Umeå University
Department of Computer Science
Master's Thesis Project D, 20 credits

2006-03-20

J1939 – CANopen gateway

- A CANopen gateway according to DSP-413 -

Author: Adam Åström

Supervisors: Bertil Persson, CC Systems AB
Stefan Johansson, Umeå University

Abstract

Since the mid 90's computer communication has become more and more common in cars and trucks. CAN based networks with sensors transmitting small data packets are utilized in the automotive industry to operate and supervise vehicles' functionality. To ease communication several higher layer protocols for CAN based networks have been developed. In some applications it is necessary to exchange information between networks using different protocols, and by connecting the two networks to a gateway, the information is translated and forwarded and intercommunication is enabled.

This master thesis was conducted at CC Systems AB headquarters in Alfta. The thesis presents two higher layer protocols for CAN communication: CANopen and J1939. It also describes a standard for CANopen gateways, DSP-413. Finally the design and analysis of a software implementation according to the CANopen gateway standard is presented. The gateway is based on a CANopen stack from IXXAT and a minimal J1939 stack implementation. The target hardware for the gateway demonstrator was a Fujitsu MB90F543 microprocessor.

Acknowledgements

First I would like to thank Bertil Persson, my supervisor at CC Systems AB, for his support and guidance throughout the thesis. I would also like to thank Stefan Johansson at Umeå University for his valuable feedback and advice which helped improve the thesis report. Finally, I would like to send a big thanks to everyone at CC Systems AB who has contributed to the completion of this master thesis.

Abbreviations

CAL	CAN Application Layer
CAN	Controller Area Network
CiA	Can In Automation
COB	Communication Object
DCF	Device Configuration File
DS	Draft Standard
DSP	Draft Standard Proposal
ECU	Electronic Control Unit
EDS	Electronic Data Sheet
IVN	In Vehicle Network
NMT	Network Management
OD	Object Dictionary
PDO	Process Data Object
PDU	Protocol Data Unit
PG	Parameter Group
PGN	Parameter Group Number
RPDO	Receive Process Data Object
RTR	Remote Transmission Request
SAE	Society of Automotive Engineers
SDO	Service Data Object
TPDO	Transmit Process Data Object

Contents

List of Figures

List of tables

List of graphs

Introduction

Computer networks come in many different shapes and sizes. From the huge Internet with millions of computers connected worldwide, to small embedded systems. Different network and application areas have different demands on the communication. These varying perquisites have led to the development of many communication frameworks, called protocols. To be able to exchange information, all units in a network have to use the same protocol. If different protocols are used in the same network the data has to be translated to enable communication. This translation is done by a gateway.

1 Background

The Swedish company CC Systems AB's solutions are often based on distributed systems with several nodes cooperating to achieve the requested functionality. CC Systems often use Controller Area Network (CAN, see Section 2.2) for intercommunication between nodes in control systems and has transitioned from using their own protocol and started using the standardized CANopen protocol instead. CANopen is a widely used and flexible standard which can be adapted to many different applications.

Another protocol standard based on the CAN data link layer is J1939. The protocol is dedicated to diesel engine applications and has predefined messages for engine-, transmission- and brake data.

CC Systems customers often want to add extra equipment to their vehicles using a separate CANopen bus. To retrieve data from the vehicles' J1939 bus, the two buses have to exchange information although different protocols are used. This is accomplished by connecting the buses to a J1939-CANopen gateway which forwards and translates the information between the buses.

1.1 Purpose

The purpose of this master thesis project is to evaluate the truck gateway standard DSP-413 with respect to CC Systems' demands. Existing implementations of the standard are also evaluated with concern taken to performance, resource requirements and cost. If no satisfying software is found, an implementation of the standard should be conducted in C/C++.

1.2 CC Systems AB

CC Systems AB develops and delivers control and information systems for vehicles and machines in harsh environments. Their products range from small modules and controllers to custom made system solutions.

The company started in 1991 and has grown rapidly over the years. Today there are 130 co-workers located in five different offices in Sweden and Finland and among their customers you can find companies like John Deere, Atlas Copco and BAE Systems Hägglunds AB.

1.3 Report outline

Chapter 2 describes some fundamental terms for the thesis. Chapter 3 contains information about the flexible high-layer protocol CANopen and chapter 4 talks about the main features of the J1939 protocol. Chapter 5 presents the CANopen device profile for truck gateways, DSP-413. Chapter 6 contains information about the software tools and environment used during the development phase. Chapters 7 and 8 describes the design and functionality of the gateway implementation. Then chapter 9 presents the test result of the implementation and finally chapter 10 discusses the result, limitations and the future work of the implementation.

Theory

This section contains a short introduction to some basic terms which is followed by a presentation of two higher layer protocols for CAN communication, CANopen and J1939. Finally a walkthrough of the DSP-413 profile for truck gateways is presented.

2 Theory background

These chapters give a brief walkthrough of some important terms which have a central role in this master thesis.

2.1 Protocols

Protocols are used in many different contexts. For example, to greet a good friend with a hug or to introduce oneself to a stranger with a handshake are part of protocols which basically are agreements between the participating parties on how the communication is to proceed. Computer communication is also controlled by a specified set of rules which is called the communication protocol. A communication protocol is based on strict rules which determine exactly how two network units shall communicate. Many network protocols start the communication with an initialization phase where nodes exchange information about sequence numbers, keys, addresses, etc. This stage is often called the handshake phase and can be compared to human communication where participants often initialize the communication with some form of greeting, for example a handshake.

Communication protocols do not only specify how the actual communication is done, they also contain guidelines about medium access method, allowed physical topologies, types of cabling and speed of data transfers. Which protocol is best suited for a specific network depends on the communication perquisites: Is the communication wired or wireless? How high bit error rates can be tolerated? What data transfer speed should be used? Etc.

The various demands on the communication have led to the development of many different network protocols [12].

2.2 Controller Area Network

Controller Area Network (CAN) [19] is a network protocol which was introduced in 1986 by Robert Bosch and was originally designed for the car industry. Since data communication in cars often have many sensors transmitting small data packets, CAN features small data frames with sizes only up to 8 bytes. This means that several thousand of messages have to be sent per second to achieve the highest data rate, 1 Mbps.

CAN uses a large amount of overhead, which combined with a 15-bit CRC makes CAN very secure and reliable, and it is today one of the dominating bus protocols in vehicle and industrial controls[1]. Its rapid growth led to the establishment of the *CAN in Automation (CiA)* international users and manufacturers association in 1992 [2]. One of its first tasks was to specify a *CAN Application Layer (CAL)* [5]. Today they work with development and support of CAN based higher layer protocols [1].

2.3 Gateways

Today, many different network technologies exist which use different communication protocols and different hardware. To level out the differences and enable communication, gateways are often used to perform the necessary translation [12]. Gateways can also be used as entrance points

between networks using the same protocol. In this context the gateway's primary function is often to detach parts of the network to minimize the damage caused by a network breakdown. Moreover, computers controlling traffic at your company's network or at your local service provider also acts as gateways [17].

3 CANopen

CANopen [3] is a higher layer protocol for CAN based networks. It is an offspring from CAL (see Section 2.2) and it uses a subset of CAL services and communication protocols. In 1993, within the scope of Esprit project ASPIC, a European consortium led by Bosch started the development of a prototype of what would become CANopen. A first version of the CANopen communications profile, CiA DS-301, was released in 1995.

The CANopen specifications define different device, interface and application profiles as well as a framework for programmable devices (see Figure 1). It contains only a small set of mandatory and a lot of optional functionality, which makes it easy for system designers to customize it to fit a specific application [1].

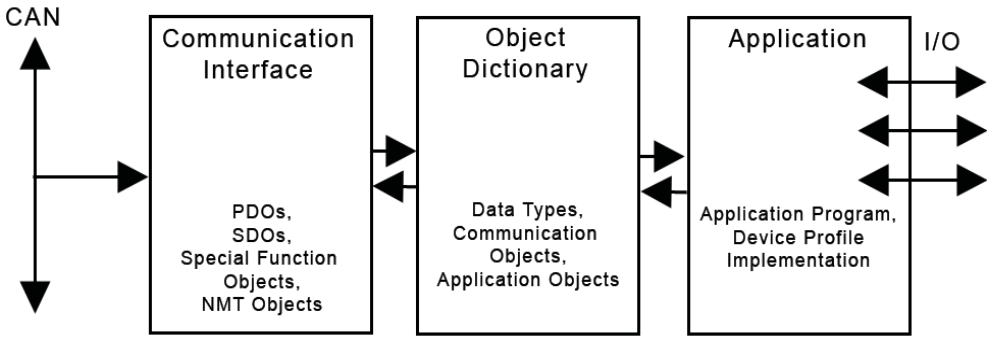


Figure 1. Schematic overview of a CANopen device [24].

3.1 Object dictionary

The central part of a CANopen device is the *object dictionary*. It is essentially a grouping of objects stored in a lookup table. It can be accessed from the network through a 16-bit index and an 8-bit subindex for individual data structure elements. The use of a 16-bit index gives the object dictionary 65 536 possible indices.

Index (hex)	Object
0000	Reserved
0001 – 001F	Static data types
0020 – 003F	Complex data types
0040 – 005F	Manufacturer specific data types
0060 – 007F	Device profile specific static data types
0080 – 009F	Device profile specific complex data types
00A0 – 0FFF	Reserved for further use
1000 – 1FFF	Communication profile area
2000 – 5FFF	Manufacturer specific profile area
6000 – 9FFF	Standardized device profile area
A000 – FFFF	Reserved for further use

Table 1. Object dictionary layout [5].

The object dictionary divides all indices into sections defining the entries (see Table 1). This solution makes it easy to structure data for the communication, but how do you know exactly which entry in the dictionary is used for what? Depending on the device functionality different entries needs to be implemented in the CANopen node. Some entries are mandatory and have to be supported by all nodes, while others are optional and can be added to extend a CANopen node's functionality.

The CANopen specifications define several device profiles that describe all the communication parameters and object dictionary entries supported by a specific type of CANopen module. A description of an object dictionary entry typically has these headings [1]:

- **Index** – The object dictionary index.
- **Object** – The object type (Variable, Array, Record etc.).
- **Name** – The name of the entry.
- **Type** – The data type (Integer16, Boolean, Unsigned32 etc.).
- **Access Attributes** – Read and write attributes.
- **Category** – Indicates if the entry is mandatory (must be implemented), optional (may be implemented) or conditional (must be implemented if certain other entries or features are implemented).

3.2 Service Data Object

To access the data stored in a node's object dictionary, a certain type of message, called *Service Data Object* (SDO), is used (see Table 2). Apart from accessing stored data, SDOs can also be used to write configuration data to a node's object dictionary, (e.g. setting device parameters, assign application data to PDOs, assign message identifiers, etc). This allows for a configuration tool to adapt a systems behaviour dynamically.

SDO communication follows the client/server model where every node implements a server to handle read/write requests to its object dictionary. The node that requests data is called the *client* [1].

Byte 0	**Byte 1-2**	**Byte 3**	**Byte 4-7**
SDO Command Specifier	Object Index	Object Subindex	– Up to 4 bytes of data (**expedited**) – 4 bytes byte counter (**segmented**) – Parameters regarding **block transfer**

Table 2. Basic structure of an SDO [5].

There are three different mechanisms for SDO transfers:

Expedited transfer – All transferred data is sent during the initialization phase with a maximum data payload of 4 bytes. Initialization message overhead describes the data segment to be transferred.

Segmented transfer – No data is sent during the initialization phase. The initialization message contains information about the data transfer. The transferred data is segmented into several packets which can contain up to seven bytes of data. Each data packet is acknowledged by the receiving node.

Block transfer – Used for large data amounts. Works like segmented transfer but with only one confirm message when all segments in a block are transferred. Each block can consist of up to 127 segments.

A SDO transfer may be aborted at any time by any of the two participating nodes by sending an SDO Abort message [5].

3.3 Process Data Object

SDOs provide access to all object dictionary entries in a node, but the large amount of overhead makes it inefficient for handling process data. A more efficient way to exchange process data is to use *Process Data Object* (PDO). PDOs correspond to objects in the device object dictionary and provide an interface to application objects.

There are two types of PDO, Transmit PDO (TPDO) and Receive PDO (RPDO). TPDO and RPDO are used for transmitting data from and receiving data to a device object dictionary. Their functionality and contents are described by a number of parameters stored at predefined indices in the object dictionary [1].

3.3.1 Mapping parameters

PDO mapping parameters are stored in the communication profile area in the object dictionary (see Table 1). The data content of each PDO is described by a mapping parameter (see Table 3). A PDO mapping parameter specifies the contents of a PDO by the index and subindex of the communicated object dictionary entries. The maximum amount of data transferred by a PDO is 8 bytes where each bit of the PDO mapping can be set individually [5].

Index	Sub-Index	Field in PDO Parameter Mapping Record	Data Type
0021h	0h	number of mapped objects in PDO	UNSIGNED8
	1h	1st object to be mapped	UNSIGNED32
	2h	2nd object to be mapped	UNSIGNED32
.....	.....	.....	.....
	40h	64th object to be mapped	UNSIGNED32

Table 3. PDO mapping parameter record specification [5].

3.3.2 Communication parameters

PDO communication parameters are stored in the communication profile area in the object dictionary (see Table 1). A communication parameter entry describes the communication capabilities of a PDO (see Table 4).

- **COB-ID:** The CAN identifier used by the PDO.
- **Transmission type:** How the PDO is communicated. PDO transmission can be synchronous, asynchronous, cyclic, remotely requested or event driven.
- **Inhibit time:** May be used to specify the minimum time between two PDO transmissions. Used to prevent starvation of low priority PDOs.
- **Reserved:** Reserved for future use.
- **Event timer:** This timer may be used to trigger events which cause transmission of the PDO.

Index	Sub-Index	Field in PDO Communication Parameter Record	Data Type
0020h	0h	number of supported entries in record	UNSIGNED8
	1h	COB-ID	UNSIGNED32
	2h	transmission type	UNSIGNED8
	3h	inhibit time	UNSIGNED16
	4h	reserved	UNSIGNED8
	5h	event timer	UNSIGNED16

Table 4. PDO communication parameter record specification [5].

The transfer of PDOs is performed with no protocol overhead because all information regarding PDO transmission and reception are stored in the object dictionary of the CANopen nodes. When a PDO is sent to the network it is assembled by the objects defined by the mapping parameters and sent according to the communication parameters. The receiver uses the same procedure which gives a transmission without overhead data [5].

3.3.3 Transmission

PDO transmissions conform to the producer/consumer relationship (see Figure 2). A TPDO is transmitted to the network by the PDO producer and received as a RPDO by zero or more PDO consumers. The PDO write service uses the push model where the producer performs an unconfirmed data transmission which is received by zero or more consumers; the data is pushed from the producer to the consumer(s).

A PDO read is initialized by a consumer sending a Remote Transmission Request (RTR) frame to the network. When the producer of the requested PDO acquires the RTR, it transmits the PDO and all consumers of the PDO receive it. This approach is called the pull model since the consumer pulls data from the producer.

The transmit trigger method decides when a PDO is sent to the network. In CANopen four different methods can be distinguished [5]:

- *Event driven*

The transmission of a PDO is triggered by the occurrence of an event. The definition of an "event" is usually defined by the device profile. A typical event is a change in the process data.

- *Timer driven*

If an event has not occurred within a specified time frame, an event is triggered by a timer and a PDO is transmitted.

- *Remotely requested*

A remote request is sent to a node, which transmits the requested PDO.

- *Synchronized*

The synchronized polling mode is intended for applications where it is crucial to get all inputs at the same time and apply all outputs at the same time. To achieve this, a SYNC signal is transmitted, usually by the network management master (see 3.4), on a fixed time basis. All synchronized nodes keeps their transmit buffer updated with the current data. Reception of a SYNC message triggers transmission of the data.

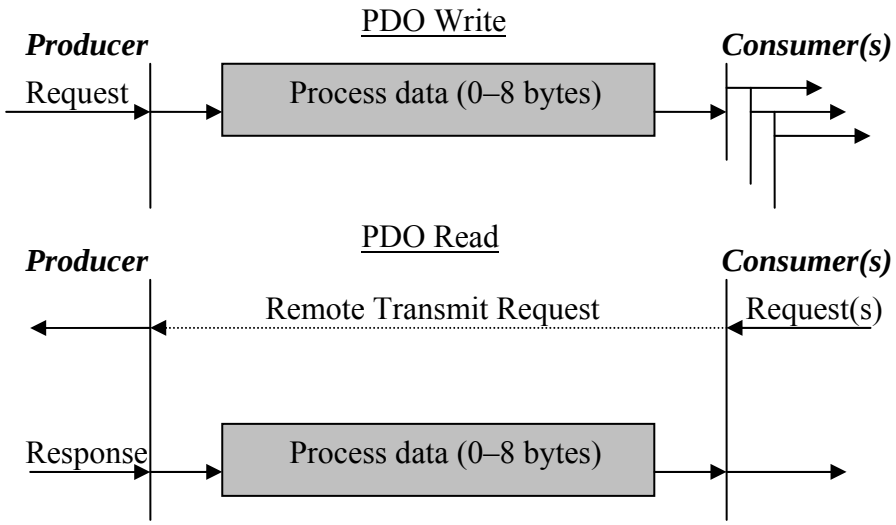


Figure 2. PDO read and write protocol [5].

3.4 Network Management

Network management (NMT) in CANopen follows the master/slave relationship. One device in the network acts as NMT master controlling the state of the slave nodes with NMT master messages. To realize the use of different operating states, every NMT slave has to implement a state machine.

The three major states of NMT slaves are:

- *Stopped* – No communication is allowed in this state except guarding and heartbeat (see Section 3.4.1).

- *Preoperational* – In this state SDO communication is possible and configuration of the node may be done. PDO communication is not allowed. A node automatically enters this state after initialization.

- *Operational* – In this state all communication is active. All PDOs are created according to mappings in the object dictionary.

When a node is started, CAN/CANopen interfaces are initialized and communication parameters are setup. When the initialization phase is completed the node sends a boot-up message and transits to preoperational mode. The boot-up message tells the NMT master that the node is initialized and in preoperational mode. When a node is initialized and in preoperational state it is the NMT masters task to control all state transitions [5].

3.4.1 Node monitoring

To detect node failures there are two different node monitoring techniques: node guarding and heartbeat. According to the CANopen specifications [5] one of the two has to be implemented.

Node guarding is a polling method where the NMT master polls all NMT slaves periodically and the slaves respond with their communication state. If a slave does not respond to the poll within a specified time window the NMT master signals a node guarding event to its application. The same goes for the NMT slaves which signal a life guarding event if its status was not polled within a specified time.

The other method is called heartbeat. With this technique all nodes signal their communication status with cyclic transmissions, heartbeats. Heartbeats are produced and consumed with certain time intervals called heartbeat producer/consumer time. The consumer time should be 1.5 – 2 times longer than the producer time to ensure that a missing heartbeat really is lost and not just delayed. Heartbeat messages can be monitored by all nodes in a network. If a heartbeat consumer does not receive a heartbeat within the consumer time, a heartbeat error is sent to the application.

3.4.2 Electronic Data Sheets and Device Configuration Files

Electronic Data Sheets (EDS) is a standardized way of expressing information about a CANopen device (see Figure 3). EDS files are primarily used by software tools to reduce the complexity of the planning, configuration and analysis process of a CANopen system.

The files have the same structure as Windows .ini files, using the ASCII coded ANSI character set and brackets to define different sections. An example of a mandatory section is the device information:

```
[DeviceInfo]

VendorName=Hurvas Ltd.
VendorNumber=156378
ProductName=E/A 74
ProductNumber=45670
RevisionNumber=1
OrderCode=BUY ME - 137/35/0615
LSS_Supported=0
```

```
BaudRate_50=1
BaudRate_250=1
BaudRate_500=1
BaudRate_1000=1
SimpleBootUpSlave=1
SimpleBootUpMaster=0
NrOfRxPdo=1
NrOfTxPdo=2
```

Device Configuration Files (DCF) and EDS files have almost identical formats, but instead of storing structural information, it describes the concrete incarnation of the device configuration and stores, for example, values of Object Dictionary entries (see Figure 3). The data stored in a node's EDS and DCF reveal both structure and contents of its Object Dictionary. This information enables software tools to monitor, analyze and configure CANopen nodes [14].

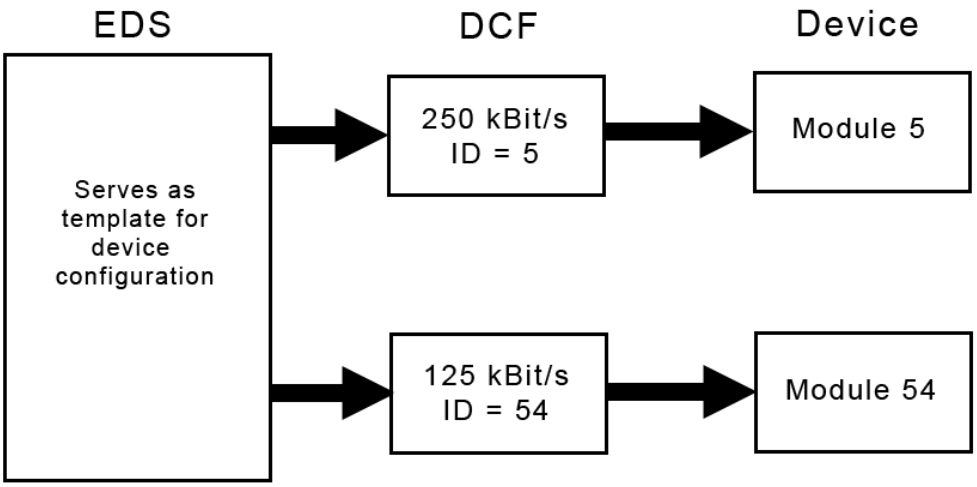


Figure 3. EDS and DCF overview [23].

3.4.3 Device profiles

CANopen device profiles are included in the standard as additional documents to avoid the need to further modify and enhance existing standards. They describe how a specific CANopen device is used, what communication parameters are available and how the object dictionary is set up. This guarantees interoperability between devices which comply with a device profile. There are a number of different device profiles, describing various devices such as I/O modules, encoders, battery chargers etc [1].

4 J1939

J1939 is a CAN based, diesel engine dedicated application profile used mainly in trucks. It was developed by the Society of Automotive Engineers (SAE) in the early 90's and is designed to support real-time closed-loop control functions in distributed in-vehicle networks [4]. The use of CAN permits any *Electronic Control Unit (ECU)* to transmit a message whenever the bus is idle. Messages have different priorities defined by their identifier. This allows message arbitration; if several ECUs are attempting to transmit simultaneously the message with the highest priority prevails [18].

4.1 Message format

J1939 CAN messages are called *Parameter Groups (PG)* because each message consists of a number of parameters defined in the protocol specifications [16]. PGs use the CAN extended frame format with a 29-bit identifier. To organize the information contained in the message, J1939 utilizes a framework called *Protocol Data Unit (PDU)*. A PDU consists of seven data fields where six of them are contained in the 29-bit identifier and the last field contains the message data (see Figure 4). Each PG is described with the PDU framework and uses one CAN data frame [18].

Field name	Priority	Reserved bit	Data Page	PDU Format	PDU Specific	Source address	Data
Bits	3	1	1	8	8	8	<= 8 bytes

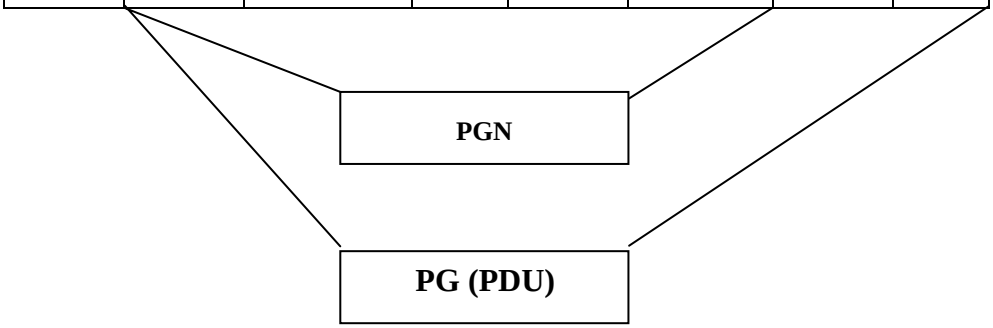


Figure 4. J1939 message format [18].

4.1.1 Parameter Group Number

Every PG has a unique identifier called *Parameter Group Number (PGN)* assigned to it, which is determined by the priority, reserved bit, data page, PDU format and PDU specific fields in the message header (see Figure 4 and Table 5).

Message header field	Field value
Priority	0x06
Reserved bit	0x00
Data page	0x00
PDU Format	0xFF
PDU Specific	0x01
PGN	**0x18FF01**

Table 5. PGN determination

PGN is used to identify the information contained in one or more CAN data frames. Each PGN describe one or more parameters where a parameter is a piece of data, like an engines revolutions per minute [18]. All PGNs and their data contents are specified in the SAE J1939 documents [16].

Priority
A message priority is determined by the first three bits of its identifier where 0 is the highest priority and 7 is the lowest. Each PGN has a recommended priority and messages with time critical information gets higher priority. By using message priorities transmission latency can be optimized.

Reserved bit
This bit is reserved for future use by the SAE. By default it shall be set to zero for all transmitted messages.

Data page
The data page bit works as a page selector. Page 0 contains all presently defined messages and page 1 is intended to provide extra capacity when all PGNs available in page 0 are used.

PDU Format
This 8-bit field determines the message PDU format. There are two different PDU formats, PDU1 and PDU2, which one is used is determined by the value of the PDU Format field. PDU1 is used between 0 and 239 and PDU2 between 240 and 255. The PDU format tells the message receiver how the value of the PDU Specific field should be interpreted (see 4.1.6). Since the PDU format is determined by the four most significant bits of the PF field, the four least significant bits can be used in conjunction with the PS to describe up to 4096 PGs per data page. The PDU1 format provides another 240 parameter groups, which gives a total of 8672 available parameter groups.

PDU Specific
The interpretation of this data field depends on the PDU Format. PDU1 implies a *Destination address* and PDU2 implies a *Group extension.* A Destination address specifies to which device the message is sent and all other devices ignore this message, but if it contains a Group extension the

message is broadcasted to all nodes. The Group Extension field extends the number of available parameter groups.

Source Address
The Source Address field contains the unique address of the device sending the message. It is determined through a handshake procedure during the node initialization phase. Since each device has a unique address this also assures that every CAN identifier becomes unique, which is required by CAN.

Data Field
The Data Field contains 0 to 1785 bytes expressing a given PG. When eight bytes or less are needed, a single CAN frame can be used. If the data amount is larger than eight bytes the data needs to be packetized and sent using several CAN data frames. To send multipacket PGs the transport protocol is used to transfer data in a series of CAN data frames [18].

4.2 ECU Identification

Each node in a J1939 network is an ECU. To identify and separate nodes, at least one *NAME* and one address is assigned to each ECU [18]. The *NAME* is a 64-bit entity which describes the functionality of the ECU and since it is unique it can also be used in address arbitration. The functionality of the ECU is described by a number of fields in the *NAME* entity. Many of the field-values are predefined and contain information about the manufacturer, the vehicle system and the industry group, while other fields provide information about the ECU functionality.

Each ECU can be uniquely identified by its *NAME* entity, but with a length of 64 bits it is not practical to use it as an identifier. Therefore an 8 bit address is utilized to provide uniqueness to message identifiers and to determine the source of a message. To minimize the number of ECUs trying to claim the same address, preferred addresses are used. This means that each ECU has a list of addresses it intends to use and announces these to the network. All other ECUs on the network compare this address to a table of addresses which are already in use. If multiple ECUs claim the same address, the one with the lowest *NAME* value gets to use the address.

An ECU can also determine its address by sending a request for all the addresses claimed in the network and then selecting an available address from its list of addresses [15].

4.3 Message transmission and reception

J1939 message transmission can be destination specific or broadcasted. Destination specific messages are directed to a specified ECU while broadcast messages are received by all nodes which then determine whether the message is relevant or not [18].

5 DSP-413 truck gateway specification

The device profile that is investigated and implemented in this thesis is the DSP-413 device profile for truck gateways. The profile describes gateways to connect CANopen networks to CAN based In Vehicle Networks (IVN), such as J1939. The specification consist of six parts:

- Part 1: General definitions and default communication objects [6].
- Part 2: Application objects for braking and running gear [25].
- Part 3: Application objects for equipment other than brake and running gear [26].
- Part 4: Application objects for diagnostics (not released).
- Part 5: Application objects for superstructure [27].
- Part 6: Framework for J1939 based networks [7].

The standard mainly describes where and how application objects should be mapped into the object dictionary. These objects can be read from another device by using SDO communication, or they can be mapped into TPDOs and sent to the network. SDO communication can also be used to configure the gateway by writing to its object dictionary.

5.1 Object dictionary definitions

To be compliant with the device profile the gateway share the object dictionary entries from 6000_h to $6FFF_h$. Each device only implements those objects relevant to its function and unused objects are replaced with dummies.

5.2 Error handling

Internal device errors shall be reported to the network via Emergency Messages. These shall be assigned the highest possible priority to ensure quick bus access. Emergency Messages contain predefined error numbers which give information about the occurred error.

When a serious device failure is detected the default behaviour for the gateway is to enter the preoperational state, but the device can also be configured to enter the stopped state or remain in the current state.

5.3 Framework for J1939 based networks

The framework for J1939 based networks is described in part 6 of DSP-413 [7]. This part of the specifications is extra interesting for this master thesis, since it describes the requirements and functionality of a J1939-CANopen gateway.

From the view of the IVN, the gateway shall be a J1939-compliant ECU according to the standard behaviour given by the SAE J1939 specification [18]. Some mandatory object entries are stored in the OD and provide the gateway name and address as used in the J1939 network. These entries can be accessed from the CANopen side.

The supported PGs used to communicate with the J1939 network are described by record objects. Each record defines the structure of the PG parameters (see Table 6).

Value	Description
enable	TRUE if PG is enabled.
	FALSE if PG is disabled
pg_state	Received PGs: Contains the number of received bytes.
	Transmitted PGs: Content depends on transmission_type.
pdu_format	Contains the PDU Format value.
pdu_specific	Contains the PDU Specific value.
data_page	Contains the Data Page value.
source	Contains the senders device address.
priority	Contains the message priority.
transmission_type	Cyclic transmission: The PG is sent cyclically with the transmission_rate time.
	Cyclic reception: The node monitors the reception time of the PG and notifies the application if a time_out occurs.
	Asynchronous transmission: The application decides when the transmission shall be started.
	Asynchronous reception: Timeouts are not reported.
	BAM: Transmission and reception is set up using the BAM transport protocol.
	CMDT: Transmission and reception is set up using the CMDT transport protocol.
data_reference	A reference to a location in the process picture where the transferred data shall be stored.
transmission_rate	The rate in which the PG shall be transmitted or received.
time_out	The timeout value given in ms.
max_data_length	Maximum length of the PG data.

Table 6. PG record description [6].

Development

These chapters provide a detailed description of the gateway implementation. It also contains information about the software tools used during the development and testing phase and furthermore some information about the hardware for the demonstrator.

6 Environment

The software was developed under Microsoft Windows XP. Microsoft .Net 2003 IDE was used to implement a simulated gateway version which later on was ported to the hardware evaluation board.

6.1 Development tools

During the development process several different software tools were used to ease implementation and testing.

6.1.1 Microsoft Visual Studio .NET 2003

Microsoft Visual Studio .NET 2003 [20] is Microsoft's complete set of development tools for developing everything from ASP Web applications to standard C console applications. This tool was used for the main development of the gateway software.

6.1.2 CanTool

CanTool is part of CC Systems simulation platform, CCSimTech [10]. CanTool is used to simulate and monitor data traffic on the CAN bus and was frequently used during the development and testing phase.

6.1.3 Softune workbench

Softune workbench [8] is a combined integrated development environment (IDE) and debugger for Fujitsu microprocessors/microcontrollers. Softune was used to port the gateway software to the demonstrator hardware and implement hardware specific functionality.

6.1.4 Accemic MDE

Accemic MDE [21] is a very powerful debugger environment for embedded C applications. It came in handy during the software porting and testing phase and many hardware specific issues could easily be found and solved thanks to its advanced features. Some of the features are:

- Execution control with single step, step into and step over.
- Breakpoints.
- Watch windows for local and global variables.
- Processor visualization with symbolic and numeric presentation of the processor's register fields.

6.1.5 Fujitsu Flash-Kit Programmer 2.9

Fujitsus serial programming tool was primarily used to write configuration data to the flash. Since the microcontroller's flash memory is divided into sectors, the gateway software code and configuration data can be written separately to different memory areas. This ability enables new configuration data to be written to the flash without interfering with the program code, which makes it easy to change the gateways behaviour.

6.2 CANopen stack

To get the CANopen functionality a CANopen stack had to be chosen and integrated with the gateway software. There were two available stack implementations: one from Embedded Systems Academy (ESA) called MicroCANopen and one from IXXAT.

6.2.1 MicroCANopen

MicroCANopen [9] is a minimal CANopen implementation which lacks some functionality of the CANopen standard. One big disadvantage with this stack is that it does not implement an object dictionary. Because the stack does not implement all the mandatory functionality of the CANopen standard, nodes based on this stack are not CANopen conformant.

The advantages with this implementation are the low resource requirements and that it is easy to use and quick to learn. If functionality demands can be fulfilled, MicroCANopen is a good choice for applications where code size and memory usage is critical.

6.2.2 IXXAT CANopen Master/Slave software

IXXAT CANopen stack implementation [11] is more extensive than MicroCANopen and has more CANopen functionality. The additional functionality increases the resource demands, but the stack can be configured to remove unnecessary functionality and thereby reduce the stack size. The increased functionality also contributes to a more complex implementation. A detailed description of the CANopen stack implementation is given in Chapter 7.1.1.

6.2.3 Gateway implementation

For the gateway software implementation IXXAT CANopen Master/Slave software was used. The main reason why MicroCANopen was not chosen was the fact that no object dictionary was available, but the stack also lacks some other important functionality required by the gateway.

6.3 Hardware

To be able to choose a microprocessor for the task, several different aspects had to be considered. One requirement was that it had to have two CAN interfaces to be able to work as a gateway. The memory demands as well as CPU performance were hard to determine since the software only had been tested in a PC simulated environment. The CANopen stack software was configured to minimize memory requirements and the gateway software resource need was estimated through calculations and measurements.

My external supervisor came up with some suggestions of suitable hardware for the demonstrator and finally we decided on an evaluation board from Accemic based on a Fujitsu microprocessor.

6.3.1 DIMM-line MB90F543 Evaluation-Board

The evaluation board [13] (see Figure 5) from Accemic is based on a Fujitsu MB90F543GS microprocessor [22]. It is a 16-bit microprocessor which runs at 16 MHz, has dual CAN interfaces, 6 kB internal RAM and 128 kB internal flash memory. The evaluation board has 128 kB external SRAM and also 128 kB external flash memory. These resources are only partially used by the gateway software. The intention with the extra resources was to ensure that the gateway could be put into operation and later on more accurate resource requirements could be calculated (see Section 9.3).

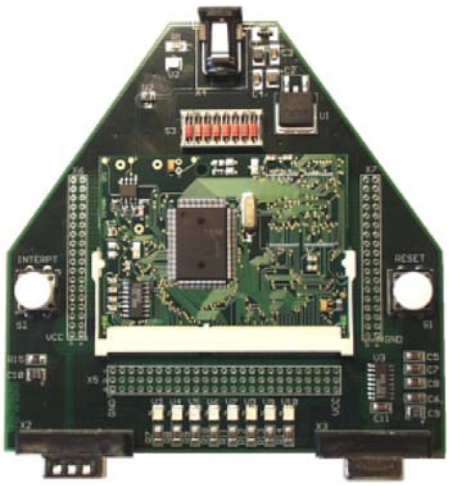


Figure 5. DIMM-line MB90F543 evaluation board.

7 Design

The software implementation has a modular design in two levels. At the higher level the software can be divided into the CAN driver, CANopen stack and gateway software. The internal design of the CANopen stack and gateway software is also divided into modules.

7.1 CANopen stack

The CANopen protocol software has a modular structure (see Figure 6) where all CANopen services are separated into modules in the source code. Not all modules were used by the gateway implementation.

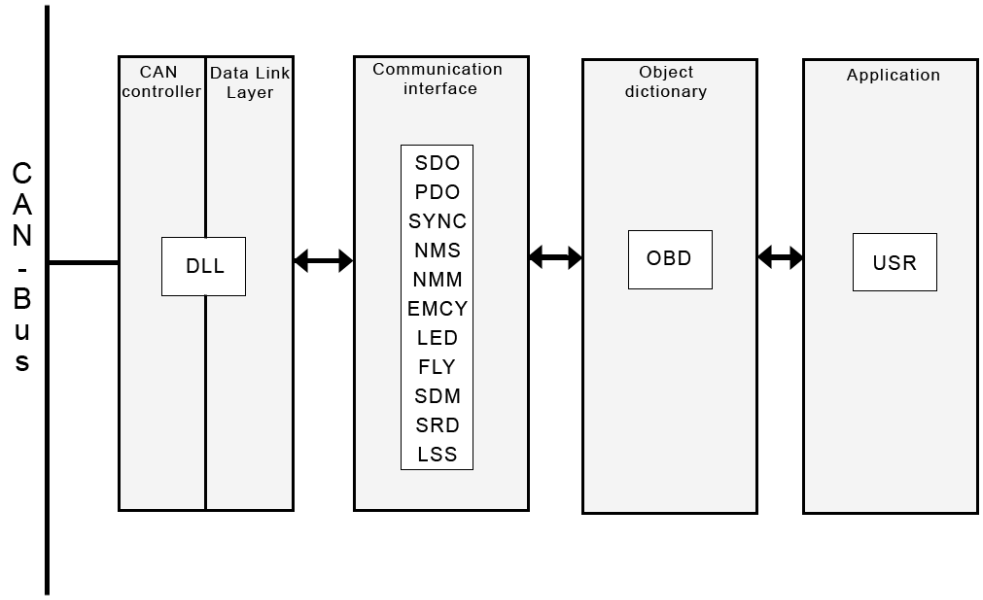


Figure 6. CANopen software structure [11].

The object dictionary is a central part of the protocol software and all modules have access to it. The stack functionality can easily be configured via defines in header files to fit the application requirements.

The following sections present an overview of the core modules of the CANopen stack implementation.

7.1.1 CANopen module

The CANopen module is a central part of the protocol software which is responsible for message distribution to other modules. The core function, *COP_Main(),* handles message distribution and has to be cyclically processed with a frequency adapted to the expected bus load. This prevents overflow of the receive queue and thus the loss of CAN messages.

7.1.2 Object Dictionary module

The Object Dictionary (OBD) module handles communication and management of the object dictionary. It provides the user with internal

access functions for reading from and writing to entries in the object dictionary. The files *obd_app.c* and *obd_com.c* define the structure and content of the object dictionary. These files are part of the application and must be altered by the user to fit the application requirements.

7.1.3 Service Data Object module

The Service Data Object (SDO) module implements functionality for server and client SDOs. Server SDOs enable nodes to download (read) and upload (write) data from/to each others object dictionaries. All SDO request messages are processed by *COP_Main()* and then passed on to the SDO module. To enhance user control, every application object access via SDO triggers a callback function. Parameters sent to the function inform the user which object was accessed and if it was a read or a write access.

7.1.4 Process Data Object module

The Process Data Object (PDO) module handles all PDO communication. Receive PDO messages are first assembled by the core function *COP_Main()* and then forwarded to the PDO module, where PDO data is written to the corresponding object and the defined callback function is called. For asynchronous PDOs the callback is called immediately upon reception, but for synchronous PDOs the callback is not executed until the next SYNC object is received.

When data of an application object has been changed, the application has to inform the PDO module about the change. The module sets up affected PDOs to be transmitted according to their transmission type. The actual transmission is performed by *COP_Main().*

7.1.5 Synchronization module

The Synchronization (SYNC) module manages tasks combined with synchronous PDOs and is integrated with the PDO module. The function *PDO_ProcessSync()* is called upon reception of a SYNC object. It first executes the callback functions of all received synchronous PDOs, then the application is informed about the reception of the SYNC object through the callback function *USR_SyncIndication().*

7.1.6 Emergency module

The Emergency (EMCY) module is used by the application to send emergency messages when internal errors occur. The module contains functionality for transmitting (producer) and receiving (consumer) emergency messages.

To transmit an emergency message the function *EMC_SetEmegency()* writes the message into the predefined error field under index 1003h and updates the error register under 1001h which triggers the transmission. All signalled emergency messages are contained in the predefined error field in a chronological order.

Received emergency messages are buffered in the emergency queue and can be accessed from the application with the function *EMC_GetEmergencyObj().*

7.1.7 Network Management Slave module

The Network Management Slave (NMS) module implements mechanisms required for an NMT slave:

- Network management
- Node guarding
- Heartbeat producer
- Heartbeat consumer

It also informs the application of node events and transmits network messages to the application via the callback functions *NMS_Event()* and *NMS_HBEAT_Event()* [11].

7.2 Gateway implementation

A description of the functionality of the gateway software modules is given below.

7.2.1 J1939

This module handles the initialization and communication with the J1939 bus. It implements some basic functionality of the J1939 protocol to fulfil the gateway specification. The main function is *J1939_Process().* It processes incoming messages and updates the process image with the received data. It also checks for messages ready to be transmitted and assembles necessary information and transmits it to the network.

7.2.2 CANopen

This module works as an abstraction layer to the CANopen stack. It initializes CANopen communication by the *CO_Init()* function. Its core function is *CO_Process()* which utilizes the CANopen stack through *COP_Main().*

7.2.3 OD

The primary task for this unit is to set up the object dictionary according to the configuration data, which is written to a specific address in the flash memory. To set up the OD configuration, data has to be mapped or copied to certain data structures used by both the CANopen stack and the J1939 module. These structures specify entries in the communication, manufacturer and standardized device profile area and define the shared memory area as well as PDO and PG mappings.

7.2.4 Timer

This module uses the hardware timer to provide timer functionality. Timers are mainly used by the J1939 module to be able to transmit cyclic PGs and trigger timeout events.

7.3 CAN driver

To utilize both CAN interfaces on the microprocessor a CAN driver had to be implemented. Since a CAN driver for a Fujitsu microprocessor with one CAN interface was available, the easiest and quickest way was to extend the existing driver with support for double CAN interfaces.

7.4 Problems

Software design is an iterative process where changed prerequisites and unforeseen events affects the development process and inflicts design changes. This implementation is not an exception and during the process the basic design has been changed several times, but the main idea with a modular structure remained unchanged.

The initial design was based on a C++ implementation. By the time the demonstrator hardware was chosen I discovered that no C++ compiler was available in the IDE for the selected hardware. This led to major design changes and rewriting of the parts which had already been implemented in C++.

One of the toughest challenges was how the external configuration should be realized. The solution with memory mapped data structures required that changes to the CANopen stack software were made. Data structures for storing J1939 data also had to be designed.

The final solution still conforms to the initial modular design, but the internal design of the modules has been changed along the way.

8 Functionality

This chapter describes the gateway functionality with a use-case scenario.

8.1 Use-case scenario

Below is the flow of a typical scenario from the initial phase, where the gateway is configured to suit user demands, until the gateway is put into operation forwarding data between the two networks described. All numbers refer to the numbers in Figure 7.

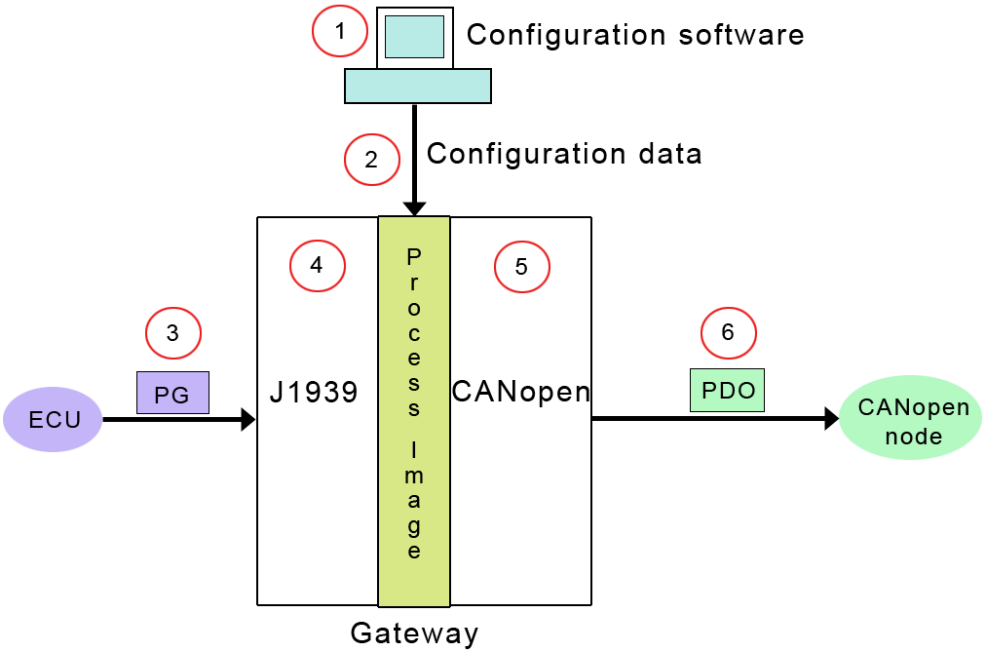


Figure 7. Use-case scenario

1. The gateway user configures the gateway with the configuration software. He/she sets up the parameter group and PDO mappings according to the gateway requirements. When the configuration is done the configuration data is written to a binary file.

2. The configuration data file is written into the gateway ROM. When the gateway is started the configuration data is mapped to data structures describing the gateway functionality and parameter mappings.

3. A PG is sent from an ECU on the J1939 bus. The unique identifier, the PGN, is included in the CAN message header.

4. The PG is received by the gateway. The J1939 module checks its PG mappings to see if the received PG has a mapping. If it has a mapping the received data is written to the shared data memory, the process image. After writing the data the J1939 module signals the CANopen module, telling it that it that the process image data was updated.

5. The CANopen module checks its PDO mapping parameters to see if a PDO with the new data should be sent. If a PDO containing the updated data exists it is sent to the CANopen network according to its communication parameters.

6. The PDO is transmitted to the CANopen network with the COB-id specified by its communication parameters and the data content received from the PG.

8.2 Gateway configuration

One big issue during the design phase was how the gateway should be configured to be as dynamic as possible. After studying other solutions and discussing with my external supervisor, we agreed that it should be programmable through external software from a PC.

The next problem was how to set up the CANopen stack software to get the object dictionary dynamically configurable. The object dictionary used in the CANopen stack software is based on static structures defined in header files and is set up at compile time. To be able to change the behaviour of the CANopen stack from the static configuration the object dictionary had to be set up in runtime.

9 Test results

In this chapter the tests and measurements conducted on the gateway implementation are presented.

9.1 Test environment

To achieve a realistic test environment an entire system would have to be set up with several CANopen nodes on one side and some kind of diesel engine consisting of many ECUs on the other. This however was not feasible and all tests have been done in a simulated environment. Simulated ECUs transmitting J1939 messages according to the J1939 specifications and a simulated CANopen node transmitting PDOs were used to create a test scenario that was as realistic as possible.

As seen in Table 7 and Table 8, the gateway was configured to receive ten different parameter groups from the IVN and three parameters from the CANopen network. The received parameter groups were then mapped into PDOs. All implemented transmission types (TT_SYNC, TT_ASYNC, TT_BAM) are also represented in the test scenario. Because PDO transfers only support up to 8 bytes of data, the *Trip fan information* (16 bytes) had to be verified by reading the data from the data reference object of the PG with SDO and comparing it to the data sent by the ECU. All parameter group definitions refer to the J1939 Vehicle Application Layer document [16].

Parameter	Transmission rate (ms)	Data length (bytes)	PGN	PDO COB-ID
EEC1	20	8	00F004	100
Intake manifold information	500	8	00FEA6	101
Engine temperature #2	1000	8	00FEA4	102
ETC1	10	8	00F002	103
Vehicle position	5000	8	00FEF3	104
Fuel information	On request	8	00FEB3	105
ETC6	On request	8	00FEAB	106
TC1	50	8	100	107
EBC1	100	8	00F001	108
Trip fan information	On request	16	00FEBB	-

Table 7. IVN to CANopen test parameters.

Parameter	Transmission rate (ms)	Data length (bytes)	PGN	PDO COB-ID
Request PGN	-	3	00EA00	200
Retarder configuration	5000	19	00FEE1	201
Transmission configuration	-	8	00FEE2	202

Table 8. CANopen to IVN test parameters.

9.2 Real time properties

Message transmission latency was measured to verify the real time properties of the gateway implementation. This was done using a test program which simulated a J1939 ECU and sent a CAN message (parameter group) from the IVN which was forwarded by the gateway to the CANopen node on the CANopen network (see Figure 8, broken lines). Messages were also sent from the CANopen network to the IVN (see Figure 8, solid lines).

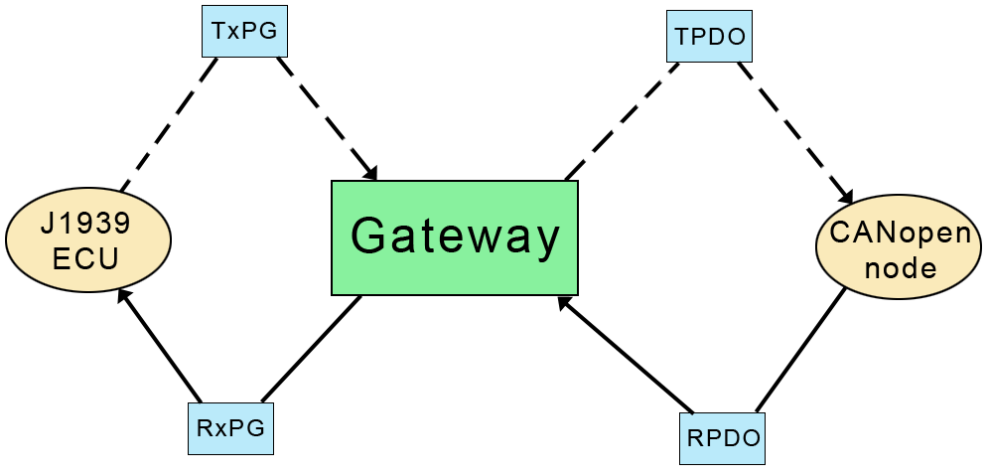
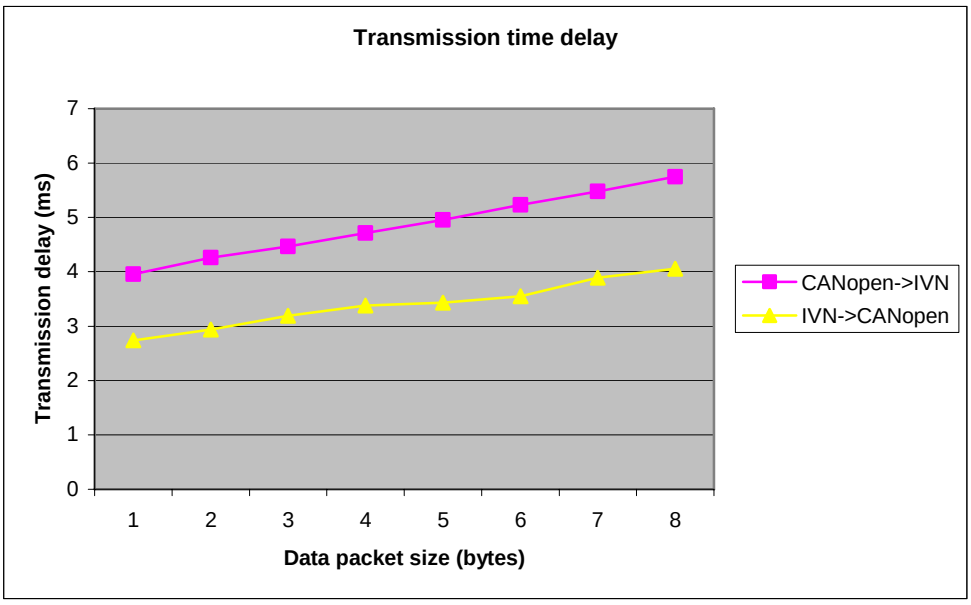


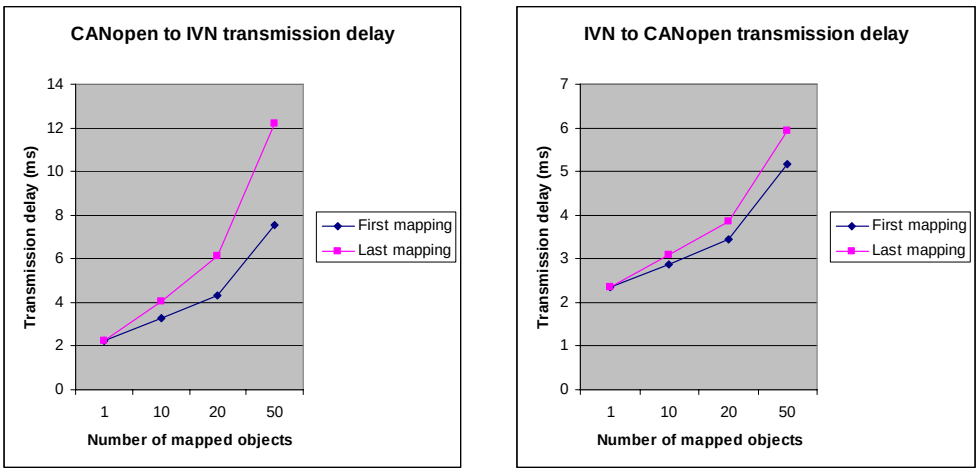
Figure 8. Message transmission

The transmission delay from one network to the other was measured depending on packet sizes (see Figure 9) and the number of mapped objects (see Figure 10).



Graph 1. Transmission delay as a function of data packet sizes.

The increase of the average transmission delay induced by larger data packets is 0.17 ms/byte for IVN to CANopen and 0.22 ms/byte for CANopen to IVN. The increased transmission time is generated from handling the additional data which has to be forwarded in every step of the transmission chain.



Graph 2. Transmission delay as function of the number of objects.

The transmission delay increases with the number of configured PGs and PDOs because the size of the object dictionary increases. This renders an increased search time. The lower line (first mapping) in Graph 2 represents the object with the shortest search time and the upper line (last mapping) represents the longest search time (worse-case).

When the gateway is pushed hard, transmission delay times increase even more. In a test run with 50 PGs, which were transmitted cyclically, the transmission delay rose with approximately 300%. This test-case is an extreme scenario which shows that the CPU performance in extreme cases is insufficient which causes increased delays.

Another time aspect that was investigated during the test phase was the execution time of the program's main loop. To achieve the wanted behaviour from the CANopen stack its main function, *COP_main()*, has to be called with "*a frequency adapted to the expected bus load*" [11].

These measurements were done by measuring the execution time for the J1939 module and the CANopen module. The results showed that processing the J1939 side of the gateway was up to 20 times slower than the CANopen processing. After extensive testing the bottleneck was discovered and some minor changes in the solution decreased the execution time to the same level as the CANopen module.

9.3 Memory requirements

Another important issue was the software's memory requirements. Both the RAM and ROM memory usage varies depending on the number of PG and PDO mappings. All data regarding PGs, PDOs and their mappings are stored in data structures which are part of the object dictionary. To be able to estimate the memory required for each mapping the memory used by each configured PDO and PG had to be calculated. These calculations were based on the sizes of the data structures and the number of instances of these structures needed to describe a PG or PDO. With the estimated memory requirements it is possible to make an approximation of the number of PG and PDO which can be supported by the gateway given the memory specifications of the target hardware. Table 9 shows the ROM and RAM memory used by the PDO and the PG.

Mapping parameter	ROM	RAM
PG	460 bytes	320 bytes
PDO	50 bytes	40 bytes

Table 9. Mapping parameter's memory requirements.

Given these values a gateway implementation with 10 mapped PGs and 10 mapped PDOs would have the following memory requirements:

ROM: (10 * 460) + (10 * 50) ≈ 5 kB
RAM: (10 * 320) + (10 * 40) ≈ 3,5 kB

10 Conclusions

In this master thesis the two CAN based higher layer protocols J1939 and CANopen were described. CANopen is a commonly used protocol with a generic design which offers great flexibility. Its flexibility facilitates system adoption in order to fit various functionality demands.

The J1939 protocol is dedicated for diesel engine applications and the data communication is based on predefined messages. Message contents and identifiers are defined by the specification documents of J1939.

The report also gave a walkthrough of the CANopen gateway standard, DSP-413, and a description and evaluation of a software implementation in accordance with the standard specifications. The gateway standard defines a CANopen device profile which determines how the gateway node should interact with other nodes in the J1939 and CANopen networks. It also specifies the data content and structure of the device specific object dictionary entries. Since the gateway implementation is based on a CANopen standard, all core functionality is based on information stored in the object dictionary. Utilizing a CANopen device profile facilitates the basic design process when several information structures are already defined by the object dictionary. DSP-413 do not only describe the information content of the object dictionary, but also how to make use of the information to achieve gateway functionality.

The CANopen device profile for truck gateways has not yet reached the market, which I discovered during the evaluation process. There were no available software implementations of the standard and the impression I got from the vendors I contacted was that they had no intention of implementing the standard in any of their CANopen gateway products. I imagine that an implementation with equal performance and functionality would be easier to realize without the requirements of the device profile. Furthermore, in small system specific cases where a generic implementation is not necessary, the most efficient way is to customize the solution to the system requirements. CANopen device profiles are adapted to the CANopen standard in a well planned manner and utilize the central parts of CANopen very efficiently. The well thought-out design compensates for the additional implementation work and for a generic gateway product I think the final result is improved by using the device profile.

10.1 Limitations and future work

To be able to complete the thesis on time, the work amount was decreased by excluding the J1939 multipacket transfer protocol CMDT support from the implementation. However, CMDT data transfers are not frequently used so this lack of functionality does not affect the final result mentionably.

Future work with this prototype is to evaluate alternatives to the external configuration solution. One option to configure PDOs is to use dynamic mapping which is supported by the CANopen stack software. With this method PDO mappings can be changed during runtime. Furthermore,

scaling of the CANopen stack should be more carefully investigated to minimize memory requirements further.

References

[1] O. Pfeiffer, A. Ayre, C. Keydel: *Embedded Networking with CAN and CANopen*, RTC Books, ISBN 0-929392-78-7, 2003.

[2] CiA homepage, http://www.can-cia.org, 2005-11-03.

[3] CANopen homepage, http://www.can-cia.org/canopen/, 2005-11-08.

[4] J1939 homepage, http://www.can-cia.org/j1939based/, 2005-11-09.

[5] CAN in Automation, *CANopen Application Layer and Communication Profile*, CiA Draft Standard 301, CAN in Automation, Version 4.02, 13 February 2002.

[6] CAN in Automation, *CANopen Device Profile for truck gateways, Part 1: General definitions and default communication objects*, CiA Draft Standard Proposal 413-1, Version 2.0, December 2003.

[7] CAN in Automation, *CANopen Device Profile for truck gateways, Part 6: Framework for J1939-based networks*, CiA Draft Standard Proposal 413-6, Version 1.0, December 2003.

[8] Fujitsu semiconductor, *F^2MC^{TM}-16 Family Softune^{TM} Workbench user's manual*, 2004.

[9] MicroCANopen homepage, http://www.microcanopen.com/, 2005-10-25.

[10] A. Möller, Product description CCSimTech, 3 December 2004.

[11] IXXAT, *CANopen Master/Slave Software, CANopen Protocol Stack for Embedded Solutions*, Software version 4.10.

[12] A. S. Tannenbaum: *Computer Networks 3rd edition*, Prentice Hall PTR, ISBN 0-13-349945-6, 1996.

[13] R. Harthaus, *Getting started with Accemic MDE and the DIMM-line MB90F543 Evaluation board*, Revision 1.0.

[14] CAN in Automation, *CANopen Electronic data sheet specification for CANopen*, CiA Draft Standard 306, Version 1.3, January 2005.

[15] SAE, J1939-81, *Recommended practice for serial control and communications vehicle network - part 81 - network management*, July 1997.

[16] SAE, J1939-71, *Vehicle Application Layer*, October 1998.

[17] Gateway definition, http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212176,00.html, 2006-01-20

[18] SAE, J1939, *Recommended practice for a Serial and Communications Vehicle Network*, April 2000.

[19] Robert Bosch GmbH, *CAN specification 2.0*, September 2001.

[20] Microsoft Visual Studio .NET 2003, http://msdn.microsoft.com/vstudio/previous/2003/, 2006-03-18.

[21] Accemic MDE - Overview, http://www.accemic.com/modules.php?name=Products_AccemicMDE&pid=2, 2006-01-15.

[22] Fujitsu semiconductor data sheet, 16-bit Proprietary Microcontroller, F^2MC-16LX MB90540/G/545/G Series.

[23] CANopen overview, http://www.datamicro.ru/can/standards/pdf/CANopen.pdf, 2005-11-25

[24] H. Boterenbrood, *CANopen high-level protocol for CAN-bus,* Version 3.0, NIKHEF Amsterdam, March 2000.

[25] CAN in Automation, *Device profile for truck gateways, Part 2: Application objects for brake and running gear,* CiA Draft Standard Proposal 413-2, Version 2.0, December 2003.

[26] CAN in Automation, *Device profile for truck gateways, Part 3: Application objects for other than brake and running gear equipment,* CiA Draft Standard Proposal 413-3, Version 2.0, December 2003.

[27] CAN in Automation, *Device profile for truck gateways, Part 5: Application objects for superstructure,* CiA Draft Standard Proposal 413-5, Version 1.0, December 2003.