



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

---

# Hardware-in-the-loop Communication Interface

Master's thesis in Embedded Electronic System Design

Yang Zhang  
Jiaxing Zhu

---

Department of Computer Science and Engineering  
CHALMERS UNIVERSITY OF TECHNOLOGY  
UNIVERSITY OF GOTHENBURG  
Gothenburg, Sweden 2020



MASTER'S THESIS 2020

# Hardware-in-the-loop Communication Interface

Yang Zhang  
Jiaxing Zhu



UNIVERSITY OF  
GOTHENBURG

---



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering  
CHALMERS UNIVERSITY OF TECHNOLOGY  
UNIVERSITY OF GOTHENBURG  
Gothenburg, Sweden 2020

## Hardware-in-the-loop Communication Interface

YANG ZHANG  
JIAXING ZHU

© YANG ZHANG, 2020.

© JIAXING ZHU, 2020.

Supervisor: Roger Johansson, Department of Computer Science and Engineering

Supervisor: Tobias Falkman, Volvo Truck Corporation

Examiner: Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2020

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

SE-412 96 Gothenburg

Telephone +46 31 772 1000



YANG ZHANG

JIAXING ZHU

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

## Abstract

A system has been designed to perform real-time transmission, between a master node and single slave node, of two kinds of protocols, EtherCAT and CAN FD. Implementing such a system requires the design of a PCB for EtherCAT and CAN FD communication interface and to achieve protocol conversion in C.

The system is able to perform message transmission successfully and correctly in most test cases except that unstable results occur when 64 bytes CAN FD message is transmitted. Other aspects of the system including message frame encapsulation, bit time and voltage are verified, indicating the system is reliable in principle.

There are several potential further developments for this system, among which improving it into multi-slave message broadcast mode is the most interesting since such a mode is more flexible and closer to realistic needs.

Keywords: protocol conversion, EtherCAT, CAN FD



# Acknowledgements

We would like to express our most sincere gratefulness to our examiner Per Larsson-Edefors and supervisor Roger Johannsson for their patience and instructions during the thesis work.

We would also like to thank the TRIP team from Volvo Group Technology, especially Richard Prestedge and Tobias Falkman, for providing us with meticulous care, useful helps and valuable advice so that the thesis can progress smoothly and successfully.

Yang Zhang Jiaxing Zhu, Gothenburg, June 2020



# Contents

<b>Abbreviations</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem description . . . . .	1
1.2 Related work . . . . .	2
1.3 Thesis outline . . . . .	2
<b>2 Technical Background</b>	<b>5</b>
2.1 Printed Circuit Board . . . . .	5
2.2 SPI . . . . .	5
2.3 EtherCAT . . . . .	6
2.4 CAN . . . . .	6
2.5 CAN FD . . . . .	7
2.5.1 CAN FD Frame Format . . . . .	7
2.5.2 Bit Time . . . . .	8
2.5.3 Message Transmission . . . . .	9
2.5.4 Message Filtering . . . . .	10
2.5.5 Message Reception . . . . .	11
<b>3 System Design</b>	<b>13</b>
3.1 System Architecture . . . . .	13
3.1.1 EtherCAT . . . . .	14
3.1.2 SPI . . . . .	14
3.2 Design Decision . . . . .	14
3.3 Hardware . . . . .	15
3.3.1 Microcontroller . . . . .	15
3.3.2 Automotive Networking Board . . . . .	16
3.3.3 EtherCAT Interface Board . . . . .	16
3.3.4 CAN FD Controller Board . . . . .	17
3.4 Test Setup . . . . .	17
3.4.1 Bit Time Measurement. . . . .	17
3.4.2 Voltage Measurement for CAN FD States . . . . .	18
3.4.3 Message Frame Encapsulation . . . . .	19
<b>4 Results</b>	<b>21</b>
4.1 Hardware Design . . . . .	21
4.2 Implementation Results . . . . .	22

4.3	Test Results . . . . .	26
4.3.1	Bit Time Measurement . . . . .	26
4.3.2	Voltage Measurement . . . . .	26
4.3.3	Message Frame Encapsulation Measurement . . . . .	27
<b>5</b>	<b>Discussion and Potential Improvements</b>	<b>29</b>
5.1	Discussion . . . . .	29
5.1.1	General Performance . . . . .	29
5.1.2	Unstable Transmission . . . . .	29
5.2	Potential Improvements . . . . .	31
<b>6</b>	<b>Conclusion</b>	<b>33</b>
	<b>Bibliography</b>	<b>35</b>
<b>A</b>	<b>Appendix</b>	<b>I</b>

# Abbreviations

<b>BRS</b>	Bit Rate Switch
<b>CAN Bus</b>	Controller Area Network
<b>CAN FD</b>	Controller Area Network Flexible Data-Rate
<b>CRC</b>	Cyclic Redundancy Check
<b>DBRP</b>	Data Bit Rate Prescaler
<b>DBR</b>	Data Bit Rate
<b>DBT</b>	Data Bit Time
<b>DLC</b>	Data length code
<b>DTQ</b>	Data Time Quanta
<b>EBS</b>	Electronic Braking System
<b>ECU</b>	Electronic Control Unit
<b>EDA</b>	Electronic Design Automation
<b>EDL</b>	Extended Data Length Bit
<b>EMI</b>	Electromagnetic Interference
<b>EOF</b>	End of Frame
<b>ESC</b>	EtherCAT Slave Controller
<b>ESI</b>	Error Status Indicator Bit
<b>EtherCAT</b>	Ethernet for Control Automation Technology
<b>HIL</b>	Hardware-In-the-Loop
<b>I<sup>2</sup>C</b>	Inter-Integrated Circuit
<b>IDE</b>	Identifier Extension Bit
<b>LIN</b>	Local Interconnect Network
<b>MISO</b>	Master-In Slave-Out
<b>MOSI</b>	Master-Out Slave-In
<b>NBRP</b>	Nominal Bit Rate Prescaler
<b>NBR</b>	Nominal Bit Rate
<b>NBT</b>	Nominal Bit Time
<b>NTQ</b>	Nominal Time Quanta
<b>PCB</b>	Printed Circuit Board
<b>PHSEG1</b>	Phase Segment 1
<b>PHSEG2</b>	Phase Segment 2
<b>PIM</b>	Plug-In Module
<b>PRSEG</b>	Propagation Segment
<b>RTR</b>	Remote Transmission Bit
<b>RXMAB</b>	Receive Message Assembly Buffer
<b>SID</b>	Standard Identifier
<b>SOF</b>	Start of Frame

<b>SPI</b>	Serial Peripheral Interface
<b>SSC</b>	Slave Stack Code
<b>SYNC</b>	Synchronization Segment
<b>TQ</b>	Time Quanta
<b>TXQ</b>	Transmit Queue



# 1

## Introduction

### 1.1 Problem description

A communication protocol is a system of rules that allow two or more entities of a communications system to transmit information via any kind of variation of a physical quantity. Serial peripheral interface (SPI), the controller area network (CAN), Ethernet for control automation technology (EtherCAT) and controller area network flexible data-rate (CAN FD) are widely used for communication. CAN bus is developed by BOSCH [1] as a multi-master, message broadcast system that specifies a maximum bandwidth of 1 Mbit/s [2]. Due to its advantages such as low cost, robustness and flexibility, CAN bus is widely used in many applications like cars, trucks, tractors and industrial robots. Due to the fact that a rise in vehicle functionality is driving an explosion in data and networks are increasingly limited by the 1 Mbit/s bandwidth, CAN FD [3] emerges as an alternative. It has two main advantages over standard CAN without changing the physical layers, a faster bandwidth up to 8 Mbit/s and a variable data payload of up to 64 bytes. Inevitably next generation electronic control units (ECUs) are using CAN FD for vehicle network communication. EtherCAT [4] is a real-time industrial Ethernet technology originally developed by Beckhoff Automation. The EtherCAT protocol is suitable for hard and soft real-time requirements in automation technology, in testing and measurement and many other applications. EtherCAT has advantages such as short cycle times, low jitter for accurate synchronization and low hardware costs.

Today the Hardware-In-the-Loop (HIL) test rigs at Volvo Group Trucks Technology use a commercially available device for a gateway or fieldbus conversion between EtherCAT and CAN. The HIL rig simulation models use this device to interface with the real truck Electronic Braking System (EBS) and Active Safety systems. However the HIL test rigs will be required to implement simulations via EtherCAT to CAN FD and at present there is no commercially available solution.

In order to solve this problem, the goals of our thesis are as follows:

1. Redesign a merged PCB, merging PIC32 minimum system board and EtherCAT slave board from Microchip, for EtherCAT and CAN FD communication interface.
2. Implement protocol conversion between EtherCAT and CAN FD with the help of external CAN FD controller. In this case, it would be CAN FD controller board.

3. Verify whether the whole redesigned prototype works as the expected requirements. That is both the signals transmitted between CAN FD and EtherCAT should be received correctly via ECU. Testing would be performed with the designed PCB acting as a protocol conversion tool. A signal database is used for sending random messages through EtherCAT interface and messages received from CAN FD interface would be displayed with Vector CANoe [5]. Several preliminary testing criteria could be CAN FD recessive/dominant bit time measurement with ambient temperature between 18°C and 28°C following REQ-CAN-28 v3 [6] and verifying whether message frame encapsulation, such as the extended data length (EDL) bit, identifier extension bit (IDE) bit and bit rate switch (BRS) bit, meets REQ-CAN-04 v1 [6].

Some of the research challenges that have to be addressed are:

1. Find the cause of the extra empty data when receiving SPI data on ECU and then figure out a proper method to solve it.
2. Set up an appropriate testing architecture and testing method for verifying the function of our prototype.

## 1.2 Related work

In recent years, a significant amount of papers describe advantages of CAN FD. In [7], the authors concludes CAN FD has faster object pool transference, lower bus load usage, shorter worst case response time, and lower jitter. However, CAN FD has some potential problems such as the same security problems as standard CAN. In [8], the authors propose a practical security architecture for in-vehicle CAN FD communication which is tested by using three microcontrollers and CANoe software. Furthermore, as shown in [9], the authors provide a formula to calculate transmission time in CAN FD frames. Their effort brings a framework to solve the optimisation problem in pseudo-polynomial time. Based on these works, from theoretical design to practical optimization and testing, CAN FD can be the next CAN technology used in automation industries [10].

However, there are no commercial solutions which are compatible with both EtherCAT and CAN FD. The existing products only work between EtherCAT and standard CAN. The typical products are EtherCAT-CAN Gateway from Esd Electronics [11], and Anybus Communicator CAN-EtherCAT [12]. The maximum CAN data-rate of these products is limited to 1 Mbit/s.

## 1.3 Thesis outline

Chapter 2 explains the basic theory of CAN FD, including frame format, message transmission and reception mechanism.

Chapter 3 presents the thesis step by step. It is divided into three parts: system architecture overview, hardware used in thesis and testing methodology.

Chapter 4 lists the current implementation results of the thesis work.

Chapter 5 discusses important results in result section, and gives some potential improvements for future work.

Chapter 6 lists the final conclusion for the thesis project.



# 2

## Technical Background

This chapter lays a technical foundation for the thesis, including printed circuit board (PCB), SPI, EtherCAT, CAN and CAN FD.

### 2.1 Printed Circuit Board

A PCB provides robust mechanical support and electrical connection by using conductive wires, holes, copper planes and other non-conductive materials. PCBs have played an important role and have been widely used, especially in embedded electronics systems [13]. According to amount of layers, PCBs can be classified as single-side boards, double-side boards and multi-layer boards. Components can only be placed on the same side of single-side boards. For double-side boards, components are allowed to be placed on both sides of boards. The multi-layer boards are suitable for higher density design. It provides extra inner signal layers, which allow routing and internal planes for power and ground connection.

Generally, the PCB design starts from a schematic design. According to project requirements, proper circuits and components should be decided. The PCB parameters, including size of PCB, layer stack configuration and design rules should be decided. After placing components and routing, manufacture files including gerber files and NC drill files can be sent to the manufacturer.

The Altium company provides an electronic design automation (EDA) tool called Altium Designer. This tool can support schematic design, PCB design and manufacture output [14]. In this project, Altium Designer 20 is used.

### 2.2 SPI

SPI [15] is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. Due to its multiple advantages, such as simple hardware, higher throughput than Inter-Integrated Circuit (I<sup>2</sup>C) and simple software implementation, SPI is one of the most widely used interfaces between microcontroller and peripheral ICs. SPI devices communicate in full duplex within a master-slave architecture. The master uses SCLK line to synchronize with slave devices before transmitting messages using Master-Out Slave-In (MOSI) line and receiving messages using Master-In Slave-Out (MISO) line. Similarly, slave devices receive messages on MOSI line and output messages on MISO

line. The SS line can be used to select which slave device to carry out the work if there are several slaves in a system.

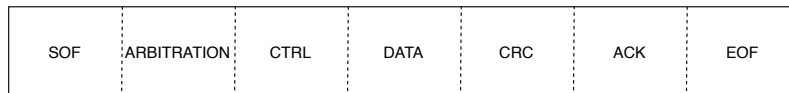
### 2.3 EtherCAT

EtherCAT enjoys a unique principle called "processing on the fly", which means as a single message issued by the EtherCAT master walks through each node in the network, each node reads its input and adds its output to the message. The EtherCAT message continues going to the next node while the former node processes the input. This unique principle enables an EtherCAT network to achieve maximum bandwidth utilization.

### 2.4 CAN

A CAN bus allows devices to communicate with each other without a host computer. The highest Nominal Bit Rate (NBR) of CAN bus is up to 1 Mbit/s, and the payload is from 0 byte to 8 byte. CAN message is transmitted only with NBR.

For frame types, a CAN frame includes data frame, remote frame, error frame and overload frame. Detailed data frame is depicted in Fig. 2.1, composed of 7 different bit fields, namely Start of Frame (SOF), Arbitration Field, Control Field, Data Field, Cyclic Redundancy Check (CRC) Field, Acknowledge (ACK) Field and End of Frame (EOF). The CAN bus also supports two frame formats which are base format and extended format. The only difference between these two types of formats is the length of the identifier.



**Figure 2.1:** CAN data frame

Data length code (DLC) in Control Field specifies the number of data bytes to be transmitted, as illustrated in Table 2.1.

DLC	Number of Data Bytes
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

**Table 2.1:** CAN DLC Encoding

## 2.5 CAN FD

CAN FD is an update of the original CAN bus, to increase capability of transmitting without changing physical layers. The maximum bit rate is up to 8 Mbit/s, and CAN FD can transmit 64 byte data at most in a single frame rather than 8 byte data at most in CAN. Another improvement is that CAN FD supports dynamically switching to different data-rate and message size.

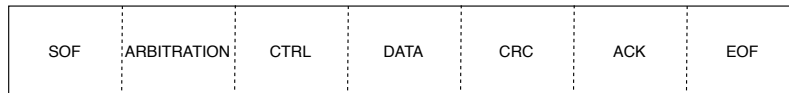
### 2.5.1 CAN FD Frame Format

In our thesis work, CAN FD base frame is used. Detailed frame format is depicted in Fig. 2.2, composed of 7 different bit fields, SOF, Arbitration Field, Control Field, Data Field, CRC Field, ACK Field EOF.

Standard Identifier (SID) in Arbitration Field is used for message priority comparison when there are a couple of nodes competing for the bus. It can also be used to enable filters to filter messages during message reception.

Bit rate switch (BRS) in Control Field decides whether the whole frame will be transmitted using Nominal Bit Rate (NBR) only or using both NBR and Data Bit Rate (DBR).

DLC in Control Field specifies the number of data bytes to be transmitted, as illustrated in Table 2.2.

**Figure 2.2:** CAN FD base frame

DLC	Number of Data Bytes
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	12
10	16
11	20
12	24
13	32
14	48
15	64

**Table 2.2:** CAN FD DLC Encoding

### 2.5.2 Bit Time

As mentioned before CAN frame is only transmitted using NBR. Two bit rates, NBR and DBR, are used for transmitting bits in a CAN FD frame so that higher bandwidth is achieved. NBR is the number of bits per second, which is used from the arbitration field to the sample point of BRS, and from the sample point of the CRC delimiter to EOF. It is the inverse of the Nominal Bit Time (NBT). Similarly DBR is the number of bits per second, which is used during the data phase and CRC field. It is the inverse of Data Bit Time (DBT).

Nominal Bit Rate Prescaler (NBRP) and Data Bit Rate Prescaler (DBRP) are used for calculating Nominal Time Quanta (NTQ) and Data Time Quanta (DTQ) respectively (see equation 2.1 and equation 2.2 ). Time quanta is the basic unit of bit time and there can be multiple time quanta in a bit time.

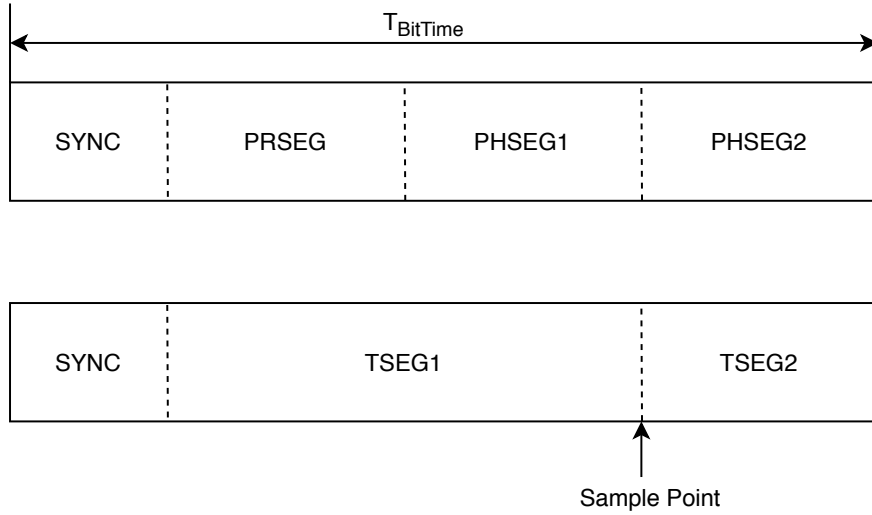
$$NTQ = \frac{NBRP}{F_{sysclk}} \quad (2.1)$$

$$DTQ = \frac{DBRP}{F_{sysclk}} \quad (2.2)$$

As specified in ISO11898-1:2015 there are four segments in CAN bit time (see Fig. 2.3), namely Synchronization Segment (SYNC), Propagation Segment (PRSEG), Phase Segment 1 (PHSEG1) and Phase Segment 2 (PHSEG2). In the Bit Time registers, PRSEG and PHSEG1 are combined to create TSEG1. PHSEG2 is called



TSEG2. Each segment has multiple Time Quanta (TQ). The sample point lies between TSEG1 and TSEG2.



**Figure 2.3:** Partition of Bit Time [16]

The number of time quanta in a bit time is calculated using equation 2.3 and equation 2.4.

$$\frac{NBT}{NTQ} = NSYNC + NTSEG1 + NTSEG2 \quad (2.3)$$

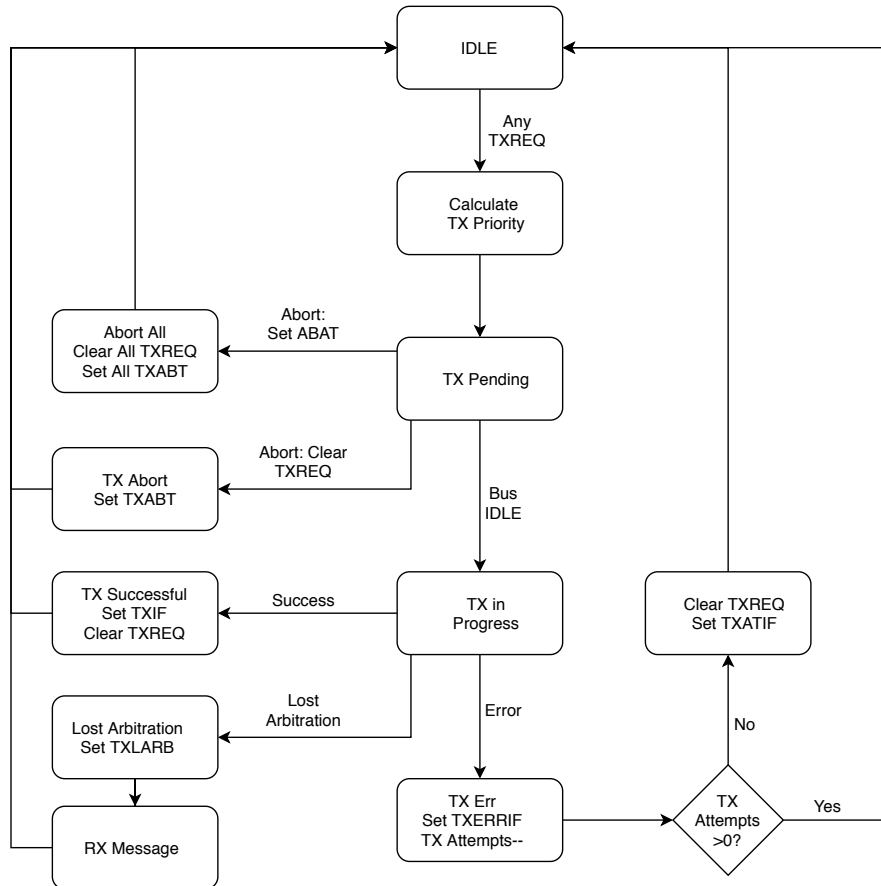
$$\frac{DBT}{DTQ} = DSYNC + DTSEG1 + DTSEG2 \quad (2.4)$$

### 2.5.3 Message Transmission

Fig. 2.4 demonstrates that CAN and CAN FD messages queued for transmission follow the following rules:

1. Whenever a message is loaded into a FIFO or Transmit Queue(TXQ) and is ready for transmission, the TXREQ bit is set.
2. Determine the transmit priority. The respective priority of FIFOs and TXQ will be compared and the transmit order for messages inside a FIFO or TXQ is calculated as well.
3. The highest priority message can not start transmission, also known as pending for transmission, until the bus is idle.
4. A pending message can only be aborted before SOF is transmitted.
5. During the course of message transmission, the CAN FD Protocol Module checks if there are loss of arbitration or transmit errors.
6. The TXREQ will only be cleared after all messages inside the FIFO are transmitted.

7. In case of arbitration loss, transmission will be aborted and the device will switch to message reception.
8. In case of transmission error, an error frame will still be transmitted and the corresponding error flags of FIFO or TXQ will be set. Retransmission attempts will be performed.



**Figure 2.4:** Transmit state diagram [16]

### 2.5.4 Message Filtering

Since all messages on a CAN network will be received by all nodes, message filtering should be implemented so that only messages of interest for a particular node will be processed. There are 32 filters in CAN FD module and each filter can be configured to only receive messages with specific SID and message data. A simple filtering procedure is as follows:

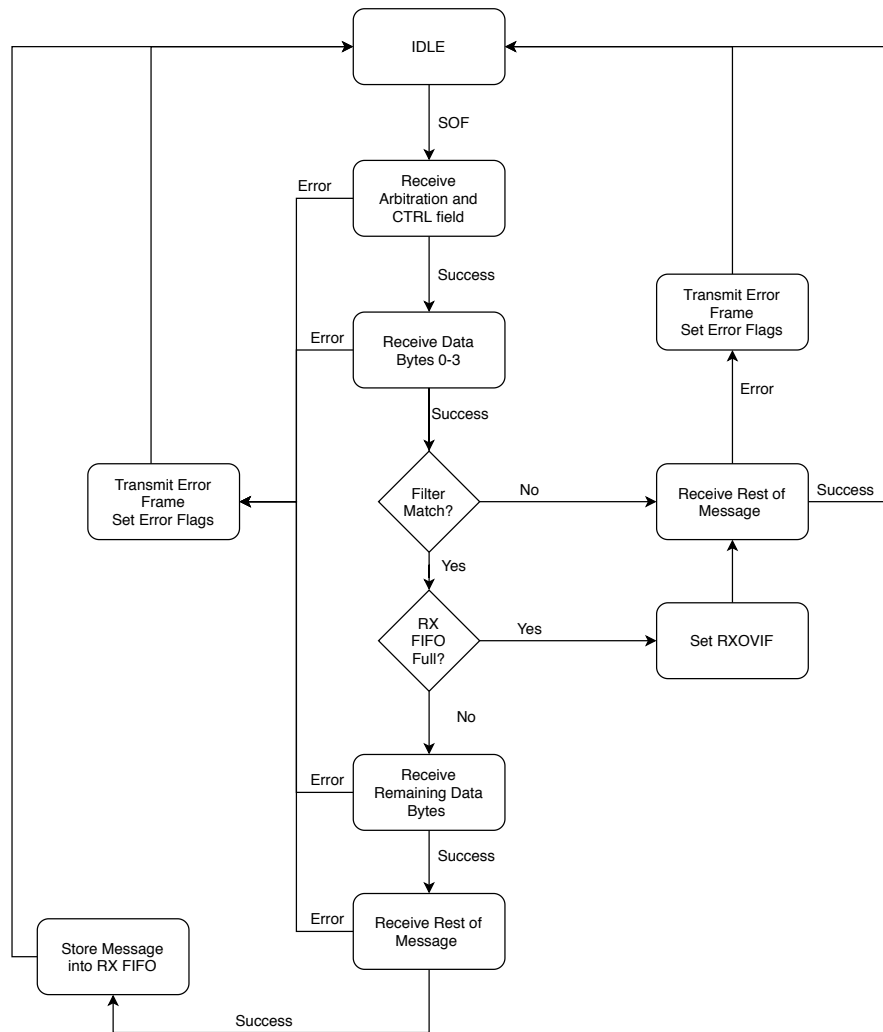
1. After arbitration field and the first three data bytes are received, CAN FD protocol module commences message filtering, starting with Filter 0.
2. Messages received in Receive Message Assembly Buffer(RXMAB) are compared to the filter and mask. If matched and received with no errors, the messages will be stored into the corresponding RX FIFO.
3. The module will loop through all filters. If none of the filters match, the received message will be discarded.

4. Though the message is matched with filter, it will not be stored into RX FIFO if FIFO is full and RXOVIF will be set.

### 2.5.5 Message Reception

Fig. 2.5 illustrates how CAN and CAN FD messages are received, following the steps below:

1. Once a SOF is detected, the CAN FD Protocol Module will continue receiving arbitration field and control field.
2. Message filtering starts.
3. If a filter matches and the corresponding RX FIFO is not full, the rest of data bytes will be received and stored into FIFO. The FIFO status flag will be updated as well.
4. If a filter matches and the corresponding RX FIFO is full, the RXOVIF bit will be set. The rest of the messages will still be received but will not be stored.
5. If none of the filters match, the messages will be received without storing.
6. In case an error is detected during the reception of a message, an error frame will be transmitted and the appropriate error flags will be set.



**Figure 2.5:** Receive state diagram [16].

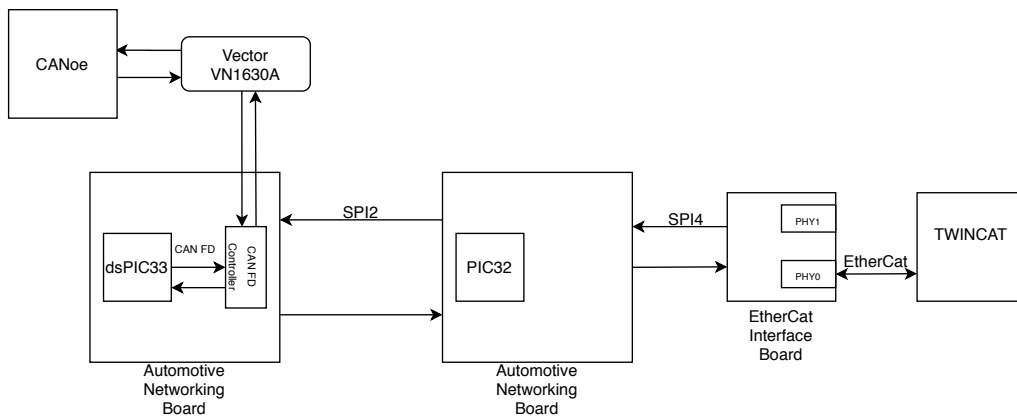
# 3

## System Design

### 3.1 System Architecture

An overview of the current system is shown in Fig. 3.1. The current implementation consists of the following modules: TWINCAT [17], EtherCAT interface board [18][19][20], one automotive networking board [21] with PIC32MX795F512L [22], another automotive networking board with dsPIC33CK256MP508 [23] and CAN FD controller board [24], Vector VN1630A [25] and CANoe. The function of each module is briefly described as follows:

1. TWINCAT is used to define the messages types that should be transmitted and received, and displayed the content of the messages.
2. EtherCAT interface board facilitates messages transmission between TWINCAT and PIC32.
3. function of PIC32 is a middle message transfer station.
4. dsPIC33 encapsulates messages into CAN FD frame format and unpacks CAN FD messages. CAN FD controller is a transceiver for CAN FD messages.
5. Vector VN1630A is a bus transceiver. It can be considered as a communication bridge between CAN FD controller and CANoe.
6. CANoe is a comprehensive tool for ECU network analysis. In our thesis, it is used to display and define the content CAN FD messages.



**Figure 3.1:** System Overview.

#### 3.1.1 EtherCAT

The EtherCAT messages to be sent are configured using Slave Stack Code (SSC) tool, with the specific excel file defining the interface nodes of EtherCAT messages, as shown in Fig. 3.2. After appropriate settings, SSC tool generates device description files and corresponding slave stack code, in which the source codes are modified accordingly which copies the application data to/from the EtherCAT Slave Controller (ESC) memory to the local application memory.

TWINCAT is then used to fill EtherCAT messages with data and visualize the content of received data, which facilitates verifying whether the prototype works as intended or not.

ObjectCode	SI	DataType	Name	Default	Min	Max
Input Data of the Module (0x6000 - 0x6FFF)						
RECORD						
		1 UINT	CanFdMsgIn	0	0	0xffff
		2 UINT	DLCH	0	0	0xffff
		3 UINT	DLCL	0	0	0xffff
		4 UINT	SIDH	0	0	0xffff
		4 UINT	SIDL	0	0	0xffff
		5 ARRAY [0..63] OF SINT	MSG	0	0	0xff

**Figure 3.2:** Example of EtherCAT Message.

#### 3.1.2 SPI

Necessary configurations have to be set before SPI channel 2 of PIC32 can function correctly. Once it is set, all of the EtherCAT messages (Fig. 3.2) are stored in a transmit buffer before they are sent. Similarly a receive buffer is used for keeping the received data. The SPI of PIC32 allows writing and reading SPI2BUF register simultaneously, meaning that transmitting data from the transmit buffer and storing received data into the receive buffer are feasible.

### 3.2 Design Decision

#### The number of CAN FD message

As shown in Fig. 3.2, the EtherCAT messages are equivalent to one CAN FD message when they are encapsulated into CAN FD frame format. The reason why only one CAN FD message is defined in TWINCAT is that if multiple equivalent CAN FD messages are transmitted from TWINCAT, the EtherCAT interface board can not function properly due to limited memory. In order to simplify the thesis work and increase our chances of successfully implementing a prototype, it is decided to define the EtherCat messages shown in Fig. 3.2.

#### Choosing suitable baud rate for CAN FD

Since the SPI bus speed is set to 2 MBd/s in Volvo's previous TWINCAT application, the bus speed for CAN FD should not be too fast. Due to the way the code is designed, the CAN FD module in dsPIC33 can not store all of the CAN FD messages from CANoe if CAN FD baud rate is much faster than that of SPI. So in

our project, the SPI baud rate is set to 2Mbit/s and the NBR and DBR are set to 500kbit/s and 4Mbit/s respectively.

### 3.3 Hardware

#### 3.3.1 Microcontroller

The PIC32MX795F512L and dsPIC33CK256MP508 microcontrollers are used in the project. They are all designed by Microchip. The main frequency of PIC32 (see Fig. 3.3) is up to 80 MHz. PIC32 supports different communication interfaces, including up to four 4-wire SPI whose clock is up to 25 MHz, one CAN module, I<sup>2</sup>C module and UART module.

The dsPIC33 (see Fig. 3.4) is a 16-bit microcontroller, and its main frequency is up to 100 MHz. The microcontroller has three 4-wire SPI interface and one CAN FD module.



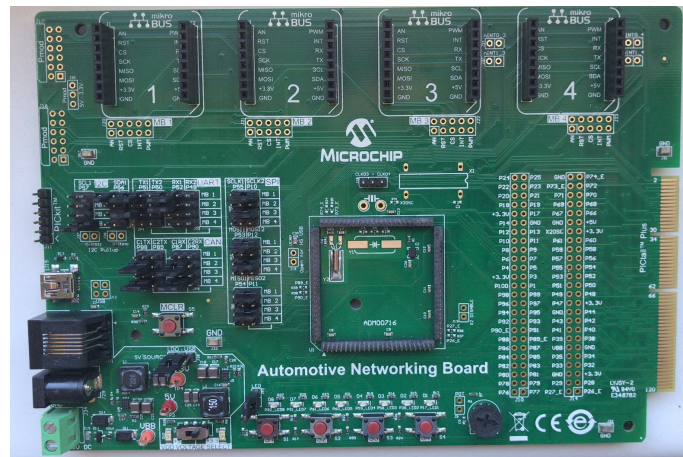
**Figure 3.3:** Microchip PIC32MX795F512L Plug-in Module.



**Figure 3.4:** Microchip dsPIC33CK256MP508 Plug-in Module.

#### 3.3.2 Automotive Networking Board

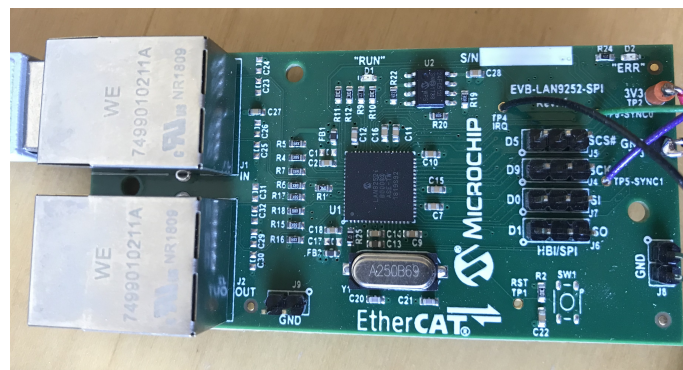
The Microchip automotive networking board (see Fig. 3.5) is a modular development system which mainly includes a 100 pin Plug-In Module (PIM) connector and four mikroBUS<sup>TM</sup> sockets. The PIM connector is compatible with Microchip's 8-bit, 16-bit and 32-bit microcontrollers, and four mikroBUS<sup>TM</sup> sockets [26] support CAN communication and Local Interconnect Network (LIN) communication. Therefore, the development board is very general by swapping different microcontrollers' PIM and adding various MikroElektronika Click<sup>TM</sup> add-on boards which is compatible with mikroBUS<sup>TM</sup> sockets.



**Figure 3.5:** Microchip Automotive Networking Board.

#### 3.3.3 EtherCAT Interface Board

The EtherCAT interface board (see Fig. 3.6) whose part number is EVB-LAN9252-SPI in Microchip company. The core is LAN9252 EtherCAT slave controller chip from Microchip [27]. The board includes dual integrated high-performance 100Mbps Ethernet PHYs.

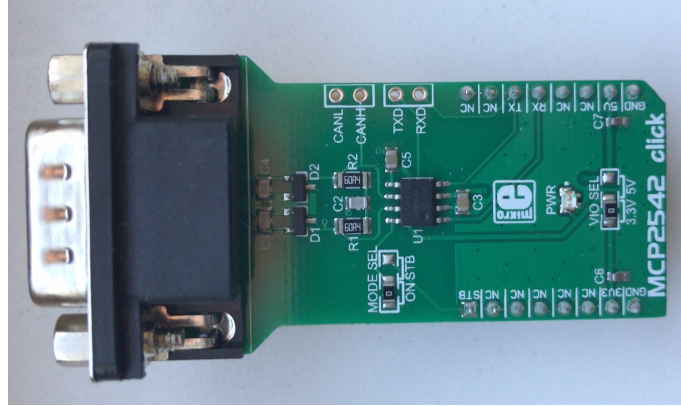


**Figure 3.6:** Microchip EtherCAT Slave Controller Board.



### 3.3.4 CAN FD Controller Board

The CAN FD controller board (see Fig. 3.7) is compatible with mikroBUS<sup>TM</sup> socket on automotive networking board, and it uses MCP2542FD designed by Microchip, a CAN FD transceiver supporting both CAN and CAN FD.



**Figure 3.7:** Microchip CAN FD Module.

## 3.4 Test Setup

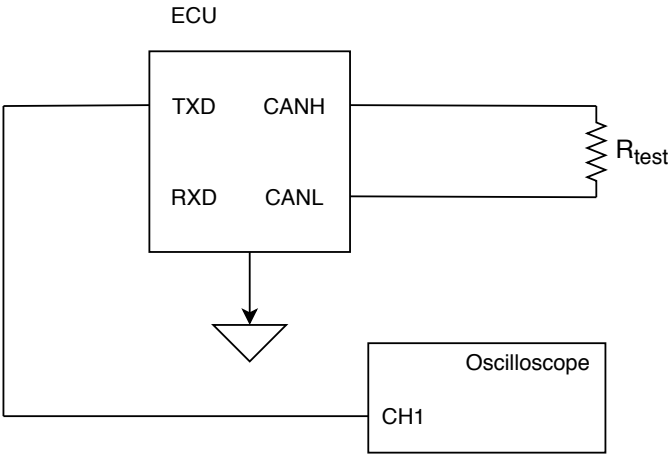
The test is based on the ECU CAN interface test specification from Volvo [6]. The test includes bit time measurement, voltage measurement for CAN FD and message frame encapsulation test.

### 3.4.1 Bit Time Measurement.

The bit time measurement configuration is shown in Fig. 3.8. A termination resistor ( $R_{test}$ ) should be connected between CANH and CANL outputs. The distance between ECU and  $R_{test}$  should be 1 meter. If there is a termination resistor inside the ECU, the value of  $R_{test}$  should be 120  $\Omega$ , otherwise a 60  $\Omega$  resistor should be used.

The specific requirements of testing are:

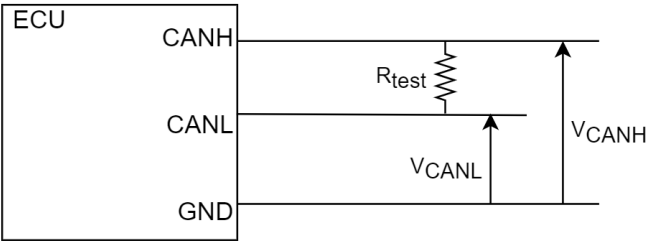
1. The measurement should use a very specific sequence. For recessive bit time measurement, the recessive bit occurs after five consecutive dominant bits. Similarly, the dominant bit occurs after five consecutive recessive bits for dominant bit time measurement.
2. Both arbitration phase measurement and data phase measurement should be performed at least twice.
3. According to Volvo requirement, tolerance of arbitration phase bit time and data phase bit time at TXD output shall not exceed  $\pm 0.4\%$ , including clock skew, jitter and asymmetry.



**Figure 3.8:** CAN FD Bit Time Measurement.

### 3.4.2 Voltage Measurement for CAN FD States

The voltage measurement configuration for CAN FD is shown in Fig. 3.9. A  $60\ \Omega$  resistor is used for ECU without a termination resistor inside, otherwise  $R_{test}$  should be  $120\ \Omega$ . The voltage for CAN FD is measured at ambient temperature between  $18^{\circ}\text{C}$  and  $28^{\circ}\text{C}$ . The Volvo requirement for CAN FD recessive state and dominant state is shown in Fig. 3.10 and Fig. 3.11 separately.



**Figure 3.9:** Voltage Measurement for CAN FD Dominant State and Recessive State.

		Volvo Requirement		
		Min	Normal	Max
Output bus voltage Normal mode	$V_{CANH}$	2.0 V	2.5 V	3.0 V
	$V_{CANL}$			
Differential output bus voltage	$V_{CANH}-V_{CANL}$	-500 mV	0 V	50 mV

**Figure 3.10:** Voltage Requirement for CAN FD Recessive State.

		Volvo Requirement		
		Min	Normal	Max
Output bus voltage Normal mode	$V_{CANH}$	2.75 V	3.5 V	4.5 V
	$V_{CANL}$	0.5 V	1.5 V	2.25 V
Differential output bus voltage	$V_{CANH} - V_{CANL}$	1.5 V	2.0 V	3.0 V

**Figure 3.11:** Voltage Requirement for CAN FD Dominant State.

### 3.4.3 Message Frame Encapsulation

It is necessary to check CAN FD frame bit by bit to make sure that the transmission is successful. According to the Volvo test specification, a CAN FD logging tool and an oscilloscope with CAN FD decoder are used.

The arbitration field should be checked as follows:

1. In arbitration field in CAN FD, the remote transmission bit (RTR) which is used in CAN is removed.
2. The identifier extension bit (IDE) should be set to dominant in standard mode.
3. The r0 and r1 bit should be set to dominant.
4. The EDL bit, which is also called FD format (FDF) bit, should be set to recessive for CAN FD.
5. The BRS and error status indicator bit (ESI).



# 4

## Results

### 4.1 Hardware Design

The new EtherCAT slave controller board includes LAN9252 and PIC32 to transfer data via Ethernet and SPI protocol. Top schematic (see Fig. A.1) includes four sub-sheets that is EtherCAT\_IO, EtherCAT\_controller, PIC32\_MCU and power. In EtherCAT\_IO sub-sheet (see Fig. A.2), two RJ45 connectors are configured, and they are connected with LAN9252 via eight data transfer pins. The SCL pin and SDA pin are used as I<sup>2</sup>C protocol to interface with 24FC512-I/SM that is a EEPROM chip to store data. In EtherCAT\_controller sheet (see Fig. A.3), LAN9252 is configured [19]. The SPI4 of the chip is used to interface with PIC32 microcontroller. Similarly, PIC32\_MCU sheet (see Fig. A.4) includes configuration circuits of microcontroller. The power sub-sheet (see Fig. A.5) includes circuits to filter 3V and 5v from external power supplier.

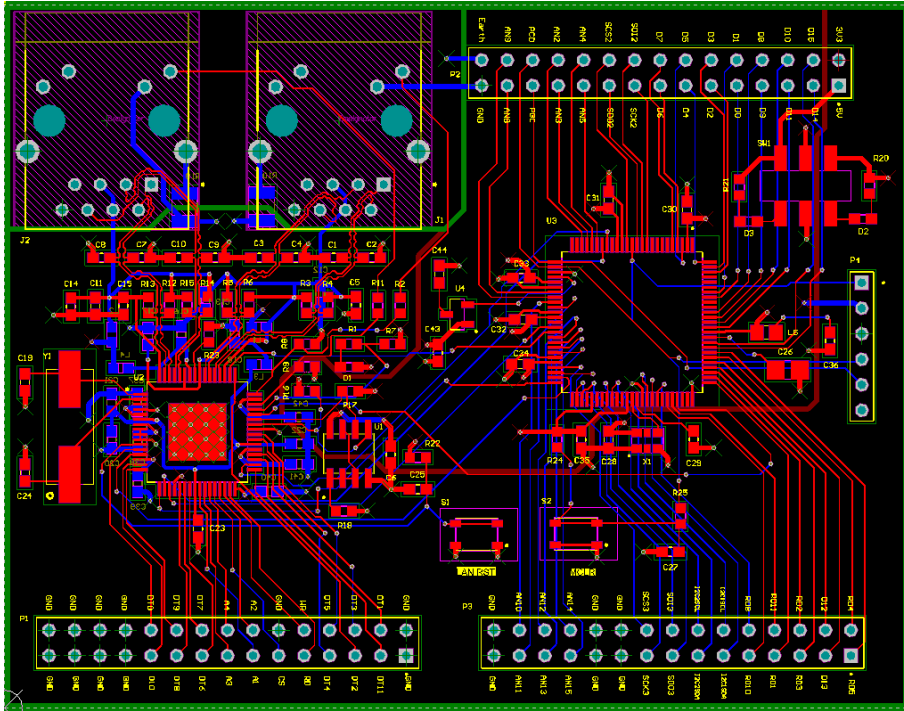


Figure 4.1: PCB design of EtherCAT slave Controller Board.

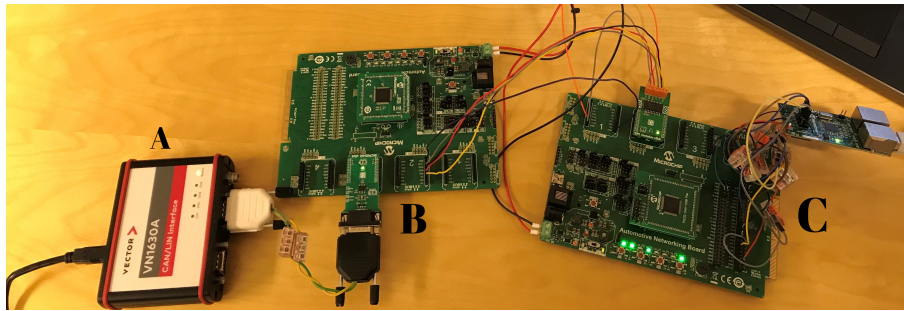
For PCB design as shown in Fig. 4.1, the EtherCAT slave Controller Board is a 4-layer board including 2 signal layers, one power plane and one ground plane. The ground plane is split as *earth* area that is connected with the shell of RJ45 and *GND* area which is connected with other ground pins. The power plane is divided into 3.3V area, 5V area and VDD area. The VDD can be connected with either 3.3V or 5V by toggling a switch.

The board size is 90 mm  $\times$  70 mm, and the basic rule to decide board size is to find a balance between smaller size and less layers. The board uses three 30-pin standard 2.54 mm connectors to interface with a motherboard designed by Volvo Truck Corporation. Almost all pins of LAN9252 and PIC32 microcontroller, except for power pins and ground pins, are kept in these three connectors. Therefore, the new slave controller board has better compatibility, and it can be used for further development.

In order to get better Electromagnetic Interference (EMI) performance, decoupling capacitors are used for the power pins of LAN9252 and PIC32 microcontroller[19][28]. The data transfer wires between RJ45 and LAN9252 are set as differential pairs to equalize their length. Regarding different pins, three special net classes are set. Main\_power class includes 3.3V and 5V from external supplier, and VDD pins. Sub\_power class contains power pins of chips. The *earth* and *GND* are included in Ground class. Other pins are set as normal pins. Then, in order to have better electrical performance, different trace width is matched for different classes. The trace width for main\_power and Ground class is 20 mil. Sub\_power class uses trace with 15 mil, and other traces are 6 mil.

## 4.2 Implementation Results

The final prototype is presented in Fig. 4.2 and it is the physical realization of system architecture displayed in Fig. 3.1. According to the plan, the C part in Fig. 4.2 which includes PIC32 module, automotive networking board and EtherCAT slave controller board should be replaced by the new designed EtherCAT slave controller board (see Fig. 4.1), and the B part in Fig. 4.2 should be replaced by a motherboard designed by Volvo. However, due to the corona virus and the layoff of Volvo, the new designed board could not be manufactured on time and Volvo could not finish the motherboard design on time either. Therefore, old boards are used in the prototype test.



**Figure 4.2:** Prototype under test. A is the Vector VN1630A tool which is used with CANoe software. B is Microchip automotive networking board with CAN FD controller and dsPIC33. C is the microchip automotive networking board with PIC32 and EtherCAT slave controller.

In order to test function of the prototype, manual testing is chosen which means that testing cases are changed manually. Automatic testing based on database should be a better choice. However, in database provided from Volvo, the most of testing cases are 0. It is not obvious to verify whether the transmission works or not.

In general the prototype is able to perform message transmission from TWINCAT to CANoe correctly. As shown in Fig. 4.3, the transmission between TWINCAT and CANoe works without any error. The DLC is set to 15 which means that the data payload is 64. In Fig. 4.3 (a), a message is configured in TWINCAT. The MAIN.iIN $x$  and MAIN.iOUT $x$  is corresponding to the position of data byte in CAN FD data frame, where  $x$  is a number from 1 to 64. For instance, MAIN.iIN6 and MAIN.iOUT6 is connected with the sixth data byte in CAN FD data frame. Therefore, the first data byte in CAN FD data field is set to 0x01. The other positions are similar. CANoe receives this message (see Fig. 4.3 (b)). The first data byte is 0x01, and other data match with the configuration in TWINCAT. In an opposite direction, a message is configured in CANoe (see Fig. 4.3 (c)), the first data byte in data field is set to 0x1c, and other positions are shown in the figure. Fig. 4.3 (d) shows the received message in TWINCAT, the MAIN.iIN1 is 0x1c which matches with the first data byte in CANoe configuration. Other positions match with configuration as well. Similarly, when the DLC is configured to 10 which means that the payload is 16, the transmission works properly as well (see Fig. 4.4). The message that CANoe receives (see Fig. 4.4 (b)) matches with the message configured in TWINCAT (see Fig. 4.4 (a)). In an opposite direction, TWINCAT receives the exactly same message (see Fig. 4.4 (d)) that configured in CANoe (see Fig. 4.4 (c)). It shows that the prototype can transmit messages correctly under different DLC setting.

Minor problems, for example some messages received in TWINCAT cannot be stable, will be discussed in next Chapter.

## 4. Results

MAIN.iOUT1	X	0x01	SINT	1.0	385044.0	MSG[0] . MSG . CanFdM
MAIN.iOUT10	X	0x10	SINT	1.0	385053.0	MSG[9] . MSG . CanFdM
MAIN.iOUT11	X	0x11	SINT	1.0	385054.0	MSG[10] . MSG . CanFc
MAIN.iOUT12	X	0x12	SINT	1.0	385055.0	MSG[11] . MSG . CanFc
MAIN.iOUT13	X	0x13	SINT	1.0	385056.0	MSG[12] . MSG . CanFc
MAIN.iOUT14	X	0x14	SINT	1.0	385057.0	MSG[13] . MSG . CanFc
MAIN.iOUT15	X	0x15	SINT	1.0	385058.0	MSG[14] . MSG . CanFc
MAIN.iOUT16	X	0x16	SINT	1.0	385059.0	MSG[15] . MSG . CanFc
MAIN.iOUT17	X	0x17	SINT	1.0	385060.0	MSG[16] . MSG . CanFc
MAIN.iOUT18	X	0x18	SINT	1.0	385061.0	MSG[17] . MSG . CanFc
MAIN.iOUT19	X	0x19	SINT	1.0	385062.0	MSG[18] . MSG . CanFc
MAIN.iOUT2	X	0x02	SINT	1.0	385045.0	MSG[1] . MSG . CanFdM

(a)

(b)

(c)

Name	[X]	Online	Type	Size	>Addre...	Linked to
MAIN.iIN1	X	0x1c	SINT	1.0	385116.0	MSG[0] . MSG . CanFdM
MAIN.iIN10	X	0x00	SINT	1.0	385125.0	MSG[9] . MSG . CanFdM
MAIN.iIN11	X	0x74	SINT	1.0	385126.0	MSG[10] . MSG . CanFc
MAIN.iIN12	X	0x01	SINT	1.0	385127.0	MSG[11] . MSG . CanFc
MAIN.iIN13	X	0x12	SINT	1.0	385128.0	MSG[12] . MSG . CanFc
MAIN.iIN14	X	0x1c	SINT	1.0	385129.0	MSG[13] . MSG . CanFc
MAIN.iIN15	X	0x32	SINT	1.0	385130.0	MSG[14] . MSG . CanFc
MAIN.iIN16	X	0x68	SINT	1.0	385131.0	MSG[15] . MSG . CanFc
MAIN.iIN17	X	0x52	SINT	1.0	385132.0	MSG[16] . MSG . CanFc
MAIN.iIN18	X	0x4a	SINT	1.0	385133.0	MSG[17] . MSG . CanFc
MAIN.iIN19	X	0x6c	SINT	1.0	385134.0	MSG[18] . MSG . CanFc
MAIN.iIN2	X	0x00	SINT	1.0	385117.0	MSG[1] . MSG . CanFdM
MAIN.iIN20	X	0x4d	SINT	1.0	385135.0	MSG[19] . MSG . CanFc

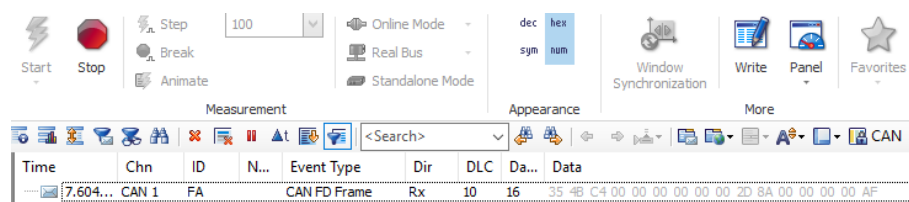
(d)

**Figure 4.3:** Transmission results when the data payload is 64. (a) shows messages sent from TWINCAT, and (b) shows that CANoe receives exactly the same data from TWINCAT. (c) shows CAN FD frame sent from CANoe, and (d) illustrates TWINCAT receives exactly the same data from CANoe.

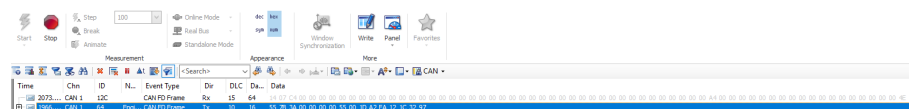


Name	[X]	Online	Type	Size	> Addr...	Linked to
MAIN.iOUT1	X	0x35	USINT	1.0	385044.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT10	X	0x2d	USINT	1.0	385053.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT11	X	0x8a	USINT	1.0	385054.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT12	X	0	USINT	1.0	385055.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT13	X	0	USINT	1.0	385056.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT14	X	0	USINT	1.0	385057.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT15	X	0	USINT	1.0	385058.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT16	X	0xaf	USINT	1.0	385059.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT17	X	0	USINT	1.0	385060.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT18	X	0	USINT	1.0	385061.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT19	X	0	USINT	1.0	385062.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT2	X	0x4b	USINT	1.0	385045.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT20	X	0	USINT	1.0	385063.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT21	X	0	USINT	1.0	385064.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT22	X	0	USINT	1.0	385065.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT23	X	0	USINT	1.0	385066.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT24	X	0	USINT	1.0	385067.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT25	X	0	USINT	1.0	385068.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT26	X	0	USINT	1.0	385069.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT27	X	0	USINT	1.0	385070.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT28	X	0	USINT	1.0	385071.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT29	X	0	USINT	1.0	385072.0	MSG . CanFdmMsgOut process data mapping .
MAIN.iOUT3	X	0xc4	USINT	1.0	385046.0	MSG . CanFdmMsgOut process data mapping .
























(a)



(b)



(c)

Name	[X]	Online	Type	Size	>Addr...	Linked to
 MAIN.iin1	X	0x55	USINT	1.0	385116.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin10	X	0x1d	USINT	1.0	385125.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin11	X	0xa2	USINT	1.0	385126.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin12	X	0xfa	USINT	1.0	385127.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin13	X	0x12	USINT	1.0	385128.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin14	X	0x1c	USINT	1.0	385129.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin15	X	0x32	USINT	1.0	385130.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin16	X	0x97	USINT	1.0	385131.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin17	X	0x00	USINT	1.0	385132.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin18	X	0x00	USINT	1.0	385133.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin19	X	0x00	USINT	1.0	385134.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin2	X	0x7b	USINT	1.0	385117.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin20	X	0	USINT	1.0	385135.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin21	X	0	USINT	1.0	385136.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin22	X	0	USINT	1.0	385137.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin23	X	0	USINT	1.0	385138.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin24	X	0	USINT	1.0	385139.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin25	X	0	USINT	1.0	385140.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin26	X	0	USINT	1.0	385141.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin27	X	0	USINT	1.0	385142.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin28	X	0	USINT	1.0	385143.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin29	X	0	USINT	1.0	385144.0	MSG . CanFdmMsgln process data mapping .
 MAIN.iin3	X	0x3a	USINT	1.0	385118.0	MSG . CanFdmMsgln process data mapping .

(d)

**Figure 4.4:** Transmission results when the data payload is 16.

### 4.3 Test Results

Following the test setup mentioned in section 3.3, several tests are accomplished .

The bit time configuration is shown in Table 4.1 and this bit time configuration is based on the CANoe configuration file from Volvo which requires that the sampling point should be 70%. The TSEG1 and TSEG2 are calculated based on it. Other parameters are set automatically by MPLAB tool [29] from Microchip.

<b>CAN FD Controller Configuration</b>	<b>Arbitration Phase</b>	<b>Data Phase</b>
Clock Frequency	80 MHz	80 MHz
BR Prescaler	1	1
Number of Time Quanta	80	10
TSEG1	54	5
TSEG2	23	2
SJW	23	2

**Table 4.1:** Bit time configuration of CAN FD controller.

#### 4.3.1 Bit Time Measurement

Bit time measurement results are shown in Table 4.2 and Table 4.3, where the arbitration phase bit rate is set to 500 kbit/s and data phase bit rate is set to 4 Mbit/s.

	<b>Recessive State</b>	<b>Dominant State</b>	<b>Requirements</b>
Average Test Result	2.006 $\mu s$	2.008 $\mu s$	1.992-2.008 $\mu s$

**Table 4.2:** CAN FD Bit time measurement for arbitration phase.

	<b>Recessive State</b>	<b>Dominant State</b>	<b>Requirements</b>
Average Test Result	250.5 ns	250.6 ns	249-251 ns

**Table 4.3:** CAN FD Bit time measurement for data phase.

#### 4.3.2 Voltage Measurement

Voltage measurement results are shown in in Fig. 4.5 and Fig. 4.6, suggesting that the CAN FD controller operates in a normal state.

		Results	Volvo Requirement		
			Min	Normal	Max
Output bus voltage Normal mode	$V_{CANH}$	3.824 V	2.75 V	3.5 V	4.5 V
	$V_{CANL}$	1.177 V	0.5 V	1.5 V	2.25 V
Differential output bus voltage	$V_{CANH} - V_{CANL}$	2.647 V	1.5 V	2.0 V	3.0 V

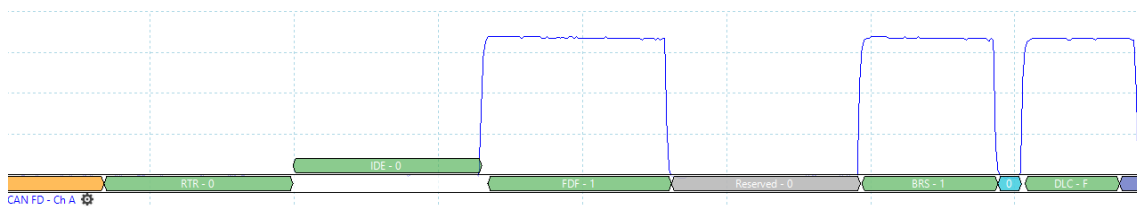
**Figure 4.5:** Test result of Dominant state voltage.

		Results	Volvo Requirement		
			Min	Normal	Max
Output bus voltage Normal mode	$V_{CANH}$	2.521 V	2.0 V	2.5 V	3.0V
	$V_{CANL}$	2.481 V			
Differential output bus voltage	$V_{CANH} - V_{CANL}$	39.92 mV	-500 mV	0 V	50 mV

**Figure 4.6:** Test result of recessive state voltage.

### 4.3.3 Message Frame Encapsulation Measurement

Detail CAN FD frame is verified and part of the arbitration phase is presented in Fig. 4.7, where IDE, r0 and ESI are set to dominant while FDF and BRS are set to recessive. It should be pointed out that r1 is named RTR by the oscilloscope and r1 (RTR) is set to dominant. In all the logic sequence of Fig. 4.7 is consistent with its requirement.



**Figure 4.7:** CAN FD arbitration field.



# 5

## Discussion and Potential Improvements

### 5.1 Discussion

#### 5.1.1 General Performance

As shown in Chapter 4, the prototype implements an interface between TWINCAT and CANoe successfully, and it meets the requirements of Volvo regarding to bit time, voltage measurement and CAN FD frame encapsulation. However, unstable transmission from CANoe to TWINCAT happens infrequently. TWINCAT loses one or more logic-high bits for some data transmission.

#### 5.1.2 Unstable Transmission

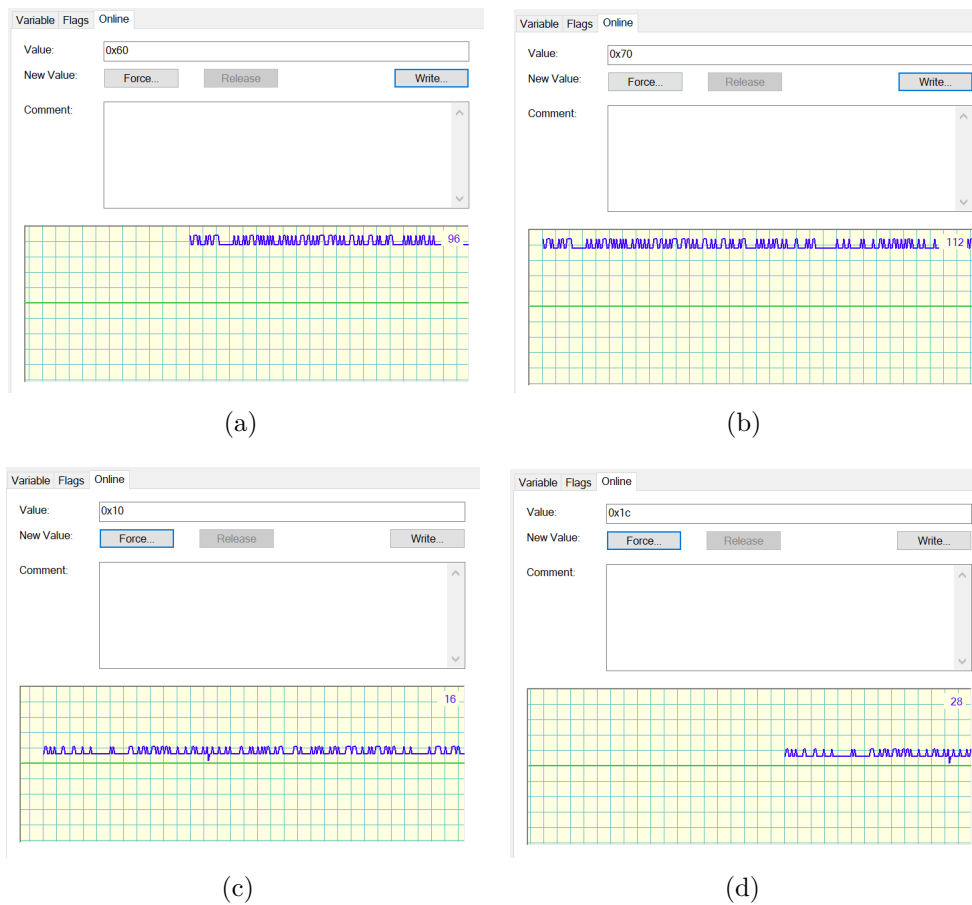
Currently, the communication between TWINCAT and CANoe works for most of the test cases, as shown in section 4.2. However, there is a problem which is the unstable transmission for some certain test cases. In Fig. 5.1, the 64th data byte is configured to 0x70 in CANoe, and TWINCAT should receive the same result if the system works properly. However, the result jumps between 0x70 and 0x60. If the other position in data frame is configured to 0x70, the result in TWINCAT is stable and correct. Similarly, the same phenomenon happens on the 14th data byte if it is set to 0x1c.

Based on our code (see Fig. 5.2), the CAN FD data frame is operated as a buffer. CAN FD transmitting and receiving are executed in `mSPI1_Exchange16bitBuffer` function controlled by an iteration. Theoretically, either 14th data byte or 64th data byte should not have logic difference compared with other positions, and the function should not be sensitive to some certain values either. Furthermore, when we check the data sent from SPI back to EtherCAT slave controller board, as shown in Fig. 5.3, the 64th data byte (0x70) is correct. However, the oscilloscope sometimes gets the wrong data due to some noise within transmission back to TWINCAT.

One of the potential reasons for this problem is that we only use one SPI isolator click board [30] [31] between two automotive networking boards. When we check the data sent from TWINCAT to CANoe, it is always correct and stable. In the opposite direction, as mentioned before, oscilloscope sometimes gets the wrong value because of noise and disturbance. The reason why we cannot use pair of iso-

## 5. Discussion and Potential Improvements

lators boards is that the clock port of the board is uni-directional. The noise and disturbance might be reduced if suitable isolators can be used in pairs.



**Figure 5.1:** Unstable transmission of the 14th and the 64th data byte. (a) and (b) show that TWINCAT receives either 0x60 (96 in decimal) or 0x70 (112 in decimal) of the 64th byte, where 0x70 should be the correct result. (c) and (d) show that TWINCAT receives either 0x10 (16 in decimal) or 0x1c (28 in decimal) of the 14th byte, where 0x1c should be the correct result.

```

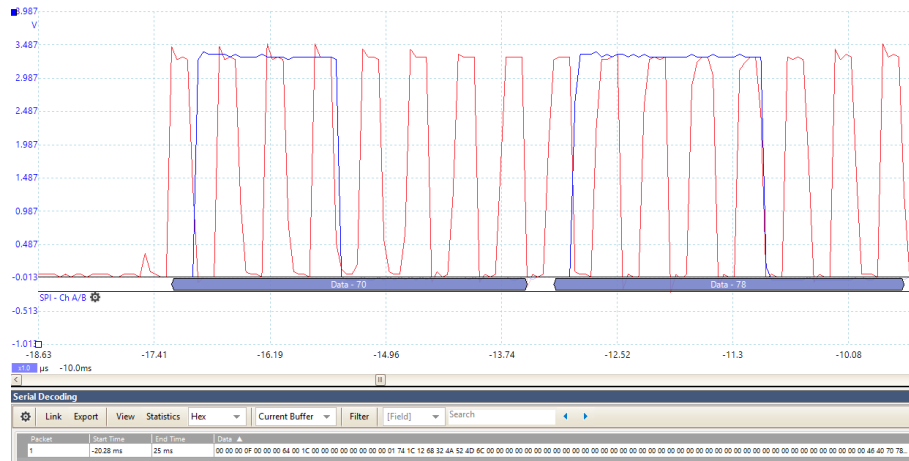
while (1)
{
    rxStatus=CAN1_Receive(&rxCanMsg);
    GetCAN1Msg(&rxCanMsg,mySendBuffer);

    for(i=0;i<36;i++){
        dataSent = mSPI1_Exchange16bitBuffer(&myReceiveBuffer[i],&mySendBuffer[i]);
    }
    /* reason for this extra SPI receive function is that somehow one extra 16 bit data is added
    at the beginning of received data. So MSG63 and MSG64 are stored in receivebuffer[36].*/
    dataSent=mSPI1_Receive16bitBuffer(&myReceiveBuffer[36]);

    setTxCanMsg(myReceiveBuffer,&txCanMsg);
    canStatus=CAN1_Transmit(CAN1_FIFO_1, &txCanMsg);
}

```

**Figure 5.2:** CAN FD transceiving code.



**Figure 5.3:** The last two bytes sent from SPI back to EtherCAT slave controller board.

## 5.2 Potential Improvements

There are several improvements that could be performed to get better performance for the whole system. This section contains a list of potential aspects.

### Using the new EtherCAT slave controller board

Due to the corona virus and the layoff in Volvo truck, the new-designed EtherCAT slave controller board (see section 4.1) can not be manufactured on time, and Volvo can not finish the new motherboard design on time either. In the prototype, two Microchip automotive networking boards are connected with enameled wires. These wires might bring unnecessary noise to make the system become unstable and unrobust. Therefore, it is better to use robust standard connectors to connect slave controller and motherboard, instead of using enameled wires.

### Implementing multiple slave nodes mode

In the prototype, the transmission between TWINCAT and single slave node is achieved. Although it has met the requirement of Volvo truck, it is meaningful to investigate the possibility to let the system be suitable for multiple slave nodes. Furthermore, the new designed EtherCAT slave controller slave board provides this possibility, as far as hardware is concerned. The most of the spare pins of the chips are wired to standard connectors which are connected with the motherboard. These pins contain spare communication channels which can be used for multiple slave nodes. Therefore, implementing multiple slave nodes mode can make the system more flexible and generic.

### Testing transmission performance using dynamic inputs

In the testing, test cases use either static inputs or can be changed manually. However, both TWINCAT and CANoe configurations can change frequently. For example, CANoe can be configured by a database which includes large test cases. Therefore, it is valuable to do transmission testing by using dynamic inputs. The

## 5. Discussion and Potential Improvements

---

transmission delay and potential bugs can be found, which can improve robustness of the whole system.

### **Using ring buffer**

To reduce transmission delay when using dynamic inputs, an more effective data storage structure, like using ring buffer, should be investigated.



# 6

## Conclusion

The goal of this thesis is to implement a communication interface for EtherCAT and CAN FD messages. It involves designing a PCB for EtherCAT and CAN FD communication interface and writing software to test and verify the prototype design.

The PCB has been approved by Volvo and the system has been tested in real-time as well. The results shown in Section 4.2 and Section 4.3 demonstrate that the system is capable of transmitting EtherCAT and CAN FD messages correctly in terms of bit time, voltage measurement, message frame encapsulation and message reception.

Unstable transmission however occurs infrequently and we can not conclude what kind of message would suffer from such a phenomenon and when it could happen. One possible reason for the unstable transmission is that environment noise affects the logic state of some bits due to the lack of an SPI isolator, which is discussed in Section 5.1.2.

There are several possible improvements to make the system more generic, flexible and robust, including using a new EtherCAT slave controller board, implementing multiple slave nodes mode, testing transmission performance using dynamic inputs and using ring buffer.



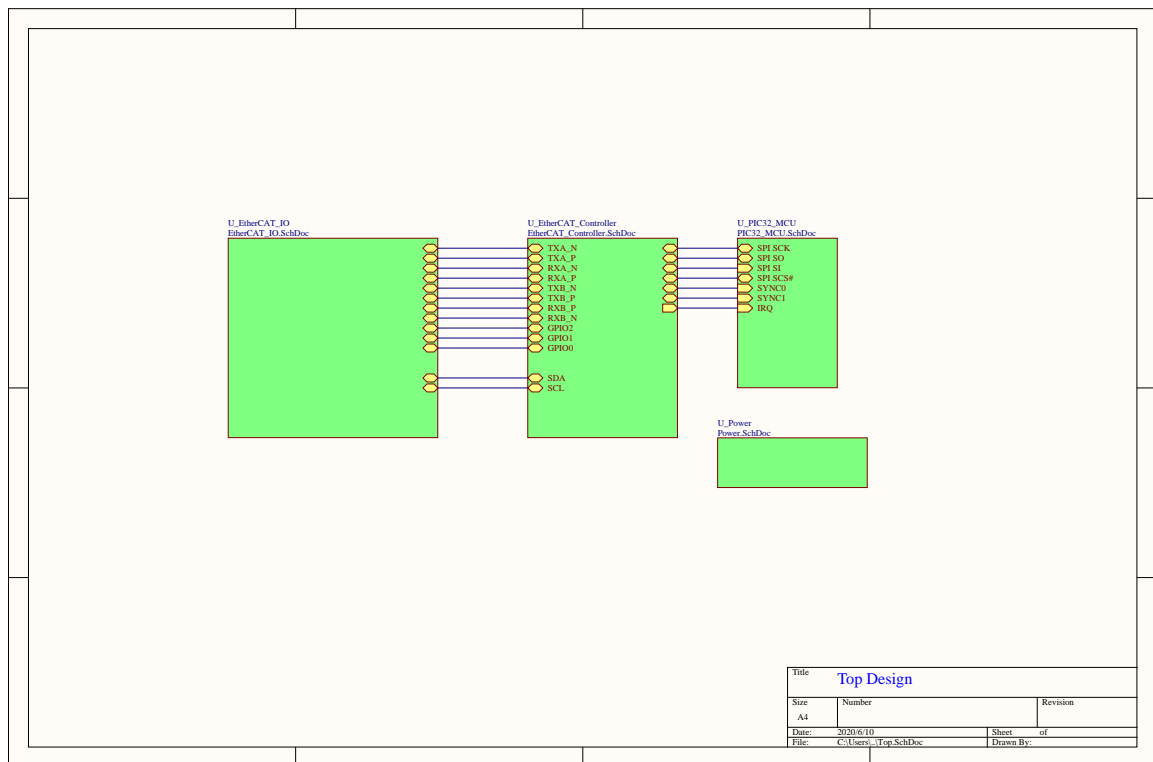
# Bibliography

- [1] Bosch White Paper. CAN with flexible data rate, 2011, [online] Available: <http://www.bosch-semiconductors.de>.
- [2] Introduction to the Controller Area Network (CAN), TEXAS INSTRUMENTS, MAY. 2016, [online] Available: <http://www.ti.com/lit/an/sloa101b/sloa101b.pdf>
- [3] R. B. Gmbh, CAN with flexible data-rate, 2012.
- [4] EtherCAT Introduction, [Online]. Available: <https://www.ethercat.org/en/technology.html>
- [5] CANoe Introduction, [Online]. Available: <https://www.vector.com/int/en/products/products-a-z/software/canoe/>
- [6] ECU CAN Interface Test Specification, VOLVO, July. 2018.
- [7] G. M. Zago and E. P. de Freitas, "A quantitative performance study on can and can FD vehicular networks," IEEE Trans. Ind. Electron., vol. 65, no. 5, pp. 4413–4422, May 2018.
- [8] S. Woo, H. J. Jo, I. S. Kim and D. H. Lee, "A Practical Security Architecture for In-Vehicle CAN-FD," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 8, pp. 2248–2261, August. 2016.
- [9] U. D. Bordoloi and S. Samii, "The Frame Packing Problem for CAN-FD," 2014 IEEE Real-Time Systems Symposium, Rome, 2014.
- [10] H. Eisele and K.P. Orlando, "What CAN-FD offers for automotive networking?", In Stuttgart International Symposium on Automotive and Engine Technology, 2014.
- [11] EtherCAT-CAN Gateway Data Sheet, ESD Electronics, 2013.
- [12] Anybus Communicator CAN-EtherCAT User Manual, Anybus, November. 2013.
- [13] Joseph LaDou, Printed circuit board industry, International Journal of Hygiene and Environmental Health, Volume 209, Issue 3, 2006, Pages 211–219.
- [14] Altium Designer 20 Feature, [Online]. Available: <https://www.altium.com/altium-designer/feature-playlists>
- [15] SPI Block Guide, MOTOROLA, February. 2003.
- [16] dsPIC33/PIC24 FRM, CAN Flexible Data-Rate (FD) Protocol Module, MICROCHIP, January. 2019
- [17] TWINCAT Introduction, [Online]. Available: <https://www.beckhoff.com/twincat/>
- [18] EVB-LAN9252-SPI Quick Start Guide, MICROCHIP, May. 2017.
- [19] EVB-LAN9252-SPI-Schematics-Rev-A0, MICROCHIP, March. 2019.
- [20] EVB-LAN9252-SPI-Board Design Files-Altium-Rev-A0, MICROCHIP, March. 2019

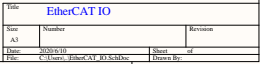
- [21] Automotive Networking Development Board User's Guide, MICROCHIP, October. 2016.
- [22] PIC32MX5XX/6XX/7XX Family Data Sheet, MICROCHIP, September. 2019.
- [23] dsPIC33CK256MP508 Family Data Sheet, MICROCHIP, October. 2019.
- [24] MCP2517FD External CAN FD Controller with SPI Interface, MICROCHIP, October. 2017
- [25] VN1600 Interface Family Manual, VECTOR, January. 2020.
- [26] mikroBUS<sup>TM</sup> standard, MikroElektronika, May. 2015.
- [27] LAN9252 - 2/3-Port EtherCAT Slave Controller with Integrated Ethernet PHYs, MICROCHIP, January. 2015.
- [28] PIC32MX795F512L 100-pin to 100-pin TQFP CAN-USB Plug-in Module (PIM) Info Sheet, MICROCHIP, January. 2014.
- [29] MPLAB IDE Introduction, [Online]. Available:<https://www.microchip.com/mplab/mplab-x-ide>
- [30] SPI Isolator Click Introduction, [Online]. Available:<https://www.mikroe.com/spi-isolator-click>
- [31] Digital Isolator for SPI Data Sheet, ANALOG DEVICES, July. 2017.

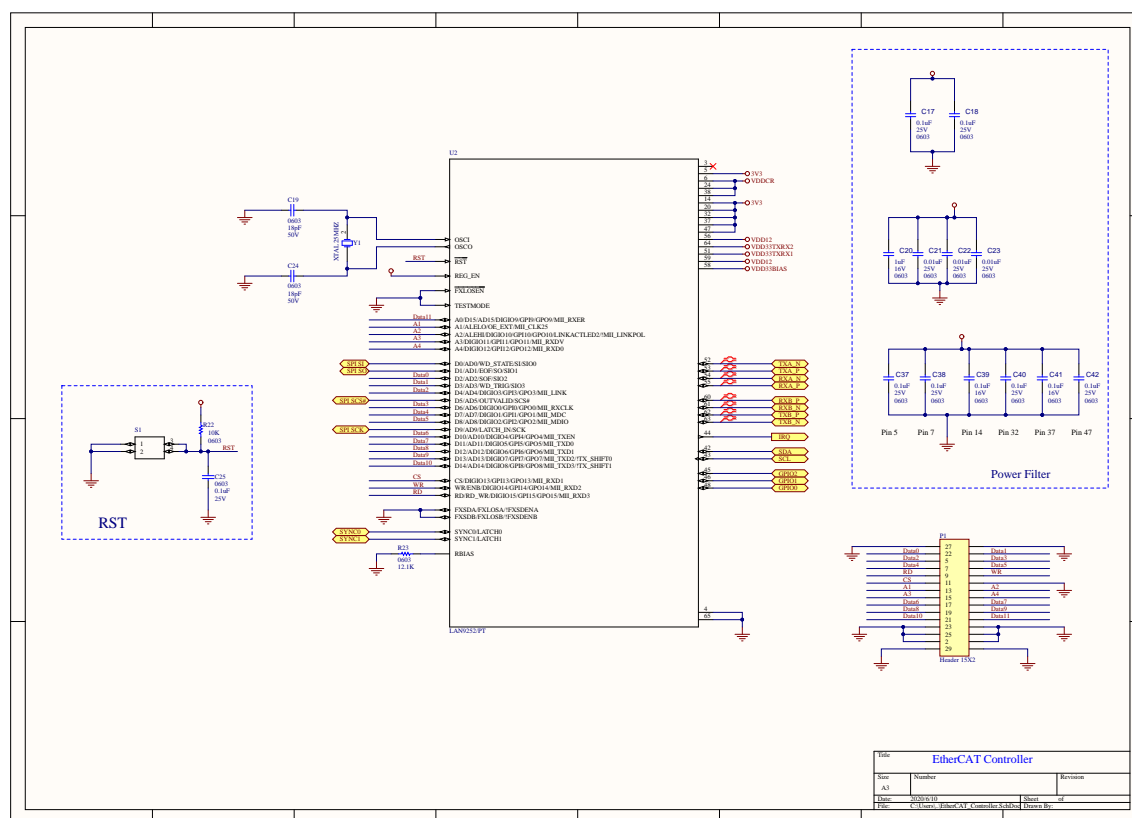
# A

## Appendix



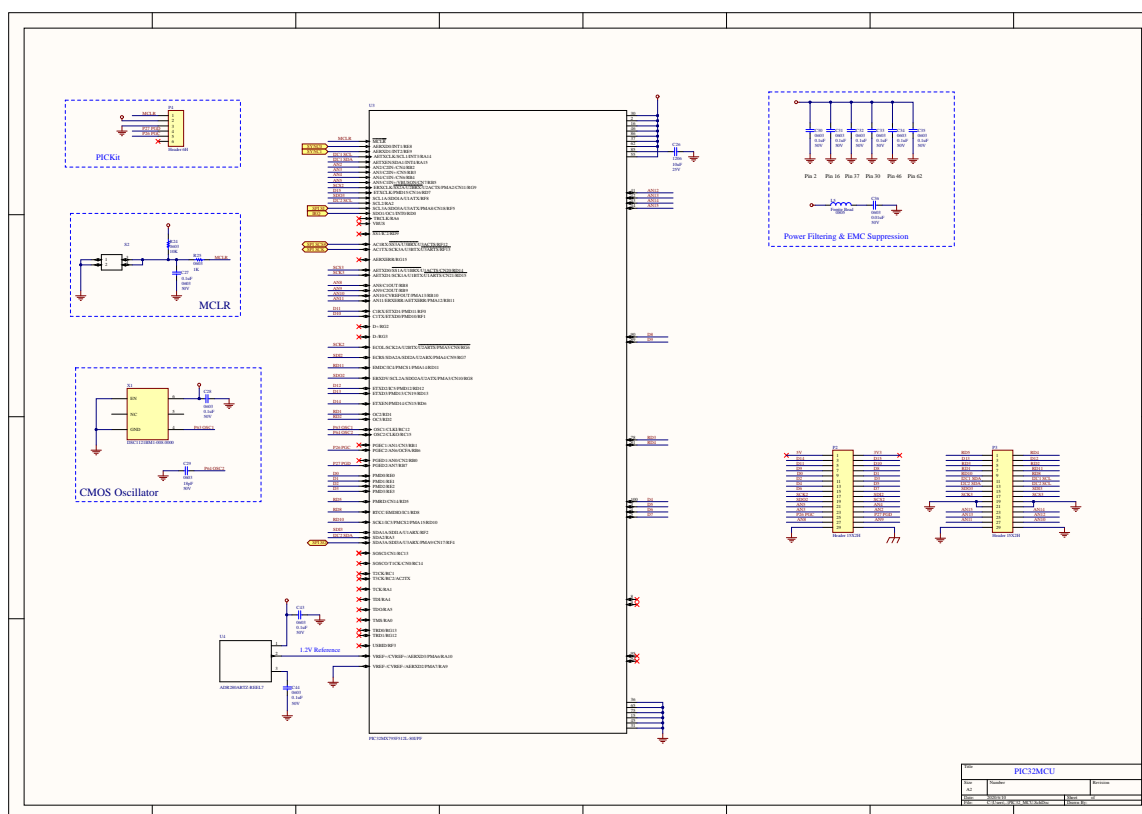
**Figure A.1:** Top schematic





**Figure A.3:** LAN9252 configuration schematic

## A. Appendix



**Figure A.4:** PIC32 configuration schematic



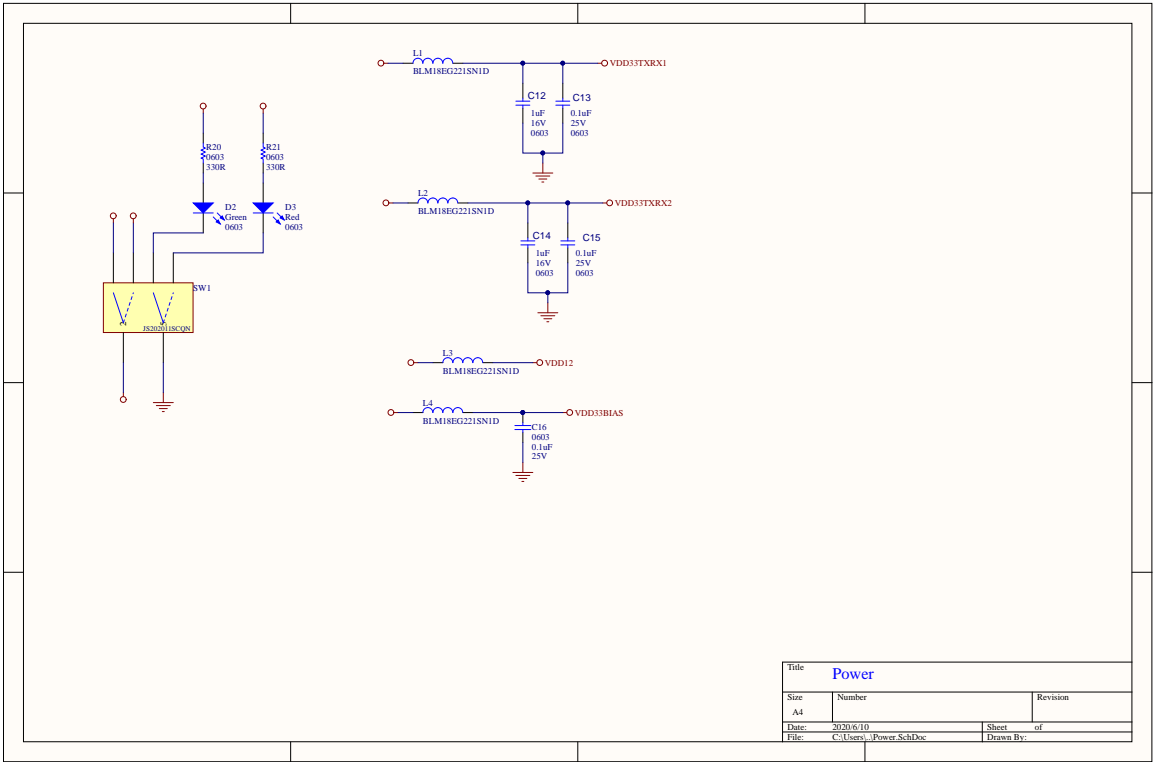


Figure A.5: Power schematic