

# Acquisition of distributed CAN traffic for centralized analysis at functional and electrical levels

Master's Thesis  
Electrical Measurements



**LUND**  
UNIVERSITY

Sebastian Mellström  
Supervisor: Christian Antfolk  
Completed at Scania CV

Department of Biomedical Engineering  
Faculty of Engineering LTH



## **Abstract**

This thesis covers specification and development of a distributed measurement system intended for CAN bus. It was completed at Scania CV, Södertälje in collaboration with Lund University. The goal of the thesis is to specify and develop a distributed measurement system that helps Scania with acquisition of functional and electrical data. Several topics are addressed such as filtering, at what speed the system shall run at and much more. By a thorough pre-study that began with reading of specifications and continuing into measurements of the bus, a number of requirements was produced to reach a specification to be able to begin development.

Development was done by producing a custom hardware controller that handles the incoming data and performs a conversion to logical data. This controller is in turn connected to a system bus on the evaluation board that makes it possible to write directly into the CPU's RAM. Due to this, valuable CPU time that would be spent fetching data is saved and the CPU can be used for signal processing and handling the upstream connection to a server instead. In addition to this, an input stage that handles interaction with the CAN bus was produced. It is essential that the bus is not affected by the measurement system. Furthermore this stage filters the signal to minimize aliasing from the ADCs.

The developed system aims to increase knowledge of the electrical behaviour of the bus as well as a new method to acquire logical information from the bus. With more development, a custom PCB could be manufactured that could be placed on-board for increased vehicle diagnostics. Finally, tips and ideas for further development are suggested.



## Acknowledgements

The thesis was carried out at Scania CV in Södertälje, Sweden in collaboration with Lund University. The academic supervisor Christian Antfolk as well as the external supervisor Mikael Borgström at Scania CV have provided exceptional motivation and support. Without their support this thesis would not have been possible.

Pico Technologies deserves an acknowledgement for their exceptional measurement tools, as well as user gmack on NI forums for his or her invaluable help with Multisim software.

This document was typeset using the  $\LaTeX$  document processing system originally developed by Leslie Lamport, based on  $\TeX$  typesetting system created by Donald Knuth.

# Contents

- 1 Introduction** **1**
- 1.1 Background . . . . . 1
- 1.2 Task . . . . . 3
  - 1.2.1 Specification part . . . . . 4
  - 1.2.2 Implementation part . . . . . 5
- 1.3 Limitations . . . . . 5
- 1.4 Solution proposal . . . . . 5
- 1.5 Purpose . . . . . 7
- 1.6 Disposition . . . . . 7
- 1.7 Controller Area Network . . . . . 7
  - 1.7.1 Physical Layer . . . . . 9
- 1.8 A/D sampling . . . . . 10
- 1.9 Signal Conditioning . . . . . 12
  
- 2 Method** **13**
- 2.1 Theory . . . . . 13
  - 2.1.1 Termination of signals . . . . . 13
  - 2.1.2 Filtering . . . . . 14
  - 2.1.3 Low-pass filter . . . . . 15
  - 2.1.4 Band-pass filter . . . . . 15
  - 2.1.5 Phase . . . . . 16
  - 2.1.6 Bandwidth . . . . . 17
- 2.2 Quantative data analysis . . . . . 17
- 2.3 Requirements specification . . . . . 19
- 2.4 Hardware . . . . . 20
- 2.5 Construction of input stage . . . . . 24
- 2.6 Development . . . . . 27
  - 2.6.1 FPGA development . . . . . 27

2.6.2	Hardware development, SoC side . . . . .	32
2.6.3	Software development, SoC side . . . . .	34
2.6.4	Software development, desktop side . . . . .	34
2.7	Testing & Quality assurance . . . . .	35
2.7.1	Testing of VHDL . . . . .	35
<b>3</b>	<b>Result</b>	<b>39</b>
3.1	Hardware oriented . . . . .	39
3.1.1	Simulations . . . . .	41
3.1.2	Measured data . . . . .	43
3.2	Software oriented . . . . .	46
3.2.1	Embedded side . . . . .	46
3.2.2	Desktop side . . . . .	48
<b>4</b>	<b>Discussion &amp; Conclusions</b>	<b>49</b>
4.1	Improvements . . . . .	49
4.2	Benefits for the group . . . . .	50
4.3	Limitations of implementation . . . . .	51
4.4	Alternative solutions . . . . .	51
4.5	Conclusions . . . . .	52
<b>A</b>	<b>Attachments</b>	<b>53</b>
A.1	Requirements specification . . . . .	53
A.2	Utilization log . . . . .	56
A.3	Bill of Materials . . . . .	56
A.4	Datasheets . . . . .	57
A.5	Attached Code . . . . .	57
A.5.1	Vivado . . . . .	57
A.5.2	HDL . . . . .	66
A.5.3	Demonstration software . . . . .	98
A.5.4	Desktop and client interaction . . . . .	100







# Abbreviations

$CAN_H$	CAN High 9, 31, 40, 41, 54
$CAN_L$	CAN Low 9, 31, 40, 41, 54
$\omega_c$	Cut-off Frequency 15
$f_c$	Cut-off Frequency 14, 16, 24, 27
ACK	Acknowledgement 8, 21, 22
ADC	Analog-to-Digital Converter 5, 6, 10–12, 20–22, 25, 28, 32, 36, 46, 51, 52
AWG	Arbitrary Waveform Generator 46
AXI	Advanced eXtensible Interface 30–32, 50, 56
BFM	Bus Functional Model 32
CAN	Controller Area Network 1, 2, 4, 7–10, 14, 18, 19, 21, 22, 27, 35, 39, 43, 46, 50, 51
CAN FD	CAN Flexible Datarate 9, 51
CPU	Central Processing Unit 5, 8, 30, 32, 39, 46, 47
CRC	Cyclic Redundancy Check 8, 37, 46
CSMA/CD	Carrier Sense Multiple Access with Collision Detection 9
DAQ	Data Acquisition 51
DDR	Double Data Rate Synchronous Dynamic Random Access Memory 32
DMA	Direct Memory Access 30, 31
DTC	Diagnostic Trouble Code 50
DUT	Device Under Test 12

ECU	Electronic Control Unit 1–4, 7, 8, 14, 50
EDA	Electric Design Automation 24, 42
EMC	Electromagnetic Compability 7
EMI	Electromagnetic Interference 15
EOF	End Of Frame 28, 31, 35
FFT	Fast Fourier Transformation 11, 23, 50
FPGA	Field Programmable Gate Array 5, 6, 27, 31, 32, 34, 43, 46, 49, 50, 52, 56
FSM	Finite State Machine 28, 31, 32
GND	Ground 9
GPIO	General Purpose IO 52
GUI	Graphical User Interface 34, 48
HDL	Hardware Description Language 50
HIL	Hardware-In-the-Loop 1, 50, 51
I/O	Input/Output 7, 8, 40
IC	Integrated Circuit 24, 25
IFFT	Inverse FFT 23, 50
IP	Intellectual Property 32, 46
Lab	Laboratory 1
MMU	Memory Management Unit 46, 47
MWE	Minimal Working Example 27, 34, 48
NRZ	Non-Return-To-Zero 22
OP-amp	Operational Amplifier 17, 22–24
OS	Operating System 34, 46
PCB	Printed Circuit Board 27, 42, 52
PDN	Pull Down Network 43
PWM	Pulse Width Modulation 15
RAM	Random Access Memory 30–32, 34, 39, 47, 51
SDK	Source Development Kit 27, 34

SoC	System on Chip 5, 6, 31, 32, 34, 46, 52
SOF	Start Of Frame 28, 31, 35
SR	Shift Register 28, 32, 51
STDIO	Standard IO 46
STDO	Standard Out 34
TCP	Transmission Control Protocol 27, 34
TLB	Translation Lookaside Buffer 47
TP	Twisted-Pair 28
UART	Universal Asynchronous Receiver/Transmitter 46
USB	Universal Serial Bus 51
VCC	IC Power-Supply Pin 9
VHDL	Very High Speed Integrated Circuit HDL 41, 42



# Chapter 1

## Introduction

This chapter aims to help the reader understand the background to the problem by presenting known information about the case in a structured manner. First a background of the group and their mission is given, then the task, limitations and finally a solution is proposed. These are here to immediately give the reader a clue about the problem, how to solve it and why someone wants to solve it.

### 1.1 Background

The group RESI – Integration Test Automation at Scania CV tests the electrical systems in the vehicle with the help of one of the largest Hardware-In-the-Loop (HIL) labs in the world. The Laboratory (Lab) contains numerous Electronic Control Unit (ECU) which all are connected to one of the multiple Controller Area Network (CAN) buses that are set up in the lab. This is needed since Scania offer their clients a modularised system where almost everything can be changed depending on what the clients are interested in. This makes the number of combinations that a vehicle can be specified as extremely high and it is therefore necessary to have a lab of this sort. The tests run in the lab are mostly large over-night suites of regression tests<sup>1</sup>.

This modularised system is excellent for customers and professional drivers who can specify their vehicle as they find appropriate and then contact Scania to have it

---

<sup>1</sup>A regression test is a software test used to discover bugs introduced by new software releases.

manufactured. However for the ones working at research and development departments it requires a whole other level of development. Today there are nearly 100 ECUs placed in the lab along with numerous sensors, actuators and huge Simulink models to be able to simulate a working lorry or bus. All these factors introduce possible faults and therefore troubleshooting can be time consuming. This thesis, and the demonstrator developed, aims to help RESI with additional information of the system and hopefully lead to less troubleshooting and in the long run increase the uptime of the lab.



Figure 1.1: Ilab3 before installation

Today there is no possibility to record electrical signal information automatically and in a controllable way that leads to a deterministic log file. The goal is to specify and implement such a system that makes this possible as well as saving logical data in a new way. By placing multiple units of this kind in the lab and on the same CAN bus it will be possible to see how the electric signal propagates along the bus. Voltage drops, signal propagation, distortion[1], differential mode interference, common mode interference and so on would be possible to study in a way that is not possible today. This obviously presents challenges when specifying the system. One must design a system that samples fast and accurately, without affecting neither the CAN bus nor the sampled signal. To make it possible to compare data from two different units a high accuracy time base must be shared between the units. It must also be possible to send a synchronize command downstream to multiple units.

Furthermore when sampling high-speed and with high bit resolution, data communication speeds becomes an issue. There are many more topics and aspects that needs to be addressed when specifying a measurement system like this, and this

report will cover every aspect that was possible to address during the limited time of 20 weeks. The report begins with a theoretical background and what aspects that can be addressed there, continues to a quantitative chapter where measurements are provided. Finally a specification of the system is presented and the final implementation.

## 1.2 Task

To specify and implement a node belonging to a distributed measurement system that samples the electrical bus, performs necessary checks on the sampled data and processes it. By storing electrical data and enabling offline analysis and comparison between arbitrarily chosen measurement points on the physical bus, the knowledge regarding the lab is further strengthened and common errors could be identified. When the embedded part of the system is implemented, the thesis can continue into developing software for offline analysis of gathered data, by comparing the data from different nodes.

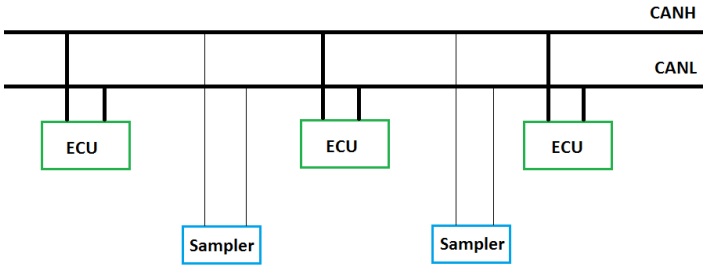


Figure 1.2: Idea of use

The figure above, 1.2, roughly displays the idea of how the system is to be used in the future. By placing multiple nodes on the system bus, the load and electrical characteristics of the data communication can be studied. Multiple nodes placed between ECUs enables comparison of received data, timestamps and so on. It also makes it possible to sample the bus and to store data for further analysis in a controllable manner.

One great benefit is that it is possible to detect where ECUs that send certain in-



formation are located. As seen in figure 1.3, by comparing the time stamp and received data it is possible to calculate where the ECU is located. This is incredibly useful if one ECU were to behave in a non-deterministic manner, and it would be possible to disconnect it from the bus to study the difference.

If an ECU were to send out erroneous data at  $t_0$  and the CAN samplers would receive this message at  $t_1$  and  $t_2$  respectively, a high level software could study this difference and locate in what direction the ECU would be. Depending on bus layout, how the CAN sampler nodes are placed and how much information that is known about the system in the software, it would be possible to pinpoint exactly what ECU that is malfunctioning.

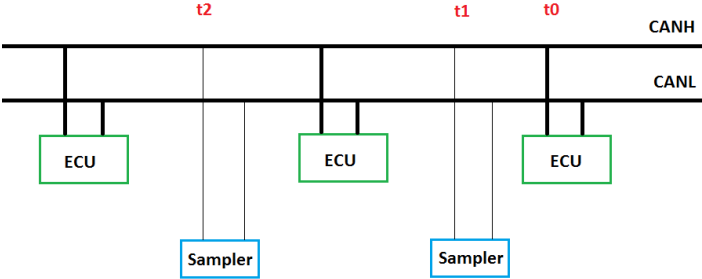


Figure 1.3: Locate ECU via time stamp

### 1.2.1 Specification part

The first part of the thesis constituted of specifying a complete system that samples the CAN bus and does initial processing of the data collected. The system had certain requirements that were known before the thesis started, and some that were discovered along the way. In the requirements specification, appendix A.1, all of this is listed. This part mostly consists of discovering how to implement certain parts, how to avoid pitfalls and what techniques that are best practice. This specification will not only improve the quality of the prototype and understandability of how it works, but it will also make it easier to change the underlying hardware and/or software when every detail is known.

## **1.2.2 Implementation part**

When the first stage of study was done, development of the system started. To make it easier to read the report and study certain parts, the implementation chapters are divided in the following manner

1. Input stage development
2. Hardware development
3. Field Programmable Gate Array (FPGA) development
4. System on Chip (SoC) development
5. Software development

The development was not done in this particular order, however it is appropriate to document the process in a bottom-up like manner. Information regarding interconnects as well as the system itself is provided, assumptions and ideas are motivated. Since this was a rather practical thesis this is the part that took most of the time. Included in this is testing and verification since the results need to be verified to be trustworthy.

## **1.3 Limitations**

This report describes the work before and after the implementation, as well as additional discoveries and knowledge gained while working with the project. Since there is a time limit of 20 weeks only one of the nodes described above will be implemented. Furthermore, the implementation will use evaluation boards where applicable, reducing development time further.

Measurements will be conducted with the unit and an oscilloscope as reference with a comparison between the two. Comparison will also be made against simulated data.

## **1.4 Solution proposal**

The solution proposed, at a first glance, was to build the system with an evaluation board containing either an FPGA, a Central Processing Unit (CPU) or a SoC-solution. This would, together with Analog-to-Digital Converter (ADC) constitute

the main block of the solution. However, after some research, it became obvious that one had to construct some analog circuitry to avoid aliasing of the signal. The more mature solution proposal included the following:

- Evaluation Board containing SoC (FPGA+CPU)
- Two ADCs
- Two low-pass filters

The reason for ending up with a SoC-based solution instead of a pure FPGA-solution was the increased computational power compared to the price. Since the demands for this application might change in the future, an over specified alternative was chosen to be sure that the evaluation board would never be the limit. Rather fast ADCs were proposed, 10 MS/s with a bit-width of 14 bits. However, this in turn sets constraints on what input we can accept and still trust the output. Therefore it was necessary to design and implement an input stage that does not put any strain on the bus, and filters the incoming signal. During the specification phase, the design of the input stage changed multiple times. The final solution is an attempt to minimize the number of discrete components used while keeping best possible transfer characteristics, making it as simple as possible to replicate.

In addition to this, software for connecting to the evaluation boards and viewing the data should be developed. It should be possible to send a reset in such a way that it reaches all hardware units simultaneously. This makes it possible to measure propagation time on the bus along with other data.

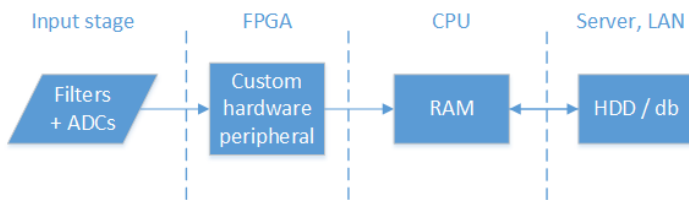


Figure 1.4: Coarse top block design

Figure 1.4 displays an overview of the separate blocks. The idea is as mentioned to be able to instantiate the custom hardware placed on the FPGA and manufacture multiple input stages, and thereby enable easy deployment of the system.

# 1.5 Purpose

Today, when there is an error on the bus, i.e. one of the ECUs is sending error frames because of malfunction or undesired behaviour, it is not always possible to know what unit that sent it. The solution to the problem is disconnecting the ECUs one by one to find which one that has entered an error state [2].

Having a dedicated electrical measurement system with multiple nodes, it would be possible to compare the electrical data sampled from the bus between nodes. This would make it possible to discover where errors, Electromagnetic Compability (EMC)-problems, and other unknown problems occur. Since the wires used for CAN and Input/Output (I/O) in the lab are not routed with EMC in mind, there are problems with interference. Furthermore, if one ECU is having problems and therefore transmitting error frames or behaving erratically, it is possible to calculate which one it is. And when the ECU is identified, remove it from the bus until a solution of the error has been produced.

# 1.6 Disposition

This report is divided into chapters which in turn are divided into sub chapters. The main chapters include Introduction, Method, Result, Discussion & Conclusions, Appendix and finally References. Sub chapters are more in-depth covering a specific matter, and in some cases there are sub-sub chapters describing in even more detail. The first chapter introduces the problem, background and task. The second provides the reader with needed background information and methods to reach the goal. The third chapter presents results and data, and the final chapter of interest discusses the implementation, conclusions and suggestions for further development.

# 1.7 Controller Area Network

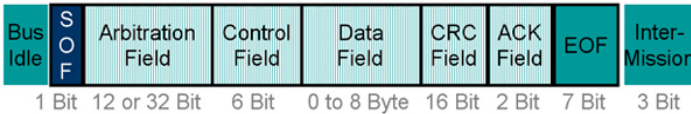


Figure 1.5: CAN data frame[3]

In figure 1.5, the CAN data frame with its separate fields are displayed. This section aims to describe what the different fields do and how the CAN protocol works.

The CAN bus is a serial, differential bus used in vehicles and heavy industry for communication between ECUs connected to the bus. Since the vehicles produced today are growing more and more electrically dependant and complex, a reliable communication protocol is required. Features of CAN is no addressing, requesting of remote frames and error frames. Due to the no adressing scheme, everything is broadcasted on the bus and every ECU decides how to act on a message. If an ECU needs data from another ECU, they send a remote frame and the receiving ECU will read this, identify the frame as a remote frame and fill in the rest of the message with the missing data. This feature enables the requesting ECU to get an update of sensor values, without having the sensor connected directly to I/O and thereby have to waste precious I/O and CPU time for processing of this data. This feature of the CAN protocol is implemented in two different ways in the ECUs, either letting the CAN hardware controller have a snapshot of every value that it could be asked to transmit or by letting the host in the ECU, i.e. CPU, know the message details and act accordingly[4].

Another feature of the protocol is error frames. CAN employs five different techniques to make sure that the message placed on the bus is identical with what the ECU intended to send. The techniques are the following: Bit monitoring, Stuff check, Form check, Cyclic Redundancy Check (CRC)[5] and Acknowledgement (ACK) check. Bit monitoring is rather simple, but very effective. By comparing the voltage output from the CAN transceiver to the bus, with the voltage read by the CAN transceiver from the bus, it is possible to detect that data is sent correctly. Stuff check counts the number of identical bits in a row, there can only be five bits in a row without an inverted bit placed in between as stuffing. If a sixth bit is found were it is not allowed to have sixth consecutive bits in a row, a stuffing error has occurred. The third check, form check, checks that the message employs all field delimiters in a correct way.

CRC is an easily implemented check-sum method that calculates a value to be used to correlate with the payload of the packet. If the packet does not equal the calculated CRC on the receiving side, an error has occurred. ACK check is a simple check that at least one ECU receives the message correctly. If no ACK check is sent on the bus an acknowledgement error has occurred.

If one or multiple errors occur during transmission, either the sender or receiver asserts an error frame on the bus. The unit that discovers the error is responsible for asserting the error frame. The frame consists of a primary error flag of 6 bits, then a secondary error flag of 0-6 bits and finally an error delimiter of 8 bits. The speed

of the bus depends on the controllers and range from 125 kbits/s to 1 Mbit/s for CAN Flexible Data Rate (CAN FD). There is even a single-wire CAN version that communicates at 33.3 to 83.3 kbit/s. Real-time demands is what sets the minimum limit for the bus. A component that does not need to signal as often as another, can be placed on a bus with lower overall speed, or they could even reside on the same physical bus. This is something that is up to the architects of the system to decide.

To avoid collisions, CAN uses a bus access strategy where the transceivers employ bit wise prioritisation rounds. The transceiver who have lower priority yield, and let the transceiver with higher priority send the message. After this, a new prioritisation round can start. Due to this, the CAN bus can be scheduled very high. In short this protocol technique can be described as Carrier Sense Multiple Access with Collision Detection (CSMA/CD)[6].

### 1.7.1 Physical Layer

The CAN protocol defines multiple communication speeds. With the multiple speeds the physical layer is slightly changed. There are three configurations:

- Single-wire CAN
- Fault Tolerant CAN
- High-Speed CAN

Which, in turn, employ different termination styles. The three techniques are termination between CAN Low ( $CAN_L$ ) and CAN High ( $CAN_H$ ), termination from respective wire to IC Power-Supply Pin (VCC)/Ground (GND) and single termination for single-wire CAN. The terminations styles for fault tolerant CAN and high-speed CAN are viewed in figure 1.6 and figure 1.7.

Single wire CAN, low-speed CAN, uses a single termination resistor of 9.09 k $\Omega$  for termination. Because of the physical implementation, this configuration can only be used for low speed due to lack of noise immunity.

The fault tolerant configuration uses a node-side termination.  $CAN_H$  and  $CAN_L$  are terminated separately. One is connected to GND and the other to VCC.[7] Depending on the number of nodes connected to the bus, the resistor termination values are changed. The resistor values range from 500  $\Omega$  to 5 k $\Omega$ [8].

The high speed configuration only terminates at the ends of the bus, and terminates between  $CAN_H$  and  $CAN_L$ . The connections for CAN transceivers should be kept less than 1 m long.[9]

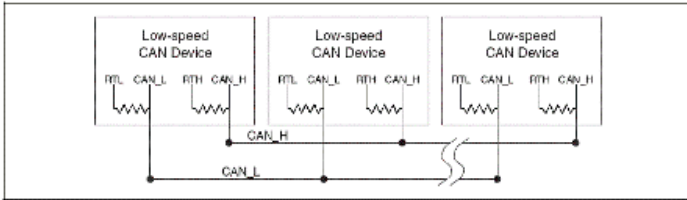


Figure 1.6: Fault tolerant CAN termination[9]

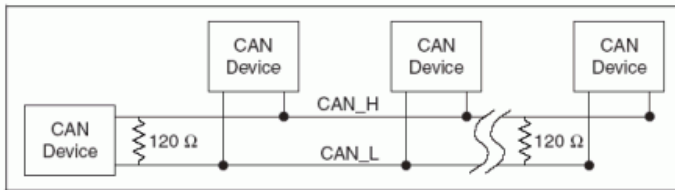


Figure 1.7: High-Speed CAN termination[9]

The system specified and implemented in this thesis only supports communication at 500 kbit/s, 250 kbit/s and 125 kbit/s, and therefore only the high-speed CAN configuration is supported. With modification of the system it would, however, be possible to also connect to a CAN bus running one of the other two configurations and thus on lower speed[9].

The reason for the different kinds of termination is because when the communication speed increases, the requirements on termination is increased to meet requirements on noise immunity.

## 1.8 A/D sampling

When sampling an analog signal for analog to digital conversion there are multiple aspects that needs to be taken care off. How many bits one sample should contain, at what sample rate shall the samples be recorded, how shall the ADC be interfaced and so on. But this is just the tip of the iceberg, when these parameters have been decided upon, one must carefully study datasheets to make sure that converter characteristics are really what is needed for the application. Constraints on bandwidth, voltage and/or current supply and similar enforces a careful selection. Choosing

how many bits the ADC shall operate with is simple. Since every code represent a specific value, by knowing the required resolution, one can calculate what bit width of the ADC that is required. An example is displayed below.

Assume we have an incoming signal with an amplitude of 4V, and need a resolution of 4mV:

$$2^2/2^{-8} = 2^{10} \tag{1.1}$$

As seen above in (1.1), the example gives us a bit width of 10. This does not account for extra codes for detection of over- and underflows, but can be used as an initial guideline.

However, this is as mentioned only a theoretical bit-resolution. Some ADC's have unreachable codes and usually imperfect linearity which might make some codes unreachable. It is also wise, as mentioned, not to use the full range of codes available, since there is a high risk of saturation. By having some space, this is avoided and no data is missed, caveat the loss in potential precision.

The sample rate of the ADC, i.e. the speed, can be chosen in multiple ways. If one knows the frequency of the desired signal to sample, there are several valid responses. Depending on how well the signal should be able to be reproduced, one might consider a multiple of the frequency required according to Nyquist theorem. Therefore a signal is usually oversampled with a factor to be able to reproduce the original waveform with certainty[10].

Some signals, like square waves and sawtooth waves, are composites of multiple frequencies that continue up to a certain frequency depending on the rise and fall time. A perfect composite wave would contain multiples up to infinity. With these kinds of waves it is for obvious reasons not possible to take the previous approach. Instead, there are two, quantified, methods that can be used. Either, one can measure the rise and fall time of the signal and via Nyquist's theorem determine the sample-speed that is required to be able to detect the level transition. The other approach is to use a Fast Fourier Transformation (FFT)[11] to transform the signal from the time base to frequency base. By doing this, one can study were the amplitude of the harmonics have so low amplitude that they will almost have no affect on the sampled signal, and then simply choose that frequency times two as a sample-speed.



## 1.9 Signal Conditioning

Another important aspect to cover is signal conditioning. Depending on what ADC that is chosen the need for pre-processing might vary. Most of the ADC available at the market today offer user selectable reference voltages. This might eliminate the need for amplification or attenuation of the signal. However, the input voltage range must be with such an amplitude that the ADC can detect the changes, if not, signal conditioning is needed. When measuring, it is important not to put any strain on the Device Under Test (DUT) since this might change the value of the unit measured. One might have a high-impedance input in the first stage, and then do attenuation and filtration after that. However this introduces the need to study the frequency dependency of the first stage so that it does not affect the amplitude and phase of the signal. Even though discrete components are linear at low frequencies, when reaching higher frequencies their characteristics change. To compensate for this and the malicious transfer function, either an analog or digital approach can be taken. This is however out of scope for this project, but it is worth mentioning.

Just as the input stage might affect the transfer function in a way that is not intended, analog filtering of the signal will affect amplitude and phase in a way that is not desired. When sampling a signal, it is desirable to use a low pass filter to attenuate signals who have a frequency that is greater than half the sampling frequency. This is due to the fact that higher frequency signals are aliased into the sampling spectrum with the frequency

$$|f - n \cdot f_s|, n \in N = f_{alias}(N) \quad (1.2)$$

If they are not attenuated sufficiently. This causes false entries in the spectrum plot and might lead to erroneously conclusions.[12]

# Chapter 2

## Method

This chapter covers the theory used during the thesis and the development of custom hardware and software. It is rather condense and instead sources are provided to the reader to study certain topics if it is needed.

### 2.1 Theory

The theory provided in this section covers analog filtering of signals and how the signal is affected by said action as well as termination of signals.

#### 2.1.1 Termination of signals

When a signal propagates along a wire and reaches an end, the contents of the signal is reflected. Depending on the material, properties of the cable and properties of the signal, this may or may not be a problem. By terminating the end of a transmission line, a counter-measure to accomplish impedance matching is taken. This minimizes, ideally eliminates, the reflection. The variable  $Z_0$  in formula 2.1 is the charecteristic impedance of the transmission line.

$$\frac{Z_{term} - Z_0}{Z_{term} + Z_0} = \rho \tag{2.1}$$

There are three extreme cases here that needs to be adressed.

Variable	Value
$Z_{term}$	0
$Z_{term}$	$Z_0$
$Z_{term}$	$\infty$

The first case,

$$Z_{term} = 0 \quad (2.2)$$

is the result of a short circuit, and results in a wave of opposite polarity in the other direction, cancelling out the original wave.

The second case,

$$Z_{term} = Z_0 \quad (2.3)$$

is the state usually desired. With a perfect impedance matching, no signals are reflected.

Third and last,

$$Z_{term} = \infty \quad (2.4)$$

is equal to a open-circuit, and results in a wave of the same polarity returning from the stub, doubling the signal level.[13]

However, as the individual ECUs connected to the CAN bus in the high-speed configuration are not terminated as long as the connecting stubs are less than one meter long, the same approach will be taken for this system.

## 2.1.2 Filtering

When building a measurement were the input is unbound, i.e. the frequency of the incoming signals are in an unlimited spectra, care must be taken. This is due to frequencies higher than the sample frequency will be folded into the result, causing undesired aliasing[14]. Fortunately, there are ways to counter-act this phenomena. By using a filter to limit the undesired aliasing, wether it being a low-pass or band-pass – this can be avoided. The idea of a filter is to leave the desired frequency range unaffected, but at a certain frequency, the Cut-off Frequency ( $f_c$ ), starts to attenuate the signal so undesired frequencies do not infer. The cut-off frequency is defined as the frequency were the amplitude of the signal has been attenuated by 3 dB.

It is important to remember that filters affect the phase. In this case, the sampled signal is modeled as a square wave that consists of a base frequency with multiple harmonics. Since it is the harmonics that give the signal the desired appearance, it is of paramount importance to affect the phase as little as possible.

### 2.1.3 Low-pass filter

The low-pass implementation would typically be of a high-order and have a cut-off frequency near half the sampling rate. There are many different ways in how to implement a low-pass filter[15]. Either by passive components, i.e. RC-net<sup>1</sup> or RC-net combined with OP amplifiers. It is not uncommon either to cascade aforementioned filters to further increase the attenuation and thereby the desired behaviour. This is the type of filter that was chosen for this application, however not a passive filter, but a 2nd order low-pass Butterworth filter.

The simplest of low-pass filters have a transfer function as below:

$$\frac{1}{s + \frac{1}{CR}} = \frac{V_{out}(s)}{V_{in}(s)} \quad (2.5)$$

With the Cut-off Frequency ( $\omega_c$ ) frequency at

$$\frac{1}{RC} = \omega_c \quad (2.6)$$

### 2.1.4 Band-pass filter

A band-pass filter would accomplish nearly the same thing as a low-pass filter, but with the addition that it can filter out signals that are below the lower cut-off frequency as well. This would attenuate interference from power grids (50 Hz in EU)[16] , Pulse Width Modulation (PWM) signals (<250 kHz)[17] and other appliances that radiate Electromagnetic Interference (EMI). Even if the filter usually is implemented as a combination of a low-pass filter and a high-pass filter[18], it increases complexity in the circuit which could be unnecessary. Typical band-pass filter characteristics are shown below in figure 2.1.

---

<sup>1</sup>A RC-net is a resistor and capacitor connected with an outlet in between.

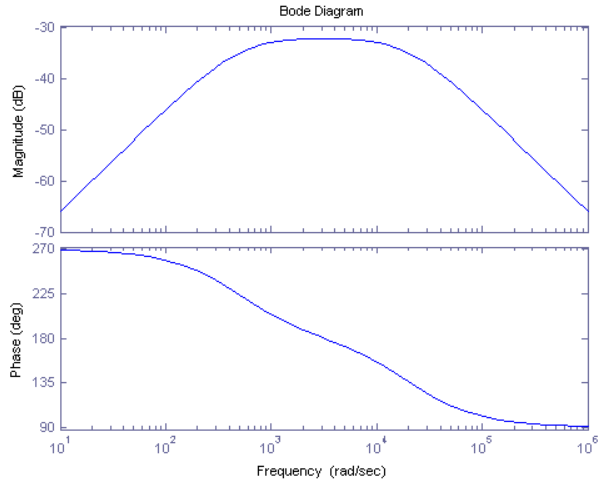


Figure 2.1: Band-pass filter, attenuation and phase margin[19]

### 2.1.5 Phase

By looking at the phase, it is possible to deduce how a signal is affected by the filter. For some frequencies this may have no affect, however when getting closer to the cut-off frequency the signal's phase is changed. This is not necessarily a bad thing, but in data communication applications, where distinct level differences between high and low are needed, it matters. The change in phase results in ringing at the level changes, which is not a problem as long as one is aware of it. The picture below, figure 2.2, displays what happens with the input signal when the phase is changed depending on the frequency of the signal. The blue wave is input to the filter, and the blue is the output signal with phase affected.

How the phase is affected in a more quantified way for isolated frequencies is displayed in the dashed line in figure 2.3. For frequencies significantly lower than  $f_c$ , the phase is not affected. However when approaching  $f_c$ , 5 MHz, the phase is affected resulting in a distorted signal.

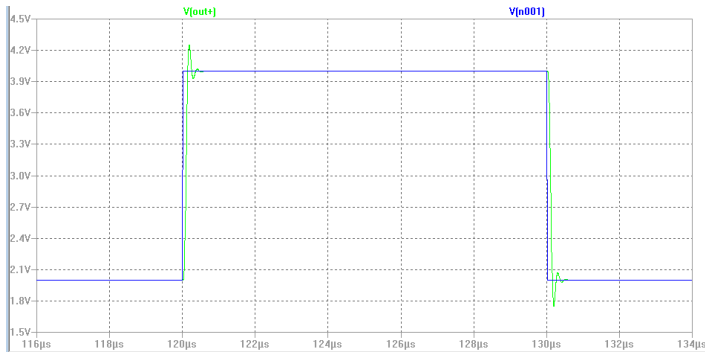


Figure 2.2: A square wave input to LT1568 Filter, simulation

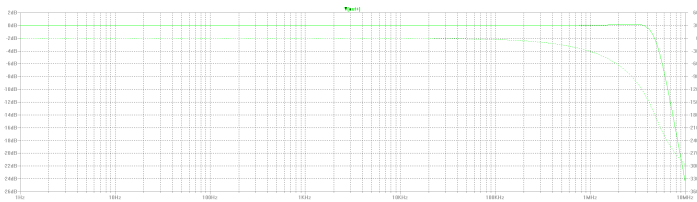


Figure 2.3: Phase changes for LT1568 Filter, simulation

### 2.1.6 Bandwidth

The filter bandwidth is the interval between where the signal is unaffected to where it is attenuated by 3 dB. For filters using only discrete components the bandwidth limit is the same as where the cut-off frequency is, but when using more complex filter solutions involving Operational Amplifier (OP-amp)s the latter might affect the bandwidth, and has to be addressed.

## 2.2 Quantative data analysis

A good way to acquire certain *know-how* about analog-to-digital conversion and signal properties is by measuring with an oscilloscope directly on the bus. This does not only show voltage levels, but also noise, under/overshooting, slew rate and so on. By transforming the signal, one gets an insight in what frequency components that are active. As previously mentioned, it is *not* enough to sample the

signal according to Nyquist's Sampling Theorem, but rather a multiple of that. This is since the physical appearance of the signal must be saved. Sampling of the signal was done and then compared to square waves generated by Matlab with a variety of harmonics included. Since the commercially available CAN transceivers sample rather late in the bit, oscillations are not acceptable. Therefore it is needed to include numerous harmonics to avoid aforementioned oscillations. Also a high sample rate is needed to be able to actually study the measured data and conversion to logical data is needed to be able to track samples. By the mentioned studies and approximation, harmonics up to 5 MHz would have to be included. This requires, as of Nyquist, a sampling rate of  $10MS/s$ . This immediately raises concern.

$$10MS/s \cdot 14bit \Rightarrow 140Mbit/s \quad (2.7)$$

And since we sample each channel separately:

$$2 \cdot 140Mbit/s \Rightarrow 280Mbit/s \quad (2.8)$$

Even if the measurements are only short bursts, the hardware has to handle this. The bus load can be very high due to the bit wise prioritisation protocol employed, and therefore if the system would be used to relay all activity on the bus, a good LAN connection would be required.

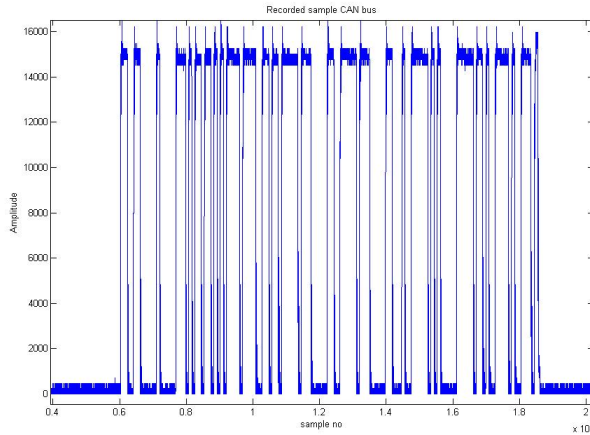


Figure 2.4: A CAN message recorded with an oscilloscope

During this phase, data from the lab was collected and analyzed in Matlab. By looking at the frequency components, amplitude, rise- and fall times just to name

a few, one can determine what the properties of the input stage needs to be. The next reasonable step to take is to transform the data to the frequency plane to study individual frequency amplitudes. By doing this the harmonics are clearly visible. The two figures 2.5 and 2.6 displays this, as well as included noise.

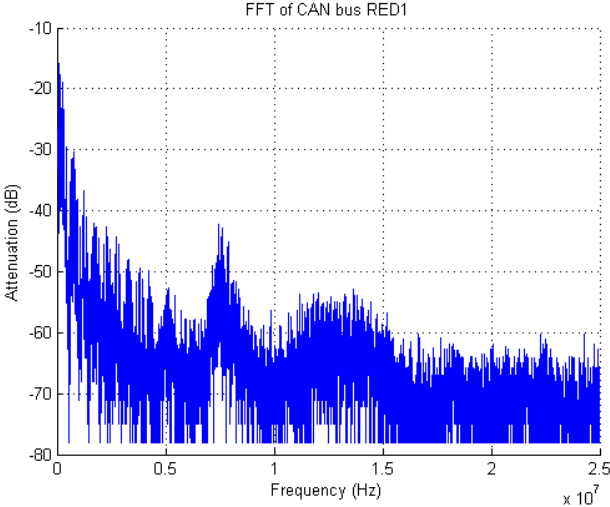


Figure 2.5: FFT of a sampled CAN message, whole spectrum

In figure 2.4, a CAN data frame sampled is displayed. The data was pre-processed in Matlab to re-interpret the Y-value to a 14 bit unsigned integer. Except that, it is as received. By studying the data in question we can calculate rise and fall times, which gives us a more practical idea of what potential frequency components that are present in the signal. Also note the overshoot and ringing at transitions of the signal. This must be considered when parametrization of the ADC is performed, since overflows are undesired. As expected, the noise is clearly visible in the figure, as well as the distinct signal levels.

### 2.3 Requirements specification

The first document produced was the requirements specification. The plan was to early on recognize requirements and scenarios that the system should be able to handle. This would obviously save time later since some problems would be known



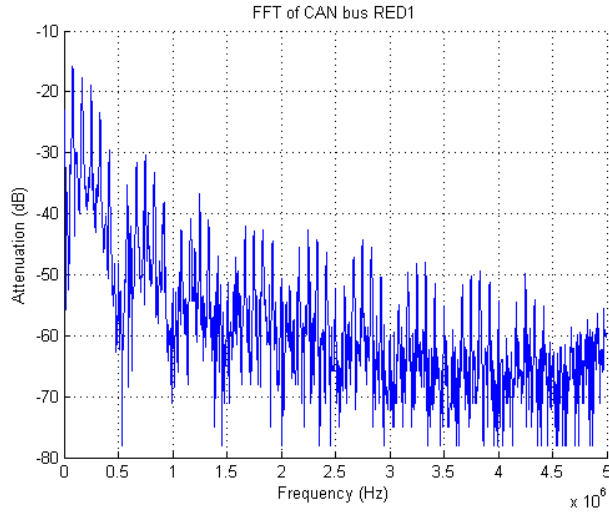


Figure 2.6: FFT of a sampled CAN message, sub 5 MHz

before even starting with the thesis. The system had certain constraints that were agreed upon, as well as requirements, especially for analog-to-digital conversion. Attached in the appendix A.1 is the requirements specification. The chapters are divided between the separate parts during development.

## 2.4 Hardware

The first step when all measurements and approximations were done, was to figure out what hardware that should be ordered. It was decided upon a 10 MS/s sample rate for the ADC. This gives a good view of the signal and lets us sample harmonics up to 5 MHz. It would have been possible to choose an ADC with a much lower sample rate, but since the goal is to be able to study the signal, it is feasible to have a high oversampling. Ideally, a 500 kbit/s data signal can be compared to a 250 kHz square wave, alternating between high and low, giving us about 20 samples per bit

$$samplespeed/(250k \cdot 2) = 20samples/bit \quad (2.9)$$

This enables us to choose a sampling point down to 5% of the period, which is great since we want to avoid to try to convert the sampled data to logical data when the voltage level is not stable. In figure 2.7 a beginning of a CAN frame is displayed, notice the noise. Another example is seen in figure 2.8, where the ACK is heavily overshoot.

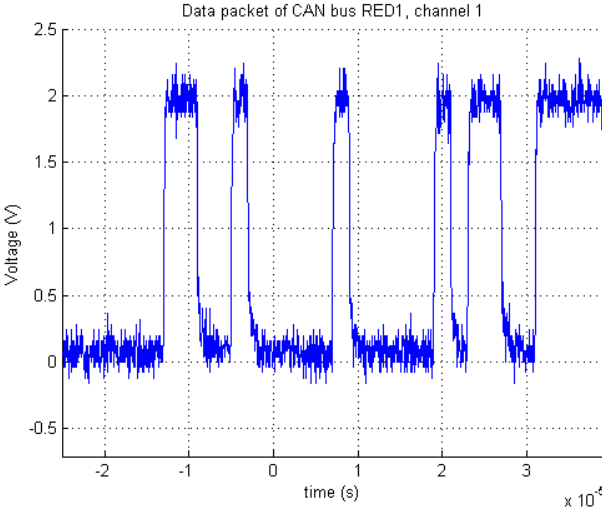


Figure 2.7: Beginning of CAN data frame

This overshooting unfortunately lowers the resolution of the ADC. Since we need to be able to detect voltage levels well out of the voltage levels used by CAN bus, the reference voltages belonging to the ADC have to be set so that no information is lost. The current bit resolution,

$$(v_h - v_l) / 2^{14} \text{V/bit} \tag{2.10}$$

$v_h$	8
$v_l$	0

gives us a corresponding bit value of 488  $\mu\text{V}$ .

When converting from sampled measurements to logical data, it is important that a good sampling point is chosen. Sampling point in this context means which of the multiple samples that should be chosen for conversion to logical data, i.e. logical high or logical low. In theory any value should be as good as any other. But since

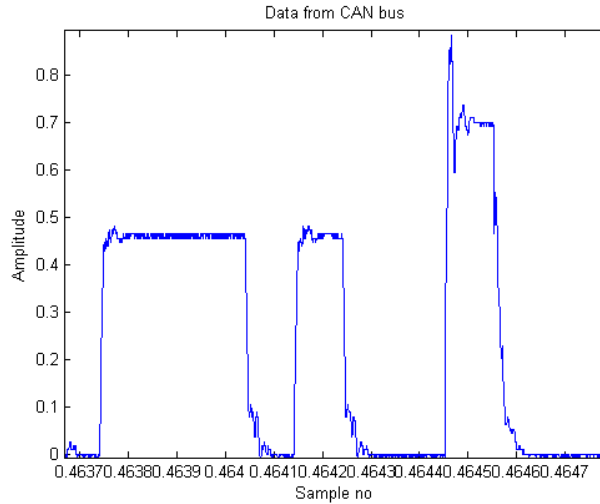


Figure 2.8: CAN ACK overflow

we employ a low pass filter on the input as well as the ADC only samples with 10 MS/s and the fact that the signal is very noisy, care has to be taken when sampling. By studying the datasheets of two classic CAN transceivers, it was found that the sampling point lies around 62.5% and can be moved around by setting special registers inside the controller. Another mode is also available, by choosing three samples in a row and then performing a majority decisions on aforementioned samples. Unfortunately this resolution on sample point is not available in this implementation, but 60% or 65% can be chosen. The majority decisions scheme has not been implemented, as the data sheets suggested that this was for the slower versions of CAN.[20]

If a bad sampling point is chosen, e.g. in either the beginning or the end of the bit, unstable signal level might lead to the wrong decision being taken by the hardware.

When specifying an OP-amp for the input stage, certain known facts can be exploited. Data sent usually alternates between HIGH and LOW, giving a maximum frequency of 250 kHz. With the same technique, the minimum frequency can be calculated, since the CAN protocol employs a Non-Return-To-Zero (NRZ) coding which means that no synchronization information is sent with the signal, and therefore the signal level must change repeatedly so that none of the receivers lose synchronization. This boils down to that every sixth consecutive bit a level change

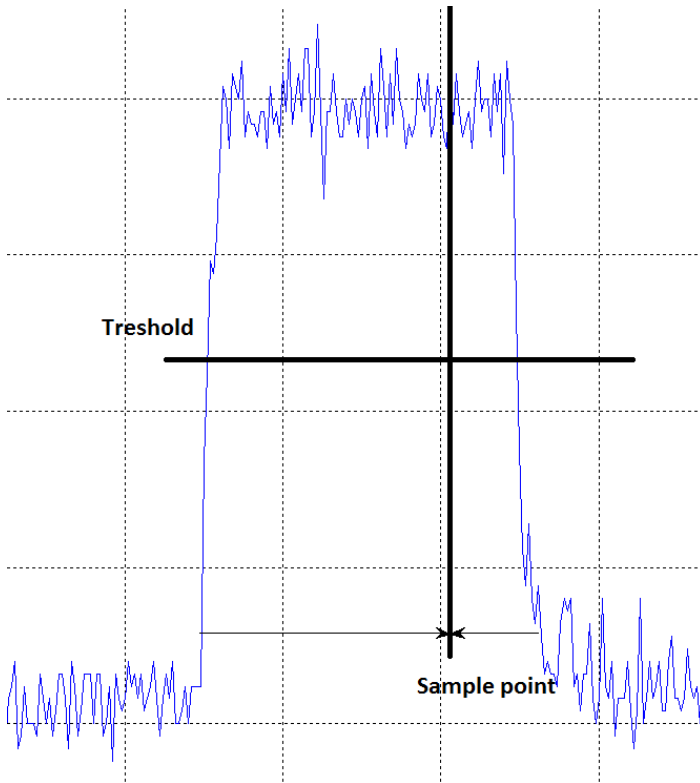


Figure 2.9: Sample point in a bit

must occur, otherwise a stuffing error has occurred. Due to this, the lowest possible frequency allowed, by the protocol, is 250 kHz divided by 5, which is 50 kHz. So the OP-amp used in the thesis need to have a bandwidth of at least 5 MHz, and a linear gain from 50 kHz to 5 MHz. The linearity of the gain can be discussed, as that is highly unlikely. However, if the gain is close to linear, it can be compensated for in HW. This compensation would be done by transforming the signal with FFT, multiply the bins with a coefficient, then Inverse FFT (IFFT). This is not done in the project, but is definitely possible. The most important thing however is to have a high impedance input. Without this, the voltage level on the bus is affected and the ECUs might not be able to communicate. The input stage is preferably implemented as a voltage follower located before the attenuation of the signal is done. Then it does not affect the bus, as the signal is isolated from the electrical bus.

The slew rate must be considered as well. If the changes in signal level is more rapid than the OP-amp can handle, the signal will be distorted. Since the data is sent as a square wave on the bus, we will have harmonics that together build up the waveform.

Unfortunately this is something that is very difficult to avoid. Every limit like this affects the signal and makes it look differently. Just as when quantization of the signal is done, information is lost. It is important that all these steps affect the original signal as little as possible.

## 2.5 Construction of input stage

By simulating different filter solutions, discussions with the supervisors and communication with retailers of electrical components, several implementations were simulated and thoroughly investigated. The first design contained a chain of Low-pass filters, simple RC-filters. The transfer function of this filter was mostly acceptable, however phase margin and signal level was affected. Signal level is easily counteracted by amplifying the signal, but the problem with phase margin is not as easy to address.

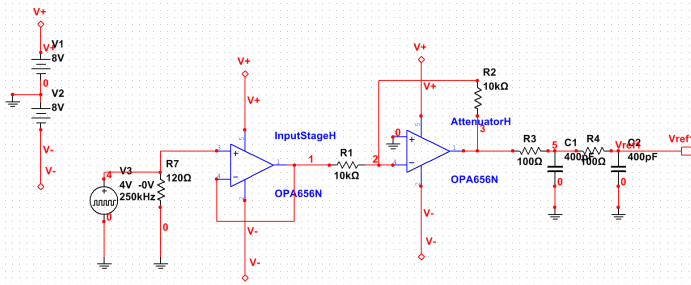


Figure 2.10: RC-chain in an Electric Design Automation (EDA)

By continuing simulations and searches for different filter solutions, an Integrated Circuit (IC) named LT1568 was discovered.[21]

The IC called LT1568 from Linear Technology contains a via resistor programmable low-pass and band-pass filter, and by using this a filter that had satisfactory transfer characteristics could be both simulated and created. The filter is a 4-pole filter that can be configured in two ways, either as a single 4-pole filter, or two 2-pole filters that allows two separate inputs.  $f_c$  is configured between 0 MHz and 10 MHz.

The latter is excellent in this application since it lowers the number of discrete components required to be able to create an operational filter.

The IC was soldered on an experiment board which in turn was connected to a breadboard via cords. This was needed since the IC is in a 16-SSOP socket, as seen in figure 2.11. On the breadboard, appropriate resistors are connected to create the desired filter functionality. Placing the experiment board on a breadboard is for signal integrity not the best choice, but it enables multiple ways of connecting the filter as well as adding decoupling capacitors to increase performance. The way suggested for connection from the data sheet is seen in figure 2.12 were decoupling capacitors have been added and resistors to act as a two-pole lowpass filter.

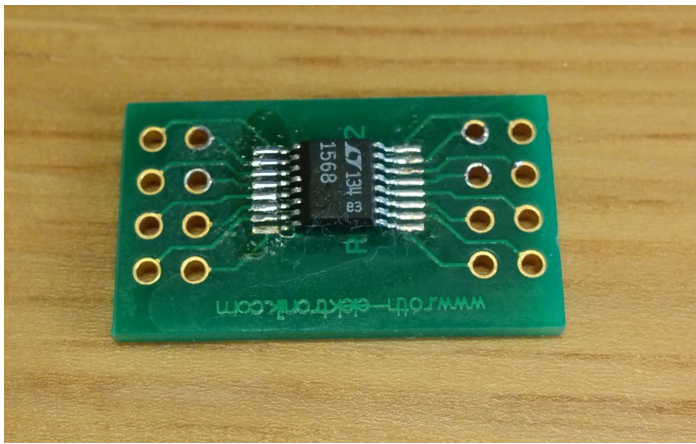


Figure 2.11: Filter circuit on experiment PCB

With appropriate resistors connected, the filter design is complete, and the only additional component required is the ADC. This solution keeps down the number of discrete components required to construct the whole stage, and the filtered signal is fed to the ADC evaluation board. The connection with filter, breadboard and required resistors and capacitors is seen in figure 2.13.

### Dual 2nd Order Lowpass Filter, Single Supply Operation

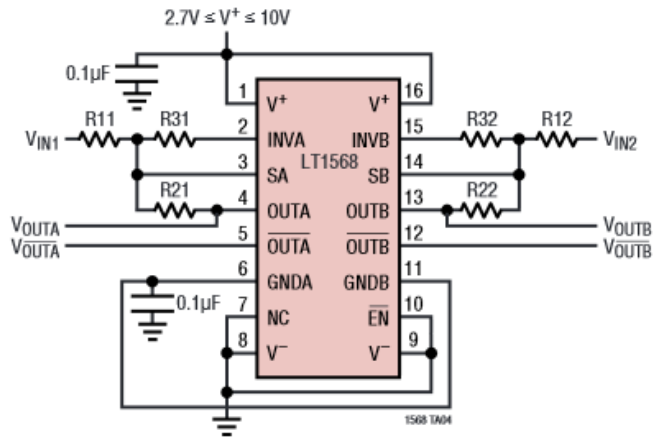


Figure 2.12: Schematic of LT1568[21]

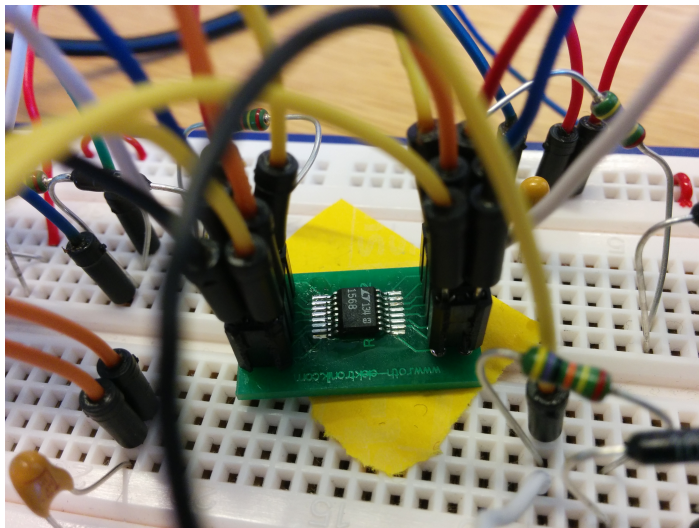


Figure 2.13: LT1568 with jumper cables to resistors

By soldering on connectors similar to those of a breadboard, it was possible to connect the circuit to a breadboard using jumper cables. Then the filter was programmed with resistors and two decoupling capacitors were used. The  $f_c$  was set at 5 MHz as specified to be able to measure actual transfer characteristics.

Since this is a prototype, evaluation boards were used when available. A more mature solution would use a custom Printed Circuit Board (PCB), but due to the time constraint, this was not done.

## 2.6 Development

In the following sections, development of the embedded system is described. Development was done in a bottom-up manner, starting with the CAN receiver placed on the FPGA together with connections from said receiver to the ARM Cortex processor placed on-chip. The input stage was soldered together and verified. When all blocks except the receiving client were finished, everything was verified so that the output of the system matched what was expected. Unfortunately, due to the time limit, the interaction over Transmission Control Protocol (TCP) was not completed. However, a Minimal Working Example (MWE) has been developed. The reason for doing it in this order was that the most crucial part to get working was the hardware parts, and there were less requirements regarding the final software on the desktop side.

Tools used during development were Xilinx's ISE, Vivado Design Suite, XPS Source Development Kit (SDK), NI's Multisim, Linear Technology's LTSpice, Mathworks's Matlab.

### 2.6.1 FPGA development

Several high-level hardware designs were proposed initially and discussed together with the supervisor at Scania. Since the system must handle multiple communication speeds on the CAN bus, the first stage on the FPGA must handle switching communication speeds while being on-line. Without this, the system needs a re-configuration every time either the bus setting is changed or the physical bus is changed. This is obviously not possible if the system should be possible to use with minimum maintenance. The idea proposed to solve this matter was to send a command to the processor to change communication speed.



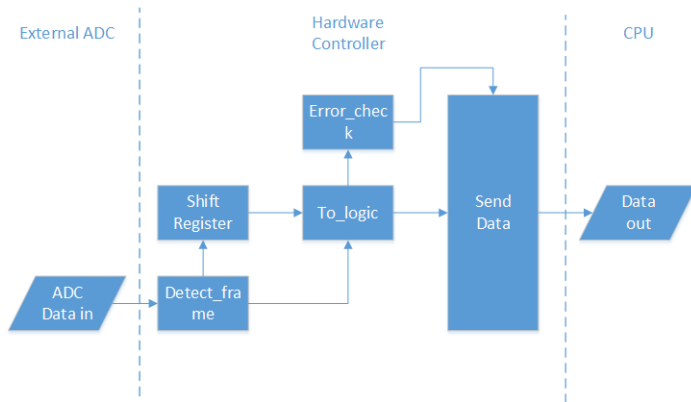


Figure 2.14: Datapath of Proposed Design 1

The first proposed design did not address how to send data from the custom hardware controller to a storage solution, instead it focused on how to handle the incoming data and perform necessary checks on it in order to convert from samples to logical information. Common to all designs are that they have a Shift Register (SR) in the beginning of the data path. This is used to be able to sample and save approximately one bit time of data before the message begins. This is to be able to study the signal level on the bus before the message begins. The various Start Of Frame (SOF) detection methods used look on samples incoming to the SR to be able to trigger data saving. The protocol was studied and quickly it was discovered that certain care had to be taken with edge detection, SOF and End Of Frame (EOF). Therefore another design was proposed that handled this in a more clear way.

Figure 2.15 clearly divides the tasks of finding edges and EOF to specific blocks. Since the first edge also indicates a SOF it was decided unnecessary to have a block that only waits for the first edge. Instead a Finite State Machine (FSM) is used to indicate when a SOF has been discovered, and when an EOF is signaled, the FSM returns to a state called *wait\_for\_SOF*

As the specification moment continued, a requirement to save both  $CAN_H$  and  $CAN_L$  separately presented itself. This is displayed in figure 2.16. Therefore, instead of letting an ADC card do the subtraction, a block is introduced to handle this. This significantly increases the memory requirement, but provides more electrical properties of the signal. Even if the signal is sent over a Twisted-Pair (TP)-cable, there are parts where the cable runs in parallel. Therefore it is beneficial to have the

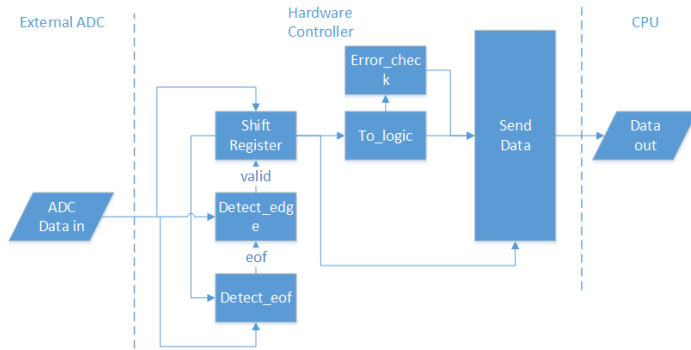


Figure 2.15: Datapath of Proposed Design 2

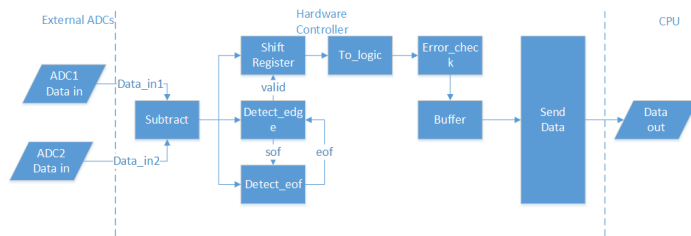


Figure 2.16: Datapath of Proposed Design 3

signal values stored separately. Apart from, that only a buffer is introduced to hold data if the method used to send data is not ready.

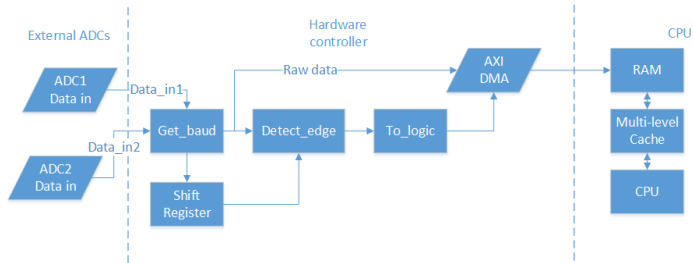


Figure 2.17: Datapath of Proposed Design 4

The major difference in figure 2.17 from earlier blocks is that the error checking block that used to reside in the hardware controller has been moved to software. This is to enable having all error checking on the same level. It might be interesting to have application specific error checking as well and since it is reasonable to do all of this in the same place. The error checking algorithms are simple and do not require many processor instructions, so from a scheduling point of view it is ok to place it on one of the CPUs. This block design is also the first with a specified connection to the rest of the system. There are multiple bus implementations available, but one of the most popular is Advanced eXtensible Interface (AXI) by ARM. It is a bus targeted at high performance, high clock frequency systems and is therefore adequate to use. Instead of using a pure AXI connection an IP provided by Xilinx, AXI Direct Memory Access (DMA)[22], was suggested. This would eliminate the need of an inferred Random Access Memory (RAM) in the design and instead directly write to the CPU's RAM and therefore reducing read/write time.

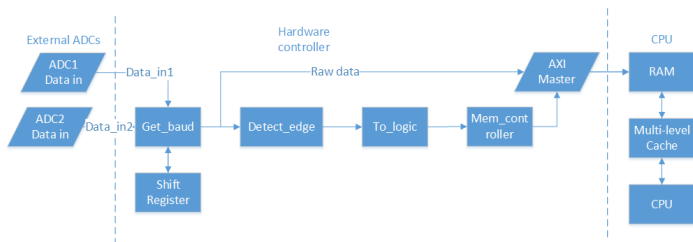


Figure 2.18: Datapath of Proposed Design 5

The final design, seen in figure 2.18, and thus the one implemented, has absorbed functions into blocks to make it easier to get a grasp of the block design. The first block subtracts the incoming data to get the signal value, and concatenates  $CAN_H$  and  $CAN_L$  to be able to store it in one fixed length SR. The next block, detect\_edge, handles edge detection, and incorporates SOF and EOF detection via a FSM. The two most important FSMs are seen in figure 2.19 and figure 2.20, they are responsible for notifying the rest of the controller when edges have been detected and SOFs and EOFs.

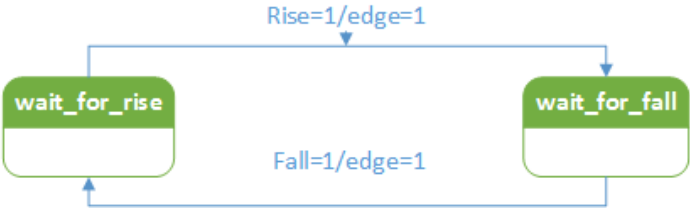


Figure 2.19: FSM edge detection

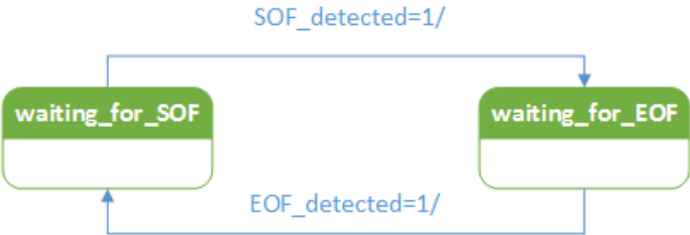


Figure 2.20: FSM SOF and EOF state

For communication uplink, a connection to the AXI system bus is used. The AXI DMA controller has been replaced by an AXI Master, since the interaction with the RAM is taken care of the hard memory controller, no custom memory controller has to be inferred. The Master is connected directly to the RAM and is given it's own memory space, increasing the simplicity further when using multiple nodes on the SoC. This is also beneficial for software development when using multiple nodes on a single FPGA. Since the nodes operate on separate memory segments, no conflicts occur.

This enables the system to be connected directly to the processors RAM, and by doing so making the whole implementation run faster than inferring an immediate

RAM that stores data, and let the CPU fetch data from the aforementioned RAM. Further on, it greatly reduces address-space issues when using multiple nodes on one FPGA. An AXI Master node is given an address-space to act on when used, and therefore multiple controllers would simply have different address-spaces in RAM. This in turn makes software development for the CPU easier.

The block design uses multiple constants that are shared between the blocks in a shared package. This is useful not only for aliasing of constants but also since some design parameters were not known at design time and instead, a mock value had to be chosen for simulations to work to a satisfactory extent. Further on, it increases the quality of the development process since there is no need of magic numbers being scattered in the code, instead clear aliases are used.

Since the development is done on a Xilinx Zynq SoC platform there are multiple generators available for what ever Intellectual Property (IP) one might seem fit. For this design, SR were the only block that was used for the hardware controller. These IP-generators provide a step by step wizard that enables one to specify exactly what is needed, which indeed is handy. To be able to determine what properties that are necessary, the information collected from the quantitative data analysis together with arithmetic was used.

Most of the blocks have FSMs that handle action based on prior input, since the FPGA itself is run with a higher clock frequency than the ADCs, the data path is controlled with *valid* signals that inform blocks when new data is available at the input.

A custom AXI Master interface had to be developed, which proved itself to be a challenge since Xilinx have licensed the Bus Functional Model (BFM), and development had to done without mentioned BFM. This ment that simulations had to be done on platform which increased the iteration time during testing.

## 2.6.2 Hardware development, SoC side

In figure 2.21, the system placed and configured on the SoC, although abstract, is displayed. The custom hardware developed is shown to the left, with connections to other subsystems. As shown, the output from the controller goes to an AXI memory interconnect, which changes protocol from the AXI lite protocol to AXI4. From the memory interconnect, the data is sent to the processor block, which internally connects the hardware controller directly to the RAM via the AXI bus. There is also a clock generator block that generates the sample clock used for ADCs. Finally, there is some fixed IO routed as well as fix Double Data Rate Synchronous

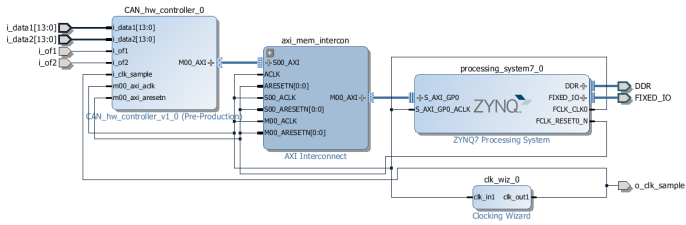


Figure 2.21: System overview, SoC

### Dynamic Random Access Memory (DDR) IO.

Everything is done by Xilinx tool-chain and especially the software Vivado[23], which synthesises, implements and exports the generated bit-file. This file is then loaded over to the evaluation board to program the board.

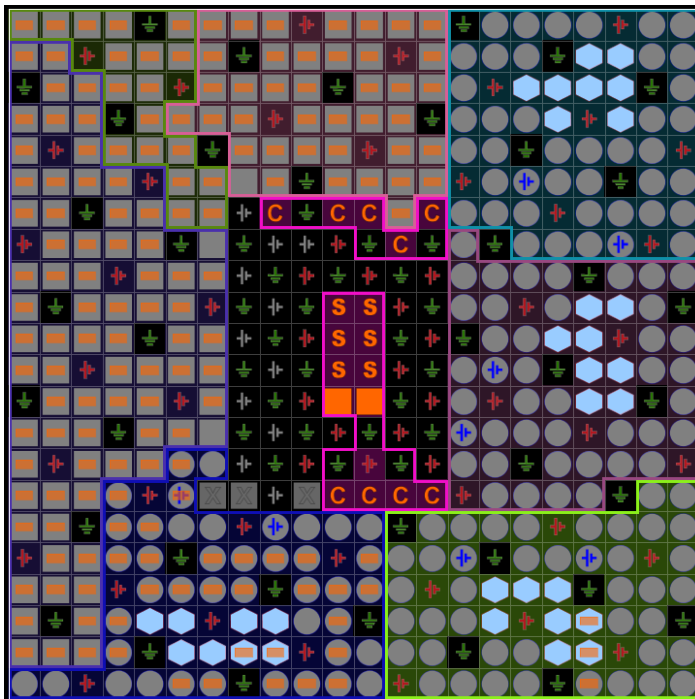


Figure 2.22: Die IO

The FPGA die has several IO banks that are customizable. The reason for this is that certain parts may operate at different speeds and signal level, and by making the banks customizable one can fully utilize the strength of an FPGA. An orange rectangle means that the specific IO is occupied, and as seen in figure 2.22, the upper left corner is fully utilized. This is due to hard routing of certain IO as communication to the RAM controller, other communications and the push buttons available on the evaluation board. The bottom side, pink and dark blue bank, are the banks used by the custom hardware controller. They operate at 2.8 V and 3.3 V.

The utilization reports of both the whole system and only the hardware controller and AXI master are available in appendix, A.2

### **2.6.3 Software development, SoC side**

Xilinx provides multiple libraries of different abstraction level, as well as a complete linux distribution. Since the software requirements were clear and the system needed no underlying Operating System (OS) to be able to fulfill its task, no OS is used. Instead the application is run directly on the processor, with adequate libraries included.

### **2.6.4 Software development, desktop side**

Regarding the software running on desktop side, a MWE has been produced. The MWE consists of one application written in C representing a node in the system, and one application written in Python that represents the application to be run on a desktop computer. Communication between the two applications is today done via a TCP-socket. Since it could not be allowed to drop data packets, TCP-sockets had to be used. The node application connects to a TCP-server socket and writes raw data to it, and when this is complete, it disconnects. The receiving application retrieves data from the socket and prints it out on Standard Out (STDO). A more mature application would probably save the message in a database, notify the user via a Graphical User Interface (GUI) that there is new data available and allow plotting of the data and various mathematical operations on the data set.

This MWE has been produced to display the not needed complicity of a program of this type. The idea is to show that development can be done without interaction with the node, and when the software is finished it can be ported to Xilinx's SDK and run on the SoC's processor.

## 2.7 Testing & Quality assurance

### 2.7.1 Testing of VHDL

Verifying correct timing and behaviour from the hardware controller is important and needs to be verified before expecting to receive correct results when connected with the other blocks. Therefore, it is important to first perform unit tests of the independent blocks, and when they work satisfactory, start with testing the whole controller at once. When this seem to work, get real data to test with. Throughout this said process one needs to be very critical and study the result closely to not let anything go uncaught or neglect any error.

During this project, initial unit tests and tests of blocks mostly tested that when applying data to the inputs, somewhat what was expected appeared at the output. This was done since it was considered the most efficient way of testing in this case. When a block performed as expected, work continued on the next. Soon it was ready to connect all the blocks and make sure that the interaction worked as expected.

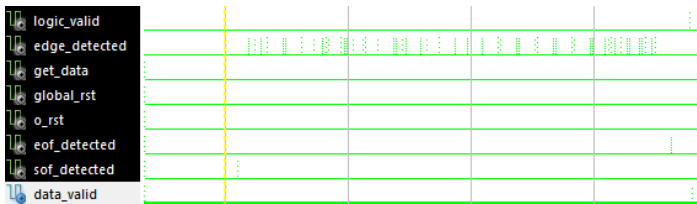


Figure 2.23: Test bench displaying trigger signals from real, sampled, input data

In figure 2.23, the custom hardware controller is fed real data. Signal assertion is quite clear, when the controller detects a SOF, a logic signal named `sof_detected` is asserted, held for one clock cycle, and then de-asserted. For every level transition of the input that occurs, a logic signal named `edge_detected` is asserted in a similiar manner to notify that the input signal changes polarity. Finally a signal named `eof_detected` is asserted when an EOF is detected. When the sampling is complete and logical conversion is complete, a signal named `logic_valid` is asserted. This notifies following controllers that one whole message has been sampled. When retrieval of necessary data such as timestamps and data length are collected, a signal named `data_valid` is raised to indicate a successful sample of one CAN message.



By using the USB-Oscilloscope manufactured by Pico Technologies, it was possible to save sampled data as a comma separated file. Since the interface from the ADC is 14 bit parallel out, this means that the input will come as an unsigned integer 14 bit long. The simplest way to do this is read the file via Matlab, pre-process it and save it back to a file. This file can then be used in a final test bench for the whole system.

Listing 2.1: Preprocessing of sampled data

```
1 fileID = fopen('data_matlab_rounded.txt', 'w+');
2 % Scaling chosen to match bit width
3 for i=1:length(d.A())
4     elm = d.A(i);
5     if elm >= 0
6         elm = elm * 1024 * 6;
7     else
8         elm = 0;
9     end
10    fprintf(fileID, '%d\n', round(elm));
11 end
12 fclose(fileID);
```

In listing 2.1 one of the many scaling scripts is displayed. Data collected from the oscilloscope was stored as a double precision value in Volts, and the controller work with 14 bit unsigned values. Therefore, scaling had to be carried out on the data in order to get it to the format that the ADC's would output. The scaled data is then written to a textfile which is newline-separated. This file is in turn fed to a test bench that reads line for line, and inputs the data as stimuli to the test bench, which in turn stimulates the controller. The test bench is seen in listing 2.2.

Listing 2.2: VHDL test bench with sampled data as stimuli

```
1
2 FILE samplefile: text IS IN
3 "C:\Users\smerk9.GLOBAL\Desktop\matlab\data.txt";
4 VARIABLE l : LINE; — line buffer
5 VARIABLE value, j : INTEGER;
6 VARIABLE c : CHARACTER;
7 VARIABLE str_idata : string(14 downto 1);
8
9 BEGIN
10
11   wait for 200 ns;
12   j := 0;
13   WHILE not endfile(samplefile) loop
14     readline(samplefile, l);
15     read(l, value);
16     — Get the next integer
17     if (j mod 5 = 0) then
18       — Sampled at 50MS/s, downsample to 10MS/s
19       data1 <= std_logic_vector(to_unsigned(value, 14));
20       report " j equals =_" & integer'image(j);
21       i_data1 <= data1;
22       wait for 100 ns;
23       — valid data hold for one sample clock
24     end if;
25     j := j + 1;
26     — update index variable
27   end loop;
28
29   report "End_of_testbench";
30
31   wait;
32 end process getSamples;
```

The test bench seen in listing 2.2 opens a pre-processed newline separated file, reads a line and converts it into a 14-bit unsigned integer value. Then the value is sent as stimuli to the controller developed and holds that value for 100 ns. Since the PicScope ran at 50 MS/s as sample speed, only every fifth sample is fed to the controller.

The steps above enables running system with real data, albeit pre-processed. This is ideal to discover errors regarding sampling, CRC and so on.



# Chapter 3

## Result

### 3.1 Hardware oriented

Below is a comparison between hardware simulations and measurements. This range from measurements for the so called input stage containing filters, to the evaluation board itself. In this case, measurements are made by Xilinx software/IP ChipScope inside Vivado. Measurements are done with the help of an oscilloscope, function generator and other miscellaneous measurement tools.

The final implemented system on hardware is one CAN controller that handles both logical data and electrical data. This receiver is connected to the CPU's RAM from where the CPU can read data and write over TCP/IP to a connected computer that stores data and is responsible for post-processing.

Due to the sample speed of 10 MS/s, the shortest detectable distance is 20 m. This is derived by assuming that the propagation speed of the wave on the CAN bus is  $0.67c$

$$\frac{0.67 \cdot c}{10M/s} \approx 20m \tag{3.1}$$

Which is very coarse. By increasing the sample speed the accuracy of this measurement is increased, see figure 3.1 for attached graph.

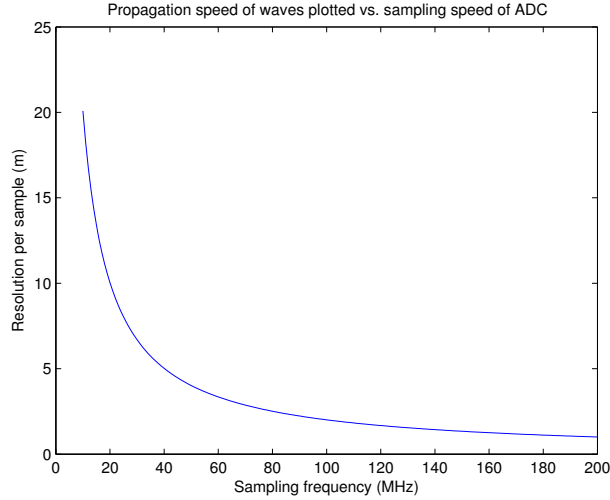


Figure 3.1: Plot displaying how the metric distance between measurements is decreased as sample speed is increased

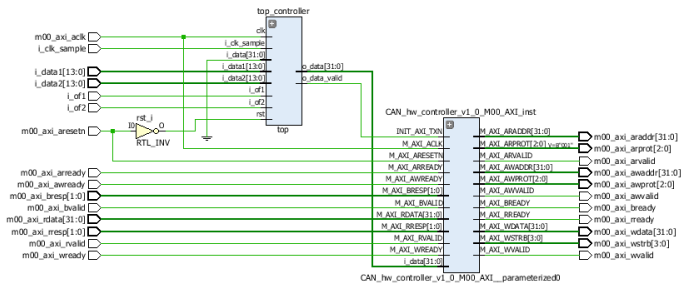


Figure 3.2: Schematic overview, custom hardware controller

In figure 3.2, an overview of the schematic for the custom controller is displayed. Since the choice of communication bus was decided late in the development phase, the bus controller is completely separated from the data controller. This makes it possible to change to another system bus if the controller were to be used on another platform. This also reduces complexity which in turn makes further development and debugging easier. Most of the I/O belongs to the AXI master node, with reset and clock signal shared. The input for  $CAN_H$ ,  $CAN_L$  and overflow

indicators as well as the sample clock are connected to the controller, as expected.

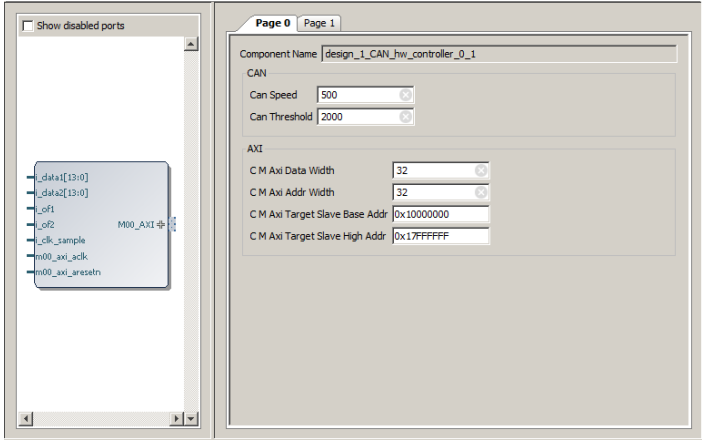


Figure 3.3: Options when instantiating controller

To make instantiation as easy as possible whilst still having it as powerful as possible, VHDL generics were used when appropriate. This makes it possible to instantiate multiple controllers quickly and controlled. Options that are available are seen in figure 3.3, including threshold between  $CAN_H$  and  $CAN_L$ , address range and CAN speed.

The LT1568 filter signal characteristics are displayed in figure 3.4, where a 250 kHz square wave is sent as stimuli to mimic real CAN data. The blue signal is the output from the filter and the red signal is the input. Thereby visualizing the difference in amplitude in a simple manner.

### 3.1.1 Simulations

For simulations of hardware, two different types of programs were used. One type of program for simulating circuit design, and one for simulating the Very High Speed Integrated Circuit HDL (VHDL) code prior to synthesis. For circuit simulation, National Instrument’s Multisim and Linear Technology’s software LTSpice were used. They enable one to study the characteristics of the circuit one is about to design. By doing this, it was decided that a two-pole low-pass filter was enough, instead of having a four-pole which would have imply more discrete components

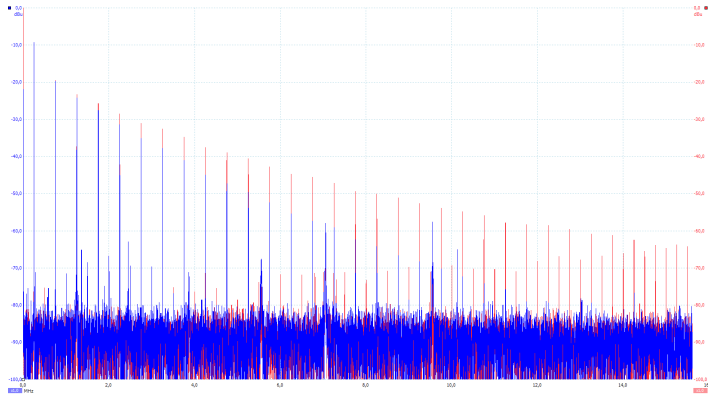


Figure 3.4: LT1568 filter with 250 kHz square wave stimuli

on the circuit board. Further on, various designs with different OP amplifiers, filter design and termination were tested.

Multisim is an EDA software developed by National Instruments that allows one to draw circuits on a computer, and simulate them with multiple techniques. This allows one to simulate the circuit, induce errors and see how the circuit will handle it, without having to manufacture the circuit. This saves a lot of time and lets engineers try out concepts without building anything on a PCB. Furthermore, Multisim provides the user with multiple simulators and handy tools to increase product quality. In this project the software was used to evaluate different methods and techniques concerning filtering of signals, attenuation and amplification.

Another EDA software is LTSpice. When in contact with Linear Technology's support engineers, they claim that their implementation of Spice is much more accurate than the normal. The downside is that they only support their own discrete components and ideal components, limiting the available circuits drastically. Because of this both LTSpice and Multisim were used in parallel.

For simulations of the hardware controller, Isim was used. Isim is a tool that simulates written VHDL code using test benches so that a tester can analyze the behaviour to make sure that it is as desired. During the thesis, several test benches were used to verify the behaviour of every part, and finally one larger to check the design.

### 3.1.2 Measured data

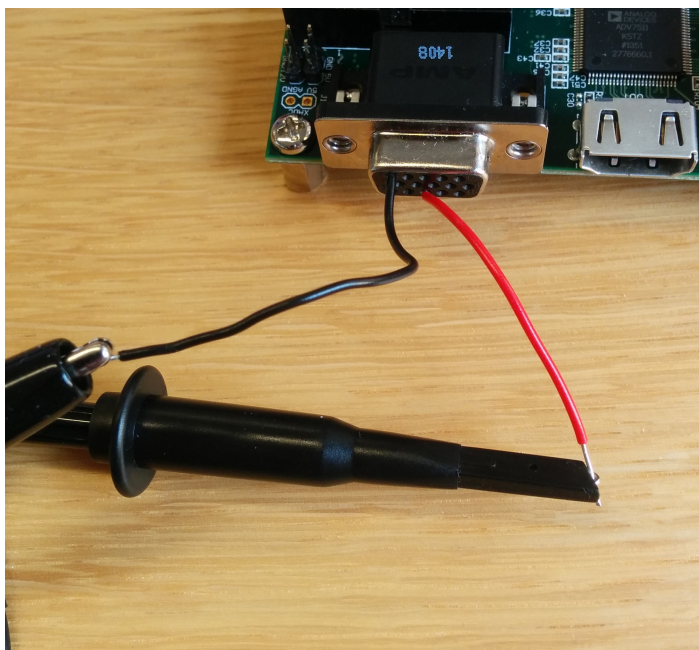


Figure 3.5: Measurement of the clock used by the ADC from the FPGA

The setup to measure the produced clock is seen in figure 3.5. In this instance the VGA connector is used to measure on.

To be able to do this, every input must be in a deterministic state. By connecting unused inputs to a Pull Down Network (PDN), this was sufficiently performed. This is displayed in figure 3.6.

The result of the measurement is shown in figure 3.7.

As mentioned earlier an USB-oscilloscope, PicoScope 3025B, was used for multiple purposes. In figure 3.8 it is used to verify that the sample clock used to sample analog data does not drift.

A previously sampled CAN message was used as stimuli to the FPGA. This message was looped to study how the system handles continuous activity on the bus. Since much of the communication on the CAN bus is real-time communication, where the same message is sent over and over again, the bus load is usually high.



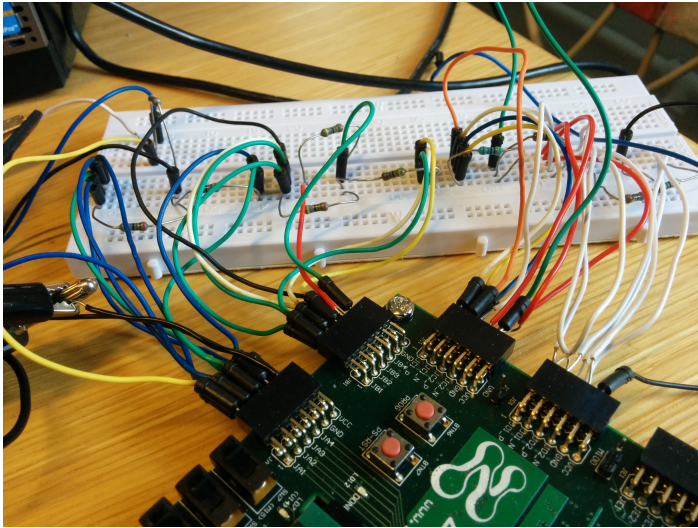


Figure 3.6: Test bed connectors

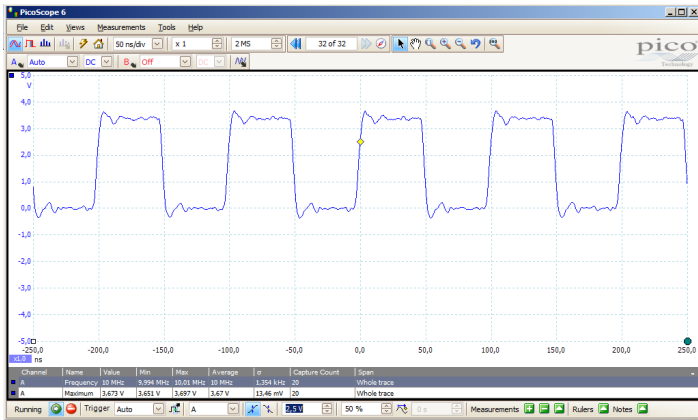


Figure 3.7: Clock used by ADC from FPGA

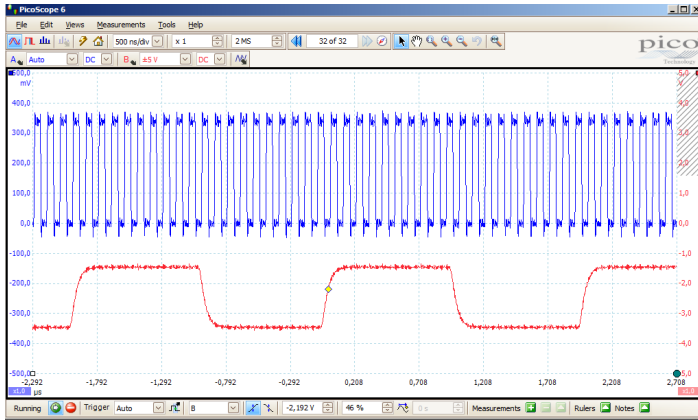


Figure 3.8: PicoScope

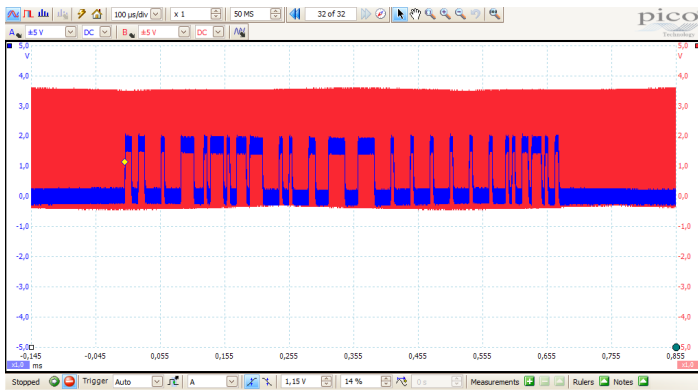


Figure 3.9: Data sent to the FPGA, with the ADC sample clock running as well

Sending *real* data to the system is straight forward. By using a signal generator that contains an Arbitrary Waveform Generator (AWG), it is possible to sample interesting signals, modify them appropriately by eg. inducing CRC-errors, over-underflows and such, and then sending it as stimuli to the system. In figure 3.9, the stimuli together with the ADC sample clock is displayed. The reason for measuring the sample clock is that it is possible to verify the result directly by counting samples.

This is something that is quite unnecessary, since PicoScope ships with a CAN protocol analyzer. It can automatically find the baud rate of the packet, as well as highlighting bit delimiters and everything else that might be convenient.

In parallel with the PicoScope sending data to the system, a desktop computer is connected to the SoC Evaluation board via Universal Asynchronous Receiver/Transmitter (UART). Via ChipScope and by a special IP core placed on the FPGA, it is possible to trig and read out signals from the custom controller. This is useful for validation of how the system is performing while online, and of course for finding out why, when it does not.

## 3.2 Software oriented

The software was developed in C and Python and runs on both the evaluation board and on a desktop computer. The software that runs on the evaluation board is written in C and ran directly on the processor with libraries provided by Xilinx. This software is responsible for retrieving sampled CAN messages from the custom developed hardware and writing it to a socket.

To allow the custom hardware controller to write to the DDR memory, we need to configure the Memory Management Unit (MMU)[24] in an appropriate way.

### 3.2.1 Embedded side

Xilinx provides a Linux kernel that can run on the cores, a micro kernel and other solutions for software development. In this application, a 'bare-metal' application<sup>1</sup> is run on the core with libraries for the TCP/IP-stack<sup>2</sup>, Standard IO (STDIO), and

---

<sup>1</sup>A 'bare-metal' application is an application that runs directly on the CPU without an OS or other support to handle scheduling etc.

<sup>2</sup>A TCP/IP stack handles the four lower levels of communication, Application, Transport, Internet and Link layers

RAM access. This provides seamless interaction over TCP/IP with the desktop side.

Configuration of the MMU is performed to allow multiple masters to write to the RAM. By setting special registers in the MMU to deny the Translation Lookaside Buffer (TLB) of caching as well as allowing multiple masters to write to the physical RAM controller, it is possible to use the RAM as intended. Caching is turned off since we do not want to mistakenly let the TLB exploit the fact that we are reading the RAM in a sequential order. A pre-fetch could result in erroneous result since the CPU might get ahead of the hardware controller writing to RAM. The code in listing 3.1 displays exactly this, by iterating through the address space of the hardware controller and setting the range to appropriate settings, this is performed.

Listing 3.1: Disabling caching and making memory shareable

```
1 // Use defines for adress space of hw controller
2 #define SHAREABLE_DISABLE_CACHE 0x14DE2
3 #define BASE_ADDR 0x10000000
4 #define HIGH_ADDR 0x17FFFFFF
5 /**
6  * Sets the desired memory range to shareable .
7  * turning off caching , making it possible for the
8  * CAN hw controller to write in it .
9  * Sets 1MB chunks at a time , therefore i+0x10 0000
10 */
11 void init_DDR ()
12 {
13     int i ;
14     for (i = BASE_ADDR; i < HIGH_ADDR; i = i + 0x100000) {
15         Xil_SetTlbAttributes(i , SHAREABLE_DISABLE_CACHE);
16     }
17 }
```

The demonstration software listed in 3.2 acts as a proof of concept. Since the controller writes faster than it is possible to write to UART, it is not possible to view a complete message before it is overwritten. However by reading short segments it is possible to verify that content is written to RAM and that the content is as expected. Since the two differential samples are concatenated into a single 32 bit word, to look at individual samples one needs to do a bit-wise *AND* with either *0xFFFF* or *0xFFFF0000* to mask out correct data. The *init\_platform* functions called below is platform-specific instructions and is generated by the tool-chain from the board support package.

### Listing 3.2: Demonstration software – iterating through the RAM and writing to UART

```
1  /*  
2  * Demonstration software, prints via UART  
3  * Data is accessed by (data & 0xFFFF) which is i_data1,  
4  * and (data & 0xFFFF0000) which is i_data2  
5  */  
6  int main()  
7  {  
8      init_platform();  
9      init_DDR();  
10  
11     int i, data = 0;  
12  
13     while(1) {  
14         for (i = BASE_ADDR; i < BASE_ADDR + 0x200; i = i + 0x4) {  
15             data = Xil_In32LE(i);  
16             xil_printf("DDR_content_at_%x: %x\n", i, data);  
17         }  
18         xil_printf("End_of_iteration");  
19         sleep(4);  
20     }  
21     return 0;  
22 }
```

## 3.2.2 Desktop side

Due to the time constraint, there was no time available to develop a receiving application, instead a MWE has been produced. Originally planned and recommended if the work were to be continued would be a desktop application that consists of an application with a GUI that lists data sent from the nodes on the network, as well as options for interaction with the nodes. Additional features such as viewing data in a plot, signal processing and such would be a neat feature. Since the data received contains a common time-reference, it would be possible to synchronize data and display them side-by-side. Further on the software should include a database-connection for storing the sampled analog data. Every received data packet contains a lot of samples and therefore need to be stored in an appropriate manner.

# Chapter 4

## Discussion & Conclusions

This report describes measures taken and methods to fully specify and develop a distributed measurement system. Due to the limited time, the project was itself limited and therefore a product has not been produced, but rather a prototype. It acts as a proof-of-concept of a distributed measurement system for Ilab3. Starting out with nothing but prior academic experience, this project explored multiple possible solutions and techniques, as well as implementing the solution that was evaluated as superior.

Every decision is thoroughly motivated, if not explicitly, implicitly. Furthermore, extensive documentation is available and reading suggestions to help the reader understand decisions taken.

Due to the FPGA, exact timing analysis and signal handling is possible, therefore real-time demands are met.

### 4.1 Improvements

There are multiple additions that would have been possible to implement already from the beginning, or be added in the future. In this section an effort is made to identify as many additions as possible. Some more realistic than other.

One of these said improvements is to allow communication via TCP/IP between nodes. Nodes could share a timebase as previously mentioned, but also propa-

gate communication speed, optimal sampling points for the logic conversion and similar. If the system were to implement sequence detection, special triggering features and logic analysis, this could also be propagated between nodes. This would require modification of both the custom Hardware Description Language (HDL) controller and the software running on the processor, but it is feasible since the AXI allows two-way communication. Regarding the shared time base, there are multiple protocols that allow this, some of which allow sub- $\mu$ S precision.[25]

If this were to be implemented, following up with improved capabilities of the software running on the desktop client is appropriate. An addition of signal processing as FFT, filtering and such as well as statistical analysis could enable discovering of previously unknown characteristics. Analyzing rise time and fall time could show previously unknown properties of specific ECUs.

Adding tools for mapping between sampled logical data and what the payload translates to would be of use. Since this could indicate and eventually prove that when the system is in a specific state, the electric performance of the bus is affected. There are multiple vendors that provide most of the functionality mentioned above. However, since there are multiple HIL-labs at Scania including the flagship Ilab3, by developing their own custom solution they might get exactly all the functions they want. This could include translation to specific Diagnostic Trouble Code (DTC)s, why a message is sent and so on.

Improvements of the hardware would enable higher resolution and precision for sampling and therefore conversion to logical data, but would increase the bus and disk load significantly. This could be counter-measured with enabling compression of data. Another interesting improvement would be to, in addition of the analog filtering, add digital filtering. By changing to a fast, high-resolution ADC and transforming the data to the frequency-plane, it would be possible to completely remove unwanted frequency components. This would result in clean sampled data, however it would eliminate the possibility to study noise on the bus. With the help of an FFT-core and an IFFT-core placed on the FPGA, it would be done in a controllable, high-performance manner.

## **4.2 Benefits for the group**

The system implemented will give the group the possibility to store raw electrical data from the CAN bus for offline analysis. This in turn will increase the possibility to rule out electrical interference when the lab is not working to a satisfactory extent. With multiple nodes connected to a central server, it is possible to study how

the signal is changed when it propagates on the CAN bus, measure the propagation time of the signal among other properties of electrical signals

### **4.3 Limitations of implementation**

The CAN receiver only handles the basic CAN protocol, with three data-rates 500 kbit/s, 250 kbit/s and 125 kbit/s. CAN FD is not supported. This is because CAN FD operates at a higher data rate and would therefore need even faster ADCs and filters with a different bandwidth. It would further on require larger SR, RAM and so on.

The system is meant to be used in Scania's largest HIL-lab, but could be in any lab-environment that uses CAN. Since the setup is rather bulky and non-mobile, it is not recommended to use it in a vehicle. However future versions of the system could be used with success.

### **4.4 Alternative solutions**

There are multiple other possible solutions that could have been used and implemented instead. Both National Instruments and Mathworks have large support for interaction between a computer based model and specialized hardware to be used with the respective software. Building a model-based solution in Simulink or LabView is a realistic alternative solution, especially since the group uses Matlab/Simulink and have a lot of experience on that subject. Further on, this could enable easy deployment of multiple nodes if using a multi-purpose board like compactRIO.

The negative side is that there are few Data Acquisition (DAQ)-cards that are fast enough and up to specification, making the implementation hard. Since the idea is to place multiple units in the lab, it has to be a standalone application and therefore normal USB-DAQs are not good enough as they require a host computer. Building a data acquisition system with components from eg. NI would be expensive and might not be possible due to budget constraints/limitations. Disregarding the budget constraint, it would be possible.

One alternative solution that is realistic to implement and probably more convenient than the one chosen, would be to use the Universal Serial Bus (USB) oscilloscope used as a tool during the thesis, together with custom software running



on a laptop. This solution would minimize the parts needed in the system, and no need of custom PCBs manufactured for a complete system. Further on, the software could be written in a language that the group have vast knowledge of, eg. C++, removing the risk of essential competence disappearing when the workforce changes. This setup would be very mobile and easy to use. The idea of using the system as distributed nodes would not be a problem either, since there are servers placed in the lab running virtual machines, and more nodes could be distributed with either laptops or small computers, as the raspberry Pi.[26]

This leads us to the chosen solution. A multi-purpose stand-alone card (FPGA/SoC evaluation board) with analog-to-digital converters connected via General Purpose IO (GPIO) and a custom built filter-solution. The benefits of this is a versatile system that can be reconfigured and edited into what ever might be the need. This is however a prototype and thus connections between the PCB and SoC evaluation board are not very rugged.

## 4.5 Conclusions

This thesis shows that it is absolutely feasible to develop a system of this sort with the chosen method. However, to be able to use this to a satisfactory extent more work is needed. Developing a custom PCB containing the SoC used in this thesis, ADCs, preferably with at least a factor 10 higher sample rate, and the analog filter proposed in this report as well.

Furthermore the development of a custom PCB would make it possible to use the system in more harsh environments such as on vehicles in use, or vehicles running experimental software. Even if RESI's labs attempt to mimic a working vehicle as much as possible, there are multiple differences, especially how wiring is done in the lab in comparison with a vehicle.

# Appendix A

## Attachments

### A.1 Requirements specification

1. Overall system requirements
  - 1.1. The system should run continuously for two weeks without reboot
  - 1.2. Electrical connections should be rugged
2. Analog input stage
  - 2.1. LP-filter,  $f_c$  at 5 MHz
  - 2.2. No load on CAN bus
  - 2.3. No custom termination
  - 2.4. High impedance input stage
  - 2.5. Use LT1568 as filter
  - 2.6. Decoupling capacitors at V+ and GND, 100  $\mu$ F
  - 2.7.  $v_h$  and  $v_l$  should be set by voltage division
  - 2.8. Overflow indicator
3. HW controller
  - 3.1. Development should be done in VHDL

- 3.2. 10 MS/s sampling rate
- 3.3. Detect overflow
- 3.4. 14 bit wide ADC
- 3.5. Parallel data in
- 3.6. Parallel data out
- 3.7. Perform conversion to logical data
- 3.8. Store electrical data
- 3.9. Store logical data
- 3.10. Store timestamp
- 3.11. Store  $CAN_H$  and  $CAN_L$
- 3.12. Data should be available for the CPU when there is a valid CAN message stored
- 3.13. Communicate over AXI bus directly to CPU's RAM
4. CPU
  - 4.1. Development should be done in C
  - 4.2. Do CRC check on incoming data
  - 4.3. Do Error check on incoming data
  - 4.4. TCP/IP for communication to server
  - 4.5. Reset should be broadcast on system bus
  - 4.6. Receive reset over TCP
5. Evaluation board
  - 5.1. SoC solution or large FPGA
  - 5.2. Gigabit Ethernet
  - 5.3. NIC soldered to the board
  - 5.4. The unit should handle 3 HW-units simultaneously
  - 5.5. Minimum 128 MB AXI-accessible RAM
  - 5.6. 91 GPIO available ( $3*HW\text{-unit req, } (28+2)*3 + 1$ )

## 6. Server

- 6.1. Development should be done in Python
- 6.2. Send reset via TCP/IP to SoC
- 6.3. Connect multiple SoCs
- 6.4. Reset should be broadcasted over IP

Resource	Utilization	Available	Utilization %
FF	659	106400	1
LUT	486	53200	1
Memory LUT	84	17400	1
IO	145	200	73
BUFG	1	32	3

Table A.1: Utilization report for hardware controller, post implementation

Resource	Utilization	Available	Utilization %
FF	632	106400	1
LUT	621	53200	1
Memory LUT	84	17400	1
IO	31	200	16
BUFG	1	32	3
MMCM	1	4	25

Table A.2: Utilization report for system, post implementation

## A.2 Utilization log

There are two utilization logs attached. The first log, A.1 is for the hardware controller itself and AXI Master interface. The second, A.2, is of the complete system with one node inferred. The reason for having two logs is that if one wants to estimate the needed space on an FPGA for multiple nodes a log reporting only controller and CPU would not be sufficient. Since the system currently only consists of one hardware controller and a protocol converter, the footprint is very small. This was however expected since the beginning and the real constraint is IO. It is important to note that MMCM and BUFG resources will not scale but rather be static since they are used for clocking.

## A.3 Bill of Materials

The materials listed in A.3 are used in this thesis. Due to export restrictions the ADCs were never received. Not listed are prototyping materials such as jumper cables, soldering materials and bread boards.

Product	Function	Units
Zedboard Xilinx Zynq-7000	Development platform	1
LT1568	Analog Filter	1
Experiment Board SSOP-16	Experiment board	1
LTC2245	ADC evaluation board	2
10 $\Omega$ High-Precision (0.1%)	Resistor	6
243 $\Omega$ High-Precision (0.1%)	Resistor	6
0.1 $\mu$ F	Capacitor	2

Table A.3: Bill of Materials

## A.4 Datasheets

SoC – [http://www.xilinx.com/support/documentation/data\\_sheets/ds190-Zynq-7000-Overview.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf)

SoC Evaluation Board – [http://zedboard.org/sites/default/files/documentations/ZedBoard\\_HW\\_UG\\_v2\\_2.pdf](http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf)

RAM on SoC Evaluation Board – [http://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/2gb\\_1\\_35v\\_ddr3l.pdf](http://www.micron.com/~/media/documents/products/data-sheet/dram/ddr3/2gb_1_35v_ddr3l.pdf)

Experiment Board SSOP [https://www1.elfa.se/data1/wwwroot/assets/datasheets/re931-02\\_eng\\_bro.pdf](https://www1.elfa.se/data1/wwwroot/assets/datasheets/re931-02_eng_bro.pdf)

LT1568 Filter – <http://cds.linear.com/docs/en/datasheet/1568f.pdf>

ADC – <http://cds.linear.com/docs/en/datasheet/2245fa.pdf>

## A.5 Attached Code

### A.5.1 Vivado

Listing A.1: Design wrapper

```

1 *timescale 1 ps / 1 ps
2
3 module design_1_wrapper
4   (ARESETN,
5    AXI_CLK,
```

```

6   DDR_addr,
7   DDR_ba,
8   DDR_cas_n,
9   DDR_ck_n,
10  DDR_ck_p,
11  DDR_cke,
12  DDR_cs_n,
13  DDR_dm,
14  DDR_dq,
15  DDR_dqs_n,
16  DDR_dqs_p,
17  DDR_odt,
18  DDR_ras_n,
19  DDR_reset_n,
20  DDR_we_n,
21  FIXED_IO_ddr_vrn,
22  FIXED_IO_ddr_vrp,
23  FIXED_IO_mio,
24  FIXED_IO_ps_clk,
25  FIXED_IO_ps_porb,
26  FIXED_IO_ps_srstb,
27  i_data1,
28  i_data2,
29  i_of1,
30  i_of2,
31  o_clk_sample);
32  output [0:0]ARESETN;
33  output AXI_CLK;
34  inout [14:0]DDR_addr;
35  inout [2:0]DDR_ba;
36  inout DDR_cas_n;
37  inout DDR_ck_n;
38  inout DDR_ck_p;
39  inout DDR_cke;
40  inout DDR_cs_n;
41  inout [3:0]DDR_dm;
42  inout [31:0]DDR_dq;
43  inout [3:0]DDR_dqs_n;
44  inout [3:0]DDR_dqs_p;
45  inout DDR_odt;
46  inout DDR_ras_n;
47  inout DDR_reset_n;
48  inout DDR_we_n;
49  inout FIXED_IO_ddr_vrn;
50  inout FIXED_IO_ddr_vrp;
51  inout [53:0]FIXED_IO_mio;
52  inout FIXED_IO_ps_clk;
53  inout FIXED_IO_ps_porb;
54  inout FIXED_IO_ps_srstb;
55  input [13:0]i_data1;
56  input [13:0]i_data2;
57  input i_of1;
58  input i_of2;
59  output o_clk_sample;
60
61  wire [0:0]ARESETN;
62  wire AXI_CLK;
63  wire [14:0]DDR_addr;
64  wire [2:0]DDR_ba;
65  wire DDR_cas_n;
66  wire DDR_ck_n;
67  wire DDR_ck_p;
68  wire DDR_cke;
69  wire DDR_cs_n;
70  wire [3:0]DDR_dm;
71  wire [31:0]DDR_dq;
72  wire [3:0]DDR_dqs_n;
73  wire [3:0]DDR_dqs_p;
74  wire DDR_odt;
75  wire DDR_ras_n;
76  wire DDR_reset_n;
77  wire DDR_we_n;
78  wire FIXED_IO_ddr_vrn;
79  wire FIXED_IO_ddr_vrp;
80  wire [53:0]FIXED_IO_mio;
81  wire FIXED_IO_ps_clk;
82  wire FIXED_IO_ps_porb;
83  wire FIXED_IO_ps_srstb;
84  wire [13:0]i_data1;

```

```

85  wire [13:0]i_data2;
86  wire i_of1;
87  wire i_of2;
88  wire o_clk_sample;
89
90  design_1 design_1_i
91    (.ARESETN(ARESETN),
92     .AXI_CLK(AXI_CLK),
93     .DDR_addr(DDR_addr),
94     .DDR_ba(DDR_ba),
95     .DDR_cas_n(DDR_cas_n),
96     .DDR_ck_n(DDR_ck_n),
97     .DDR_ck_p(DDR_ck_p),
98     .DDR_cke(DDR_cke),
99     .DDR_cs_n(DDR_cs_n),
100    .DDR_dm(DDR_dm),
101    .DDR_dq(DDR_dq),
102    .DDR_dqs_n(DDR_dqs_n),
103    .DDR_dqs_p(DDR_dqs_p),
104    .DDR_odt(DDR_odt),
105    .DDR_ras_n(DDR_ras_n),
106    .DDR_reset_n(DDR_reset_n),
107    .DDR_we_n(DDR_we_n),
108    .FIXED_IO_ddr_vrn(FIXED_IO_ddr_vrn),
109    .FIXED_IO_ddr_vrp(FIXED_IO_ddr_vrp),
110    .FIXED_IO_mio(FIXED_IO_mio),
111    .FIXED_IO_ps_clk(FIXED_IO_ps_clk),
112    .FIXED_IO_ps_porb(FIXED_IO_ps_porb),
113    .FIXED_IO_ps_srstb(FIXED_IO_ps_srstb),
114    .i_data1(i_data1),
115    .i_data2(i_data2),
116    .i_of1(i_of1),
117    .i_of2(i_of2),
118    .o_clk_sample(o_clk_sample));
119  endmodule

```



## Listing A.2: Design constraints

```
1 set_property CFGBVS VCCO [current_design]
2 set_property CONFIG_VOLTAGE 2.5 [current_design]
3
4 set_property PACKAGE_PIN Y11 [get_ports {i_data1[13]}]
5 set_property PACKAGE_PIN AA11 [get_ports {i_data1[12]}]
6 set_property PACKAGE_PIN AA19 [get_ports o_clk_sample]
7
8 set_property PACKAGE_PIN AA9 [get_ports {i_data1[11]}]
9 set_property PACKAGE_PIN AB11 [get_ports {i_data1[10]}]
10 set_property PACKAGE_PIN AB10 [get_ports {i_data1[9]}]
11 set_property PACKAGE_PIN AB9 [get_ports {i_data1[8]}]
12 set_property PACKAGE_PIN AA8 [get_ports {i_data1[7]}]
13 set_property PACKAGE_PIN W12 [get_ports {i_data1[6]}]
14 set_property PACKAGE_PIN W11 [get_ports {i_data1[5]}]
15 set_property PACKAGE_PIN V10 [get_ports {i_data1[4]}]
16 set_property PACKAGE_PIN W8 [get_ports {i_data1[3]}]
17 set_property PACKAGE_PIN V12 [get_ports {i_data1[2]}]
18 set_property PACKAGE_PIN W10 [get_ports {i_data1[1]}]
19 set_property PACKAGE_PIN V9 [get_ports {i_data1[0]}]
20 set_property PACKAGE_PIN V8 [get_ports i_of1]
21 set_property PACKAGE_PIN AB6 [get_ports {i_data2[13]}]
22 set_property PACKAGE_PIN AB7 [get_ports {i_data2[12]}]
23 set_property PACKAGE_PIN AA4 [get_ports {i_data2[11]}]
24 set_property PACKAGE_PIN Y4 [get_ports {i_data2[10]}]
25 set_property PACKAGE_PIN T6 [get_ports {i_data2[9]}]
26 set_property PACKAGE_PIN R6 [get_ports {i_data2[8]}]
27 set_property PACKAGE_PIN U4 [get_ports {i_data2[7]}]
28 set_property PACKAGE_PIN T4 [get_ports {i_data2[6]}]
29 set_property PACKAGE_PIN W7 [get_ports {i_data2[5]}]
30 set_property PACKAGE_PIN V7 [get_ports {i_data2[4]}]
31 set_property PACKAGE_PIN V4 [get_ports {i_data2[3]}]
32 set_property PACKAGE_PIN V5 [get_ports {i_data2[2]}]
33 set_property PACKAGE_PIN W5 [get_ports {i_data2[1]}]
34 set_property PACKAGE_PIN W6 [get_ports {i_data2[0]}]
35 set_property PACKAGE_PIN U5 [get_ports i_of2]
36
37 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[13]}]
38 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[12]}]
39 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[11]}]
40 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[10]}]
41 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[9]}]
42 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[8]}]
43 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[7]}]
44 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[6]}]
45 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[5]}]
46 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[4]}]
47 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[3]}]
48 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[2]}]
49 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[1]}]
50 set_property IOSTANDARD LVCMOS25 [get_ports {i_data1[0]}]
51 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[13]}]
52 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[12]}]
53 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[11]}]
54 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[10]}]
55 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[9]}]
56 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[8]}]
57 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[7]}]
58 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[6]}]
59 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[5]}]
60 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[4]}]
61 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[3]}]
62 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[2]}]
63 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[1]}]
64 set_property IOSTANDARD LVCMOS25 [get_ports {i_data2[0]}]
65 set_property IOSTANDARD LVCMOS25 [get_ports i_of1]
66 set_property IOSTANDARD LVCMOS25 [get_ports i_of2]
67 set_property IOSTANDARD LVCMOS33 [get_ports o_clk_sample]
68
69 set_property IOSTANDARD LVCMOS33 [get_ports {ARESETN[0]}]
70 set_property PACKAGE_PIN AB19 [get_ports {ARESETN[0]}]
71 set_property IOSTANDARD LVCMOS33 [get_ports AX1_CLK]
72 set_property PACKAGE_PIN Y19 [get_ports AX1_CLK]
73
74 set_false_path -from [get_clocks clk_out1_design_1_clk_wiz_0_0] -to [get_clocks clk_fpga_0]
75 set_property LOC BSCAN_X0Y0 [get_cells dbg_hub/inst/bscan_inst/SERIES7_BSCAN.bscan_inst]
76
77 #####
```

```

78 ##
79 ## Xilinx, Inc. 2006      www.xilinx.com
80 #####
81 ## File name :      ps7_constraints.xdc
82 ##
83 ## Details :      Constraints file
84 ##             FPGA family:      zynq
85 ##             FPGA:             xc7z020c1g484-1
86 ##             Device Size:      xc7z020
87 ##             Package:          c1g484
88 ##             Speedgrade:      -1
89 ##
90 ##
91 #####
92 #####
93 #####
94 # Clock constraints      #
95 #####
96 create_clock -name clk_fpga_0 -period "10" [get_pins "PS7_i/FCLKCLK[0]"]
97 set_input_jitter clk_fpga_0 0.3
98 #The clocks are asynchronous, user should constrain them appropriately.#
99
100
101 #####
102 # I/O STANDARDS and Location Constraints      #
103 #####
104
105 # UART 1 / rx / MIO[49]
106 set_property iostandard "LVCMOS18" [get_ports "MIO[49]"]
107 set_property PACKAGE_PIN "C14" [get_ports "MIO[49]"]
108 set_property slew "slow" [get_ports "MIO[49]"]
109 set_property drive "8" [get_ports "MIO[49]"]
110 set_property PIO_DIRECTION "INPUT" [get_ports "MIO[49]"]
111 # UART 1 / tx / MIO[48]
112 set_property iostandard "LVCMOS18" [get_ports "MIO[48]"]
113 set_property PACKAGE_PIN "D11" [get_ports "MIO[48]"]
114 set_property slew "slow" [get_ports "MIO[48]"]
115 set_property drive "8" [get_ports "MIO[48]"]
116 set_property PIO_DIRECTION "OUTPUT" [get_ports "MIO[48]"]
117 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_VRP"]
118 set_property PACKAGE_PIN "N7" [get_ports "DDR_VRP"]
119 set_property slew "FAST" [get_ports "DDR_VRP"]
120 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_VRP"]
121 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_VRN"]
122 set_property PACKAGE_PIN "M7" [get_ports "DDR_VRN"]
123 set_property slew "FAST" [get_ports "DDR_VRN"]
124 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_VRN"]
125 set_property iostandard "SSTL15" [get_ports "DDR_WEB"]
126 set_property PACKAGE_PIN "R4" [get_ports "DDR_WEB"]
127 set_property slew "SLOW" [get_ports "DDR_WEB"]
128 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_WEB"]
129 set_property iostandard "SSTL15" [get_ports "DDR_RAS_n"]
130 set_property PACKAGE_PIN "R5" [get_ports "DDR_RAS_n"]
131 set_property slew "SLOW" [get_ports "DDR_RAS_n"]
132 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_RAS_n"]
133 set_property iostandard "SSTL15" [get_ports "DDR_ODT"]
134 set_property PACKAGE_PIN "P5" [get_ports "DDR_ODT"]
135 set_property slew "SLOW" [get_ports "DDR_ODT"]
136 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_ODT"]
137 set_property iostandard "SSTL15" [get_ports "DDR_DRSTB"]
138 set_property PACKAGE_PIN "F3" [get_ports "DDR_DRSTB"]
139 set_property slew "FAST" [get_ports "DDR_DRSTB"]
140 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DRSTB"]
141 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS[3]"]
142 set_property PACKAGE_PIN "V2" [get_ports "DDR_DQS[3]"]
143 set_property slew "FAST" [get_ports "DDR_DQS[3]"]
144 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS[3]"]
145 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS[2]"]
146 set_property PACKAGE_PIN "N2" [get_ports "DDR_DQS[2]"]
147 set_property slew "FAST" [get_ports "DDR_DQS[2]"]
148 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS[2]"]
149 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS[1]"]
150 set_property PACKAGE_PIN "H2" [get_ports "DDR_DQS[1]"]
151 set_property slew "FAST" [get_ports "DDR_DQS[1]"]
152 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS[1]"]
153 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS[0]"]
154 set_property PACKAGE_PIN "C2" [get_ports "DDR_DQS[0]"]
155 set_property slew "FAST" [get_ports "DDR_DQS[0]"]
156 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS[0]"]

```

```

157 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS_n[3]"]
158 set_property PACKAGE_PIN "W2" [get_ports "DDR_DQS_n[3]"]
159 set_property slew "FAST" [get_ports "DDR_DQS_n[3]"]
160 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS_n[3]"]
161 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS_n[2]"]
162 set_property PACKAGE_PIN "P2" [get_ports "DDR_DQS_n[2]"]
163 set_property slew "FAST" [get_ports "DDR_DQS_n[2]"]
164 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS_n[2]"]
165 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS_n[1]"]
166 set_property PACKAGE_PIN "J2" [get_ports "DDR_DQS_n[1]"]
167 set_property slew "FAST" [get_ports "DDR_DQS_n[1]"]
168 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS_n[1]"]
169 set_property iostandard "DIFF_SSTL15_T_DCI" [get_ports "DDR_DQS_n[0]"]
170 set_property PACKAGE_PIN "D2" [get_ports "DDR_DQS_n[0]"]
171 set_property slew "FAST" [get_ports "DDR_DQS_n[0]"]
172 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQS_n[0]"]
173 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[9]"]
174 set_property PACKAGE_PIN "G1" [get_ports "DDR_DQ[9]"]
175 set_property slew "FAST" [get_ports "DDR_DQ[9]"]
176 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[9]"]
177 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[8]"]
178 set_property PACKAGE_PIN "G2" [get_ports "DDR_DQ[8]"]
179 set_property slew "FAST" [get_ports "DDR_DQ[8]"]
180 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[8]"]
181 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[7]"]
182 set_property PACKAGE_PIN "F1" [get_ports "DDR_DQ[7]"]
183 set_property slew "FAST" [get_ports "DDR_DQ[7]"]
184 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[7]"]
185 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[6]"]
186 set_property PACKAGE_PIN "F2" [get_ports "DDR_DQ[6]"]
187 set_property slew "FAST" [get_ports "DDR_DQ[6]"]
188 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[6]"]
189 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[5]"]
190 set_property PACKAGE_PIN "E1" [get_ports "DDR_DQ[5]"]
191 set_property slew "FAST" [get_ports "DDR_DQ[5]"]
192 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[5]"]
193 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[4]"]
194 set_property PACKAGE_PIN "E3" [get_ports "DDR_DQ[4]"]
195 set_property slew "FAST" [get_ports "DDR_DQ[4]"]
196 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[4]"]
197 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[3]"]
198 set_property PACKAGE_PIN "D3" [get_ports "DDR_DQ[3]"]
199 set_property slew "FAST" [get_ports "DDR_DQ[3]"]
200 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[3]"]
201 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[31]"]
202 set_property PACKAGE_PIN "Y1" [get_ports "DDR_DQ[31]"]
203 set_property slew "FAST" [get_ports "DDR_DQ[31]"]
204 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[31]"]
205 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[30]"]
206 set_property PACKAGE_PIN "W3" [get_ports "DDR_DQ[30]"]
207 set_property slew "FAST" [get_ports "DDR_DQ[30]"]
208 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[30]"]
209 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[2]"]
210 set_property PACKAGE_PIN "B2" [get_ports "DDR_DQ[2]"]
211 set_property slew "FAST" [get_ports "DDR_DQ[2]"]
212 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[2]"]
213 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[29]"]
214 set_property PACKAGE_PIN "Y3" [get_ports "DDR_DQ[29]"]
215 set_property slew "FAST" [get_ports "DDR_DQ[29]"]
216 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[29]"]
217 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[28]"]
218 set_property PACKAGE_PIN "W1" [get_ports "DDR_DQ[28]"]
219 set_property slew "FAST" [get_ports "DDR_DQ[28]"]
220 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[28]"]
221 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[27]"]
222 set_property PACKAGE_PIN "U2" [get_ports "DDR_DQ[27]"]
223 set_property slew "FAST" [get_ports "DDR_DQ[27]"]
224 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[27]"]
225 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[26]"]
226 set_property PACKAGE_PIN "AA1" [get_ports "DDR_DQ[26]"]
227 set_property slew "FAST" [get_ports "DDR_DQ[26]"]
228 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[26]"]
229 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[25]"]
230 set_property PACKAGE_PIN "U1" [get_ports "DDR_DQ[25]"]
231 set_property slew "FAST" [get_ports "DDR_DQ[25]"]
232 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[25]"]
233 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[24]"]
234 set_property PACKAGE_PIN "AA3" [get_ports "DDR_DQ[24]"]
235 set_property slew "FAST" [get_ports "DDR_DQ[24]"]

```

```

236 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[24]"]
237 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[23]"]
238 set_property PACKAGE_PIN "R1" [get_ports "DDR_DQ[23]"]
239 set_property slew "FAST" [get_ports "DDR_DQ[23]"]
240 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[23]"]
241 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[22]"]
242 set_property PACKAGE_PIN "M2" [get_ports "DDR_DQ[22]"]
243 set_property slew "FAST" [get_ports "DDR_DQ[22]"]
244 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[22]"]
245 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[21]"]
246 set_property PACKAGE_PIN "T2" [get_ports "DDR_DQ[21]"]
247 set_property slew "FAST" [get_ports "DDR_DQ[21]"]
248 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[21]"]
249 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[20]"]
250 set_property PACKAGE_PIN "R3" [get_ports "DDR_DQ[20]"]
251 set_property slew "FAST" [get_ports "DDR_DQ[20]"]
252 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[20]"]
253 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[1]"]
254 set_property PACKAGE_PIN "C3" [get_ports "DDR_DQ[1]"]
255 set_property slew "FAST" [get_ports "DDR_DQ[1]"]
256 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[1]"]
257 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[19]"]
258 set_property PACKAGE_PIN "T1" [get_ports "DDR_DQ[19]"]
259 set_property slew "FAST" [get_ports "DDR_DQ[19]"]
260 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[19]"]
261 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[18]"]
262 set_property PACKAGE_PIN "N3" [get_ports "DDR_DQ[18]"]
263 set_property slew "FAST" [get_ports "DDR_DQ[18]"]
264 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[18]"]
265 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[17]"]
266 set_property PACKAGE_PIN "T3" [get_ports "DDR_DQ[17]"]
267 set_property slew "FAST" [get_ports "DDR_DQ[17]"]
268 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[17]"]
269 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[16]"]
270 set_property PACKAGE_PIN "M1" [get_ports "DDR_DQ[16]"]
271 set_property slew "FAST" [get_ports "DDR_DQ[16]"]
272 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[16]"]
273 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[15]"]
274 set_property PACKAGE_PIN "K3" [get_ports "DDR_DQ[15]"]
275 set_property slew "FAST" [get_ports "DDR_DQ[15]"]
276 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[15]"]
277 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[14]"]
278 set_property PACKAGE_PIN "J1" [get_ports "DDR_DQ[14]"]
279 set_property slew "FAST" [get_ports "DDR_DQ[14]"]
280 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[14]"]
281 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[13]"]
282 set_property PACKAGE_PIN "K1" [get_ports "DDR_DQ[13]"]
283 set_property slew "FAST" [get_ports "DDR_DQ[13]"]
284 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[13]"]
285 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[12]"]
286 set_property PACKAGE_PIN "L3" [get_ports "DDR_DQ[12]"]
287 set_property slew "FAST" [get_ports "DDR_DQ[12]"]
288 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[12]"]
289 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[11]"]
290 set_property PACKAGE_PIN "L2" [get_ports "DDR_DQ[11]"]
291 set_property slew "FAST" [get_ports "DDR_DQ[11]"]
292 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[11]"]
293 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[10]"]
294 set_property PACKAGE_PIN "L1" [get_ports "DDR_DQ[10]"]
295 set_property slew "FAST" [get_ports "DDR_DQ[10]"]
296 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[10]"]
297 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DQ[0]"]
298 set_property PACKAGE_PIN "D1" [get_ports "DDR_DQ[0]"]
299 set_property slew "FAST" [get_ports "DDR_DQ[0]"]
300 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DQ[0]"]
301 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DM[3]"]
302 set_property PACKAGE_PIN "AA2" [get_ports "DDR_DM[3]"]
303 set_property slew "FAST" [get_ports "DDR_DM[3]"]
304 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DM[3]"]
305 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DM[2]"]
306 set_property PACKAGE_PIN "P1" [get_ports "DDR_DM[2]"]
307 set_property slew "FAST" [get_ports "DDR_DM[2]"]
308 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DM[2]"]
309 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DM[1]"]
310 set_property PACKAGE_PIN "H3" [get_ports "DDR_DM[1]"]
311 set_property slew "FAST" [get_ports "DDR_DM[1]"]
312 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DM[1]"]
313 set_property iostandard "SSTL15_T_DCI" [get_ports "DDR_DM[0]"]
314 set_property PACKAGE_PIN "B1" [get_ports "DDR_DM[0]"]

```

```

315 set_property slew "FAST" [get_ports "DDR_DM[0]"]
316 set_property PIO_DIRECTION "BIDIR" [get_ports "DDR_DM[0]"]
317 set_property iostandard "SSTL15" [get_ports "DDR_CS_n"]
318 set_property PACKAGE_PIN "P6" [get_ports "DDR_CS_n"]
319 set_property slew "SLOW" [get_ports "DDR_CS_n"]
320 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_CS_n"]
321 set_property iostandard "SSTL15" [get_ports "DDR_CKE"]
322 set_property PACKAGE_PIN "V3" [get_ports "DDR_CKE"]
323 set_property slew "SLOW" [get_ports "DDR_CKE"]
324 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_CKE"]
325 set_property iostandard "DIFF_SSTL15" [get_ports "DDR_Clk"]
326 set_property PACKAGE_PIN "N4" [get_ports "DDR_Clk"]
327 set_property slew "FAST" [get_ports "DDR_Clk"]
328 set_property PIO_DIRECTION "INPUT" [get_ports "DDR_Clk"]
329 set_property iostandard "DIFF_SSTL15" [get_ports "DDR_Clk_n"]
330 set_property PACKAGE_PIN "N5" [get_ports "DDR_Clk_n"]
331 set_property slew "FAST" [get_ports "DDR_Clk_n"]
332 set_property PIO_DIRECTION "INPUT" [get_ports "DDR_Clk_n"]
333 set_property iostandard "SSTL15" [get_ports "DDR_CAS_n"]
334 set_property PACKAGE_PIN "P3" [get_ports "DDR_CAS_n"]
335 set_property slew "SLOW" [get_ports "DDR_CAS_n"]
336 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_CAS_n"]
337 set_property iostandard "SSTL15" [get_ports "DDR_BankAddr[2]"]
338 set_property PACKAGE_PIN "M6" [get_ports "DDR_BankAddr[2]"]
339 set_property slew "SLOW" [get_ports "DDR_BankAddr[2]"]
340 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_BankAddr[2]"]
341 set_property iostandard "SSTL15" [get_ports "DDR_BankAddr[1]"]
342 set_property PACKAGE_PIN "L6" [get_ports "DDR_BankAddr[1]"]
343 set_property slew "SLOW" [get_ports "DDR_BankAddr[1]"]
344 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_BankAddr[1]"]
345 set_property iostandard "SSTL15" [get_ports "DDR_BankAddr[0]"]
346 set_property PACKAGE_PIN "L7" [get_ports "DDR_BankAddr[0]"]
347 set_property slew "SLOW" [get_ports "DDR_BankAddr[0]"]
348 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_BankAddr[0]"]
349 set_property iostandard "SSTL15" [get_ports "DDR_Addr[9]"]
350 set_property PACKAGE_PIN "H5" [get_ports "DDR_Addr[9]"]
351 set_property slew "SLOW" [get_ports "DDR_Addr[9]"]
352 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[9]"]
353 set_property iostandard "SSTL15" [get_ports "DDR_Addr[8]"]
354 set_property PACKAGE_PIN "J5" [get_ports "DDR_Addr[8]"]
355 set_property slew "SLOW" [get_ports "DDR_Addr[8]"]
356 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[8]"]
357 set_property iostandard "SSTL15" [get_ports "DDR_Addr[7]"]
358 set_property PACKAGE_PIN "J6" [get_ports "DDR_Addr[7]"]
359 set_property slew "SLOW" [get_ports "DDR_Addr[7]"]
360 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[7]"]
361 set_property iostandard "SSTL15" [get_ports "DDR_Addr[6]"]
362 set_property PACKAGE_PIN "J7" [get_ports "DDR_Addr[6]"]
363 set_property slew "SLOW" [get_ports "DDR_Addr[6]"]
364 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[6]"]
365 set_property iostandard "SSTL15" [get_ports "DDR_Addr[5]"]
366 set_property PACKAGE_PIN "K5" [get_ports "DDR_Addr[5]"]
367 set_property slew "SLOW" [get_ports "DDR_Addr[5]"]
368 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[5]"]
369 set_property iostandard "SSTL15" [get_ports "DDR_Addr[4]"]
370 set_property PACKAGE_PIN "K6" [get_ports "DDR_Addr[4]"]
371 set_property slew "SLOW" [get_ports "DDR_Addr[4]"]
372 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[4]"]
373 set_property iostandard "SSTL15" [get_ports "DDR_Addr[3]"]
374 set_property PACKAGE_PIN "L4" [get_ports "DDR_Addr[3]"]
375 set_property slew "SLOW" [get_ports "DDR_Addr[3]"]
376 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[3]"]
377 set_property iostandard "SSTL15" [get_ports "DDR_Addr[2]"]
378 set_property PACKAGE_PIN "K4" [get_ports "DDR_Addr[2]"]
379 set_property slew "SLOW" [get_ports "DDR_Addr[2]"]
380 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[2]"]
381 set_property iostandard "SSTL15" [get_ports "DDR_Addr[1]"]
382 set_property PACKAGE_PIN "M5" [get_ports "DDR_Addr[1]"]
383 set_property slew "SLOW" [get_ports "DDR_Addr[1]"]
384 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[1]"]
385 set_property iostandard "SSTL15" [get_ports "DDR_Addr[14]"]
386 set_property PACKAGE_PIN "G4" [get_ports "DDR_Addr[14]"]
387 set_property slew "SLOW" [get_ports "DDR_Addr[14]"]
388 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[14]"]
389 set_property iostandard "SSTL15" [get_ports "DDR_Addr[13]"]
390 set_property PACKAGE_PIN "F4" [get_ports "DDR_Addr[13]"]
391 set_property slew "SLOW" [get_ports "DDR_Addr[13]"]
392 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[13]"]
393 set_property iostandard "SSTL15" [get_ports "DDR_Addr[12]"]

```

```

394 set_property PACKAGE_PIN "H4" [get_ports "DDR_Addr[12]"]
395 set_property slew "SLOW" [get_ports "DDR_Addr[12]"]
396 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[12]"]
397 set_property fostandard "SSTL15" [get_ports "DDR_Addr[11]"]
398 set_property PACKAGE_PIN "G5" [get_ports "DDR_Addr[11]"]
399 set_property slew "SLOW" [get_ports "DDR_Addr[11]"]
400 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[11]"]
401 set_property fostandard "SSTL15" [get_ports "DDR_Addr[10]"]
402 set_property PACKAGE_PIN "J3" [get_ports "DDR_Addr[10]"]
403 set_property slew "SLOW" [get_ports "DDR_Addr[10]"]
404 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[10]"]
405 set_property fostandard "SSTL15" [get_ports "DDR_Addr[0]"]
406 set_property PACKAGE_PIN "M4" [get_ports "DDR_Addr[0]"]
407 set_property slew "SLOW" [get_ports "DDR_Addr[0]"]
408 set_property PIO_DIRECTION "OUTPUT" [get_ports "DDR_Addr[0]"]
409
410 # file: design_1_clk_wiz_0_0.xdc
411 #
412 # (c) Copyright 2008 – 2013 Xilinx, Inc. All rights reserved.
413 #
414 # This file contains confidential and proprietary information
415 # of Xilinx, Inc. and is protected under U.S. and
416 # international copyright and other intellectual property
417 # laws.
418 #
419 # DISCLAIMER
420 # This disclaimer is not a license and does not grant any
421 # rights to the materials distributed herewith. Except as
422 # otherwise provided in a valid license issued to you by
423 # Xilinx, and to the maximum extent permitted by applicable
424 # law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
425 # WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
426 # AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
427 # BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
428 # INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
429 # (2) Xilinx shall not be liable (whether in contract or tort,
430 # including negligence, or under any other theory of
431 # liability) for any loss or damage of any kind or nature
432 # related to, arising under or in connection with these
433 # materials, including for any direct, or any indirect,
434 # special, incidental, or consequential loss or damage
435 # (including loss of data, profits, goodwill, or any type of
436 # loss or damage suffered as a result of any action brought
437 # by a third party) even if such damage or loss was
438 # reasonably foreseeable or Xilinx had been advised of the
439 # possibility of the same.
440 #
441 # CRITICAL APPLICATIONS
442 # Xilinx products are not designed or intended to be fail-
443 # safe, or for use in any application requiring fail-safe
444 # performance, such as life-support or safety devices or
445 # systems, Class III medical devices, nuclear facilities,
446 # applications related to the deployment of airbags, or any
447 # other applications that could lead to death, personal
448 # injury, or severe property or environmental damage
449 # (individually and collectively, "Critical
450 # Applications"). Customer assumes the sole risk and
451 # liability of any use of Xilinx products in Critical
452 # Applications, subject only to applicable laws and
453 # regulations governing limitations on product liability.
454 #
455 # THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
456 # PART OF THIS FILE AT ALL TIMES.
457 #
458
459 # Input clock periods. These duplicate the values entered for the
460 # input clocks. You can use these to time your system. If required
461 # commented constraints can be used in the top level xdc
462 #
463 # Connect to input port when clock capable pin is selected for input
464 create_clock -period 10.0 [get_ports clk_in1]
465 set_input_jitter [get_clocks -of_objects [get_ports clk_in1]] 0.1

```

## A.5.2 HDL

Listing A.3: AXI Master top

```
1 library ieee;
2 library work;
3
4 use ieee.std_logic_1164.all;
5 use ieee.std_logic_arith.all;
6 use ieee.std_logic_unsigned.all;
7
8 entity CAN_hw_controller_v1_0 is
9     generic (
10         -- Users to add parameters here
11         CAN_SPEED : integer := 500;
12         CAN_THRESHOLD : integer := 2000;
13         C_M_AXI_TARGET_SLAVE_HIGH_ADDR : std_logic_vector := x"17FFFFFFF";
14
15         -- User parameters ends
16         -- Do not modify the parameters beyond this line
17
18
19         -- Parameters of Axi Master Bus Interface M00_AXI
20         C_M_AXI_TARGET_SLAVE_BASE_ADDR : std_logic_vector := x"10000000";
21         C_M_AXI_ADDR_WIDTH : integer := 32;
22         C_M_AXI_DATA_WIDTH : integer := 32
23         --C_M_AXI_TRANSACTIONS_NUM : integer := 1
24     );
25 port (
26     -- Users to add ports here
27     i_data1 : in std_logic_vector(13 downto 0);
28     i_data2 : in std_logic_vector(13 downto 0);
29     i_of1 : in std_logic;
30     i_of2 : in std_logic;
31     i_clk_sample : in std_logic;
32     -- User ports ends
33     -- Do not modify the ports beyond this line
34
35
36     -- Ports of Axi Master Bus Interface M00_AXI
37     --m00_axi_init_axi_txn : in std_logic;
38     --m00_axi_error : out std_logic;
39     --m00_axi_txn_done : out std_logic;
40     m00_axi_ackl : in std_logic;
41     m00_axi_aresetn : in std_logic;
42     m00_axi_awaddr : out std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
43     m00_axi_awprot : out std_logic_vector(2 downto 0);
44     m00_axi_awvalid : out std_logic;
45     m00_axi_awready : in std_logic;
46     m00_axi_wdata : out std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
47     m00_axi_wstrb : out std_logic_vector(C_M_AXI_DATA_WIDTH/8-1 downto 0);
48     m00_axi_wvalid : out std_logic;
49     m00_axi_wready : in std_logic;
50     m00_axi_bresp : in std_logic_vector(1 downto 0);
51     m00_axi_bvalid : in std_logic;
52     m00_axi_bready : out std_logic;
53     m00_axi_araddr : out std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
54     m00_axi_arprot : out std_logic_vector(2 downto 0);
55     m00_axi_arvalid : out std_logic;
56     m00_axi_arready : in std_logic;
57     m00_axi_rdata : in std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
58     m00_axi_rresp : in std_logic_vector(1 downto 0);
59     m00_axi_rvalid : in std_logic;
60     m00_axi_rready : out std_logic
61 );
62 end CAN_hw_controller_v1_0;
63
64 architecture arch_imp of CAN_hw_controller_v1_0 is
65
66     -- component declaration
67     component CAN_hw_controller_v1_0_M00_AXI is
68         generic (
69             C_M_TARGET_SLAVE_HIGH_ADDR : std_logic_vector := x"17FFFFFFF";
70             C_M_TARGET_SLAVE_BASE_ADDR : std_logic_vector := x"10000000";
71             C_M_AXI_ADDR_WIDTH : integer := 32;
72             C_M_AXI_DATA_WIDTH : integer := 32;
73             C_M_TRANSACTIONS_NUM : integer := 1
```

```

74     );
75     port (
76         i_data      : in std_logic_vector(31 downto 0);
77         INIT_AXI_TXN : in std_logic;
78         ERROR       : out std_logic;
79         TXN_DONE    : out std_logic;
80         M_AXI_ACLK  : in std_logic;
81         M_AXI_ARESETN : in std_logic;
82         M_AXI_AWADDR : out std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
83         M_AXI_AWPROT : out std_logic_vector(2 downto 0);
84         M_AXI_AWVALID : out std_logic;
85         M_AXI_AWREADY : in std_logic;
86         M_AXI_WDATA : out std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
87         M_AXI_WSTRB : out std_logic_vector(C_M_AXI_DATA_WIDTH/8-1 downto 0);
88         M_AXI_WVALID : out std_logic;
89         M_AXI_WREADY : in std_logic;
90         M_AXI_BRESP : in std_logic_vector(1 downto 0);
91         M_AXI_BVALID : in std_logic;
92         M_AXI_BREADY : out std_logic;
93         M_AXI_ARADDR : out std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
94         M_AXI_ARPROT : out std_logic_vector(2 downto 0);
95         M_AXI_ARVALID : out std_logic;
96         M_AXI_ARREADY : in std_logic;
97         M_AXI_RDATA : in std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
98         M_AXI_RRESP : in std_logic_vector(1 downto 0);
99         M_AXI_RVALID : in std_logic;
100        M_AXI_RREADY : out std_logic
101    );
102    end component CAN_hw_controller_v1_0_M00_AXI;
103
104    component top is
105    generic (
106        CAN_SPEED : integer := 500;
107        CAN_TRESHOLD : integer := 2000
108    );
109    port (
110        rst      : in STD_LOGIC;
111        clk      : in STD_LOGIC;
112        i_data1   : in std_logic_vector(13 downto 0);
113        i_data2   : in std_logic_vector(13 downto 0);
114        i_of1     : in std_logic;
115        i_of2     : in std_logic;
116        i_data    : in std_logic_vector(31 downto 0);
117        i_clk_sample : in std_logic;
118        o_data    : out std_logic_vector(31 downto 0);
119        o_data_valid : out std_logic
120    );
121    end component top;
122
123    signal m00_axi_error , m00_axi_txn_done : std_logic;
124    signal data_to_axi_valid , rst , last_write : std_logic;
125    signal i_data , data_to_axi : std_logic_vector(31 downto 0);
126
127    begin
128        rst <= not m00_axi_aresetn;
129        i_data <= (others => '0');
130
131
132    — Instantiation of Axi Bus Interface M00_AXI
133    CAN_hw_controller_v1_0_M00_AXI_inst : CAN_hw_controller_v1_0_M00_AXI
134    generic map (
135        C_M_TARGET_SLAVE_HIGH_ADDR => C_M_AXI_TARGET_SLAVE_HIGH_ADDR,
136        C_M_TARGET_SLAVE_BASE_ADDR => C_M_AXI_TARGET_SLAVE_BASE_ADDR,
137        C_M_AXI_ADDR_WIDTH => C_M_AXI_ADDR_WIDTH,
138        —C_M_AXI_DATA_WIDTH => C_M_AXI_DATA_WIDTH,
139        —C_M_TRANSACTIONS_NUM => C_M_AXI_TRANSACTIONS_NUM
140    )
141    port map (
142        i_data => data_to_axi ,
143        INIT_AXI_TXN => data_to_axi_valid ,
144        ERROR => m00_axi_error ,
145        TXN_DONE => m00_axi_txn_done ,
146        M_AXI_ACLK => m00_axi_aclk ,
147        M_AXI_ARESETN => m00_axi_aresetn ,
148        M_AXI_AWADDR => m00_axi_awaddr ,
149        M_AXI_AWPROT => m00_axi_awprot ,
150        M_AXI_AWVALID => m00_axi_awvalid ,
151        M_AXI_AWREADY => m00_axi_awready ,
152        M_AXI_WDATA => m00_axi_wdata ,

```



```

153     M_AXI_WSTRB => m00_axi_wstrb ,
154     M_AXI_WVALID => m00_axi_wvalid ,
155     M_AXI_WREADY => m00_axi_wready ,
156     M_AXI_BRESP => m00_axi_bresp ,
157     M_AXI_BVALID => m00_axi_bvalid ,
158     M_AXI_BREADY => m00_axi_bready ,
159     M_AXI_ARADDR => m00_axi_araddr ,
160     M_AXI_ARPROT => m00_axi_arprot ,
161     M_AXI_ARVALID => m00_axi_arvalid ,
162     M_AXI_ARREADY => m00_axi_arready ,
163     M_AXI_RDATA => m00_axi_rdata ,
164     M_AXI_RRESP => m00_axi_rresp ,
165     M_AXI_RVALID => m00_axi_rvalid ,
166     M_AXI_RREADY => m00_axi_rready
167 );
168
169 — Add user logic here
170
171 top_controller : top
172   generic map(
173     CAN_SPEED => CAN_SPEED,
174     CAN_THRESHOLD => CAN_THRESHOLD
175   )
176   port map (
177     rst => rst ,
178     clk => m00_axi_aclck ,
179     i_data1 => i_data1 ,
180     i_data2 => i_data2 ,
181     i_of1 => i_of1 ,
182     i_of2 => i_of2 ,
183     i_data => i_data ,
184     i_clk_sample => i_clk_sample ,
185     o_data => data_to_axi ,
186     o_data_valid => data_to_axi_valid
187   );
188 — User logic ends
189
190 end arch_imp;

```

## Listing A.4: AXI Master

```

1  library ieee;
2  library work;
3
4  use ieee.std_logic_1164.all;
5  use ieee.std_logic_arith.all;
6  use ieee.std_logic_unsigned.all;
7
8  entity CAN_hw_controller_v1_0_M00_AXI is
9      generic (
10         -- Users to add parameters here
11         -- Will OF to C_M_TARGET_SLAVE_BASE_ADDR
12         C_M_TARGET_SLAVE_HIGH_ADDR : std_logic_vector := x"17FFFFFF";
13         -- User parameters ends
14         -- Do not modify the parameters beyond this line
15
16         -- The master will start generating data from the C_M_START_DATA_VALUE value
17         C_M_START_DATA_VALUE : std_logic_vector := x"DEADBEEF";
18         -- The master requires a target slave base address.
19         -- The master will initiate read and write transactions on the slave with base address specified here as a
20         parameter.
21         C_M_TARGET_SLAVE_BASE_ADDR : std_logic_vector := x"10000000";
22         -- Width of M_AXI address bus.
23         -- The master generates the read and write addresses of width specified as C_M_AXI_ADDR_WIDTH.
24         C_M_AXI_ADDR_WIDTH : integer := 32;
25         -- Width of M_AXI data bus.
26         -- The master issues write data and accept read data where the width of the data bus is C_M_AXI_DATA_WIDTH
27         C_M_AXI_DATA_WIDTH : integer := 32;
28         -- Transaction number is the number of write
29         -- and read transactions the master will perform as a part of this example memory test.
30         C_M_TRANSACTIONS_NUM : integer := 1
31     );
32     port (
33         -- Users to add ports here
34         i_data : in std_logic_vector(31 downto 0);
35         -- User ports ends
36         -- Do not modify the ports beyond this line
37
38         -- Initiate AXI transactions
39         -- Initiate AXI transactions
40         INIT_AXI_TXN : in std_logic;
41         -- Asserts when ERROR is detected
42         ERROR : out std_logic;
43         -- Asserts when AXI transactions is complete
44         TXN_DONE : out std_logic;
45         -- AXI clock signal
46         M_AXI_ACLK : in std_logic;
47         -- AXI active low reset signal
48         M_AXI_ARESETN : in std_logic;
49         -- Master Interface Write Address Channel ports. Write address (issued by master)
50         M_AXI_AWADDR : out std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
51         -- Write channel Protection type.
52         -- This signal indicates the privilege and security level of the transaction,
53         -- and whether the transaction is a data access or an instruction access.
54         M_AXI_AWPROT : out std_logic_vector(2 downto 0);
55         -- Write address valid.
56         -- This signal indicates that the master signaling valid write address and control information.
57         M_AXI_AWVALID : out std_logic;
58         -- Write address ready.
59         -- This signal indicates that the slave is ready to accept an address and associated control signals.
60         M_AXI_AWREADY : in std_logic;
61         -- Master Interface Write Data Channel ports. Write data (issued by master)
62         M_AXI_WDATA : out std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
63         -- Write strobes.
64         -- This signal indicates which byte lanes hold valid data.
65         -- There is one write strobe bit for each eight bits of the write data bus.
66         M_AXI_WSTRB : out std_logic_vector(C_M_AXI_DATA_WIDTH/8-1 downto 0);
67         -- Write valid. This signal indicates that valid write data and strobes are available.
68         M_AXI_WVALID : out std_logic;
69         -- Write ready. This signal indicates that the slave can accept the write data.
70         M_AXI_WREADY : in std_logic;
71         -- Master Interface Write Response Channel ports.
72         -- This signal indicates the status of the write transaction.
73         M_AXI_BRESP : in std_logic_vector(1 downto 0);
74         -- Write response valid.
75         -- This signal indicates that the channel is signaling a valid write response
76         M_AXI_BVALID : in std_logic;
77         -- Response ready. This signal indicates that the master can accept a write response.

```

```

77     M_AXI_BREADY : out std_logic;
78     — Master Interface Read Address Channel ports. Read address (issued by master)
79     M_AXI_ARADDR : out std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
80     — Protection type.
81     — This signal indicates the privilege and security level of the transaction.
82     — and whether the transaction is a data access or an instruction access.
83     M_AXI_ARPROT : out std_logic_vector(2 downto 0);
84     — Read address valid.
85     — This signal indicates that the channel is signaling valid read address and control information.
86     M_AXI_ARVALID : out std_logic;
87     — Read address ready.
88     — This signal indicates that the slave is ready to accept an address and associated control signals.
89     M_AXI_ARREADY : in std_logic;
90     — Master Interface Read Data Channel ports. Read data (issued by slave)
91     M_AXI_RDATA : in std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
92     — Read response. This signal indicates the status of the read transfer.
93     M_AXI_RRESP : in std_logic_vector(1 downto 0);
94     — Read valid. This signal indicates that the channel is signaling the required read data.
95     M_AXI_RVALID : in std_logic;
96     — Read ready. This signal indicates that the master can accept the read data and response information.
97     M_AXI_RREADY : out std_logic
98 );
99 end CAN_hw_controller_v1_0_M00_AXI;
100
101 architecture implementation of CAN_hw_controller_v1_0_M00_AXI is
102
103 — function called clogb2 that returns an integer which has the
104 — value of the ceiling of the log base 2
105 function clogb2 (bit_depth : integer) return integer is
106     variable depth : integer := bit_depth;
107     variable count : integer := 1;
108 begin
109     for clogb2 in 1 to bit_depth loop — Works for up to 32 bit integers
110         if (bit_depth <= 2) then
111             count := 1;
112         else
113             if (depth <= 1) then
114                 count := count;
115             else
116                 depth := depth / 2;
117                 count := count + 1;
118             end if;
119         end if;
120     end loop;
121     return(count);
122 end;
123
124 — Example user application signals
125
126 — TRANS_NUM_BITS is the width of the index counter for
127 — number of write or read transaction..
128 constant TRANS_NUM_BITS : integer := clogb2(C_M_TRANSACTIONS_NUM-1);
129
130 — Example State machine to initialize counter, initialize write transactions,
131 — initialize read transactions and comparison of read data with the
132 — written data words.
133 type state is ( IDLE, — This state initiates AXI4Lite transaction
134                 — after the state machine changes state to INIT_WRITE
135                 — when there is 0 to 1 transition on INIT_AXI_TXN
136                 INIT_WRITE, — This state initializes write transaction,
137                 — once writes are done, the state machine
138                 — changes state to INIT_READ
139                 INIT_READ, — This state initializes read transaction
140                 — once reads are done, the state machine
141                 — changes state to INIT_COMPARE
142                 INIT_COMPARE); — This state issues the status of comparison
143                 — of the written data with the read data
144
145 signal mst_exec_state : state ;
146
147 — AXI4LITE signals
148 — write address valid
149 signal axi_awvalid : std_logic;
150 — write data valid
151 signal axi_wvalid : std_logic;
152 — read address valid
153 signal axi_arvalid : std_logic;
154 — read data acceptance
155 signal axi_rready : std_logic;

```

```

156  —write response acceptance
157  signal axi_bready : std_logic;
158  —write address
159  signal axi_awaddr : std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
160  —write data
161  signal axi_wdata : std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
162  —read address
163  signal axi_araddr : std_logic_vector(C_M_AXI_ADDR_WIDTH-1 downto 0);
164  —Asserts when there is a write response error
165  signal write_resp_error : std_logic;
166  —Asserts when there is a read response error
167  signal read_resp_error : std_logic;
168  —A pulse to initiate a write transaction
169  signal start_single_write : std_logic;
170  —A pulse to initiate a read transaction
171  signal start_single_read : std_logic;
172  —Asserts when a single beat write transaction is issued and remains asserted till the completion of write
   transaction.
173  signal write_issued : std_logic;
174  —Asserts when a single beat read transaction is issued and remains asserted till the completion of read
   transaction.
175  signal read_issued : std_logic;
176  —flag that marks the completion of write trasactions. The number of write transaction is user selected by
   the parameter C_M_TRANSACTIONS_NUM.
177  signal writes_done : std_logic;
178  —flag that marks the completion of read trasactions. The number of read transaction is user selected by
   the parameter C_M_TRANSACTIONS_NUM
179  signal reads_done : std_logic;
180  —The error register is asserted when any of the write response error, read response error or the data
   mismatch flags are asserted.
181  signal error_reg : std_logic;
182  —index counter to track the number of write transaction issued
183  signal write_index : std_logic_vector(TRANS_NUM_BITS downto 0);
184  —index counter to track the number of read transaction issued
185  signal read_index : std_logic_vector(TRANS_NUM_BITS downto 0);
186  —Expected read data used to compare with the read data.
187  signal expected_rdata : std_logic_vector(C_M_AXI_DATA_WIDTH-1 downto 0);
188  —Flag marks the completion of comparison of the read data with the expected read data
189  signal compare_done : std_logic;
190  —This flag is asserted when there is a mismatch of the read data with the expected read data.
191  signal read_mismatch : std_logic;
192  —Flag is asserted when the write index reaches the last write transaction number
193  signal last_write : std_logic;
194  —Flag is asserted when the read index reaches the last read transaction number
195  signal last_read : std_logic;
196  signal init_txn_ff : std_logic;
197  signal init_txn_ff2 : std_logic;
198  signal init_txn_edge : std_logic;
199  signal init_txn_pulse : std_logic;
200
201  — Signals defined by user
202  signal data , data_n : std_logic_vector(31 downto 0); — set up incoming data in a register to hold value
203
204
205  begin
206  — I/O Connections assignments
207
208  —Adding the offset address to the base addr of the slave
209  M_AXI_AWADDR <= C_M_TARGET_SLAVE_BASE_ADDR + axi_awaddr;
210  —AXI 4 write data
211  M_AXI_WDATA <= axi_wdata;
212  M_AXI_AWPROT <= "000";
213  M_AXI_AWVALID <= axi_awvalid;
214  —Write Data(W)
215  M_AXI_WVALID <= axi_wvalid;
216  —Set all byte strobes in this example
217  M_AXI_WSTRB <= "1111";
218  —Write Response (B)
219  M_AXI_BREADY <= axi_bready;
220  —Read Address (AR)
221  M_AXI_ARADDR <= C_M_TARGET_SLAVE_BASE_ADDR + axi_araddr;
222  M_AXI_ARVALID <= axi_arvalid;
223  M_AXI_ARPROT <= "001";
224  —Read and Read Response (R)
225  M_AXI_RREADY <= axi_rready;
226  —Example design I/O
227  TXN_DONE <= compare_done;
228  init_txn_pulse <= ( not init_txn_ff2 ) and init_txn_ff;
229

```

```

230
231 —Generate a pulse to initiate AXI transaction.
232 process(M_AXIACLK)
233 begin
234   if (rising_edge (M_AXIACLK)) then
235     — Initiates AXI transaction delay
236     if (M_AXIARESETN = '0' ) then
237       init_txn_ff <= '0';
238       init_txn_ff2 <= '0';
239     else
240       init_txn_ff <= INIT_AXI_TXN;
241       init_txn_ff2 <= init_txn_ff;
242     end if;
243   end if;
244 end process;
245
246
247
248 —Write Address Channel
249
250
251 — The purpose of the write address channel is to request the address and
252 — command information for the entire transaction. It is a single beat
253 — of information.
254
255 — Note for this example the axi_awvalid/axi_wvalid are asserted at the same
256 — time, and then each is deasserted independent from each other.
257 — This is a lower-performance, but simpler control scheme.
258
259 — AXI VALID signals must be held active until accepted by the partner.
260
261 — A data transfer is accepted by the slave when a master has
262 — VALID data and the slave acknowledges it is also READY. While the master
263 — is allowed to generated multiple, back-to-back requests by not
264 — deasserting VALID, this design will add rest cycle for
265 — simplicity.
266
267 — Since only one outstanding transaction is issued by the user design,
268 — there will not be a collision between a new request and an accepted
269 — request on the same clock cycle.
270
271 process(M_AXIACLK)
272 begin
273   if (rising_edge (M_AXIACLK)) then
274     —Only VALID signals must be deasserted during reset per AXI spec
275     —Consider inverting then registering active-low reset for higher fmax
276     if (M_AXIARESETN = '0') then
277       axi_awvalid <= '0';
278     else
279       —Signal a new address/data command is available by user logic
280       if (start_single_write = '1') then
281         axi_awvalid <= '1';
282       elsif (M_AXI_AWREADY = '1' and axi_awvalid = '1') then
283         —Address accepted by interconnect/slave (issue of M_AXI_AWREADY by slave)
284         axi_awvalid <= '0';
285       end if;
286     end if;
287   end if;
288 end process;
289
290 — start_single_write triggers a new write
291 — transaction. write_index is a counter to
292 — keep track with number of write transaction
293 — issued/initiated
294 process(M_AXIACLK)
295 begin
296   if (rising_edge (M_AXIACLK)) then
297     if (M_AXIARESETN = '0' or init_txn_pulse = '1') then
298       write_index <= (others => '0');
299     elsif (start_single_write = '1') then
300       — Signals a new write address/ write data is
301       — available by user logic
302       write_index <= write_index + '1';
303     end if;
304   end if;
305 end process;
306
307
308

```

```

309  —Write Data Channel
310
311
312  —The write data channel is for transferring the actual data.
313  —The data generation is specific to the example design, and
314  —so only the WVALID/WREADY handshake is shown here
315
316  process(M_AXI_ACLK)
317  begin
318    if (rising_edge (M_AXI_ACLK)) then
319      if (M_AXI_ARESETN = '0') then
320        axi_wvalid <= '0';
321      else
322        if (start_single_write = '1') then
323          —Signal a new address/data command is available by user logic
324          axi_wvalid <= '1';
325        elsif (M_AXI_WREADY = '1' and axi_wvalid = '1') then
326          —Data accepted by interconnect/slave (issue of M_AXI_WREADY by slave)
327          axi_wvalid <= '0';
328        end if;
329      end if;
330    end process;
331
332
333  —Write Response (B) Channel
334
335
336
337
338  —The write response channel provides feedback that the write has committed
339  —to memory. BREADY will occur after both the data and the write address
340  —has arrived and been accepted by the slave, and can guarantee that no
341  —other accesses launched afterwards will be able to be reordered before it.
342
343  —The BRESP bit [1] is used indicate any errors from the interconnect or
344  —slave for the entire write burst. This example will capture the error.
345
346  —While not necessary per spec, it is advisable to reset READY signals in
347  —case of differing reset latencies between master/slave.
348
349  process(M_AXI_ACLK)
350  begin
351    if (rising_edge (M_AXI_ACLK)) then
352      if (M_AXI_ARESETN = '0' or init_txn_pulse = '1') then
353        axi_bready <= '0';
354      else
355        if (M_AXI_BVALID = '1' and axi_bready = '0') then
356          — accept/acknowledge bresp with axi_bready by the master
357          — when M_AXI_BVALID is asserted by slave
358          axi_bready <= '1';
359        elsif (axi_bready = '1') then
360          — deassert after one clock cycle
361          axi_bready <= '0';
362        end if;
363      end if;
364    end process;
365
366  —Flag write errors
367  write_resp_error <= (axi_bready and M_AXI_BVALID and M_AXI_BRESP(1));
368
369
370
371  —Read Data (and Response) Channel
372
373
374
375  —The Read Data channel returns the results of the read request
376  —The master will accept the read data by asserting axi_rready
377  —when there is a valid read data available.
378  —While not necessary per spec, it is advisable to reset READY signals in
379  —case of differing reset latencies between master/slave.
380
381  process(M_AXI_ACLK)
382  begin
383    if (rising_edge (M_AXI_ACLK)) then
384      if (M_AXI_ARESETN = '0' or init_txn_pulse = '1') then
385        axi_rready <= '1';
386      else
387        if (M_AXI_RVALID = '1' and axi_rready = '0') then

```

```

388      — accept/acknowledge rdata/rresp with axi_rready by the master
389      — when M_AXI_RVALID is asserted by slave
390      axi_rready <= '1';
391      elsif (axi_rready = '1') then
392          — deassert after one clock cycle
393          axi_rready <= '0';
394      end if;
395      end if;
396      end if;
397      end process;
398
399      —Flag write errors
400      read_resp_error <= (axi_rready and M_AXI_RVALID and M_AXI_RRESP(1));
401
402      -----
403      —User Logic
404      -----
405
406      —Address/Data Stimulus
407
408      —Address/data pairs for this example. The read and write values should
409      —match.
410      —Modify these as desired for different address patterns.
411
412      — Write Addresses
413      process(M_AXI_ACLK)
414      begin
415          if (rising_edge (M_AXI_ACLK)) then
416              if (M_AXI_ARESETN = '0') then — or init_txn_pulse = '1') then
417                  axi_awaddr <= (others => '0');
418              elsif (M_AXI_WREADY = '1' and axi_awvalid = '1') then
419                  — Signals a new write address/ write data is
420                  — available by user logic
421                  if (axi_awaddr >= (C_M_TARGET_SLAVE_HIGH_ADDR - C_M_TARGET_SLAVE_BASE_ADDR)) then
422                      axi_awaddr <= (others => '0');
423                  else
424                      axi_awaddr <= axi_awaddr + x"00000004";
425                  end if;
426              end if;
427          end if;
428      end process;
429
430
431      — Write data
432      process(M_AXI_ACLK)
433      begin
434          if (rising_edge (M_AXI_ACLK)) then
435              if (M_AXI_ARESETN = '0' or init_txn_pulse = '1') then
436                  axi_wdata <= data; — CHANGED from C_M_START_DATA_VALUE;
437              elsif (M_AXI_WREADY = '1' and axi_wvalid = '1') then
438                  — Signals a new write address/ write data is
439                  — available by user logic
440                  axi_wdata <= data; — CHANGED from C_M_START_DATA_VALUE + EXT(write_index ,C_M_AXI_DATA_WIDTH);
441              end if;
442          end if;
443      end process;
444
445      —implement master command interface state machine
446      MASTER_EXECUTION_PROC: process(M_AXI_ACLK)
447      begin
448          if (rising_edge (M_AXI_ACLK)) then
449              if (M_AXI_ARESETN = '0' ) then
450                  — reset condition
451                  — All the signals are ed default values under reset condition
452                  mst_exec_state <= IDLE;
453                  start_single_write <= '0';
454                  write_issued <= '0';
455                  start_single_read <= '0';
456                  read_issued <= '0';
457                  compare_done <= '0';
458                  ERROR <= '0';
459              else
460                  — state transition
461                  case (mst_exec_state) is
462                      when IDLE =>
463                          — This state is responsible to initiate
464                          — AXI transaction when init_txn_pulse is asserted

```

```

467     if ( init_txn_pulse = '1') then
468         mst_exec_state <= INIT_WRITE;
469         ERROR <= '0';
470         compare_done <= '0';
471     else
472         mst_exec_state <= IDLE;
473     end if;
474
475     when INIT_WRITE =>
476         — This state is responsible to issue start_single_write pulse to
477         — initiate a write transaction. Write transactions will be
478         — issued until last_write signal is asserted.
479         — write controller
480         if (writes_done = '1') then
481             mst_exec_state <= INIT_READ;
482         else
483             mst_exec_state <= INIT_WRITE;
484
485             if (axi_awvalid = '0' and axi_wvalid = '0' and M_AXI_BVALID = '0' and
486                 last_write = '0' and start_single_write = '0' and write_issued = '0') then
487                 start_single_write <= '1';
488                 write_issued <= '1';
489             elsif (axi_bready = '1') then
490                 write_issued <= '0';
491             else
492                 start_single_write <= '0'; —Negate to generate a pulse
493             end if;
494         end if;
495
496     when INIT_READ =>
497         mst_exec_state <= INIT_COMPARE;
498     when INIT_COMPARE =>
499         mst_exec_state <= IDLE;
500     when others =>
501         mst_exec_state <= IDLE;
502     end case;
503 end if;
504 end process;
505
506 —Terminal write count
507 process(M_AXI_ACLK)
508 begin
509     if (rising_edge (M_AXI_ACLK)) then
510         if (M_AXI_ARESETN = '0' or init_txn_pulse = '1') then
511             — reset condition
512             last_write <= '0';
513         else
514             —The last write should be associated with a write address ready response
515             if (write_index = CONV_STD_LOGIC_VECTOR(C_M_TRANSACTIONS_NUM, TRANS_NUM_BITS+1) and M_AXI_AWREADY =
516                 '1') then
517                 last_write <= '1';
518             end if;
519         end if;
520     end if;
521 end process;
522
523 —/*
524 — Check for last write completion.
525 —
526 — This logic is to qualify the last write count with the final write
527 — response. This demonstrates how to confirm that a write has been
528 — committed.
529 — */
530 process(M_AXI_ACLK)
531 begin
532     if (rising_edge (M_AXI_ACLK)) then
533         if (M_AXI_ARESETN = '0' or init_txn_pulse = '1') then
534             — reset condition
535             writes_done <= '0';
536         else
537             if (last_write = '1' and M_AXI_BVALID = '1' and axi_bready = '1') then
538                 —The writes_done should be associated with a bready response
539                 writes_done <= '1';
540             end if;
541         end if;
542     end if;
543 end process;
544

```



```
545  — Add user logic here
546
547  process (INIT_AXI_TXN)
548  begin
549    if (INIT_AXI_TXN = '1') then
550      data_n <= i_data;
551    else
552      data_n <= data;
553    end if;
554  end process;
555
556  process (M_AXI_ACLK)
557  begin
558    if (rising_edge (M_AXI_ACLK)) then
559      data <= data_n;
560    else
561      end if;
562  end process;
563  — User logic ends
564
565  end implementation;
```

## Listing A.5: top

```

1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.NUMERIC_STD.ALL;
5
6 package constants is
7
8
9     constant CAN_MAX_LENGTH : integer := (165-1); — max theoretical length of a data packet. (-1 since we
10 index from 0) 132 frame + 11 recessive bits
11 — measurement gives 165 in length..
12     constant CAN_MAX_LENGTH_WO_EOF : integer := (134-1);
13
14     constant CAN_RECESSIVE : std_logic := '1';
15     constant CAN_DOMINANT : std_logic := '0';
16
17 — Constants for Controller
18     constant INST_NOP : std_logic_vector(2 downto 0) := "000";
19     constant INST_SET_SAMPLE_POINT : std_logic_vector(2 downto 0) := "010";
20     constant INST_RESET : std_logic_vector(2 downto 0) := "001";
21     constant INST_GET_DATA : std_logic_vector(2 downto 0) := "011";
22     constant INST_SET_BAUD : std_logic_vector(2 downto 0) := "100";
23
24 — Sample bit widths at different speeds eg at 500 kbits = 20 samples
25     constant CAN_HIGH_SPEED : integer := 19; — 500 kpbs
26     constant CAN_MEDIUM_SPEED : integer := 39; — 250 kpbs
27     constant CAN_LOW_SPEED : integer := 79; — 125 kpbs
28
29 — Sample points for various speeds
30     constant SAMPLE_POINT_HIGH_SPEED : std_logic_vector(7 downto 0) := "00001100";
31     constant SAMPLE_POINT_MEDIUM_SPEED : std_logic_vector(7 downto 0) := "00011000";
32     constant SAMPLE_POINT_LOW_SPEED : std_logic_vector(7 downto 0) := "00110000";
33
34 — Cool down time for various speeds
35     constant COOL_DOWN_HIGH_SPEED : std_logic_vector(13 downto 0) := "00000010100000";
36     constant COOL_DOWN_MEDIUM_SPEED : std_logic_vector(13 downto 0) := "00000101000000";
37     constant COOL_DOWN_LOW_SPEED : std_logic_vector(13 downto 0) := "00001010000000";
38
39 — Vector defining speed
40     constant CAN_HIGH_SPEED_VEC : std_logic_vector(2 downto 0) := "001";
41     constant CAN_MEDIUM_SPEED_VEC : std_logic_vector(2 downto 0) := "010";
42     constant CAN_LOW_SPEED_VEC : std_logic_vector(2 downto 0) := "100";
43
44 — Actual no of low sample for EOF detection
45     constant EOF_HIGH_SPEED : std_logic_vector(9 downto 0) := "0011011100";
46     constant EOF_MEDIUM_SPEED : std_logic_vector(9 downto 0) := "0110111000";
47     constant EOF_LOW_SPEED : std_logic_vector(9 downto 0) := "1101110000";
48
49 — Alias for the three supported speeds, 500, 250, 125
50     constant HIGH_SPEED_ALIAS : std_logic_vector(13 downto 0) := std_logic_vector(to_unsigned(500, 14));
51     constant MEDIUM_SPEED_ALIAS : std_logic_vector(13 downto 0) := std_logic_vector(to_unsigned(250, 14));
52     constant LOW_SPEED_ALIAS : std_logic_vector(13 downto 0) := std_logic_vector(to_unsigned(125, 14));
53
54 end package constants;
55
56 library IEEE;
57 use IEEE.STD_LOGIC_1164.ALL;
58 use ieee.numeric_std.ALL;
59
60 library work;
61 use work.constants;
62
63 LIBRARY XilinxCoreLib;
64
65 entity top is
66 generic (
67     CAN_SPEED : integer := 500;
68     CAN_TRESHOLD : integer := 2000
69 );
70 port (
71     rst : in STD_LOGIC;
72     clk : in STD_LOGIC;
73     i_data1 : in std_logic_vector(13 downto 0);
74     i_data2 : in std_logic_vector(13 downto 0);
75     i_of1 : in std_logic;
76     i_of2 : in std_logic;
77     i_data : in std_logic_vector(31 downto 0);

```

```

77     i_clk_sample : in std_logic;
78     o_data       : out std_logic_vector(31 downto 0);
79     o_data_valid : out std_logic
80 );
81 end top;
82
83 architecture Behavioral of top is
84
85     component controller is
86     port (
87         clk       : in STD_LOGIC;
88         rst       : in STD_LOGIC;
89         i_data    : in std_logic_vector(31 downto 0);
90         o_rst    : out std_logic;
91         o_samplepoint : out std_logic_vector(3 downto 0);
92         o_baud_rate : out std_logic_vector(2 downto 0)
93     );
94 end component;
95
96     component detect_edge is
97     generic (
98         CAN_TRESHOLD : integer := 2000
99     );
100    port (
101        clk       : in std_logic;
102        rst       : in std_logic;
103        i_diff    : in std_logic_vector(13 downto 0);
104        o_edge_detected : out std_logic
105    );
106 end component;
107
108     component get_baud_rate is
109     generic (
110         CAN_SPEED : integer := 500;
111         CAN_TRESHOLD : integer := 2000
112     );
113    port (
114        clk       : in STD_LOGIC;
115        rst       : in STD_LOGIC;
116        i_data1   : in STD_LOGIC_VECTOR(13 downto 0);
117        i_data2   : in STD_LOGIC_VECTOR(13 downto 0);
118        i_of1     : in std_logic;
119        i_of2     : in std_logic;
120        i_clk_sample : in STD_LOGIC;
121        i_q       : in std_logic_vector(27 downto 0);
122        i_baud_rate : in std_logic_vector(2 downto 0); -- used to know baud rate, useful for eof detection !!!
123        o_data     : out std_logic_vector(13 downto 0);
124        --o_baud_rate : out std_logic_vector(2 downto 0); -- one bit for each speed (500 250 125)
125        o_sof_detected : out std_logic; -- trig a measurement upon SOF, this forces the ram to start storing
126        -- values, moves the controller to data state
127        o_eof_detected : out std_logic;
128        o_timestamp : out std_logic_vector(7 downto 0); -- unsigned number updated on i_clk_sample
129        o_of_detected : out std_logic_vector(1 downto 0);
130        o_d         : out std_logic_vector(27 downto 0)
131    );
132 end component;
133
134     component mem_controller is
135     port (
136         clk       : in std_logic;
137         rst       : in STD_LOGIC;
138         i_data    : in STD_LOGIC_VECTOR(27 downto 0);
139         i_clk_sample : in STD_LOGIC;
140         i_sof     : in std_logic;
141         i_eof     : in std_logic;
142         i_valid_timestamp : in std_logic;
143         i_valid_logic_data : in std_logic;
144         i_timestamp : in std_logic_vector(7 downto 0);
145         i_logic_data : in std_logic_vector(constants.CAN_MAX_LENGTH downto 0);
146         o_data     : out std_logic_vector(31 downto 0);
147         o_data_valid : out std_logic
148     );
149 end component;
150
151     component to_logic is
152     generic (
153         CAN_TRESHOLD : integer := 2000
154     );
155     port (

```

```

155     clk           : in STD_LOGIC;
156     rst           : in STD_LOGIC;
157     i_diff        : in STD_LOGIC_vector(13 downto 0);
158     i_clk_sample  : in STD_LOGIC;
159     i_set_point   : in STD_LOGIC_vector(3 downto 0);
160     i_baud_rate   : in STD_LOGIC_vector(2 downto 0);
161     i_edge_detected : in STD_LOGIC;
162     o_logic_data  : out STD_LOGIC_vector(constants.CAN_MAX_LENGTH downto 0);
163     o_logic_length : out std_logic_vector(7 downto 0);
164     o_logic_valid  : out std_logic
165 );
166 end component;
167
168
169 COMPONENT shift_reg
170 PORT (
171     d : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
172     clk : IN STD_LOGIC;
173     q : OUT STD_LOGIC_VECTOR(27 DOWNTO 0)
174 );
175 END COMPONENT;
176
177
178 --SIGNAL DECLARATIONS
179
180 signal logic_valid, edge_detected, global_rst, o_rst, eof_detected, sof_detected, clk_sample, clk_sample_n :
181     std_logic;
182 signal of_detected : std_logic_vector(1 downto 0);
183 signal logic_data : std_logic_vector(constants.CAN_MAX_LENGTH downto 0);
184 signal q, d : std_logic_vector(27 downto 0);
185 signal data : std_logic_vector(13 downto 0);
186 signal logic_data_length, timestamp : std_logic_vector(7 downto 0);
187 signal set_point : std_logic_vector(3 downto 0);
188 signal baud_rate : std_logic_vector(2 downto 0);
189
189 begin
190
191 global_rst <= rst or o_rst; -- Allow both reset from controller and reset generated from AXI bus
192 clk_sample_n <= i_clk_sample;
193
194 process (clk) -- delay sample clk with one cc to help timing
195 begin
196     if (rising_edge (clk)) then
197         clk_sample <= clk_sample_n;
198     else
199         end if;
200 end process;
201
202
203 cntnl : controller
204 port map (
205     clk           => clk ,
206     rst           => rst ,
207     i_data        => i_data ,
208     --o_data       => o_data ,
209     o_rst         => o_rst ,
210     o_samplepoint => set_point ,
211     o_baud_rate   => baud_rate
212 );
213
214 get_baud : get_baud_rate
215 generic map (
216     CAN_SPEED => CAN_TRESHOLD,
217     CAN_TRESHOLD => CAN_TRESHOLD
218 )
219 port map(
220     clk           => clk ,
221     rst           => global_rst ,
222     i_data1       => i_data1 ,
223     i_data2       => i_data2 ,
224     i_of1         => i_of1 ,
225     i_of2         => i_of2 ,
226     i_clk_sample  => clk_sample ,
227     i_q           => q ,
228     i_baud_rate  => baud_rate ,
229     o_data        => data ,
230     o_sof_detected => sof_detected ,
231     o_eof_detected => eof_detected ,
232     o_timestamp  => timestamp ,

```

```

233     o_of_detected => of_detected ,
234     o_d           => d
235 );
236
237 det_edge : detect_edge
238 generic map (
239     CAN_TRESHOLD => CAN_TRESHOLD
240 )
241 port map(
242     clk           => clk ,
243     rst           => global_rst ,
244     i_diff        => data ,
245     o_edge_detected => edge_detected
246 );
247
248 to_log : to_logic
249 generic map(
250     CAN_TRESHOLD => CAN_TRESHOLD
251 )
252 port map(
253     clk           => clk ,
254     rst           => global_rst ,
255     i_diff        => data ,
256     i_clk_sample  => clk_sample ,
257     i_set_point   => set_point ,
258     i_baud_rate   => baud_rate ,
259     i_edge_detected => edge_detected ,
260     o_logic_data  => logic_data ,
261     o_logic_length => logic_data_length ,
262     o_logic_valid => logic_valid
263 );
264
265 mem_cntrl : mem_controller
266 port map(
267     clk           => clk ,
268     rst           => global_rst ,
269     i_data        => q ,
270     i_clk_sample  => clk_sample ,
271     i_sof         => sof_detected ,
272     i_eof         => eof_detected ,
273     i_valid_timestamp => sof_detected, — Maybe it is interesting to see when the signal ends instead, or w
274     /e. Modify accordingly eg eof_detected
275     i_valid_logic_data => logic_valid ,
276     i_timestamp      => timestamp ,
277     i_logic_data     => logic_data ,
278     o_data           => o_data ,
279     o_data_valid     => o_data_valid
280 );
281 dual_channel_shift_reg : shift_reg
282 PORT MAP (
283     d => d ,
284     clk => clk_sample ,
285     q => q
286 );
287
288 end Behavioral;

```

## Listing A.6: controller

```

1  library IEEE;
2  library work;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use work.constants;
5
6  use IEEE.NUMERIC_STD.ALL;
7
8  -- this controller handles communication with AXI/CPU or FSL/CPU and the hardware.
9  -- As soon as a complete CAN frame is stored AXI is notified to relay the data
10 entity controller is
11 port (
12     clk           : in  STD_LOGIC;
13     rst           : in  STD_LOGIC;
14     i_data        : in  std_logic_vector(31 downto 0);
15     -- o_data      : out std_logic_vector(31 downto 0);
16     o_rst         : out std_logic;
17     o_samplepoint : out std_logic_vector(3 downto 0);
18     o_baud_rate   : out std_logic_vector(2 downto 0)
19 );
20 end controller;
21
22 architecture Behavioral of controller is
23
24     type state_type is (registers , ram);
25     signal state , state_n : state_type;
26
27     signal instruction : std_logic_vector(2 downto 0);
28     signal payload : std_logic_vector(28 downto 0);
29     signal r , r_n : get_data , get_data_n : std_logic;
30     signal samplepoint , samplepoint_n , cnt , cnt_n : std_logic_vector(3 downto 0);
31     signal data_length , data_length_n , read_length , read_length_n : std_logic_vector(13 downto 0); -- MSB in
32     signal baud_rate , baud_rate_n : std_logic_vector(2 downto 0);
33     signal data , data_n : std_logic_vector(31 downto 0);
34     --signal instruction std_logic_vector(31 downto 0);
35
36     begin
37
38     instruction <= i_data(31 downto 29);
39     payload <= i_data(28 downto 0);
40     o_rst <= r;
41     o_samplepoint <= samplepoint;
42     o_baud_rate <= baud_rate;
43     --o_data <= data;
44
45     clocked : process (clk)
46     begin
47     if (rising_edge(clk)) then
48     if (rst = '1') then
49         r <= '0';
50         samplepoint <= "0100";
51         get_data <= '0';
52         read_length <= (others => '0');
53         data_length <= (others => '0');
54         state <= registers;
55         data <= (others => '0');
56         cnt <= (others => '0');
57         baud_rate <= (others => '0'); --constants.CAN_HIGH_SPEED_VEC;
58     else
59         r <= r_n;
60         samplepoint <= samplepoint_n;
61         get_data <= get_data_n;
62         read_length <= read_length_n;
63         data_length <= data_length_n;
64         baud_rate <= baud_rate_n;
65         state <= state_n;
66         data <= data_n;
67         cnt <= cnt_n;
68     end if;
69     else
70     end if;
71     end process;
72
73     asynch : process(i_data , samplepoint , instruction , payload , cnt , data , data_length , read_length , state ,
74         baud_rate)
75     begin

```

```

76 r_n <= '0';
77 cnt_n <= (others => '0');
78 data_n <= (others => '0');
79 data_length_n <= data_length;
80 read_length_n <= read_length;
81 get_data_n <= '0';
82 samplepoint_n <= samplepoint;
83 state_n <= state;
84
85 baud_rate_n <= baud_rate;
86
87 case instruction is
88   when constants.INST_NOP =>
89     null;
90   when constants.INST_Reset =>
91     r_n <= '1'; -- reset memory, SR, all register i.e. send reset to all units.
92   when constants.INST_SET_SAMPLE_POINT =>
93     -- sets the sample point in the to_logic block,
94     samplepoint_n <= payload(3 downto 0);
95   when constants.INST_SET_BAUD =>
96     baud_rate_n <= payload(2 downto 0);
97   when constants.INST_GET_DATA => -- something ??
98     get_data_n <= '1'; -- this signal must be kept high untill everything is read!
99
100   data_length_n <= data_length;
101   read_length_n <= read_length;
102   case state is
103     when registers =>
104       data_n <= (others => '0');
105       cnt_n <= std_logic_vector(unsigned(cnt) + 1);
106
107       if (unsigned(cnt) = 1) then -- this is used to know how many clock cycles to read from mem_controller
108         data_length_n <= (others => '0');
109       elsif (unsigned(cnt) = 4) then
110         state_n <= ram;
111         cnt_n <= (others => '0');
112       else
113         state_n <= registers;
114       end if;
115     when ram =>
116       read_length_n <= std_logic_vector(unsigned(read_length) + 1);
117       data_n <= (others => '0');
118       if(unsigned(data_length) = unsigned(read_length)) then
119         state_n <= registers;
120         read_length_n <= (others => '0');
121       else
122         state_n <= ram;
123       end if;
124     when others =>
125       null;
126     end case;
127   when others =>
128     null;
129   end case;
130 end process;
131
132 end Behavioral;

```

## Listing A.7: get\_baud

```

1  library IEEE;
2  library work;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.NUMERIC_STD.ALL;
5  use work.constants;
6
7  entity get_baud_rate is
8      Generic (
9          CAN_SPEED : integer := 500;
10         CAN_TRESHOLD : integer := 2000
11     );
12     Port (
13         clk          : in  STD_LOGIC;
14         rst          : in  STD_LOGIC;
15         i_data1      : in  STD_LOGIC_vector(13 downto 0);
16         i_data2      : in  STD_LOGIC_vector(13 downto 0);
17         i_of1        : in  std_logic;
18         i_of2        : in  std_logic;
19         i_clk_sample : in  STD_LOGIC;
20         i_q          : in  std_logic_vector(27 downto 0);
21         i_baud_rate  : in  std_logic_vector(2 downto 0);
22         o_data       : out std_logic_vector(13 downto 0);
23         o_sof_detected : out std_logic;
24         o_eof_detected : out std_logic;
25         o_timestamp  : out std_logic_vector(7 downto 0);
26         o_of_detected : out std_logic_vector(1 downto 0);
27         o_d          : out std_logic_vector(27 downto 0)
28     );
29 end get_baud_rate;
30
31 architecture Behavioral of get_baud_rate is
32
33     type state_type is (init, wait_for_rise, wait_for_fall, d0,d1,d2,d3);
34     signal state, state_n : state_type;
35
36     type flag_order_type is (waiting_for_sof, waiting_for_eof);
37     signal flag_status, flag_status_n : flag_order_type;
38
39     signal sof_detected, sof_detected_n, eof_detected, eof_detected_n : std_logic;
40     signal baud_rate, baud_rate_n : std_logic_vector(2 downto 0);
41     signal of_detected, of_detected_n : std_logic_vector(1 downto 0);
42     signal eof_cnt, eof_cnt_n, EOF_NO : std_logic_vector(9 downto 0);
43     signal bit_width, bit_width_n : std_logic_vector(7 downto 0);
44     signal timestamp, timestamp_n : std_logic_vector(7 downto 0);
45     signal data, data_n, diff, can_speed_slv : std_logic_vector(13 downto 0);
46     signal d, d_n : std_logic_vector(27 downto 0);
47
48     begin
49
50         o_data      <= data;
51         o_timestamp <= timestamp;
52         o_of_detected <= of_detected;
53         o_eof_detected <= eof_detected;
54         o_sof_detected <= sof_detected;
55         o_d         <= d;
56
57         can_speed_slv <= std_logic_vector(to_unsigned(CAN_SPEED, 14));
58
59         look_up_table : process (can_speed_slv)
60         begin
61             CASE can_speed_slv IS
62                 when constants.HIGH_SPEED_ALIAS => EOF_NO <= constants.EOF_HIGH_SPEED; — 220;
63                 when constants.MEDIUM_SPEED_ALIAS => EOF_NO <= constants.EOF_MEDIUM_SPEED; — 440
64                 when constants.LOW_SPEED_ALIAS => EOF_NO <= constants.EOF_LOW_SPEED; — 880
65                 when others => EOF_NO <= constants.EOF_HIGH_SPEED; — 220, default
66             end case;
67         end process look_up_table;
68
69         sync : process (clk)
70         begin
71             if (rising_edge (clk)) then
72                 if (rst = '1') then
73                     timestamp <= (others => '0');
74                     bit_width <= (others => '0');
75                     baud_rate <= "001"; — defaults to high speed (500 kbit/s)
76                     data <= (others => '0');
77                     d <= (others => '0');

```



```

78     sof_detected <= '0';
79     eof_detected <= '0';
80     eof_cnt     <= (others => '0');
81     state      <= init;
82     flag_status <= waiting_for_sof;
83     of_detected <= (others => '0');
84
85     else
86         timestamp <= timestamp_n;
87         bit_width <= bit_width_n;
88         baud_rate <= baud_rate_n;
89         sof_detected <= sof_detected_n;
90         eof_detected <= eof_detected_n;
91         eof_cnt     <= eof_cnt_n;
92         data        <= data_n;
93         d           <= d_n;
94         state       <= state_n;
95         flag_status <= flag_status_n;
96         of_detected <= of_detected_n;
97     end if;
98 else
99 end if;
100
101 -- Simple state machine to control flags for start and end of frame
102 flag_status_proc : process(flag_status , sof_detected , eof_detected , i_clk_sample)
103 begin
104
105     flag_status_n <= flag_status;
106
107     case flag_status is
108     when waiting_for_sof =>
109         if(sof_detected = '1') then
110             flag_status_n <= waiting_for_eof;
111         end if;
112     when waiting_for_eof =>
113         if(eof_detected = '1') then
114             flag_status_n <= waiting_for_sof;
115         end if;
116     when others =>
117     end case;
118 end process flag_status_proc;
119
120 calc_baud : process(state , flag_status , data , i_clk_sample , i_q , i_data1 , i_data2 , i_baud_rate , bit_width ,
121                 timestamp , baud_rate , eof_cnt , EOF_NO , of_detected , i_of1 , i_of2)
122 begin
123     data_n <= std_logic_vector(unsigned(i_q(27 downto 14)) - unsigned(i_q(13 downto 0)));
124     diff <= std_logic_vector(unsigned(i_data1) - unsigned(i_data2)); -- used for calculating baud rate
125     timestamp_n <= std_logic_vector(unsigned(timestamp) + 1);
126
127     baud_rate_n <= i_baud_rate;
128
129     d_n <= i_data1 & i_data2;
130     bit_width_n <= bit_width;
131     eof_cnt_n <= eof_cnt;
132     sof_detected_n <= '0';
133     eof_detected_n <= '0';
134     state_n <= state;
135     of_detected_n <= of_detected;
136
137     if(i_of1 = '1') then -- Thanks to the following lines we catch any overflow and keep it until EOF
138         of_detected_n(0) <= '1';
139     end if;
140     if(i_of2 = '1') then
141         of_detected_n(1) <= '1';
142     end if;
143
144     case state is
145     when init =>
146         of_detected_n <= (others => '0'); -- reset of register
147         if(i_clk_sample = '1' and (unsigned(diff) < to_unsigned(CAN_TRESHOLD, 14))) then -- signal currently low
148             state_n <= wait_for_rise;
149         elsif(i_clk_sample = '1' and (unsigned(diff) > to_unsigned(CAN_TRESHOLD, 14))) then -- signal currently
150             state_n <= wait_for_fall;
151             if(flag_status = waiting_for_sof) then
152                 sof_detected_n <= '1';
153             end if;
154         else

```

```

155     state_n <= init;
156   end if;
157   when wait_for_rise =>
158     if (i_clk_sample = '1' and (unsigned(diff) > to_unsigned(CAN_TRESHOLD, 14))) then
159       bit_width_n <= std_logic_vector(unsigned(bit_width) + 1);
160       if (flag_status = waiting_for_sof) then
161         sof_detected_n <= '1';
162       end if;
163       state_n <= wait_for_fall;
164     elsif (i_clk_sample = '1' and unsigned(eof_cnt) > unsigned(EOF_NO)) then
165       if (flag_status = waiting_for_eof) then
166         eof_detected_n <= '1';
167       end if;
168       eof_cnt_n <= (others => '0');
169       state_n <= init;
170     elsif (i_clk_sample = '1') then
171       eof_cnt_n <= std_logic_vector(unsigned(eof_cnt) + 1);
172       state_n <= d0; -- delay
173     else
174       state_n <= wait_for_rise;
175     end if;
176   when wait_for_fall =>
177     eof_cnt_n <= (others => '0');
178     if (i_clk_sample = '1' and (unsigned(diff) < to_unsigned(CAN_TRESHOLD, 14))) then -- signal low, One bit
179       -- time measured
180       bit_width_n <= (others => '0');
181       state_n <= init;
182     elsif (i_clk_sample = '1') then -- initiate delay loop
183       bit_width_n <= std_logic_vector(unsigned(bit_width) + 1);
184       state_n <= d0;
185     else
186     end if;
187   when d0 =>
188     state_n <= d1;
189   when d1 =>
190     state_n <= d2;
191   when d2 =>
192     state_n <= d3;
193   when d3 =>
194     if (unsigned(eof_cnt) /= 0) then
195       state_n <= wait_for_rise;
196     else
197       state_n <= wait_for_fall;
198     end if;
199   when others =>
200     end case;
201 end process calc_baud;
end Behavioral;

```

## Listing A.8: detect\_edge

```

1  library work;
2  library ieee;
3
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.ALL;
6  use work.constants;
7
8  — Susceptible to jitter on the bus.
9  — Therefore 3 samples in a row need to to have changed signal level for an edge-detect event to occur.
10
11  entity detect_edge is
12  generic (
13  CAN_TRESHOLD : integer := 2000
14  );
15  port (
16  clk      : in std_logic;
17  rst      : in std_logic;
18  i_diff   : in std_logic_vector(13 downto 0);
19  o_edge_detected : out std_logic
20  );
21  end entity detect_edge;
22
23  architecture detect_edge_arch of detect_edge is
24  —signal diff_n, diff_reg : std_logic_vector(13 downto 0);
25  type state_type is (wait_for_rise, wait_for_fall);
26  signal state, state_n : state_type;
27
28  signal jitter_cnt, jitter_cnt_n : std_logic;
29
30  begin
31
32  async: process(state, i_diff)
33  begin
34
35  o_edge_detected <= '0';
36  jitter_cnt_n <= '0';
37  state_n <= state;
38
39  case state is
40  when wait_for_rise =>
41  if(unsigned(i_diff) <= to_unsigned(CAN_TRESHOLD, 14)) then
42  jitter_cnt_n <= '0';
43  state_n <= wait_for_rise;
44  else
45  jitter_cnt_n <= '1';
46  if(jitter_cnt = '1') then
47  o_edge_detected <= '1';
48  state_n <= wait_for_fall;
49  else
50  state_n <= wait_for_rise;
51  end if;
52  end if;
53  when wait_for_fall =>
54  if(unsigned(i_diff) >= to_unsigned(CAN_TRESHOLD, 14)) then
55  jitter_cnt_n <= '0';
56  state_n <= wait_for_fall;
57  else
58  jitter_cnt_n <= '1';
59  if(jitter_cnt = '1') then
60  o_edge_detected <= '1';
61  state_n <= wait_for_rise;
62  else
63  state_n <= wait_for_fall;
64  end if;
65  end if;
66  when others =>
67
68  end case;
69  end process;
70
71
72  sync : process(clk)
73  begin
74  if rising_edge(clk) then
75  if(rst = '1') then
76  state <= wait_for_rise;
77  jitter_cnt <= '0';

```

```
78     else
79         state      <= state_n;
80         jitter_cnt <= jitter_cnt_n;
81     end if;
82     else
83     end if;
84
85
86 end process;
87 end architecture detect_edge_arch;
```

## Listing A.9: to\_logic

```

1  library IEEE;
2  library work;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.NUMERIC_STD.ALL;
5  use work.constants;
6
7  entity to_logic is
8  generic (
9  CAN_TRESHOLD : integer := 2000
10 );
11 port (
12   clk       : in STD_LOGIC;
13   rst       : in STD_LOGIC;
14   i_diff    : in STD_LOGIC_vector(13 downto 0);
15   i_clk_sample : in STD_LOGIC;
16   i_set_point : in STD_LOGIC_vector(3 downto 0);
17   i_baud_rate : in STD_LOGIC_vector(2 downto 0);
18   i_edge_detected : in STD_LOGIC;
19   o_logic_data  : out STD_LOGIC_vector(work.constants.CAN_MAX_LENGTH downto 0);
20   o_logic_length : out std_logic_vector(7 downto 0);
21   o_logic_valid : out std_logic
22 );
23 end to_logic;
24
25 architecture Behavioral of to_logic is
26
27 type state_type is (init, hold, sample, cool_down, d0, d1, d2, d3);
28 signal state, state_n : state_type;
29
30 type last_added_type is (none, recessive, dominant);
31 signal last_added, last_added_n : last_added_type;
32
33 signal logic_data, logic_data_n : std_logic_vector(constants.CAN_MAX_LENGTH downto 0);
34 signal cnt, cnt_n, sample_point, cnt_length, cnt_length_n, cooldown_cnt, cooldown_cnt_n : std_logic_vector(7
35   downto 0);
36 signal cnt_rec, cnt_rec_n : std_logic_vector(2 downto 0); -- 7 recessive bits marks definite end of
37   transmission
38 signal diff, cooldown_threshold : std_logic_vector(13 downto 0);
39
40 begin
41   diff <= i_diff;
42   o_logic_data <= logic_data;
43
44   look_up_table : Process(i_baud_rate)
45   begin
46     CASE i_baud_rate IS
47     when constants.CAN_HIGH_SPEED_VEC => sample_point <= constants.SAMPLE_POINT_HIGH_SPEED;
48     when constants.CAN_MEDIUM_SPEED_VEC => sample_point <= constants.SAMPLE_POINT_MEDIUM_SPEED;
49     when constants.CAN_LOW_SPEED_VEC => sample_point <= constants.SAMPLE_POINT_LOW_SPEED;
50     when others => sample_point <= constants.SAMPLE_POINT_HIGH_SPEED;
51     end case;
52
53     CASE i_baud_rate IS
54     when constants.CAN_HIGH_SPEED_VEC => cooldown_threshold <= constants.COOL_DOWN_HIGH_SPEED;
55     when constants.CAN_MEDIUM_SPEED_VEC => cooldown_threshold <= constants.COOL_DOWN_MEDIUM_SPEED;
56     when constants.CAN_LOW_SPEED_VEC => cooldown_threshold <= constants.COOL_DOWN_LOW_SPEED;
57     when others => cooldown_threshold <= constants.COOL_DOWN_HIGH_SPEED;
58     end case;
59   end process look_up_table;
60
61   sync : process(clk)
62   begin
63     if(rising_edge(clk)) then
64       if(rst = '1') then
65         last_added <= none;
66         state <= init;
67         cnt <= (others => '0');
68         logic_data <= (others => '0');
69         cnt_rec <= (others => '0');
70         cnt_length <= (others => '0');
71         cooldown_cnt <= (others => '0');
72       else
73         state <= state_n;
74         last_added <= last_added_n;
75         cnt <= cnt_n;

```

```

76     logic_data    <= logic_data_n;
77     cnt_rec       <= cnt_rec_n;
78     cnt_length   <= cnt_length_n;
79     cooldown_cnt <= cooldown_cnt_n;
80   end if;
81   else
82   end if;
83 end process;
84
85 sample_proc : process(state, cnt, i_edge_detected, i_clk_sample, logic_data, cnt_rec, cnt, sample_point,
86                   cnt_length, diff, last_added, cooldown_cnt)
87 begin
88   cnt_n <= cnt;
89   cnt_rec_n <= cnt_rec;
90   cnt_length_n <= cnt_length;
91   logic_data_n <= logic_data;
92   o_logic_valid <= '0';
93   o_logic_length <= (others => '0');
94   last_added_n <= last_added;
95   state_n <= state;
96   cooldown_cnt_n <= cooldown_cnt;
97
98   case state is
99   when init =>
100    if(i_edge_detected = '1') then
101      state_n <= hold;
102    else
103      state_n <= init;
104    end if;
105   when hold =>
106    if(i_edge_detected = '1') then — force resync
107      cnt_n <= (others => '0');
108      state_n <= hold;
109    elsif(unsigned(cnt) < unsigned(sample_point) and i_clk_sample = '1') then — send to delay loop
110      cnt_n <= std_logic_vector((unsigned(cnt) + 1));
111      state_n <= d0;
112    elsif(unsigned(cnt) < unsigned(sample_point)) then
113      state_n <= hold;
114    else
115      cnt_n <= (others => '0');
116      state_n <= sample;
117    end if;
118   when sample =>
119    if(i_edge_detected = '1') then — resync
120      cnt_n <= (others => '0');
121      cnt_rec_n <= (others => '0');
122      state_n <= hold;
123    else
124      if(unsigned(cnt_rec) = 6) then — 7 recessive bits in a row, definite end of message!
125        state_n <= init;
126        o_logic_valid <= '1';
127        o_logic_length <= cnt_length;
128        last_added_n <= none;
129
130        — ##### Reset all sampling-dependant registers
131        cnt_n <= (others => '0');
132        cnt_rec_n <= (others => '0');
133        cnt_length_n <= (others => '0');
134        logic_data_n <= (others => '0');
135        last_added_n <= none;
136        — ##### RESET
137
138      else — must keep track of latest added!!!
139        cnt_n <= (others => '0'); — set to zero, restart counting to sample point —std_logic_vector(unsigned
          (cnt) + 1);
140        state_n <= cool_down;
141        if(unsigned(diff) > CAN_TRESHOLD) then
142          logic_data_n <= logic_data(constants.CAN_MAX_LENGTH-1 downto 0) & constants.CAN_DOMINANT;
143          last_added_n <= dominant;
144          CASE last_added IS
145            when recessive => cnt_rec_n <= (others => '0');
146            when dominant => cnt_rec_n <= std_logic_vector(unsigned(cnt_rec) + 1);
147            when others => null;
148          end case;
149          cnt_length_n <= std_logic_vector(unsigned(cnt_length) + 1);
150        else
151          logic_data_n <= logic_data(constants.CAN_MAX_LENGTH-1 downto 0) & constants.CAN_RECESSIVE;
152          last_added_n <= recessive;

```

```

153     CASE last_added IS
154         when dominant => cnt_rec_n <= (others => '0');
155         when recessive => cnt_rec_n <= std_logic_vector(unsigned(cnt_rec) + 1);
156         when others => null;
157     end case;
158     cnt_length_n <= std_logic_vector(unsigned(cnt_length) + 1);
159     end if;
160
161     end if;
162     end if;
163     when d0 =>
164         if(i_edge_detected = '1') then — force resync
165             cnt_n <= (others => '0');
166             state_n <= hold;
167         else
168             state_n <= d1;
169         end if;
170     when d1 =>
171         if(i_edge_detected = '1') then — force resync
172             cnt_n <= (others => '0');
173             state_n <= hold;
174         else
175             state_n <= d2;
176         end if;
177     when d2 =>
178         if(i_edge_detected = '1') then — force resync
179             cnt_n <= (others => '0');
180             state_n <= hold;
181         else
182             state_n <= d3;
183         end if;
184     when d3 =>
185         state_n <= hold;
186         if(i_edge_detected = '1') then — force resync
187             cnt_n <= (others => '0');
188         elsif(i_clk_sample = '1' and unsigned(cnt) = unsigned(sample_point)) then
189             state_n <= sample;
190         else
191             end if;
192     when cool_down => — Restrain from peeking at the bus
193         cnt_n <= (others => '0');
194         if(i_edge_detected = '1') then — resync
195             state_n <= hold;
196         else
197             if(unsigned(cooldown_cnt) = unsigned(cooldown_treshold)) then — ok to retrig
198                 state_n <= hold;
199                 cooldown_cnt_n <= (others => '0');
200             else — stay here
201                 cooldown_cnt_n <= std_logic_vector(unsigned(cooldown_cnt) + 1);
202             end if;
203         — if( > unsigned(sample_point(5 downto 0)) hold of for 1/4 of a bit length. Since we sample 75% in the
204         — bit, pause 25% before we restart counters.
205         end if;
206     when others =>
207     end case;
208 end process;
209 end Behavioral;

```

## Listing A.10: Controller

```

1  library IEEE;
2  library work;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use work.constants;
5
6  use IEEE.NUMERIC_STD.ALL;
7
8  entity mem_controller is
9  Port (
10     clk           : in std_logic;
11     rst           : in STD_LOGIC;
12     i_data        : in STD_LOGIC_VECTOR (27 downto 0);
13     i_clk_sample  : in STD_LOGIC;
14     i_sof         : in std_logic;
15     i_eof         : in std_logic;
16     i_valid_timestamp : in std_logic;
17     i_valid_logic_data : in std_logic;
18     i_timestamp   : in std_logic_vector(7 downto 0);
19     i_logic_data  : in std_logic_vector(work.constants.CAN_MAX_LENGTH downto 0);
20     o_data        : out std_logic_vector(31 downto 0);
21     o_data_valid  : out std_logic
22     );
23 end mem_controller;
24
25 architecture Behavioral of mem_controller is
26
27 type state_data_detected is (init, wait_for_eof, write_sample, valid_logic, wait_for_valid_data);
28 signal state_d, state_d_n : state_data_detected;
29
30 type state_delay_loop is (exec, d0, d1, d2, d3);
31 signal state_delay, state_delay_n : state_delay_loop;
32
33 type state_read_from_reg is (init, timestamp_state, logic0_state, logic1_state, logic2_state, logic3_state,
34     logic4_state, logic5_state, signal_eof_state);
35 signal state_r, state_r_n : state_read_from_reg;
36
37 signal logic0, logic1, logic2, logic3, logic4, logic5 : std_logic_vector(31 downto 0);
38 signal logic0_n, logic1_n, logic2_n, logic3_n, logic4_n, logic5_n : std_logic_vector(31 downto 0);
39 signal timestamp, timestamp_n : std_logic_vector(7 downto 0);
40
41 signal data, data_n : std_logic_vector(31 downto 0);
42 signal data_valid, data_valid_n : std_logic;
43
44 signal hold_cnt, hold_cnt_n : std_logic_vector(3 downto 0);
45
46 begin
47 -- #### Output
48 o_data <= data;
49 o_data_valid <= data_valid;
50 -- #### Output
51
52 dma : process(i_clk_sample, i_data, i_sof, i_eof, data, data_valid, state_d, state_delay, state_r, timestamp,
53     logic0, logic1, logic2, logic3, logic4, logic5, hold_cnt)
54 begin
55     data_n <= (others => '0');
56     data_valid_n <= '0';
57     state_d_n <= state_d;
58     state_delay_n <= state_delay;
59     state_r_n <= state_r;
60     hold_cnt_n <= (others => '0');
61
62     case state_d is
63     when init =>
64         if(i_sof = '1') then
65             data_n <= x"FFFFFFFF"; -- Mark beginning of message with 0xFFFFFFFF since this data can not occur as
66             a valid sample
67             data_valid_n <= '1';
68             state_d_n <= wait_for_eof;
69         else
70             state_d_n <= init;
71         end if;
72     when wait_for_eof =>
73         if(i_valid_logic_data = '1') then
74             -- no more data, send register info when available
75             state_d_n <= valid_logic;

```



```

75     elsif(i_eof = '1') then
76         state_d_n <= wait_for_valid_data;
77     elsif(i_clk_sample = '1') then
78         state_d_n <= write_sample;
79     elsif(i_sof = '1') then
80         state_d_n <= wait_for_eof;
81     else
82     end if;
83
84 when write_sample =>
85     data_n <= (others => '0');
86     data_valid_n <= '0';
87     if(i_eof = '1') then
88         state_d_n <= wait_for_valid_data;
89     else
90         state_d_n <= state_d;
91     end if;
92     case state_delay is
93     when exec =>
94         data_n <= "00" & i_data(27 downto 14) & "00" & i_data(13 downto 0);
95         data_valid_n <= '1';
96         state_delay_n <= d0;
97     when d0 =>
98         state_delay_n <= d1;
99     when d1 =>
100        state_delay_n <= d2;
101    when d2 =>
102        state_delay_n <= d3;
103    when d3 =>
104        state_delay_n <= exec;
105        state_d_n <= wait_for_eof;
106    when others =>
107    end case;
108 when wait_for_valid_data =>
109     if(i_valid_logic_data = '1') then
110         state_d_n <= valid_logic;
111     else
112         state_d_n <= wait_for_valid_data;
113     end if;
114     -- Write to DMA controller via AXI, since we use the 'raw' data sampled, we need to wait while
115     -- i_clk_sample is still high
116 when valid_logic =>
117     case state_r is
118     when init =>
119         state_r_n <= timestamp_state;
120     when timestamp_state =>
121         if(unsigned(hold_cnt) = 0) then
122             data_n <= x"000000" & timestamp;
123             data_valid_n <= '1';
124             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
125         elsif(unsigned(hold_cnt) = 10) then
126             state_r_n <= logic0_state;
127         else
128             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
129         end if;
130     when logic0_state =>
131         if(unsigned(hold_cnt) = 0) then
132             data_n <= logic0;
133             data_valid_n <= '1';
134             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
135         elsif(unsigned(hold_cnt) = 10) then
136             state_r_n <= logic1_state;
137         else
138             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
139         end if;
140     when logic1_state =>
141         if(unsigned(hold_cnt) = 0) then
142             data_n <= logic1;
143             data_valid_n <= '1';
144             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
145         elsif(unsigned(hold_cnt) = 10) then
146             state_r_n <= logic2_state;
147         else
148             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
149         end if;
150     when logic2_state =>
151         if(unsigned(hold_cnt) = 0) then
152             data_n <= logic2;
153             data_valid_n <= '1';
154             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
155         else
156             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
157         end if;
158     end case;
159 end process;

```

```

153     elsif (unsigned(hold_cnt) = 10) then
154         state_r_n <= logic3_state;
155     else
156         hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
157     end if;
158     when logic3_state =>
159         if (unsigned(hold_cnt) = 0) then
160             data_n <= logic3;
161             data_valid_n <= '1';
162             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
163         elsif (unsigned(hold_cnt) = 10) then
164             state_r_n <= logic4_state;
165         else
166             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
167         end if;
168     when logic4_state =>
169         if (unsigned(hold_cnt) = 0) then
170             data_n <= logic4;
171             data_valid_n <= '1';
172             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
173         elsif (unsigned(hold_cnt) = 10) then
174             state_r_n <= logic5_state;
175         else
176             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
177         end if;
178     when logic5_state =>
179         if (unsigned(hold_cnt) = 0) then
180             data_n <= logic5;
181             data_valid_n <= '1';
182             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
183         elsif (unsigned(hold_cnt) = 10) then
184             state_r_n <= signal_eof_state;
185         else
186             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
187         end if;
188     when signal_eof_state => — Signals end of data by writing FFFF FFFF to RAM
189         if (unsigned(hold_cnt) = 0) then
190             data_n <= x"FFFFFFFF";
191             data_valid_n <= '1';
192             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
193         elsif (unsigned(hold_cnt) = 10) then
194             state_r_n <= init;
195             state_d_n <= init;
196         else
197             hold_cnt_n <= std_logic_vector(unsigned(hold_cnt) + 1);
198         end if;
199     when others =>
200     end case;
201 when others =>
202 end case;
203 end process dma;
204
205 update_reg : process(i_valid_timestamp , i_valid_logic_data , i_timestamp , i_logic_data , logic0 , logic1 , logic2
, logic3 , logic4 , logic5 , timestamp)
206 begin
207
208     if (i_valid_timestamp = '1') then
209         timestamp_n <= i_timestamp;
210     else
211         timestamp_n <= timestamp;
212     end if;
213
214     if (i_valid_logic_data = '1') then
215         logic0_n <= x"000000" & "000" & i_logic_data(164 downto 160);
216         logic1_n <= i_logic_data(159 downto 128);
217         logic2_n <= i_logic_data(127 downto 96);
218         logic3_n <= i_logic_data(95 downto 64);
219         logic4_n <= i_logic_data(63 downto 32);
220         logic5_n <= i_logic_data(31 downto 0);
221     else
222         logic0_n <= logic0;
223         logic1_n <= logic1;
224         logic2_n <= logic2;
225         logic3_n <= logic3;
226         logic4_n <= logic4;
227         logic5_n <= logic5;
228     end if;
229 end process update_reg;
230

```

```

231 synch : process(clk)
232 begin
233   if(rising_edge(clk)) then
234     if(rst = '1') then
235       state_d   <= init;
236       state_r   <= init;
237       state_delay <= exec;
238       logic0    <= (others => '0');
239       logic1    <= (others => '0');
240       logic2    <= (others => '0');
241       logic3    <= (others => '0');
242       logic4    <= (others => '0');
243       logic5    <= (others => '0');
244       timestamp <= (others => '0');
245       data      <= (others => '0');
246       data_valid <= '0';
247       hold_cnt  <= (others => '0');
248     else
249       state_d   <= state_d_n;
250       state_r   <= state_r_n;
251       state_delay <= state_delay_n;
252       logic0    <= logic0_n;
253       logic1    <= logic1_n;
254       logic2    <= logic2_n;
255       logic3    <= logic3_n;
256       logic4    <= logic4_n;
257       logic5    <= logic5_n;
258       timestamp <= timestamp_n;
259       data      <= data_n;
260       data_valid <= data_valid_n;
261       hold_cnt  <= hold_cnt_n;
262     end if;
263   else
264   end if;
265 end process synch;
266
267 end Behavioral;

```

## Listing A.11: Shift register

```

1  --- (c) Copyright 1995–2015 Xilinx, Inc. All rights reserved.
2  ---
3  --- This file contains confidential and proprietary information
4  --- of Xilinx, Inc. and is protected under U.S. and
5  --- international copyright and other intellectual property
6  --- laws.
7  ---
8  --- DISCLAIMER
9  --- This disclaimer is not a license and does not grant any
10 --- rights to the materials distributed herewith. Except as
11 --- otherwise provided in a valid license issued to you by
12 --- Xilinx, and to the maximum extent permitted by applicable
13 --- law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
14 --- WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
15 --- AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
16 --- BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
17 --- INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
18 --- (2) Xilinx shall not be liable (whether in contract or tort,
19 --- including negligence, or under any other theory of
20 --- liability) for any loss or damage of any kind or nature
21 --- related to, arising under or in connection with these
22 --- materials, including for any direct, or any indirect,
23 --- special, incidental, or consequential loss or damage
24 --- (including loss of data, profits, goodwill, or any type of
25 --- loss or damage suffered as a result of any action brought
26 --- by a third party) even if such damage or loss was
27 --- reasonably foreseeable or Xilinx had been advised of the
28 --- possibility of the same.
29 ---
30 --- CRITICAL APPLICATIONS
31 --- Xilinx products are not designed or intended to be fail-
32 --- safe, or for use in any application requiring fail-safe
33 --- performance, such as life-support or safety devices or
34 --- systems, Class III medical devices, nuclear facilities,
35 --- applications related to the deployment of airbags, or any
36 --- other applications that could lead to death, personal
37 --- injury, or severe property or environmental damage
38 --- (individually and collectively, "Critical
39 --- Applications"). Customer assumes the sole risk and
40 --- liability of any use of Xilinx products in Critical
41 --- Applications, subject only to applicable laws and
42 --- regulations governing limitations on product liability.
43 ---
44 --- THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
45 --- PART OF THIS FILE AT ALL TIMES.
46 ---
47 --- DO NOT MODIFY THIS FILE.
48 ---
49 --- IP VLVN: xilinx.com:ip:c_shift_ram:12.0
50 --- IP Revision: 3
51 ---
52 LIBRARY ieee;
53 USE ieee.std_logic_1164.ALL;
54 USE ieee.numeric_std.ALL;
55 ---
56 LIBRARY c_shift_ram_v12_0;
57 USE c_shift_ram_v12_0.c_shift_ram_v12_0;
58 ---
59 ENTITY shift_reg IS
60 PORT (
61   D : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
62   CLK : IN STD_LOGIC;
63   Q : OUT STD_LOGIC_VECTOR(27 DOWNTO 0)
64 );
65 END shift_reg;
66 ---
67 ARCHITECTURE shift_reg_arch OF shift_reg IS
68 ATTRIBUTE DowngradeIPIdentifiedWarnings : string;
69 ATTRIBUTE DowngradeIPIdentifiedWarnings OF shift_reg_arch : ARCHITECTURE IS "yes";
70 ---
71 COMPONENT c_shift_ram_v12_0 IS
72 GENERIC (
73   C_XDEVICEFAMILY : STRING;
74   C_VERBOSITY : INTEGER;
75   C_WIDTH : INTEGER;
76   C_DEPTH : INTEGER;
77   C_ADDR_WIDTH : INTEGER;

```

```

78     C_SHIFT_TYPE : INTEGER;
79     C_OPT_GOAL : INTEGER;
80     C_AINIT_VAL : STRING;
81     C_SINIT_VAL : STRING;
82     C_DEFAULT_DATA : STRING;
83     C_HAS_A : INTEGER;
84     C_HAS_CE : INTEGER;
85     C_REG_LAST_BIT : INTEGER;
86     C_SYNC_PRIORITY : INTEGER;
87     C_SYNC_ENABLE : INTEGER;
88     C_HAS_SCLR : INTEGER;
89     C_HAS_SSET : INTEGER;
90     C_HAS_SINIT : INTEGER;
91     C_MEM_INIT_FILE : STRING;
92     C_ELABORATION_DIR : STRING;
93     C_READ_MIF : INTEGER;
94     C_PARSER_TYPE : INTEGER
95 );
96 PORT (
97     A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
98     D : IN STD_LOGIC_VECTOR(27 DOWNTO 0);
99     CLK : IN STD_LOGIC;
100    CE : IN STD_LOGIC;
101    SCLR : IN STD_LOGIC;
102    SSET : IN STD_LOGIC;
103    SINIT : IN STD_LOGIC;
104    Q : OUT STD_LOGIC_VECTOR(27 DOWNTO 0)
105 );
106 END COMPONENT c_shift_ram_v12_0;
107 ATTRIBUTE X_CORE_INFO OF shift_reg : STRING;
108 ATTRIBUTE X_CORE_INFO OF shift_reg_arch : ARCHITECTURE IS "c_shift_ram_v12_0,Vivado_2013.4";
109 ATTRIBUTE CHECK_LICENSE_TYPE OF shift_reg : STRING;
110 ATTRIBUTE CHECK_LICENSE_TYPE OF shift_reg_arch : ARCHITECTURE IS "shift_reg,c_shift_ram_v12_0,{}";
111 ATTRIBUTE CORE_GENERATION_INFO : STRING;
112 ATTRIBUTE CORE_GENERATION_INFO OF shift_reg_arch : ARCHITECTURE IS "shift_reg,c_shift_ram_v12_0,{x_ipProduct=
    Vivado_2013.4,x_ipVendor=xilinx.com,x_ipLibrary=ip,x_ipName=c_shift_ram,x_ipVersion=12.0,
    x_ipCoreRevision=3,x_ipLanguage=VERILOG,C_XDEVICEFAMILY=zynq,C_VERBOSITY=0,C_WIDTH=28,C_DEPTH=90,
    C_ADDR_WIDTH=4,C_SHIFT_TYPE=0,C_OPT_GOAL=0,C_AINIT_VAL=00000000000000000000000000000000,C_SINIT_VAL
    =0000000000000000000000000000000000000000000000000000,C_DEFAULT_DATA=00000000000000000000000000000000,C_HAS_A=0,C_HAS_CE=0,
    C_REG_LAST_BIT=1,C_SYNC_PRIORITY=1,C_SYNC_ENABLE=0,C_HAS_SCLR=0,C_HAS_SSET=0,C_HAS_SINIT=0,
    C_MEM_INIT_FILE=no_coe_file_loaded,C_ELABORATION_DIR=./,C_READ_MIF=0,C_PARSER_TYPE=0}";
113 ATTRIBUTE X_INTERFACE_INFO : STRING;
114 ATTRIBUTE X_INTERFACE_INFO OF D : SIGNAL IS "xilinx.com:signal:data:1.0_d_intf_DATA";
115 ATTRIBUTE X_INTERFACE_INFO OF CLK : SIGNAL IS "xilinx.com:signal:clock:1.0_clk_intf_CLK";
116 ATTRIBUTE X_INTERFACE_INFO OF Q : SIGNAL IS "xilinx.com:signal:data:1.0_q_intf_DATA";
117 BEGIN
118 U0 : c_shift_ram_v12_0
119     GENERIC MAP (
120         C_XDEVICEFAMILY => "zynq",
121         C_VERBOSITY => 0,
122         C_WIDTH => 28,
123         C_DEPTH => 90,
124         C_ADDR_WIDTH => 4,
125         C_SHIFT_TYPE => 0,
126         C_OPT_GOAL => 0,
127         C_AINIT_VAL => "00000000000000000000000000000000",
128         C_SINIT_VAL => "00000000000000000000000000000000",
129         C_DEFAULT_DATA => "00000000000000000000000000000000",
130         C_HAS_A => 0,
131         C_HAS_CE => 0,
132         C_REG_LAST_BIT => 1,
133         C_SYNC_PRIORITY => 1,
134         C_SYNC_ENABLE => 0,
135         C_HAS_SCLR => 0,
136         C_HAS_SSET => 0,
137         C_HAS_SINIT => 0,
138         C_MEM_INIT_FILE => "no_coe_file_loaded",
139         C_ELABORATION_DIR => "./",
140         C_READ_MIF => 0,
141         C_PARSER_TYPE => 0
142     )
143     PORT MAP (
144         A => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 4)),
145         D => D,
146         CLK => CLK,
147         CE => '1',
148         SCLR => '0',
149         SSET => '0',
150         SINIT => '0',

```

```
151     Q => Q
152   );
153 END shift_reg_arch;
```

## A.5.3 Demonstration software

Listing A.12: Demonstration software

```
1  /*
2  * Copyright (c) 2009–2012 Xilinx, Inc. All rights reserved.
3  *
4  * Xilinx, Inc.
5  * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
6  * COURTESY TO YOU. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
7  * ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
8  * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
9  * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
10 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION.
11 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
12 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
13 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
14 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
15 * AND FITNESS FOR A PARTICULAR PURPOSE.
16 *
17 */
18
19 /*
20 * helloworld.c: simple test application
21 *
22 * This application configures UART 16550 to baud rate 9600.
23 * PS7 UART (Zynq) is not initialized by this application, since
24 * bootrom/bsp configures it to baud rate 115200
25 *
26 * _____
27 * | UART TYPE   BAUD RATE     |
28 * _____
29 * uarts550     9600
30 * uartlite     Configurable only in HW design
31 * ps7_uart     115200 (configured by bootrom/bsp)
32 */
33
34 #define DEBUG 1
35
36 #define SHAREABLE_DISABLE_CACHE 0x14DE2
37
38 #define BASE_ADDR 0x10000000
39 #define HIGH_ADDR 0x17FFFFFFF
40
41 #include <xil_io.h>
42 #include <xil_mmu.h>
43 #include <xparameters.h>
44
45 #include <stdio.h>
46 #include <sleep.h>
47 #include "platform.h"
48
49
50 #include <xil_printf.h>
51
52 //void print(char *str);
53
54 int send_data();
55 void ddr_clean();
56 void init_DDR();
57 int test_DDR();
58
59 int read_data(int addr);
60
61 void use_ram();
62
63 /*
64 * Demonstration software, prints via UART
65 * Data is accessed by (data & 0xFFFF) which is i_data1, and (data & 0xFFFF0000) which is i_data2
66 */
67 int main()
68 {
69     init_platform();
70     init_DDR();
71
72     int i, data = 0;
73     xil_printf("Hello_World_1_2_3_4\n\r");
```

```

74
75
76     while(1) {
77         for (i = BASE_ADDR; i < BASE_ADDR + 0x200; i = i + 0x4) {
78             data = Xil_In32LE(i);
79             xil_printf("DDR_content_at_%x:_%x\n", i, data);
80         }
81         xil_printf("End_of_iteration");
82         sleep(4);
83     }
84     return 0;
85 }
86
87
88 /**
89  * Sets the desired memory range to shareable, turning off caching, making it possible for the CAN hw
90  * controller to write in it
91  * Sets LMB chunks at a time, therefore i+0x10 0000
92  */
93 void init_DDR()
94 {
95     int i;
96     for(i = BASE_ADDR; i < HIGH_ADDR; i = i+0x100000) {
97         Xil_SetTlbAttributes(i, SHAREABLE_DISABLE_CACHE);
98     }
99 }
100
101 /**
102  * Verify that the DDR is set to zero
103  */
104 int test_DDR()
105 {
106     int i;
107     for(i = XPAR_PS7_DDR_0_S_AXI_BASEADDR; i <= XPAR_PS7_DDR_0_S_AXI_HIGHADDR; i = i+4)
108         if(Xil_In32LE(i) != 0)
109             return -1;
110     return 0;
111 }
112
113 /**
114  * Viciously stolen from http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+-+32+Bit+DDR+Access+with+ECC+Tech+Tip
115  * Initiates memory to 0
116  */
117 void ddr_clean()
118 {
119     memset((void *)0x100000,0,0x100000);
120     memset((void *)0x200000,0,0x1000000);
121     memset((void *)0x10200000,0,0x10000000);
122     memset((void *)0x20200000,0,0x10000000);
123     memset((void *)0x30200000,0,0xFE00000);
124 }

```



## A.5.4 Desktop and client interaction

Listing A.13: MWE Server

```
1  #!/usr/bin/python # This is server.py file
2
3  import socket      # Import socket module
4  import threading
5  import struct
6
7  print 'Starting_server...'
8  host='127.0.0.1'
9  port=8881
10 s = socket.socket() # Create a socket object
11 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # reuse port..., useful when debugging/early
12   development
13 s.bind((host, port)) # Bind to the port
14 s.listen(5) # Max 5 clients in queue
15
16 # Now wait for client connection.
17 c, addr = s.accept() # Establish connection with client.
18 print 'Got_connection_from', addr
19 # read raw data
20 # read length
21 # read timestamp+samplepoint
22 # read logic data
23 for i in xrange(1, 16001):
24     a = c.recv(4)
25     myint = struct.unpack("<i", a)[0]
26     print myint
27 c.close()
```

## Listing A.14: MWE Client

```
1  /*
2   Code to run on CPU
3  */
4  #include <stdio.h> // printf
5  #include <string.h> // strlen
6  #include <stdlib.h>
7  #include <errno.h>
8  #include <sys/socket.h> // socket
9  #include <arpa/inet.h> // inet_addr
10
11 int main(int argc , char *argv[])
12 {
13     int sock;
14     struct sockaddr_in server;
15     char message[1000] , server_reply[2000];
16
17     server.sin_addr.s_addr = inet_addr("127.0.0.1");
18     server.sin_family = AF_INET;
19     server.sin_port = htons(8881);
20
21     //Create socket
22     sock = socket(AF_INET , SOCK_STREAM , 0); // TCP socket
23     if (sock == -1)
24     {
25         printf("Could_not_create_socket");
26         return 0;
27     }
28     puts("Socket_created"); // debug line
29
30     //Connect to remote server
31     if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
32     {
33         perror("connect_failed_Error");
34         return 1;
35     }
36
37     puts("Connected\n"); // debug line
38
39     int *data , i; // pointer to data vector , loop variable
40
41     data = malloc(sizeof(int) * 16000);
42     for(i = 0; i < 15999; i++)
43     {
44         data[i] = i;
45     }
46     data[15999] = 0;
47
48     int a = 1337;
49     puts("before_send");
50     int b = send(sock , data , sizeof(int)*16000, 0) ;
51     if (b < 0)
52     {
53         printf("Oh_dear ,_something_went_wrong_with_send(!_%s\n" , strerror(errno));
54     }
55     puts("after_send");
56     free(data);
57
58     close(sock);
59     return 0;
60 }
```



# Bibliography

- [1] A. R. Hambley, "Amplifiers: Specifications and External Characteristics," in *Electrical Engineering, Principles And Applications* 5th ed. New Jersey: Pearson, 2011 ch. 11, sec. 8, pp. 561-563.
- [2] Vector Informatik GmbH, "Error Tracking" [Online] Available: [http://elearning.vector.com/index.php?&wbt\\_ls\\_seite\\_id=522851&root=378422&seite=vl\\_can\\_introduction\\_en](http://elearning.vector.com/index.php?&wbt_ls_seite_id=522851&root=378422&seite=vl_can_introduction_en)
- [3] European Editors, Digikey, "Communication in Industrial Networks", (April, 2012) [Online] Available: <http://www.digikey.com/en-US/articles/techzone/2012/apr/communication-in-industrial-networks>
- [4] Embedded Systems Academy, "Full CAN" [Online] Available:<http://www.esacademy.com/en/library/technical-articles-and-documents/can-and-canopen/selecting-a-can-controller.html>
- [5] B. A. Forouzan, "Error Detection and Correction," in *Data Communications And Networking* 4th ed. New York: McGraw-Hill, 2007 ch. 10, sec. 4, pp. 284-297.
- [6] B. A. Forouzan, "Multiple Access," in *Data Communications And Networking* 4th ed. New York: McGraw-Hill, 2007 ch. 12, sec. 1, pp. 373-377.
- [7] Semiconductor Components Industries LLC, "AMIS-4168x Fault Tolerant Transceiver Design Considerations Using CANLSFT", (Januari, 2009) Datasheet pp.2 [Online] Available: [http://www.onsemi.com/pub\\_link/Collateral/AND8366-D.PDF](http://www.onsemi.com/pub_link/Collateral/AND8366-D.PDF)

- [8] Semiconductor Components Industries LLC, "AMIS-4168x Fault Tolerant Transceiver Design Considerations Using CANLSFT", (Januari, 2009) Datasheet pp.8 [Online] Available: [http://www.onsemi.com/pub\\_link/Collateral/AND8366-D.PDF](http://www.onsemi.com/pub_link/Collateral/AND8366-D.PDF)
- [9] National Instruments, "CAN Physical Layer and Termination Guide" (August 01, 2014) [Online] Available: <http://www.ni.com/white-paper/9759/en/>
- [10] P. Manolakis, "Sampling and Reconstruction of Signals" in *Digital Signal Processing: Principles, Algorithms, and Applications* 4th ed. New Jersey: Pearson, 2007 ch. 6, sec. 3, pp. 371-380.
- [11] P. Manolakis, "Efficient Computation of the DFT: Fast Fourier Transform Algorithms" in *Digital Signal Processing: Principles, Algorithms, and Applications* 4th ed. New Jersey: Pearson, 2007 ch. 8, sec. 1, pp. 476-477.
- [12] B. A. Olshausen, "Aliasing", (October, 2000) [Online] Available: <http://redwood.berkeley.edu/bruno/npb261/aliasing.pdf>
- [13] J. F. Wakerly, "Zo: Transmission Lines, Reflections, and Termination," from Supplementary material to *Digital Design Principles and Practices* 4th ed. New Jersey: Pearson, 2006 pp. Zo-3. [http://www.ddpp.com/DDPP4student/Supplementary\\_sections/Zo.pdf](http://www.ddpp.com/DDPP4student/Supplementary_sections/Zo.pdf)
- [14] A. R. Hambley, "Computer-Based Instrumentation Systems," in *Electrical Engineering, Principles And Applications* 5th ed. New Jersey: Pearson, 2011 ch. 9, sec. 3, pp. 472-473.
- [15] A. R. Hambley, "Frequency Response, Bode Plots, and Resonance," in *Electrical Engineering, Principles And Applications* 5th ed. New Jersey: Pearson, 2011 ch. 6, sec. 2, pp. 313-318.
- [16] C. H. McGregor, "Plug, socket & voltage by country" (2015) [Online] Available: <http://www.worldstandards.eu/electricity/plug-voltage-by-country/>
- [17] National Instruments, "What is a Pulse Modulation (PWM) Signal and What is it User For?", (2003) [Online] Available: <http://digital.ni.com/public.nsf/allkb/294E67623752656686256DB800508989>
- [18] A. R. Hambley, "Frequency Response, Bode Plots, and Resonance," in *Electrical Engineering, Principles And Applications* 5th ed. New Jersey: Pearson, 2011 ch. 6, sec. 5, pp. 325-329.

- [19] B. Landman et al., "Band Pass Filter", Johns Hopkins University, Baltimore, MD (2008) [Online] Available: [http://pages.jh.edu/~bmesignals/analysis\\_of\\_frequency\\_space.html](http://pages.jh.edu/~bmesignals/analysis_of_frequency_space.html)
- [20] Philips Semiconductors, "SJA1000 Stand-alone CAN controller", (November, 1992) Datasheet pp. 7-52 [Online] Available: <http://www.e-lab.de/downloads/DOCs/PCA82C200.pdf>
- [21] Datasheet LT1568, Linear Technology, (2001) [Online] Available: <http://cds.linear.com/docs/en/datasheet/1568f.pdf>
- [22] AXI DMA, Xilinx (2015) [Online] Available: [http://www.xilinx.com/support/documentation/ip\\_documentation/axi\\_dma/v7\\_1/pg021\\_axi\\_dma.pdf](http://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf)
- [23] Vivado Design Suite, Xilinx, (2015) [Online] Available: <http://www.xilinx.com/products/design-tools/vivado.html>
- [24] R. Griffin, Silica EMEA, "Controlling the Zynq MMU / caches using a custom Xilinx SDK library" (2014) [Online] Available: [http://www.silica.com/fileadmin/02\\_Products/Productdetails/Xilinx/Zynq\\_MMU\\_caches\\_control\\_ver1.0.pdf](http://www.silica.com/fileadmin/02_Products/Productdetails/Xilinx/Zynq_MMU_caches_control_ver1.0.pdf)
- [25] J. Eidson, Agilent Technologies, "IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems - A Tutorial", (2005) [Online] Available: <http://www.nist.gov/el/isd/ieee/upload/tutorial-basic.pdf>
- [26] Datasheet Raspberry Pi, Raspberry Pi Foundation [Online] Available: <http://www.farnell.com/datasheets/1524403.pdf>