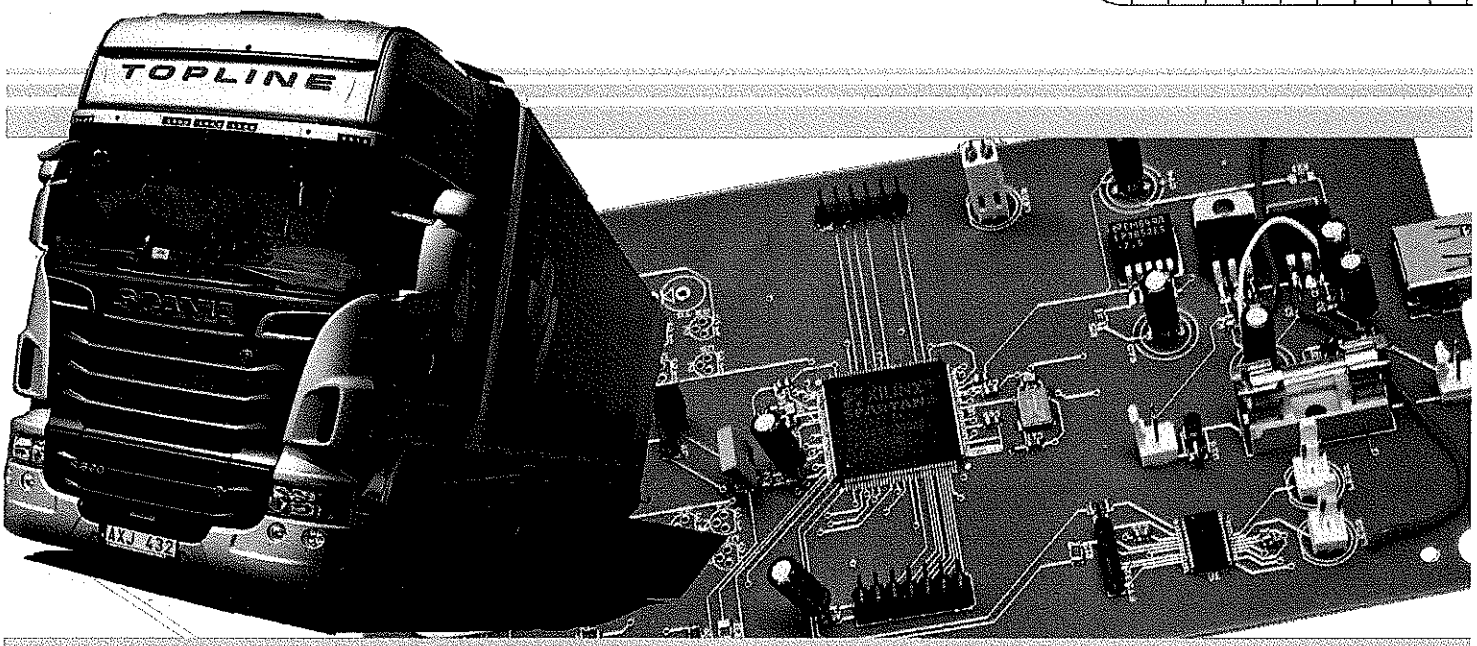
CHALMERS



CAN SAMPLE POINT ANALYZER

Implementing a CAN sample point testing methodology

Master of Science thesis

MATHIAS CARLSSON

Department of Microtechnology and nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, ~~2010~~ 2011

CAN SAMPLE POINT ANALYZER

Implementing a CAN sample point testing methodology

MATHIAS CARLSSON

Department of Microtechnology and nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2010

CAN SAMPLE POINT ANALYZER
Implementing a CAN sample point testing methodology
MATHIAS CARLSSON

Department of Microtechnology and nanoscience
Chalmers University of Technology
SE-412 96 Göteborg Sweden

Telephone + 46 (0)31-772 1000

Cover:
The cover includes a Scania R620 truck and in the background the final circuit board discussed in this thesis.

Preword:
Thanks to Henrik Persson who was my supervisor at Scania. I would also like to thank all of co-workers of the department at which the thesis was conducted for all their support during the project.

1 Abstract

The purpose of this master thesis project is to develop a method for verifying the sample point position (SPP) of an electronic control unit (ECU). The thesis is performed in two phases; first a theoretical solution of the problem and also a practical implementation. To realize the theory a circuit board were constructed. The solution of the problem is then tested and verified on the design PCB. To be able to validate the test result special test instrument was used such as an oscilloscope with a controller-area network (CAN) analyzer module.

To analyse the sample point position the method used takes advantage of the built-in error detection function, Stuff error. The Stuff error is a mechanism integrated in the CAN protocol and is provoked when it don't perceive an expected stuff bit in the message sent over the CAN bus. To generate a stuff bit error the bit length of the stuff bit is decreased until the receiver no longer can interpret it. When this occurs an error flag is set and by keeping track of the number of steps that the length has been decreased the sample point position can be calculated.

The accuracy of the calculation of the sample point position is within the given limits that have been set for this project. As seen in the report the results are SPP±2% which is close enough to know when data is sampled.

Keywords:

Controller Area Network, Bit stuffing, Vehicle Network, Sample point position, Electronic control unit, CAN , SPP, ECU

2 Contents

List of abbreviation	
ECU	Electronic Control Unit
CAN	Controller Area Network
CSMA/CA	Carrier Sensor Multiple Access/Collision Avoidance
FPGA	Field-Programmable Gate Array
SPP	Sample Point Position
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
SOF	Start-of-frame
CRC	Cyclic Redundancy Check
LCD	Liquid Crystal Display
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver/Transmitter
RTR	Remote Transmission Request
IDE	Identifier
DLC	Data Length Code
BCH code	Bose-Chaudhuri-Hocquenghem Code

1 Introduction

The electronic communication in a bus or a truck is managed by a so-called ECU (Electronic Control Unit). These units send and receive data over the CAN protocol, which is message protocol used in automotive applications. The protocol standard defines rules on how the communication shall be performed and among these rules there is a bit timing requirement that indicates when a bit shall be sampled.

The sample point position setting is a protocol parameter, which is set by the firmware of particular ECU into the CAN controller and its value is not simply visible from the outside. Wrong setting can cause higher level of frame error rate in the system, as the CAN uses broadcasts and incorrect behaviour of any node influences the performance of the whole system.

This report is based on a project at the bus and truck manufacturer Scania CV AB (henceforth only called Scania) in Södertälje.

1.1 Background and problem description

Scania is among the top manufacturers of trucks and buses worldwide and its brand symbolizes quality and trust. The company offers many different solutions of vehicles but they all have one thing in common, they are all built on electronic components.

In a truck there are about 20-25 different ECU's that manages all the electronic communication and these units' controls everything from oil pressure to dashboard lights. Timing and the reliability of data transfer are two very important criteria and therefore the accuracy of the sample point is a crucial factor. If a data bit is wrongly perceived it could have devastating consequences in terms of system failure.

Scania wants to develop a simpler method for verifying the sample point position in their ECU's, so this thesis will discuss the importance of the sample point criteria and also how to implement a testing methodology on how to perform reliable test to verify that position.

1.2 Purpose

The purpose of this thesis is to develop a testing methodology on how to perform reliable test to verify the sample point position of an ECU. To be able to verify and test the proposed testing methodology new tools had to be constructed. The tools construction includes the schematics of a circuit board containing all the components needed for CAN communication, layout of components and also on how to interface those components with software.

1.3 Delimiters

There are a couple of areas that has not been taken under consideration during the project. No analyze of possible signal disturbances from the 50MHz clock cycle has been performed.

The possibility of setting the CAN controller registers via the user interface software has not been implemented. Also the possibility to run the CAN communication at another bit rate than 250kBit/s or 500kBit/s has been neglected.

2 Theory

2.1 CAN – Basic concept

CAN is short for *Controller Area Network* and is an asynchronous multi-master serial broadcasting protocol for communication between ECU's. Since it is multi-master all nodes can transmit on the bus, but only one at the time. All nodes receive every transmission but because of individual identification on each unit only one will respond at a certain message. Chapter 2.2 CAN bus describes more in detail how nodes determine on which message to act on. (STMicroelectronics)

Its domain of application ranges from high speed networks to low cost multiplex wiring. Examples of application that has implemented CAN communication are automotive electronics, engine control units, sensors, anti-skid-systems. These applications are connected using CAN with bitrates up to 1 Mbit/s. CAN is also used in less complicated systems like for example lamp clusters and electrical windows and this is due to that it is a very cost effective solution.

The intention of this specification is to achieve compatibility between any two CAN implementations. Compatibility, however, has different aspects regarding e.g. electrical features and the interpretation of data to be transferred. (Bosch, 1991)

2.2 CAN bus

Every bit transmitted on the bus is defined as recessive or dominant, which maps to 1 or 0. If more than one node is trying to transmit, the result will carry a dominant bit if at least one node is transmitting a dominant bit (See Table 1: Bus state truth table). When a node transmits a dominant bit it directly perceive that bit on the bus and therefore don't know if someone else was trying to send data. If a node transmits a recessive bit, but a dominant bit is seen on the bus, the node knows that someone else is sending data.

Bit	**Dominant**	**Recessive**
Dominant	Dominant	Dominant
Recessive	Dominant	Recessive

Table 1: Bus state truth table

2.2.1 Data transmissions

As mentioned earlier in the report a master is not needed to communicate, when a node is ready to transmit data it checks the bus for activity and if it is available it simply writes a CAN frame onto the network. The CAN frames that are transmitted do not contain addresses of either the transmitting node or any of the intended receiving node(s). Instead, a unique network label is defined by the arbitration ID. All nodes on the network receive the message but the only unit with matching arbitration ID will accept the frame. (National Instruments, 2009)

2.2.2 Arbitration

All nodes generates messages concerning their own agenda, this can lead to several nodes trying to access the bus simultaneously. That problem is solved by bitwise arbitration, also called Carrier Sensor Multiple Access/Collision Avoidance (CSMA/CA). This principle that has been adopted into the CAN protocol solves the bus access conflicts by assigning a level of priority to each message. If a collision is detected, the message with highest priority (lowest arbitration ID) will gain access to the bus. Lower-priority nodes must wait until the bus becomes available before trying to transmit again. (Paret, 2007)

2.2.2.1 CSM/CA

The CSMA/CA principle states that when a dominant bit and a recessive bit are transmitted simultaneously on the bus, the resulting state must be dominant. And as seen in chapter 2.3 the first part of a message frame is the arbitration ID, this explains how the level of priority solves the conflict problem. (Paret, 2007)

2.2.3 Synchronization

There are no synchronizing signals attached to the data transmitted onto the bus. Instead the transmit unit starts sending frames synchronously with the bit timing. The receiving unit synchronizes itself by a change of the bus level as it receives frames. However, it could occur that the units get out of sync with respect to the other because of a clock error on either side or a phase delay in transmission path (e.g. cable or driver). Therefore, the receive unit adjusts its operation timing by means of hardware synchronization or resynchronization as it receives frames. Hardware synchronization is conducted at the beginning of a frame, at the start-of-frame bit. Resynchronization happens continuously as the unit receives the frames. Figure 1 illustrates the problem of when two nodes are out of synch. Node 1 synchronizes at the correct position but Node 2 misses the bit. (Renesas Electronics Corporation, 2006)

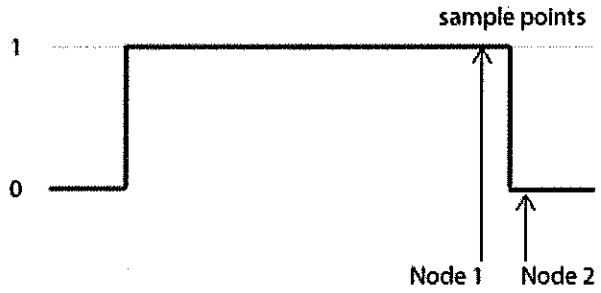


Figure 1: Unsynchronized nodes

2.4 CAN protocol

The CAN protocol supports two message frame formats, the only essential difference being in the length of the identifier. The "CAN standard frame" supports a length of 11 bits for the identifier and the "CAN extended frame" supports a length of 29 bits for the identifier.

2.4.1 Standard frame

A CAN message begins with the start bit "Start Of Frame (SOF)", this is followed by the "Arbitration field" which consist of the identifier and the "Remote Transmission Request (RTR)" bit used to distinguish between the data frame and the data request frame called remote frame. The following "Control field" contains the "IDentifier Extension (IDE)" bit to distinguish between the CAN base frame and the CAN extended frame, as well as the "Data Length Code (DLC)" used to indicate the number of following data bytes in the "Data field". If the message is used as a remote frame, the DLC contains the number of requested data bytes. The "Data field" that follows is able to hold up to 8 data byte. The integrity of the frame is guaranteed by the following "Cyclic Redundant Check (CRC)" sum. The "ACKnowledge (ACK) field" compromises the ACK slot and the ACK delimiter. The bit in the ACK slot is sent as a recessive bit and is overwritten as a dominant bit by those receivers, which have at this time received the data correctly. The receivers regardless of the result of the acceptance test acknowledge correct messages. "End Of Frame (EOF)" indicates the end of the message. The "Intermission Frame Space (IFS)" is the minimum number of bits separating consecutive messages. Unless another unit starts transmitting the bus remains idle after the IFS. (CAN CiA, 2010)

2.4.2 Extended frame

The difference between an extended frame message and a standard frame message is the length of the identifier. The 29-bit identifier consists of the 11-bit identifier (standard identifier) and an 18-bit extension (identifier extension). The distinction between CAN base frame format and CAN extended frame format is made by using the IDE bit, which is transmitted as dominant in case of an 11-bit frame, and transmitted as recessive in case of a 29-bit frame. As the two formats have to co-exist on one bus, it is laid down which message has higher priority on the bus in the case of bus access collision with different formats and the same identifier / base identifier: The 11-bit message always has priority over the 29-bit message.

CAN controllers, which support extended frame format messages, are also able to send and receive messages in CAN standard frame format. CAN controllers that just cover the base frame format do not correctly interpret extended frames. However there are CAN controllers, which only support the base frame format but recognize extended messages and ignore them. (CAN CiA, 2010)

2.4.3 Message frames

There are four different types of messages available: Data frame, remote frame, error frame and overload frame. This report only describes the data frame since this is the only one used in the project.

2.4.3.1 *Data frame*

The data frame is used when a node transmits data and this frame has two message formats, *Standard frame* and *extended frame*. As with all other frames the standard frame begins with a Start-Of-Frame (SOF) bit, which is of the dominant state and synchronization of all nodes. The structure of a message is shown in Figure 2: Standard data frame.

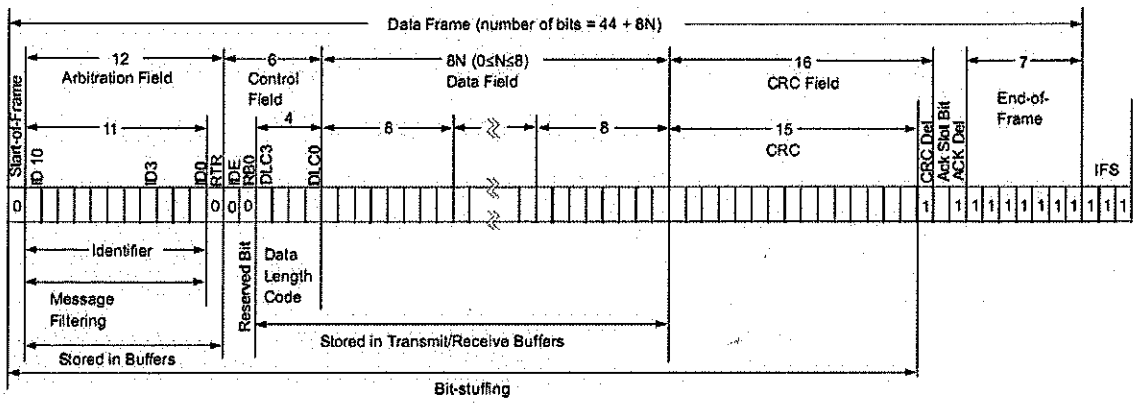


Figure 2: Standard data frame (Bosch 1991)

2.4.4 Error detection

If a CAN node receives a message with an error it destroys the frame and causes the transmitter to resend. The CAN node destroys the message by sending an error frame on the bus. The six different error detection mechanisms used by the CAN protocol is explained below.

Monitoring

The transmitter of a bit compares the signal sent with the signal perceived on the bus line (transmission channel). Except for the arbitration phase during the transmission of the identifier of a message and in the ACK slot, a transmitter starts sending an error frame if the data bit on the bus is different from the one sent. In this way bit errors affecting all stations on the bus cannot lead to non-detectable errors because they will be detected by the transmitter of the frame first.

Cyclic Redundancy Check

The 15 CRC bits are computed with every bit between the SOF to the last data bit. The BCH code used for generating the CRC leads to a hamming distance of six, including a parity check, in the unstuffed bit sequence.

Message Frame Check

The SOF, RTR, IDE, DLC, CRC delimiter and EOF fields must be consistent with the CAN specification. If a fixed format field in a received frame (except for the last EOF bit) does not conform to the standard, the receiver sends an error frame and does not accept the received frame.

Bit Stuffing

Any violation of the stuff rule between SOF and CRC is regarded as an error. This error will be explained more in detail further down in the report.

Acknowledgement

The transmitter of a data or remote frame treats a missing acknowledgement as an error and destroys the EOF field by sending an error frame.

Error Signalling

Each station that detects an error starts sending an error frame, so that other stations are notified of this condition by seeing a violation of the stuff rule or the fixed format delimiter or EOF fields. Note that due to the Bit Stuffing mechanism all stations answer a received error frame with error frames of their own. (Charzinski, 1994)

2.4.5 Bit timing

The Nominal Bit Rate of the network is uniform throughout the network and is given by:

$$f_{NBT} = \frac{1}{t_{NBT}} \tag{1}$$

Where t_{NBT} is the Nominal Bit Time. As defined in equation (1) the bit time is divided into four separate non-overlapping time segments called SYNC_SEG, PROP_SEG, PHASE_SEG1 and PHASE_SEG2. These are illustrated in Figure 3: Nominal Bit Time. (Florian & Bassemir, 1999) (Robb, 1999)

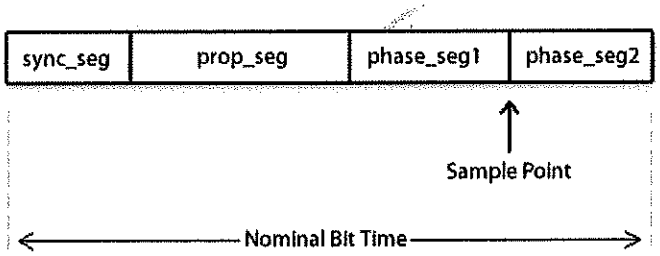


Figure 3: Nominal Bit Time

2.4.6 Bit stuffing

Since the CAN bus does not contain any clock signal there must be a way for all nodes to synchronize data. Therefore receivers use transitions in the CAN frame to synchronize their internal clock.

To ensure that there are enough transitions, the CAN protocol implements the Bit-stuffing rule. Bit stuffing adds an extra bit, which is of the opposite state compared to current one. This extra bit is inserted after five consecutive identical bits has been detected. Since the appearance of stuff bits depends on the message content, the length of the CAN frame will vary depending on the data content. The receiver automatically throws the extra stuff bits away so that the application software in the CAN node does not need to take them into consideration. (Intrepid Control System, 2010)

This process is illustrated in Figure 4 below.

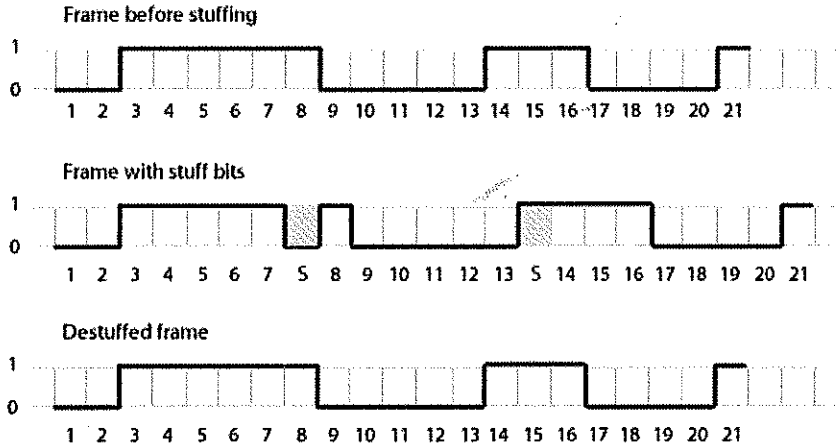


Figure 4: Bit stuffing

If more than five consecutive bits with the same polarity are detected between Start of Frame and the CRC Delimiter, the bit-stuffing rule has been violated. A Stuff-Error occurs and an Error Frame is generated. The message is then repeated. (Softing)

The remaining bits of the other segment of the data frames, such as the CRC delimiter, ACK field and End-of-Frame, have fixed structures and are not stuffed. (Paret, 2007)

3 Theoretical solution of problem

Since the sample point position is a protocol parameter set by software the only way to verify it would be to do it electrically. Of course the possibility of reading the controller register exists in software but this will not actually verify anything other than the data stored. The verification of the sample point position is not a widely discussed area and therefore hard to compare different solutions.

The solution discussed in this thesis is based on report by (Novák) but is using a slightly different hardware construction that is describe further down. It is based on that an error is deliberately provoked in the CAN protocol to make it possible to calculate the sample point position.

3.1 Issues

Since this is a rather complex protocol the mission was to identify all possible issues that could complicate the task. By reading the CAN specification (Bosch, 1991) the issue of controlling the timing is raised. Due to strict timing requirements there are several error mechanisms used in the protocol and there could be a problem controlling all of these.

3.2 Alternatives

The obvious component needed in any solution would be to have at least one CAN controller acting as a receiver. With this decided different alternatives were compared.

3.2.1 CAN generator

A possible way would be to create a CAN generator that could produce the CAN messages needed to communicate with the receiver. It would then be possible to control the generator and manage the messages sent. But due to lack of time to be able to create an own controller with the CAN protocol implemented this idea was rejected.

3.2.2 CAN message interference controller

As describe in chapter 3.2.1 the implementation of an own CAN protocol generator was not a possibility due to the workload. Instead the idea of interrupt the message between two CAN controllers was raised. This solution is also performed by (Novák).

By using two fabricated CAN controllers the protocol is integrated and therefore saves a lot of time and work.

3.3 Interfering the message

To be able to interfere the message sent between two CAN controllers a third component is needed. Several possibilities were discussed, e.g PIC or a ARM processor, but since that the impact of interfering the message has to be as small as possible the best solution were to use a FPGA. The FPGA can receive and transmit the message with the delay of only a few clock cycles.

Figure 5: Interfering the message

3.4 Finding the sample point position

Since the method for how to hijack the CAN message has been decided the remaining problem would be how to actually find the sample point position (SPP). Seen in Figure 3: Nominal Bit Time the SPP is located between segment three and four. The segments are as described before set in software.

To be able to locate the position the integrated CAN error mechanism, Stuff bit error, is to be used. In chapter 2.4.6 this function is explained in detail.

When five consecutive bits of the same polarity has been detected on the bus the CAN protocol adds the so-called stuff bit. If the stuff bit is missing and there are more than five bits with the same polarity in row on the bus the stuff error will occur. So by altering the stuff bit the error can be provoked.

3.5 Implementation

This chapter will in theory describe the system flow from start to end of a transmitted message on the CAN bus.

3.5.1 System overview

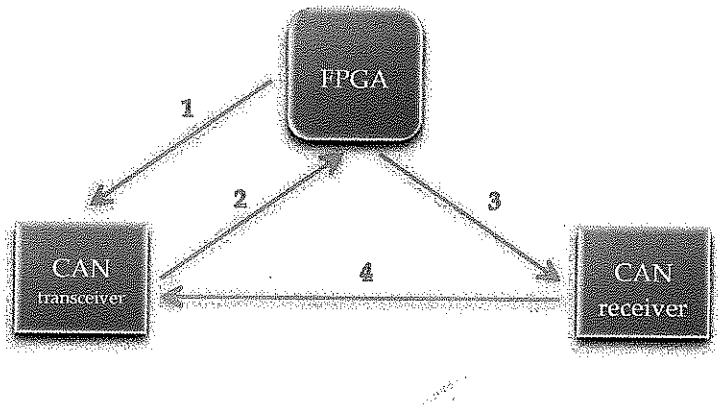


Figure 6: System overview

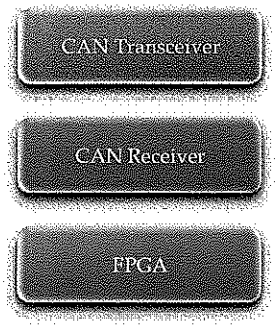
1: The FPGA sends start command to the CAN transceiver.

2: The CAN transceiver sends a defined message which the FPGA hijacks and performs its calculations.

3: The FPGA forwards the altered message to the CAN receiver.

4: The CAN receiver responds to the CAN transceiver without to involvement of the FPGA.

3.5.2 Flow chart



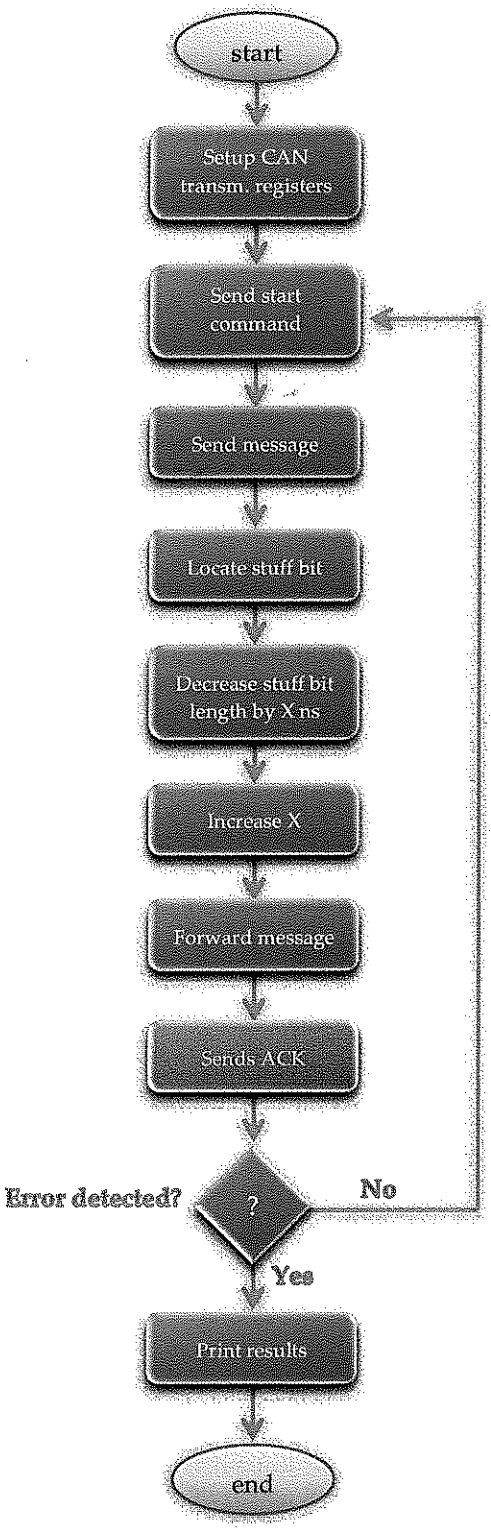


Figure 7: Flow chart

3.5.3 Possible complications

There can be some complication when trying to alter the message from the CAN transceiver. If the wrong bit is changed it can be hard to detect what type of error that have occurred. Also the problem of propagation delay has an impact on the system. If the CAN transceiver has not detected its transfer bit on its data input pin within a defined time another error, than stuff bit error, will be provoked.

The calculations performed by the FPGA are based on a fully synchronized CAN system. If the units were out of synch it would be impossible to measure an exact sample point position.

4 Practical solution of problem

To be able to realize to theoretical solution a hardware platform with the chosen components were needed. The goal was to create a platform that was user-friendly and portable.

At first, the system was developed on an off-the-shelf platform from Spartan with some external CAN controllers. This prototype worked as a debugging system and were later on in the project the base for the self designed PCB.

4.1 Hardware

Since the Spartan starter kit includes a lot of circuitry not needed for this project and is unnecessary large by size, a new slimmer version was designed. The new PCB only included the circuitry stated below. The stripped down system is shown in Figure 6.

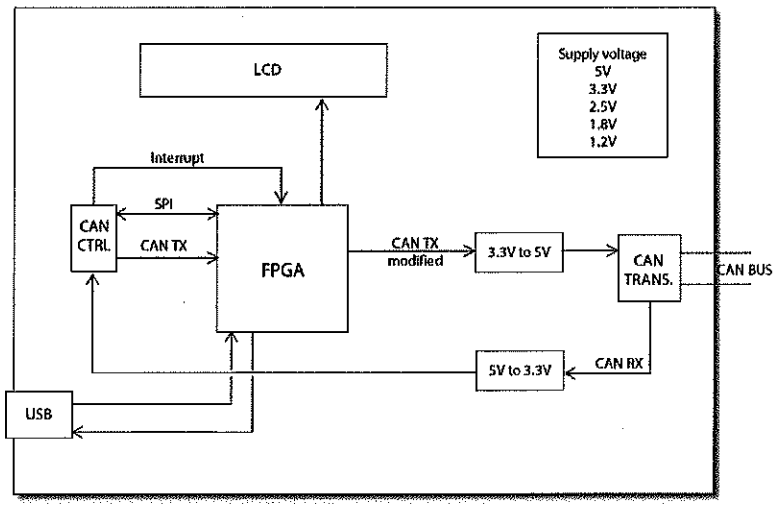


Figure 8: System overview

FPGA

The programmable logic used in this project is part of the Spartan-3 FPGA family. It is a low-cost, high-performance logic solution for consumer –oriented solutions. The device chosen for the construction of this family is a 400K gates FPGA called XC3S400. The XC3S400 is a 144-pin quad flat package chip with 16 multipliers and 4 Digital Clock Managers (DMCs).

Configuration memory

The platform flash used is part of a series of in-system programmable PROMs and is a non-volatile storage. By non- volatile means that the information stored at the chip is retained even when not powered. The chip, XCF04s, has a storage volume of 4MB and is power by 3.3V supply voltage. It supports several configuration modes and the one used in this project is a "FPGA Master Serial Mode". (Xilinx 2009)

CAN controller

The MCP2515 is stand-alone CAN controller with SPI interface that implements the CAN specifications. It has the capabilities of transmitting and receiving CAN data of both standard and extended frames. The MCP2515 has numerous acceptance filters used to filter unwanted messages. (Microchip 2010)

CAN transceiver

The MCP2551 is a high-speed CAN, fault-tolerant device that serves as the interface between a CAN protocol controller and the physical bus. The MCP2551 device provides differential transmit and receive capability for the CAN protocol controller, and is fully compatible with the ISO-11898 standard. Typically, each node in a CAN system must have a device to convert the digital signals generated by a CAN controller to signals suitable for transmission over the bus cabling (differential output). It also provides a buffer between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outside sources (EMI, ESD, electrical transients, etc.). (Microchip 2010)

USB to serial

The FT232R is a USB to serial UART interface device, which simplifies USB to serial designs and reduces external component count by fully integrating an external EEPROM, USB termination resistors and an integrated clock circuit, which requires no external crystal, into the device. It has been designed to operate efficiently with a USB host controller by using as little as possible of the total USB bandwidth available. (FTDI 2010)

DC/DC converter

Several DC/DC converters are used to provide the correct supply voltages to the different chips. The available voltages on the circuit board are 5V, 3.3V, 1.8V and 1.2V.

LCD

A 2x16 segment LCD display.

4.2 Software

This thesis has included two kinds of software development; hardware description coding (VHDL) and a graphical user interface (C#).

4.2.1 VHDL

The main part of the project has been coding the behaviour of the CAN Sample Point Analyzer. The function of the code is to listen for a message from the CAN controller and locating the stuff bit within that message. By knowing that a stuff bit always inserts after five consecutive identical bits, the software only needs to wait until this bit occurs.

First, a setup of the CAN controller is performed setting all concerned registers. These registers control for example the transmitting bit rate of the controller and the message content. When the setup is done the message programmed is sent onto the bus.

After the setup a listen state is initiated waiting for the stuff bit to appear. This state consists of a loop counting the equal bits that arrive. When the loop counter detects the fifth equal bit the next state is entered knowing that the next following bit is the stuff bit that is going to be altered.

After the stuff bit is received the decreasing of its length is started. Knowing that a bit sent at for example 250kbit/s consists of 200 system clock cycles, this state decreases the length of the stuff bit by 1/200 each time it is entered. The new altered bit is now sent onto the bus and if no error is detected the loop is restarted.

By keeping track of how many times the length has been decreased the sample point can be calculated. Once an error message occurs the loop is broken and the value printed. The flow chart of the VHDL code can be seen in Figure 7: Flow chart.

4.2.2 User interface (C#)

The user interface was not part of the project specification but was added to simplify the debugging. By communication via the RS232 protocol the CAN controller registers could be read and displayed in a user-friendly application. Some other data such as bit rate and bus termination is also printed. Figure 9 shows a print screen of the software interface used on a computer.

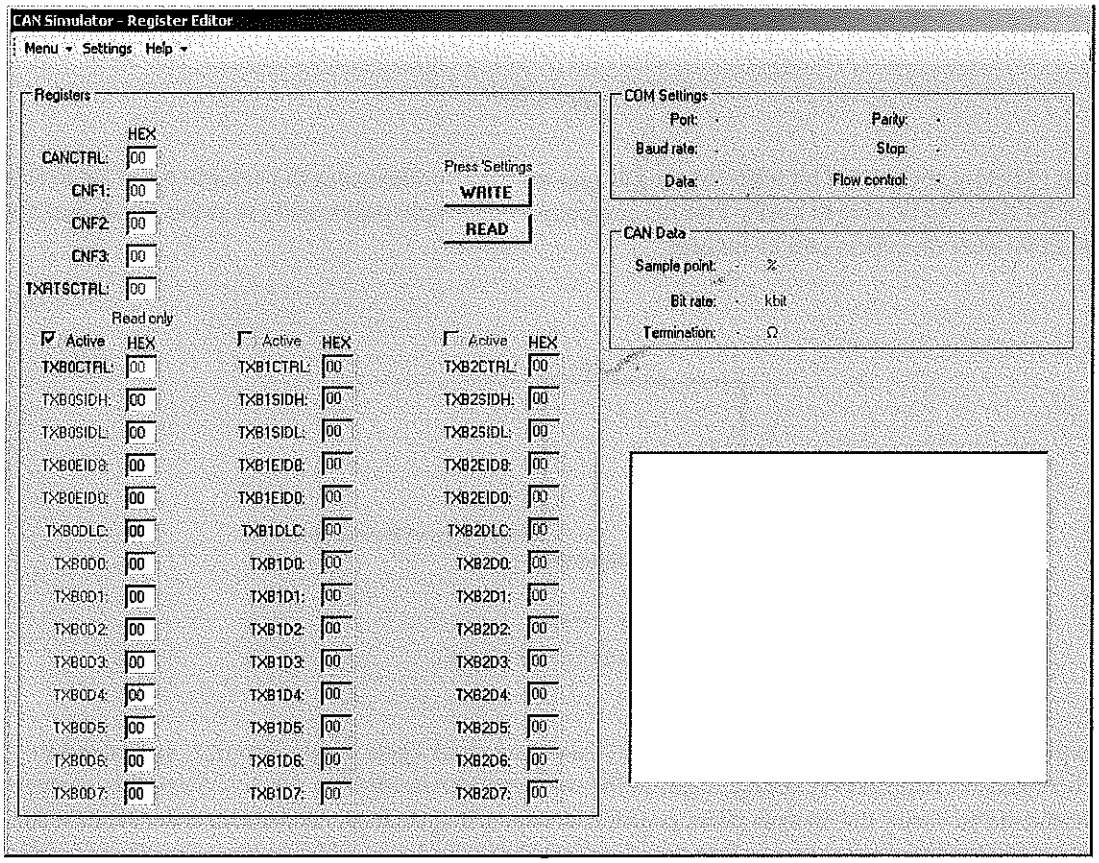


Figure 9: User interface

4.2.3 Configuration

As seen in Figure 7 one of the states manage the configuration of the CAN controller. This was done by the FPGA via the SPI protocol. For further reading on values and which registers to configure see appendix B.

4.3 Method

One way to find the sample point position of an ECU is to calculate the limit where the CAN protocol rules are violated. To do this a message is sent with a known configuration that is not violating the rules and then gradually decreases the values until an error is flagged.

As mentioned in chapter 2.4.4 Error detection, there are several mechanisms used to detect an error. In this method used to find the sample point position, the *Stuff bit error* will be the indicator of when that position is found.

If the bit rate of the CAN controller is set to 250kBit/s, the length of one bit is therefore by eq. (1):

$$t_{NBT} = 4\mu s$$

Figure 10 below shows one bit with the length t_{NBT} and it also illustrates the sample point set at 80% of that bit time. To be able to verify that the programmed sample point really is at the given position the *Stuff bit error* detection will be helpful.

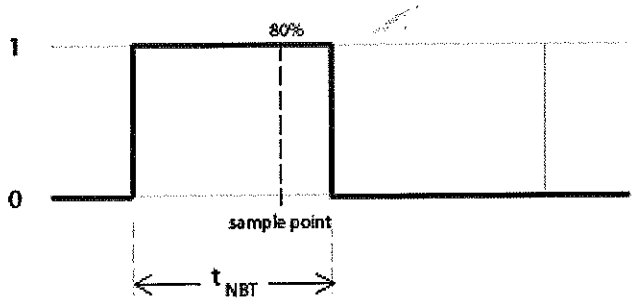


Figure 10: Bit length

4.3.1 Configuration of the message

As explained in theory the CAN nodes doesn't have any clock cycle to synchronize on, instead the synchronization happens on falling edges in the message transmitted on the bus. Since this analyzing method used to verify the sample point position is depending on high resolution of the bit time the synchronization is a crucial factor. It is very important that nodes samples at the same time of a bit to be really certain that the error provoked are not caused by a synchronization fault.

Figure 1 illustrates the problem when two nodes are out of sync. Node 1 samples at the correct bit and Node 2 that is out of synch samples at the following, which will cause an error. Therefore a message with as many transitions as possible is constructed to assure that the nodes are synchronized.

4.3.1.1 *Message setup*

When configuring the message frame there are a couple of fields that are needed to be set. The transmitted frame should be configured with to following values:

ID = 0x5555555
Data length = 0xA
Data byte 0-7 = 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xA0, 0x2A

The maximum number of bytes in the data field is 8, but since this only includes one transition the field is set to 0xA, however only 8 bytes of data are transmitted. The different frame fields are shown below in Figure 11.

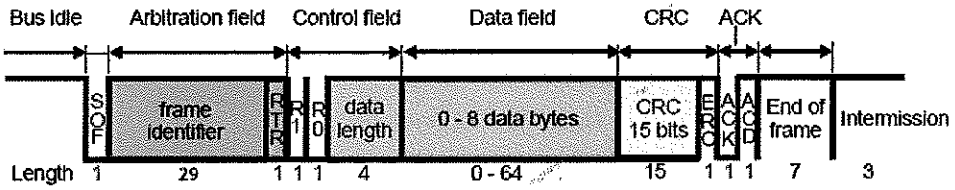


Figure 11: CAN data frame

By setting the message frame as described above the maximum number of transitions will be provided on the bus since the binary value of a 0x5 and a 0xA is 0101 respectively 1010. Data byte 6 is set to 0xA0 because it is during this byte the stuff bit will be provoked. What will happen is that the CAN controller will receive more than five zeroes, which will produce a stuff bit. The stuff bit that appears will be the one that should be altered. In Figure 13 the message is illustrated in binary code.

4.4 Locating the stuff bit

The only bit that is supposed to be altered is the stuff bit, it is therefore essential that no other bit is changed. Since the message is hardcoded the algorithm knows what sequence to look for and it is not possible for a stuff bit to appear somewhere else but between data byte 6 and 7.

A state machine is used to locate the stuff bit, and is shown in Figure 12. When five zeroes are detected the system knows that the following bit is the stuff bit. And when it reaches that state it starts to decrease that bit one step at a time.

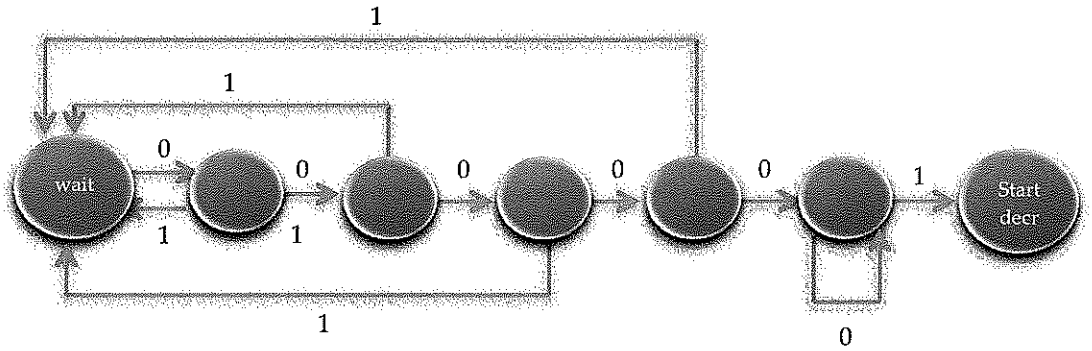


Figure 12: State machine of locating the stuff bit

4.5 Finding the sample point position

To find the sample point position the *Stuff bit error* needs to be provoked. This is done by successively decreasing the stuff bit length until the receiving node no longer can interpreter it. When bit-stuffing rule is violated the ECU sets an active error frame and the transmission is interrupted. In Figure 13 the message is illustrated before and after the insertion of a stuff bit.

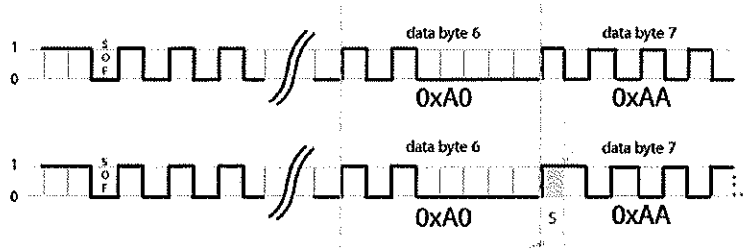


Figure 13: CAN message

When detected, the length of the stuff bit is decreased with steps of 20ns until the error is flagged. The error will be provoked when bit flank of the stuff bit has passed the sample point position, as seen in Figure 14. When the edge of the bit has passed the sample point the CAN controller will miss to sample the stuff bit and instead sample the next following bit. When this happens six zeroes will be read in a row which violates the Stuff error rule and a flag will be set.

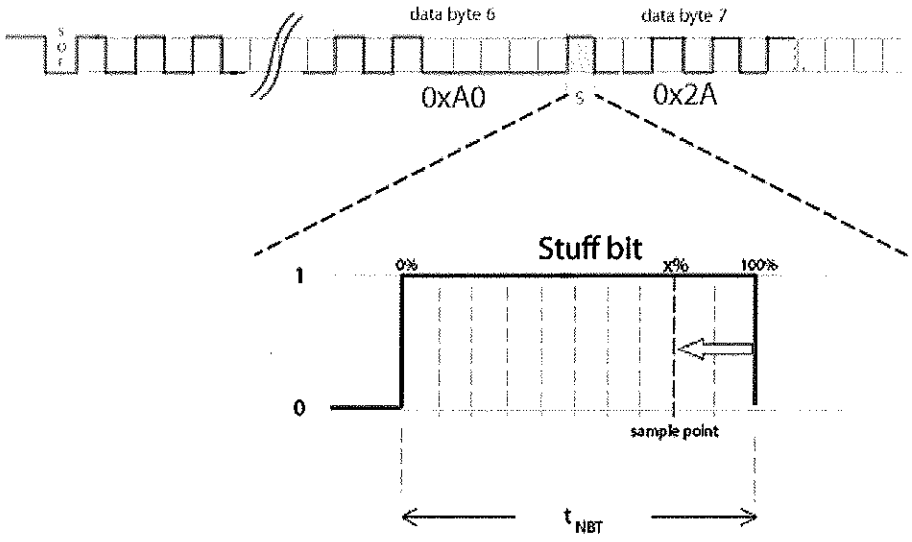


Figure 14: Decreasing the stuff bit length

By keeping track of how many steps that the stuff bit has been decreased the sample point position can be calculated.

4.6 Calculating the sample point position

A counter is keeping track of the number of steps that the bit has been decreased. This number is stored in *cntTmp* in equation (2).

$$cnt = BitRate - cntTmp \quad (2)$$

Depending on which bit rate the system is running, the length of a data bit is different. While running at 250kBit/s one bit is 4µs respectively 2µs at 500kBit/s. A bit with the length 4µs can be divided into 200 cycles of a 50MHz clock. Therefore the variable *BitRate* is either set to 200 or 100 depending on bit rate setup. The result calculated in equation (2) will thereby represent the sample point position given in number of cycles. The number of cycles calculated is the divided by the total number of cycles given by the bit rate.

Example of sample point calculation is given in equation (3) below.

$$\begin{aligned} BitRate &= 200 \\ cntTmp &= 17 \\ cnt &= 200 - 17 = 183 \\ \frac{183}{200} &= 0.915 = 91.5\% \end{aligned} \quad (3)$$

Since the system doesn't have a floating point implementation the result will lose some precision. The given example above will be printed as 91% because of this resolution problem.

4.7 Verifying

To verify the results calculated be the system an oscilloscope with a CAN analyzer has been used. This analyzer shall not be mixed up with this thesis system since they do not perform the same tasks.

With the oscilloscope the length of bits within the message could be verified. To assure that the code worked as supposed to, an untouched message were sent and analyzed. Then an altered message with a decreased length of the stuff bit was sent. And by using the oscilloscope measuring functions I could verify that my altered bit were of the correct length set by me.

The CAN analyzer has the possibility of setting the sample point of the integrated CAN controller. By using this function I could test that the code were finding the correct sample point position. Setting that sample point at different values the test result in chapter 5 were calculated by the CAN sample point analyzer.

Verification towards Scania ECU's has also been done, the sample point position in those units are programmed at a value which are not allowed to mention in this report because it is classified information.

5 Results

The testing has been an on-going task since there are a lot of different modules that are cooperating in this project.

The general test method used has been more of a trial and error technique. In the beginning of the project testing were done to just get the system to response for different commands such as setting and reading registers from the CAN controller. Next step in the testing phase was sending a CAN message and let it run through the FPGA unaltered and analyze the data on the bus. By verifying that the message was correct with the CAN analyzer tool, the following procedure was to start decreasing the length of the stuff bit.

Since the results from the above test cases only rendered "Fail" or "OK" there is no need to present this in the report. Focus has instead been put on results from the final system.

The test results in the tables below are presented as an average of 30 runs. Each run is performed in room temperature with the same setting of the CAN controller regarding frame structure, bit rate and bus termination.

The tables below show the calculated average sample point position (SPP) with settings.

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	55 %	60.03%	66.97%	72.77%	79.45%	85.90%

Table 2: 250kBit/s - 60 Ohm - Extended frame

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	55 %	60.15%	67.20%	72.30%	80%	85.98%

Table 3: 500kBit/s - 60 Ohm - Extended frame

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	55.10%	60.95%	66.82%	72.35%	81%	85.80%

Table 4: 250kBit/s- 120 Ohm - Extended frame

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	56 %	60.25%	67.42%	72.8%	80.55%	85.65%

Table 5: 500kBit/s - 120 Ohm - Extended frame

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	55.05%	60.00%	67%	73%	79.35%	86.25%

Table 6: 250kBit/s - 60 Ohm - Standard frame

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	55 %	60.89%	68.20%	73.30%	80.50%	85.40%

Table 7: 500kBit/s - 60 Ohm - Standard frame

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	55%	61.35%	67.70%	74.15%	81.03%	85.64%

Table 8: 250kBit/s - 120 Ohm - Standard frame

Programmed SPP	56.25%	62.50%	68.75%	75%	81.25%	87.50%
Measured SPP	56.05%	61.50%	67.40%	73.23%	80.45%	86.15%

Table 9: 500kBit/s - 120 Ohm - Standard frame

Table 10 presents the results from test runs performed with a Scania ECU, but due to classified information the settings of the test runs in cannot be given.

Programmed SPP	x%
Measured SPP	x-1.8%

Table 10: Scania ECU

6 Discussion

The results obtained are based on a test environment with very short cable distance; it is uncertain what will happen with delays and disturbances if the cable length is very long. It will probably cause an error due to the response time has been violated. Even though the project specifications were fulfilled there are some improvements that could be implemented.

To be able to get a better estimation of the sample point a faster system clock is needed. The clock used in this project is at 50MHz and gives a bit time of 20ns. If instead a 100MHz clock is to be used the resolution would increase by a factor 2. And with better resolution the decreasing of the stuff bit length could be more precise. But to really take advantage of such a fast system clock a floating point implementation is needed, because then it is possible to calculate tenths instead of rounding off to integers. In the implementation that is described in this report the result calculated are just integers, this gives a good estimation of were to sample point is located but not an exact value. The results shown in the tables in previous chapters are printed with decimals, but this is not from the calculation of the sample point but from the average calculation of the entire test run performed.

Another implementation that could be done is to control the message sent from the CAN controller onto the bus. Today this message is hard coded and can only be change if the code is modified. By giving the possibility to modify the message sent this module could not only work as a sample point analyzer but also as a *Message analyzer*. It could facilitate the work to see what response is given from the receiver for a certain message. For example, if the unit that controls the breaks of a truck response correct when the "break"-command is sent. To be able to set different messages the windows software created for this project needs to be modified as well. The software is prepared for this kind of functionality and could easily be expanded.

The code written for this project works for its purpose but might be problematic to alter when adding new function. Before doing so, a makeover is needed in terms of structure and variable naming.

7 Summary

The criteria's set by Scania for this project has been fulfilled and my CAN Sample Point Analyzer works according to specifications. However, there are still improvements that are needed as mentioned in the discussion. Due to lack of time in the end of the period, testing and verification of the project has not been performed enough. There are still possible cases were the CAN Sample Point Analyzer might not indicate the correct value.

However, the result of this thesis is indicating well enough on where the sample point is located. Furthermore, the testing has proven that the calculation and measurement of the sample point is only differ maximum 2% from the actual value.

The PCB designed has been patched with a few wires since there were some errors when exporting the net list from the schematics to the layout software. This has been fixed in the layout files but not been fabricated. Those errors caused the supply voltages to the different components to mix up; this resulted in that the 5V supply wire was connected to the 3,3V. By cutting the 5V supply and connected it by wires to the involved components the problem were solved.

As with all projects new interesting features appeared during the development. The given criteria's has been extended as the project proceeded, some has been implemented but since this was only a master thesis it was not possible to apply them all.

8 References

9 Litteraturförteckning

Bosch. (1991). *CAN Specification: Version 2.0.* Stuttgart: Bosch.

CAN CiA. (den 19 May 2010). *CAN in Automation (CiA).* Hämtat från CAN protocol: http://www.can-cia.org

Charzinski, J. (1994). *Performance of the Error Detection Mechanisms in CAN.* Institute of Communications Switching and Data Technics. Stuttgart: University of Stuttgart.

Florian, H., & Bassemir, A. (1999). The Configuration of the CAN Bit Timing. *6th International CAN Conference.* Turin.

Instruments, N. (2009). *Controller Area Network (CAN) Overview.* Hämtat från http://zone.ni.com/devzone/cda/tut/p/id/2732 den 12 Juny 2010

Intrepid Control System. (den 2 July 2010). Hämtat från CAN bus: http://intrepidcs.com/wavebps/CAN_bus.html

Novák, J. *Sample Point Position Measurement of Controller Area Network Nodes.* Faculty of Electrical Engineering, Czech Technical University. Prague: Czech Technical University.

Paret, D. (2007). *Multiplexed Networks For Embedded Systems.* Chicester: JOHN WILEY & SONS.

Renesas Electronics Corporation. (2006). *Introduction to CAN.* Hämtat från http://documentation.renesas.com/eng/products/mpumcu/apn/rej05b0804_m16cap.pdf den 14 June 2010

Robb, S. (1999). *CAN Bit Timing Requirements.* East Kilbride: Freescale Semiconductor.

Softing. (u.d.). *CAN bus.* Hämtat från <http://www.softing.com> den 2 July 2010

STMicroelectronics. (u.d.). *Introduction to CAN controllers.* Hämtat från http://www.st.com/stonline/products/families/automotive/microcontrollers/images/can_introduction.pdf den 18 June 2010

Xilinx. (2009). *Platform Flash PROM User Guide.* Xilinx.

10 Appendix

A. VHDL modules

B. Register map of CAN controller

C. Configuration of the CAN controller

A. VHDL modules

An explanation of every VHDL module and process seen in Figure 6 will follow below. No details such as signal names will be explained. Module and process names will be written in cursive text.

LCDDriver

The LCDDriver module initiates and setup the LCD display. It prints data regarding the sample point position.

ScaledClk

Receive an input signal from external button and sets the chosen clock frequency. Logic 1 represents the frequency 500 kHz and a 0 sets it to 250 kHz. The output from this module works as a start bit to for e.g. CanRead.

OhmCntrl

Takes an input from external button and sets the chosen CAN termination. There are two possible configurations to choose from; 60Ω and 120Ω. The module output controls a relay which is connected to each of the terminations.

Uart

This is the core to all communication sent between the FPGA and a computer and LCD display. It includes a hardcoded ASCII table to represent all the text needed to be printed along with the initiation of the serial communication. To be able to communication via the RS232 protocol certain settings need to match and this is set here.

UartTran

UartTran is a more compact version of the Uart above, it does not include any text table. It only sends the data bits as it receives them. This is done to facilitate the reception of data in the c# software CanSim, see 3.2.2 User interface in C#.

UartRec

UartRec is the opposite of the above uart modules and handles the reception of data bits from the serial port. Initiation and setup is performed to make it possible to receive data.

ShortBit

This is the component that shortens the chosen bit to perform the sample point test. Depending on the set frequency the length of the test bit is calculated.

Debounce

Component is not in use, but could easily be implemented to prevent noise on button signals.

CanCntrl

CanCntrl resets and controls the CAN controller chip, it writes all the registers needed to be set.

CanRead

This module reads from the CAN controller registers. Reading is done to show register values in the c# software.

SystemCnt

Every sample point test is run 64 times and SystemCnt handles the counting of these test runs. The number of runs is set to 64 to facilitate the calculation of the average sample point position, this is done by right shifting the final value six times.

The following components are not modules, but instead processes of the core file. Processes names are texted in cursive.

ShowPercent

The process takes a input argument from the ShortBit module and passes on the calculated value to Uart where it is printed to the user. As explained in SystemCnt above the average percent calculation is performed with an six bit right shift.

UpCounter

This is the counter used to shorten the test bit length. It increases the variable cntShortBitTmp by one every clock cycle as long as no error has been detected on the CAN bus.

BitRate

As explained above the process BitRate sets the variable bitRateLength value according to the chosen frequency.

SystemCntStateCntrl

Controls the state machine of start signals for the counting processes such as avarageCalc and systemCnt.

CanReadStateCntrl

A state machine that controls the CanRead module.

ReadCanReg

A process to choose between which of CanCtrl or CanRead to access data from.

CoreCounterDelay

A delay counter used by the system. It is used to delay some of the signals, otherwise the CAN controller will signal an error on wrong bases.

ErrorCntDelay

To prevent unwanted error signal triggers an error count delay is used. To assure that it is actual error indicated the system waits for the counter to reach 100 000 clock cycles before checking the interrupt pin.

AverageCalc

Performs the first part of average percentage calculation. It summarize the sample point positions and stores the value in the variable averagePercent. This variable is shifted in ShowPercent calculate the final value .

intCounter

This process sets the error flag indicating an error on the CAN bus. If ErrorCntDelay has reach 100 000 and the interrupt pin is still active, the error flag is set.

LCDCharReset

Handles the char pointer of the LCD display. For every character written the pointer is increased one step.

B. Register map of the CAN Controller

Table is from the datasheet of the MCP2515 CAN controller.

Lower Address Bits	Higher-Order Address Bits							
	0000 xxxx	0001 xxxx	0010 xxxx	0011 xxxx	0100 xxxx	0101 xxxx	0110 xxxx	0111 xxxx
0000	RXF0SIDH	RXF3SIDH	RXM0SIDH	TXB0CTRL	TXB1CTRL	TXB2CTRL	RXB0CTRL	RXB1CTRL
0001	RXF0SIDL	RXF3SIDL	RXM0SIDL	TXB0SIDH	TXB1SIDH	TXB2SIDH	RXB0SIDH	RXB1SIDH
0010	RXF0EID8	RXF3EID8	RXM0EID8	TXB0SIDL	TXB1SIDL	TXB2SIDL	RXB0SIDL	RXB1SIDL
0011	RXF0EID0	RXF3EID0	RXM0EID0	TXB0EID8	TXB1EID8	TXB2EID8	RXB0EID8	RXB1EID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXB0EID0	TXB1EID0	TXB2EID0	RXB0EID0	RXB1EID0
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXB0DLC	TXB1DLC	TXB2DLC	RXB0DLC	RXB1DLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXB0D0	TXB1D0	TXB2D0	RXB0D0	RXB1D0
0111	RXF1EID0	RXF4EID0	RXM1EID0	TXB0D1	TXB1D1	TXB2D1	RXB0D1	RXB1D1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXB0D2	TXB1D2	TXB2D2	RXB0D2	RXB1D2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXB0D3	TXB1D3	TXB2D3	RXB0D3	RXB1D3
1010	RXF2EID8	RXF5EID8	CNF1	TXB0D4	TXB1D4	TXB2D4	RXB0D4	RXB1D4
1011	RXF2EID0	RXF5EID0	CANINTE	TXB0D5	TXB1D5	TXB2D5	RXB0D5	RXB1D5
1100	BFPCTRL	TEC	CANINTF	TXB0D6	TXB1D6	TXB2D6	RXB0D6	RXB1D6
1101	TXRTSCTRL	REC	EFLG	TXB0D7	TXB1D7	TXB2D7	RXB0D7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL	CANCTRL

Note: Shaded register locations indicate that these allow the user to manipulate individual bits using the Bit Modify command.

Register Name	Address (Hex)	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	POR/RST Value
BFPCTRL	0C	—	—	B1BFS	B0BFS	B1BFE	B0BFE	B1BFM	B0BFM	--00 0000
TXRTSCTRL	0D	—	—	B2RTS	B1RTS	B0RTS	B2RTSM	B1RTSM	B0RTSM	--xx x000
CANSTAT	xE	OPMOD2	OPMOD1	OPMOD0	—	ICOD2	ICOD1	ICOD0	—	100- 000-
CANCTRL	xF	REQOP2	REQOP1	REQOP0	ABAT	OSM	CLKEN	CLKPRE1	CLKPRE0	1110 0111
TEC	1C	Transmit Error Counter (TEC)								0000 0000
REC	1D	Receive Error Counter (REC)								0000 0000
CNF3	28	SOF	WAKFIL	—	—	—	PHSEG22	PHSEG21	PHSEG20	00-- -000
CNF2	29	BTLMODE	SAM	PHSEG12	PHSEG11	PHSEG10	PRSEG2	PRSEG1	PRSEG0	0000 0000
CNF1	2A	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	0000 0000
CANINTE	2B	MERRE	WAKIE	ERRIE	TX2IE	TX1IE	TX0IE	RX1IE	RX0IE	0000 0000
CANINTF	2C	MERRF	WAKIF	ERRIF	TX2IF	TX1IF	TX0IF	RX1IF	RX0IF	0000 0000
EFLG	2D	RX1OVR	RX0OVR	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN	0000 0000
TXB0CTRL	30	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
TXB1CTRL	40	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
TXB2CTRL	50	—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0	-000 0-00
RXB0CTRL	60	—	RXM1	RXM0	—	RXRTR	BUKT	BUKT	FILHIT0	-00- 0000
RXB1CTRL	70	—	RSM1	RXM0	—	RXRTR	FILHIT2	FILHIT1	FILHIT0	-00- 0000

C. Configuration of the CAN controller

In the table below the affected registers are shown with their values. The address to each register can be found in the register map in appendix B.

Register	Data (Hex)
TXRTSCTRL	0x00
CNF1	0x01
CNF2	0xB8
CNF3	0x05
CANCTRL	0x0C
CANINTE	0x80
TXB0CTRL	0x03
TXB0SIDH	0x55
TXB0SIDL	0xA9
TXB0EID8	0x55
TXB0EID0	0x55
TXB0DLC	0x0A
TXB0D0	0xAA
TXB0D1	0xAA
TXB0D2	0xAA
TXB0D3	0xAA
TXB0D4	0xAA
TXB0D5	0x0A
TXB0D6	0x00
TXB0D7	0xAA

Förbättring: detektera vilken typ av fel det är. Just nu kan den bara se att det är ett error ch inte vilken typ.