of the sensors

This time only the static parameters were observed. Like in 6.1.1 the four process outputs were measured with a digital multimeter and compared with the readings of the process equipment. Generally the flow rates are ranging between 0 and 4.2 l/min and the levels are ranging between 0 and 250 mm. All physical values are represented by voltages between 0 and 10 volts. The first sensor that was observed was the flow sensor of tank 1.

<table>
<thead>
<tr>
<th>Flow rate tank 1 [l/min]</th>
<th>Process Output [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.98</td>
</tr>
<tr>
<td>0.6</td>
<td>1.4</td>
</tr>
<tr>
<td>0.8</td>
<td>1.84</td>
</tr>
<tr>
<td>1</td>
<td>2.33</td>
</tr>
<tr>
<td>1.2</td>
<td>2.74</td>
</tr>
<tr>
<td>2</td>
<td>4.67</td>
</tr>
<tr>
<td>3</td>
<td>6.92</td>
</tr>
<tr>
<td>3.8</td>
<td>8.66</td>
</tr>
<tr>
<td>4.2</td>
<td>9.7</td>
</tr>
</tbody>
</table>

Table 7-1: Flow rate / Process Output relationship

Figure 7-3 shows that the measured data can be approximated by a linear function (volts = 2.27240*Flow + 0.05196). Again MATLAB’s polyfit function was used. The sensor shows a very small zero point offset. The realization of the static function can be seen in Figure 7-4.

Figure 7-4: Flow Sensor 1 static structure
The secondly observed sensor was the level sensor of tank 1.

<table>
<thead>
<tr>
<th>Level tank 1 [mm]</th>
<th>Process Output [v]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>27</td>
</tr>
<tr>
<td>30</td>
<td>106</td>
</tr>
<tr>
<td>50</td>
<td>180</td>
</tr>
<tr>
<td>70</td>
<td>264</td>
</tr>
<tr>
<td>90</td>
<td>343</td>
</tr>
<tr>
<td>150</td>
<td>580</td>
</tr>
<tr>
<td>180</td>
<td>702</td>
</tr>
<tr>
<td>210</td>
<td>821</td>
</tr>
<tr>
<td>230</td>
<td>900</td>
</tr>
<tr>
<td>250</td>
<td>975</td>
</tr>
</tbody>
</table>

Table 7-2: Level / Process Output relationship

The measured data shown in Figure 7-5 was approximated by the linear function \( \text{volts} = 0.03969 \times \text{level} - 0.14248 \). The sensor shows a small zero point offset The Simulink model can be seen in Figure 7-6.

Figure 7-6: Level Sensor 1 static structure

The flow sensor of tank 2 could not be observed because no manual flow meter was integrated in the process. So the same behaviour like the flow sensor of tank 1 is assumed. Hence the Simulink model is carrying the same parameters (see Figure 7-4).
Last the level sensor of tank 2 was observed.

<table>
<thead>
<tr>
<th>Level Tank 2 [l/min]</th>
<th>Process Output [v]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.420</td>
</tr>
<tr>
<td>20</td>
<td>2.800</td>
</tr>
<tr>
<td>30</td>
<td>3.203</td>
</tr>
<tr>
<td>40</td>
<td>3.624</td>
</tr>
<tr>
<td>50</td>
<td>4.035</td>
</tr>
<tr>
<td>100</td>
<td>6.056</td>
</tr>
<tr>
<td>150</td>
<td>8.095</td>
</tr>
<tr>
<td>200</td>
<td>10.107</td>
</tr>
<tr>
<td>220</td>
<td>10.887</td>
</tr>
<tr>
<td>230</td>
<td>11.297</td>
</tr>
<tr>
<td>240</td>
<td>11.686</td>
</tr>
<tr>
<td>250</td>
<td>12.096</td>
</tr>
</tbody>
</table>

Table 7-3: Level / Process Output relationship

The measured data shown in Figure 7-7 was approximated by the linear function $v = 0.04039 \times \text{level} + 2.00810$. The sensor shows a big zero point offset. To avoid a negative level the Simulink model is provided with a Saturation block (holding the limits between 0 and 250 mm) which can be seen in Figure 7-8.

Figure 7-7: Level / Process Output relationship

Figure 7-8: Level Sensor 2 static structure
7.1.2 Modified Astrom test

The modified Astrom test uses a relay in closed loop like the original Astrom test. Additionally an integrator is added to achieve an oscillation at a phase of -90 degrees rather than -180 degrees like in the original test. The reason for this is to allow the remaining zero of the PI controller to provide \( \phi_m \) compensation of 90 degrees by its location. The goal is to detect the gain value \( K_u \) and the period \( P_u \). \( K_u \) can be calculated from the amplitude \( A \) of the oscillation and the relay gain \( D \).

\[
K_u = \frac{4D}{\pi A}
\]

The period \( P_u \) can be read directly from the diagram. Finally the two control parameters of the PI controller can be calculated using the following formulas. \( \phi_m \) is the desired phase margin for the loop.

\[
T_i = \frac{P_u}{2\pi \tan(\phi_m)} \quad \text{and} \quad K_c = \frac{P_u \sin(\phi_m)}{2\pi}
\]

To perform the modified Astrom test with the two loops two Simulink subsystems were created named Mod. Astrom tank 1 and Mod. Astrom tank 2 (see Figure 7-9). Beside the relay and the integrator a saturation block is implemented also. To get valid results for the test the control effort should be observed. The control effort must stay in between the limits which are 0 to 10 volts for this process otherwise the above formulas are no longer valid.

![Figure 7-9: Astrom test model](image)
7.2 SISO control of a MIMO process

The control strategy was to use two single input/single output controllers to control the double input/double output process. The two flow rate outputs were not used. It was decided to take again a PI control structure. Due to the fact that the two tanks of the process are coupled one closed control loop effects the other. For the tuning of the control parameters the “Sequential Loop Closing” technique was used. Normally the first loop tuned is the most dominant one. In this special process the two loops are equal.

Step 1: The modified Astrom test of the first loop is performed with the second loop open. The control parameters of the first loop are set for a desired phase margin. The set point dynamics of the first loop is observed.

Step 2: The modified Astrom test of the second loop is performed with the first loop closed. The control parameters of the second loop are set for a desired phase margin. The set point dynamics of both loops are observed.

Step 3: If necessary a retuning of the first control loop is performed.

7.2.1 Control system structure

As shown in Figure 7-10 the structure of the controlled system consists basically of the parts that are already used in 6.3.2. This time no process model is used.
The *Control switch 1* block allows switching between a closed loop with the PI controller, a closed loop for the modified Astrom test and an open loop *pump rate tank 1* input block. For the input of the desired value for the PI controller now a subsystem was used to create a whole sequence of set point changes. The structure of the second loop is similar and not shown in the figure.

### 7.2.2 Step 1

Figure 7-11 shows the test for the first control loop with the second loop open. The loop was tested with a with a set point change from 0 to 100 mm (blue line). The control effort gets in saturation for three cycles at the beginning of the test (green line). From the time 500 sec on the controlled value oscillates with a constant frequency and amplitude.

![Figure 7-11: Modified Astrom test](image)

To get a control effort without saturation the relay gain D was set to 0.2. Figure 7-11 reveals an amplitude of 8.469 and a period of 51.94 s. With this values and a phase margin of 45 degrees the control parameters can be calculated with the above equations. It comes to $T_i = 8.27$ sec and $K_c = 0.1758$. Figure 7-12 shows the system’s response to a set point sequence (blue line). The overshots of the controlled value (red line) are around 10 %. The green line again shows the control effort. The second tank was in open loop.

![Figure 7-12: Set point sequence](image)

### 7.2.3 Step 2

Figure 7-13 shows the test for the second control loop with the first loop closed. The loop was tested again with a with a set point change from 0 to 100 mm (blue line). The control effort gets in saturation for three cycles at the beginning of the test (green line). From the time 500 sec on the controlled value oscillates with a constant frequency and amplitude.
The relay gain $D$ was set to 0.4. Figure 7-13 reveals an amplitude of 5 and a period of 21.88 s. With this values and a phase margin of 45 degrees the control parameters can be calculated with the above equations. It comes to $T_i = 3.482$ sec and $K_c = 0.2508$. Figure 3-6 shows the system’s response of tank 1 to a set point sequence (blue line) and the influence of the second loop (tank 2).

Figure 7-14 shows the system’s response of tank 2 to a set point sequence (blue line) and the influence of the first loop. The green line represents in both figures the control effort.

The behaviour of the second loop is faster and the reaction on a disturbance is smaller. Both loops show nearly the same overshoot for set point changes. Due to the observed behaviour a third step is not required. The parameters of the two PI controllers deliver good results.
8 Conclusions

The aim of this project was to set up a CANbus system consisting of I/O modules from Selectron and a PC running the CANbus Toolset software from Scientific Computers.

The system was set up and tested with different control structures and connected to different processes. The set ups of the system were

- I/O testing with connecting the modules inputs to their outputs,
- connecting an air heating process to the modules and control it with a PI controller,
- using the same process and control it with a Fuzzy controller and finally
- connecting a coupled tanks process and control this MIMO system with two SISI PI controllers.

In the first part of the project the requirements for the usage of the modules were determined. Further more the possibility of the CANbus Toolset software was checked to meet these requirements. It was revealed that all the CANopen messages that are needed to control the modules can be realized with the software. By using Matlab’s M-files a automatic start up of the modules have been realized. So, using the Matlab environment a system can be controlled by simply entering its name in Matlab’s command line and the starting the simulation in Simulink.

While working with the air heating process it was observed that the step width of the whole system turned out to be 200ms rather than 20ms which was expected from the Simulink parameter. The reason for this was investigated and Selectron contacted but as yet no reason for this behaviour has been found. For subsequent works the following is suggested:

Due to the fact that messages are sent every 20ms but containing the same data 10 times it should be checked that the PC is acknowledging the messages. This could be done be accessing the analogue inputs with a C++ program rather than with the Matlab software. If this shows the same results the further observation should be concentrated on the modules itself.

The system worked as well with more complex control structures (Fuzzy control instead of PI control) as with more complex processes (MIMO system instead of SISO system). Thus this system can be used for research, undergraduate, postgraduate studies at this university.

For subsequent projects it is suggested to set up a multiple node network connecting several processes.
Appendix

A MATLAB M-files

A.1 Project 1: I/O Testing

A.1.1 Start.m

% Starts the sequence to activate the node
% and set the initial parameters.
% 7 M-files are called.

clear
clc;

disp(' ')
disp('Start remote node 2')
disp(' ')

nmtstart; % Start remote node

disp(' ')
disp('Initialise node 2')
disp(' ')

RxPDO2on; % Set RxPDO2 active
CheckRxPDO2on; % Check if RxPDO2 active
TxPDO2on; % Set TxPDO2 active
CheckTxPDO2on; % Check if TxPDO2 active
TxPDO2Sync; % Set TxPDO2 to synchronous mode
CheckTxPDO2Sync; % Check TxPDO2 mode

disp(' ')
disp('Node 2 successfully')
disp('started and initialised')
disp(' ')

A.1.2 Nmtstart.m
% Performs the Start Remote Node operation with
% the NMT Module Control Services
% The Simulink model DDC711_BootUp is used

% Set variable
cs=1;

% Start simulation model
sim('DDC711_BootUp');

A.1.3 Rxpdo2on.m
% Starts the CANopen communication for a SDO
% telegram. The client sends an Download Request
% to the server (Rx-SDO).
% Writing to the Communication-Parameter RxPDO2
% (Object 1401) sets PDO2 active.

% Set variables
index_hex='1401';
subindex_hex='1';
data_down_hex=770;
CCS=1;
E=1;

% Convert variables to decimal
index_dec=hex2dec(index_hex);
subindex_dec=hex2dec(subindex_hex);

% Start simulation model
sim('DDC711_SDO');

disp(' ')
disp(' ')

disp(' ')

A.1.4 CheckRxPdo2on.m

% Starts the CANopen communication for a SDO
% telegram. The client sends an Download Request
% to the server (Rx-SDO).
% Reading from the Communication-Parameter RxPDO2
% (Object 1401) delivers the COB-ID PDO.

% Set output variables
index_hex='1401';
subindex_hex='1';
data_down_dec=0;
CCS=2;
E=0;

% Convert output variables to decimal
index_dec=hex2dec(index_hex);
subindex_dec=hex2dec(subindex_hex);

% Start simulation model
sim('DDC711_SDO');

% Display output variables
disp(' ')
disp('Comm.-Parameter RxPDO2')
disp('COB-ID PDO2')
disp(dec2hex(data_up_dec)) % display hex
disp(dec2bin(hex2dec(data_up_hex))) % display binary
disp(' ')

A.1.5 TxPdo2on.m

% Starts the CANopen communication for a SDO
% telegram. The client sends an Download Request
% to the server (Tx-SDO).
% Writing to the Communication-Parameter TxPDO2
% (Object 1801) sets PDO2 active.

% Set variables
index_hex='1801';
subindex_hex='1';
data_down_dec=642;
CCS=1;
E=1;

% Convert variables to decimal
index_dec=hex2dec(index_hex);
subindex_dec=hex2dec(subindex_hex);

% Start simulation model
sim('DDC711_SDO');
disp(' ')
disp(' ')
A.1.6 CheckTxPdo2on.m

% Starts the CANopen communication for a SDO telegram. The client sends a Download Request to the server (Tx-SDO).
% Reading from the Communication-Parameter TxPDO2 (Object 1801) delivers the COB-ID PDO.

% Set output variables
index_hex='1801';
subindex_hex='1';
data_down_dec=0;
CCS=2;
E=0;

% Convert output variables to decimal
index_dec=hex2dec(index_hex);
subindex_dec=hex2dec(subindex_hex);

% Start simulation model
sim('DDC711_SDO');

% Display output variables
disp('')
disp('Comm.-Parameter TxPDO2')
disp('COB-ID TxPDO2')
disp(dec2hex(data_up_dec)) % display hex
disp(dec2bin(hex2dec(data_up_hex))) % display binary
disp('')
A.1.7 TxPdoSync.m

% Starts the CANopen communication for a SDO
% telegram. The client sends an Download Request
% to the server (Tx-SDO).
% Writing to the Communication-Parameter TxPDO2
% (Object 1801) sets PDO2 to synchronous mode.

% Set variables
index_hex='1801';
subindex_hex='2';
data_down_dec=1;
CCS=1;
E=1;

% Convert variables to decimal
index_dec=hex2dec(index_hex);
subindex_dec=hex2dec(subindex_hex);

% Start simulation model
sim('DDC711_SDO');

disp(' ')
disp(' ')
A.1.8 CheckTxPdoSync.m

% Starts the CANopen communication for a SDO
% telegram. The client sends an Download Request
% to the server (Tx-SDO).
% Reading from the Communication-Parameter TxPDO2
% (Object 1801) delivers the transmission mode.

% Set output variables
index_hex='1801';
subindex_hex='2';
data_down_dec=0;
CCS=2;
E=0;

% Convert output variables to decimal
index_dec=hex2dec(index_hex);
subindex_dec=hex2dec(subindex_hex);

% Start simulation model
sim('DDC711_SDO');

% Display output variables
disp(' ')
disp('Comm.-Parameter TxPDO2')
disp('Transmission Mode')
disp(dec2hex(data_up_dec))
disp(dec2bin(hex2dec(data_up_hex)))
disp(' ')
## B Mapping tables

<table>
<thead>
<tr>
<th>TxPDO1</th>
<th>Input</th>
<th>07</th>
<th>06</th>
<th>05</th>
<th>04</th>
<th>03</th>
<th>02</th>
<th>01</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RxPDO1</th>
<th>Output</th>
<th>07</th>
<th>06</th>
<th>05</th>
<th>04</th>
<th>03</th>
<th>02</th>
<th>01</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### RxPDO2

<table>
<thead>
<tr>
<th>Output 11</th>
<th>Sign</th>
<th>MSB</th>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

| Output 11 | x    | x   | LSB | 0   | 0   | 0   | 0   | 0   |
| Bit       | 55   | 54  | 53  | 52  | 51  | 50  | 49  | 48  |

<table>
<thead>
<tr>
<th>Output 01</th>
<th>Sign</th>
<th>MSB</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

| Output 01 | x    | x   | LSB | 0   | 0   | 0   | 0   | 0   |
| Bit       | 39   | 38  | 37  | 36  | 35  | 34  | 33  | 32  |

<table>
<thead>
<tr>
<th>Output 10</th>
<th>Sign</th>
<th>MSB</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

| Output 10 | x    | x   | LSB | 0   | 0   | 0   | 0   | 0   |
| Bit       | 23   | 22  | 21  | 20  | 19  | 18  | 17  | 16  |

<table>
<thead>
<tr>
<th>Output 00</th>
<th>Sign</th>
<th>MSB</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<p>| Output 00 | x    | x   | LSB | 0   | 0   | 0   | 0   | 0   |
| Bit       | 7    | 6   | 5   | 4   | 3   | 2   | 1   | 0   |</p>
<table>
<thead>
<tr>
<th>Input 07</th>
<th>Sign</th>
<th>MSB</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
</tr>
<tr>
<td>Input 07</td>
<td>x</td>
<td>x</td>
<td>LSB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>Input 06</td>
<td>Sign</td>
<td>MSB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bit</td>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>Input 06</td>
<td>x</td>
<td>x</td>
<td>LSB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
</tr>
<tr>
<td>Input 05</td>
<td>Sign</td>
<td>MSB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bit</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>Input 05</td>
<td>x</td>
<td>x</td>
<td>LSB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>Input 04</td>
<td>Sign</td>
<td>MSB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bit</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Input 04</td>
<td>x</td>
<td>x</td>
<td>LSB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
C Literature

[1] CAN system engineering : from theory to practical applications
    Lawrenz, W,
    Springer-Verlag New York
    –
    1997

[2] CANopen implementation : applications to industrial networks
    Farsi, M.
    Research Studies Press Ltd.
    –
    2000

[3] System manual CANopen :
    SELECONTROL® MAS
    –
    Selectron Lyss Ltd.
    –
    1997

    –
    –
    Scientific Computers GmbH
    –
    1999

    O’Dwyer, A.
    Dublin Institute of Technology
    –
    2000

    Casey, T.
    CSC, University of Sunderland
    –
    1999

[7] Survey and selection of a local area network
    Doonan, A. F.; Farsi, M.; Cox, C. S.
    SCET, University of Sunderland
    –
    –