April 2017

Encoding and Physical Study of the CANbus Sensor Network

Anna Celeste Hernandez
Worcester Polytechnic Institute

Klaudia Linek
Worcester Polytechnic Institute

ENCODING AND PHYSICAL STUDY OF THE CANBUS SENSOR NETWORK

Major Qualifying Project Submitted
to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By

Klaudia Linek

Annie Hernandez

April 2017

Advisor: Professor Alexander Wyglinski

Abstract

Within vehicles, the need for a better performing, more secure network is increasing due to the complexity of the sensors and the growing number of people who can compromise the system. The objective was to improve the performance and throughput of data while also working to improve security within the CANbus network. The approach entailed encoding the CAN frames using M-ASK, and characterizing nodes based on their emitted signal. Results included the successful creation of base2CAN and base4CAN nodes, implementing 4-ary ASK within the network, thusly achieving a doubled throughput. Another achievement was the sounding of the CAN test-bed, however, this left inconclusive results. Although the results are inconclusive, there is reason to believe that characterizing the nodes can be implemented for increasing vehicle security. Ultimately, future research can be done to improve integration of M-ASK onto the harness using a current amplifier and signal analysis, build a to-spec CAN node from scratch, or design another method to characterize CAN nodes.

Acknowledgments

Our group would like to thank the following mentors and peers for their help and support throughout the entirety of our project:

- Our project advisor, Professor Alexander Wyglinski, for his timely feedback and assistance throughout the project.
- MITRE for granting us the use of the CANbus Harness and the access to the newly renovated MITRE collab lab.
- Hristos Giannopoulos for his help in setting up the CANbus Harness and for all the tips and advice in handling the harness.
- William Appleyard for helping to gather the necessary equipment and parts for our project.
- Professor R. James Duckworth for help with the ADC and DAC peripherals.

Table of Contents

List of Figures

List of Tables

Executive Summary

The CANbus protocol provides a central bus line for each Electronic Control Unit (ECU) to connect to. While internal safety features are in place, if a malicious attacker were to gain control of a low-priority, easy-to-access node (such as a Tire Pressure Monitoring System (TPMS) sensor), this hacked node could flood the bus with messages and prevent other nodes from communicating vital information.

This MQP aims to enhance the security of the CANbus through an impedance study of the 2013 Chevy Impala bus we were given access to by MITRE. By studying the reverberations and distortions of the original signal, we can characterize the nodes and their locations. The outcome of sounding, showed that the measured data signals were barely distinguishable from each other. After conducting several trials, taking measurements and doing analysis, it appeared that the sounding process would not yield results that could be used to characterize the nodes. This characterization of the bus is still believed to be feasible, although the process of doing so in this case lead to inconclusive results.

While initially security focused, our MQP has come to include throughput optimization of the CANbus network. A very analog approach was taken to digital signals, implementing 4-ary ASK. A CAN platform, base2CAN, was developed, to send, receive, and send modified CAN frames and messages. M-ASK was implemented, using FPGAs and peripherals including DACs and ADCs. base2CAN and 4-ary ASK were then integrated to create base4CAN, a node that could send, receive, and modify CAN messages in four analog voltage levels. This has been shown successful in both digital and analog operation, as detailed in the report.

1. Introduction

1.1 Today's Technological Challenges

It is no surprise that with the advent and rise of autonomous vehicles, the industry finds itself faced with new challenges. Generations are being born surrounded by computers and technology, and are as proficient in technology as they are with their native language. This is leading to people, as young as teenagers, finding security flaws in major systems and taking them down, whether online or not [1]. The implication this has on vehicular security is enormous - people interested in autonomous vehicles are wary of trusting computers wholly, afraid of internal lockups or external attacks. When dealing with a literal ton of metal, gears, and wires at such high speeds, their reservations are completely valid.

Other issues plaguing the automotive industry center around the ever-rapidly increasing number of Electronic Control Units (ECUs) and microcontrollers. Microcontrollers and ECUs control things as trivial as power windows and seat adjustment and as crucial as steering and braking [2]. While the standard used to have the steering wheel and column physically connect to the tires making turning all-manual, active steering interjects an ECU (to modify/rectify your input), and steer-by-wire systems would no longer have a physical connection at all. While active steering is currently available in many models, steer-by-wire is banned until its safety and reliability can be thoroughly proven [2].

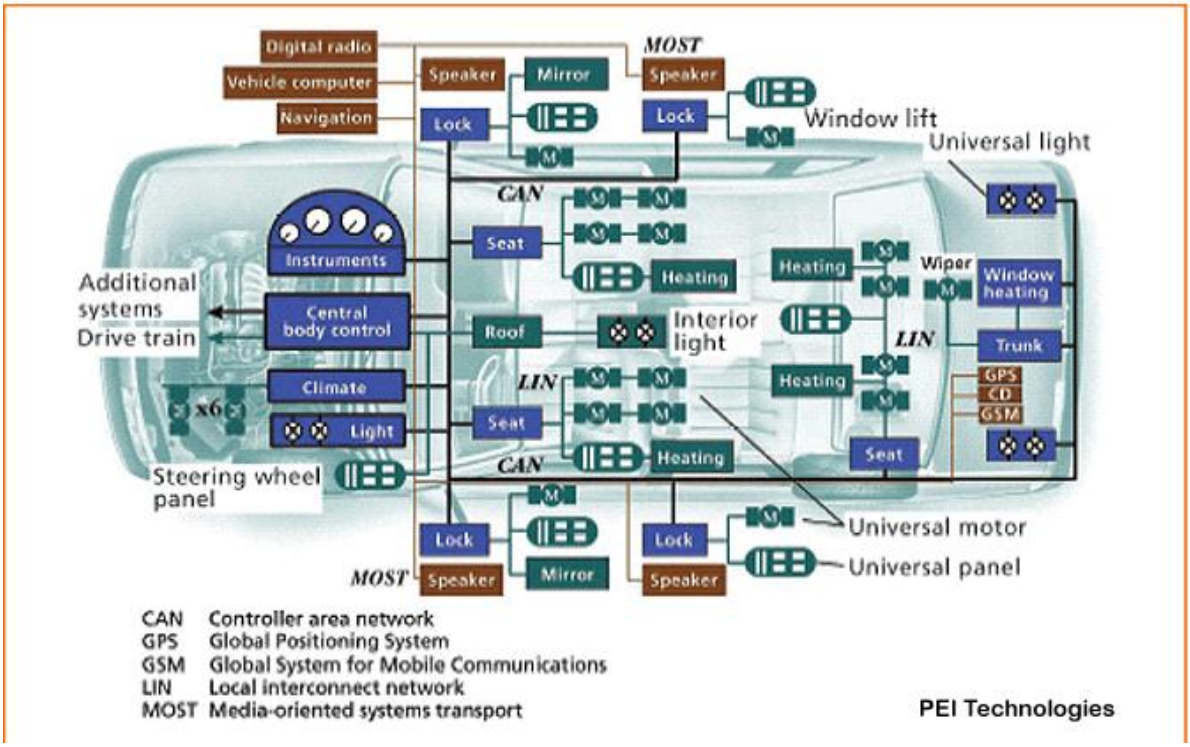

Figure 1.1: Hierarchy of a car with CANbus [3]

The Controller Area Network bus line (CANbus), shown in Figure 1.1, provides a hierarchy and central bus line for each sensor to communicate on. The CAN protocol is the *de-facto* standard in automation for vehicular security. With an updated version [4], CAN with Flexible Data-rate (CAN FD), to be on the market as late as Fall 2017, many eyes are on the technology [5]. This projects looks into researching the inner workings of the CAN and CAN FD protocols, as well as the nodes that are physically placed on the bus. After, a study of an automotive test-bed will be completed to analyze the effect M-ary Amplitude Shift Key (M-ASK) encoding will have on the signals within a CANbus network. Ideas will be proposed on reduction of effects that noise, reflections and other factors have on the M-ASK encoded transmission.

CAN is a protocol that has lasted the test of time, with 2017 marking the 30 year anniversary of the initial market release. While created, standardized, and sustained by Robert Bosch GmbH, also known as Bosch, two protocol updates have been released to remain current and innovative [6].

The original protocol, referred to as "CAN," was later updated to "CAN 2.0" in order to provide support for longer messages and more nodes on the bus at one time. While both of these are in use today, CAN 2.0 is the vehicle bus standard and part of the mandated OBD-II standard, in the U.S. since 1996 and worldwide since 2008 [7]. Bosch is still developing "CAN FD," a protocol yet to be released on vehicles today, but which is set to appear on 2018 model vehicles and predicted to be implemented in most 2020 models [8].

While the CAN network does increases speed and improves structure, CAN does not have many security measures. While there is error checking for correct message framing, there is currently no support to assure node IDs are correspond to the correct nodes. That is to say, your TPMS sensor has a lower priority than your braking system, however, an attacker has easy access to the TPMS sensor and can send a legitimate looking message with a higher priority ID. If your system were to be flooded with messages of the highest priority, none of the other systems would be able to communicate, leading to system failures [9].

Because of this, further innovation is underway in many college settings. Such examples include graduate students at Carnegie Mellon University [11], University of Washington [12], and Worcester Polytechnic Institute [10], as well as Federally Funded Research and Development Centers, such as MITRE. These efforts usually center around bus security with the goal to fend off malicious attacks [13].

1.2 Addressing and Filling Needs

This MQP, while initially security oriented in nature, has since come to include throughput optimization through M-ASK encoding. While CAN FD enhanced CAN2.0's abilities with a higher,

selective data-rates and and longer data fields, a further discovery could improve throughput further still. With autonomous vehicles on the rise, and the ever-increasing complexity of sensor networks, there higher speeds will boost reaction time and allow more information to be sent in the same period of time.

Advised by a wireless communication-based professor, the team was inspired to attempt the unconventional. Seeing the CAN-Hi and CAN-Lo lines unique pairing, they would be best utilized by implementing M-ary Amplitude Shift Keying. M-ASK seemed not only the most reasonable but also the most straightforward way to go. Access to the CANbus from a 2013 Chevy Impala as well as a space to store such a large network was graciously provided to us by MITRE, which was invaluable for our analysis and implementation.

The project's final form includes an impedance study of the Impala's bus, two base4CAN communication nodes (each acting as both a sender & receiver), two ADC and DACs (one of each per base4CAN node), and the CANbus itself. Contributions gained from this research include higher node security, throughput doubling, and a proof of M-ASK encoding feasibility.

1.3 Report Organization

This report begins with a brief background of the CANbus System in Chapter 2 which includes information about its history, applications and implementation, as well as discuss M-ASK modulation and signal transient response. Chapter 3 includes the expected timeline and the progression of the MQP. Chapter 4 analyzes the hardware options available, presents the hardware alternatives selected for the project, presents relevant information about the physical harness. Also, in this chapter is an explanation of the design implementation. Chapter 5 discusses the final results. Chapter 6 concludes the report and provides suggestions for future projects.

2. The CANbus System

2.0 Introduction

To begin, a brief background is provided below for those who have not been exposed to CAN or want to review. In this chapter we study the CAN system and discuss when it was developed, what its purpose is, why is it important, what are some of its applications, how is it implemented on a physical bus and digitally. CAN FD will also be mentioned as it predicted to take the place of CAN in the near future. Lastly, introduced are the concepts of MASK-ing and transient response which are used during project execution.

2.1 Fundamentals of a CANbus Network

Early in automation, vehicles were by-and-large analog machines. As technology progresses and advances, so do the number of sensors, microcontrollers, and ECUs. Most functions you can think of, from the indicator lights on a dashboard to the airbags and Anti-lock Brake System (ABS), are sensor-based and controlled electronically.

When sensors were fewer, communication between them was direct and hardwired. The top half of Figure 2.1 shows a star topology, where wires cross node to node forming a web. This quickly became less feasible as complexity increased.

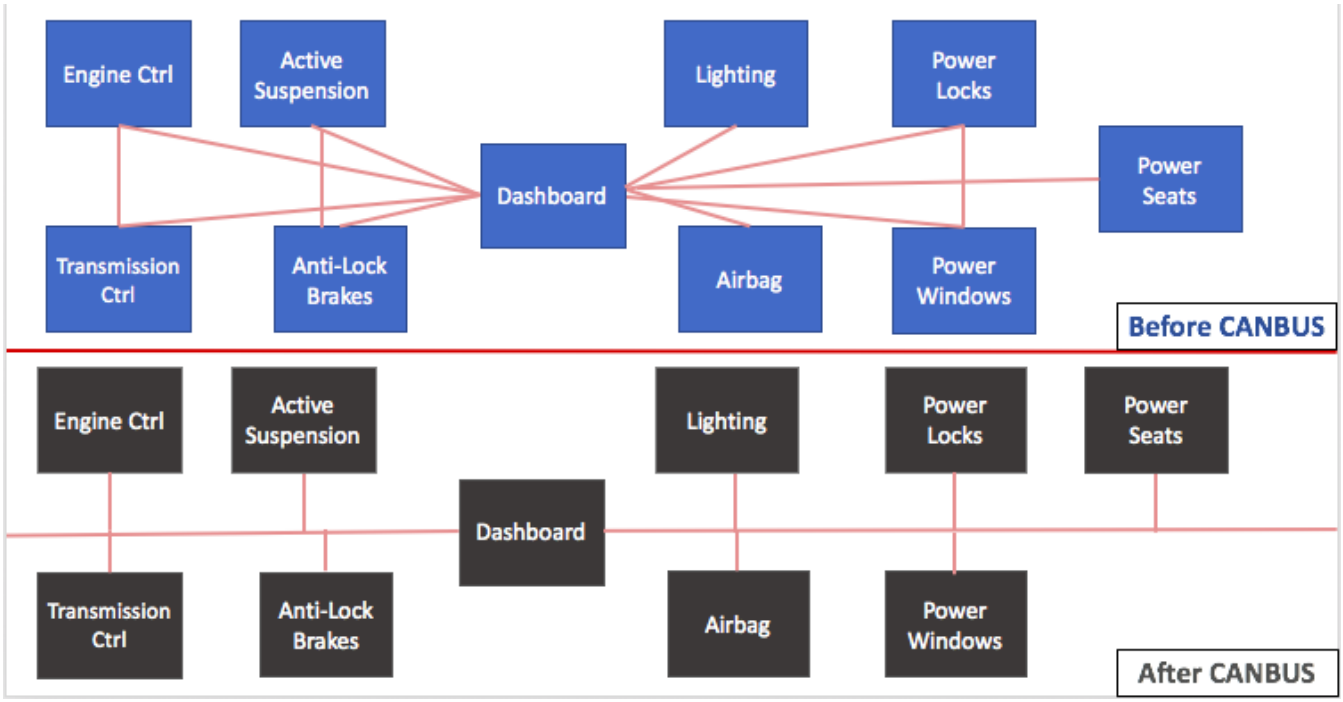

Figure 2.1: Sensor Network Structure shifting from star to bus topology

A change of hierarchy was needed, and so the CAN protocol and architecture was designed. With CAN, a central bus line is installed, providing a party line for all sensors to communicate on. Not only is less wiring needed, but also the ability for every sensor to hear everyone else on the bus is gained, as shown in the lower section of Figure 2.1. While simplified, this is not a trivial measure, and greatly increases efficiency and safety. Figure 2.2 provides a better sense of scale, in these regards.

The CANbus is a serial bus communications protocol that provides an effective and reliable means of communication between sensors and other nodes within a vehicle. It is the party line through which all messages are relayed to all electronic devices in the vehicle. CANbus is the most effective and commonly used communication network for vehicles, known for its reliability, with multiple ECUs and micro-controlled sensors connected through it.

Not only is it already being used within the majority of vehicles and other transportation systems, but it is also being further improved to become quicker and more secure with the development of CAN Flexible Data-rate, or CAN FD.

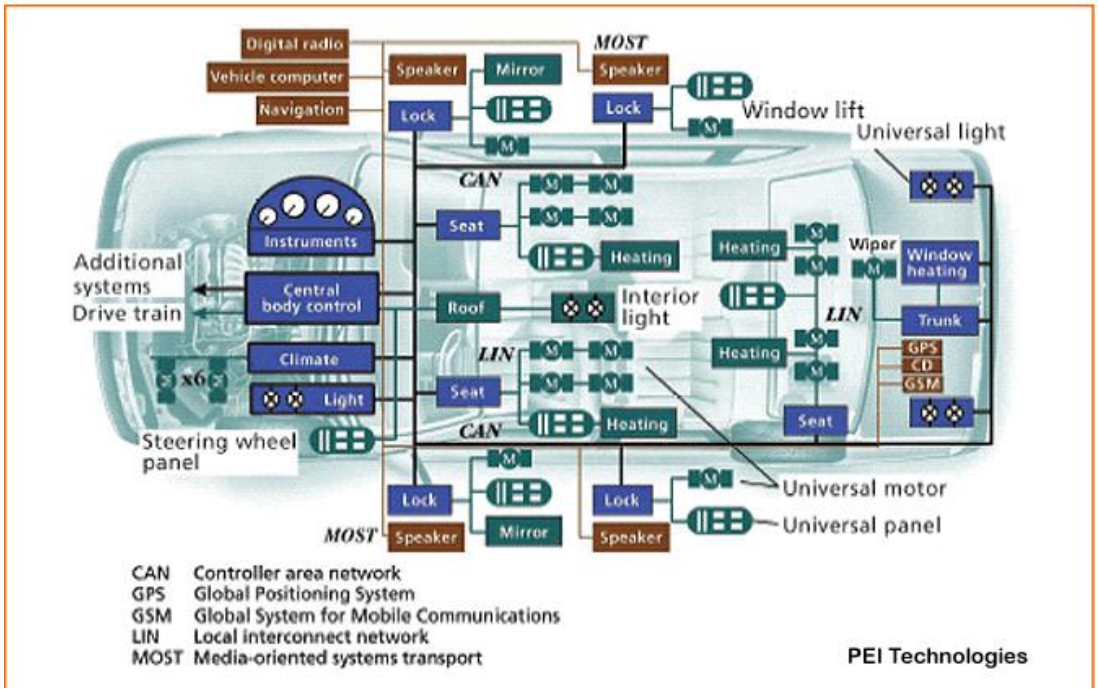

Figure 2.2: CAN Implementation Within Vehicles [3]

2.2 History of CANbus

The CANbus system was proposed by Robert Bosch GmbH, a German electronics company, which revolutionized node interaction. This system was implemented in hardware by Intel, who in 1987 created a CAN controller chip. In 1991 CAN version 2.0 was internationally standardized and a technical group of the manufacturers and users, "CAN in Automation" (CiA) was created [14].

There are various CAN protocols. The first, Bosch's prototype of CANopen, began their development phase in 1993 and thanks to the assistance of various college professors, was completed

and standardized by the CiA within two years [14]. In only five years' time, this protocol became the most important standardized embedded network in Europe. Throughout the late 1990's and early 2000's, new protocols were created, some of which include the 'Time-triggered communication on CAN' (TTCAN) and the CANopen Safety Protocol. The CAN FD protocol specifications were introduced at the 13th International CAN Conference in 2012 and are expected to be used in next generation vehicles communication networks [14].

2.3 CAN Applications

While the CANbus protocol intended use was for communication within nodes of a passenger vehicle, multiple other applications have emerged. The CANbus system provided the framework for programmable systems and different device, interface, and application profiles. This made CANopen popular in industry segments such as printing machines, maritime applications, and medical systems.

Many have adapted the CAN protocol to further fit their needs. This can be seen in the CANOpen [15] and CANAerospace protocols [16]. CANOpen is applied to industrial machinery, railways, construction machinery, mining trucks, and process-optimized operating rooms [15]. CANOpen's aim was to predefine some of the higher-level protocols to make it easier for other industries to use. CANAerospace was developed by Stock Flight Systems and has been certified by the FAA and used by NASA. In addition to planes and unmanned helicopters, it can also be seen in satellites currently orbiting the Earth and in SOFIA (the Stratospheric Observatory for Infrared Astronomy) [15]. CANAerospace was designed to be used specifically in the aerospace industry rather than most industrial adaptations. Other examples of higher level CAN protocols

include DeviceNet, CAN Kingdom, and SDS [17].

2.4 Technical Understanding

The CANbus is the central network that all the sensors in a vehicle are attached to. This includes Tire Pressure Monitoring System (TPMS) sensors, dashboards, radios, and everything in-between. It also includes vital components such as the Anti-lock Brake System and airbags [18]. For an example of just how large these networks are, the physical CANbus line this MQP is using for testing is shown in Figure 2.3. Do remember these run the perimeter of the car.

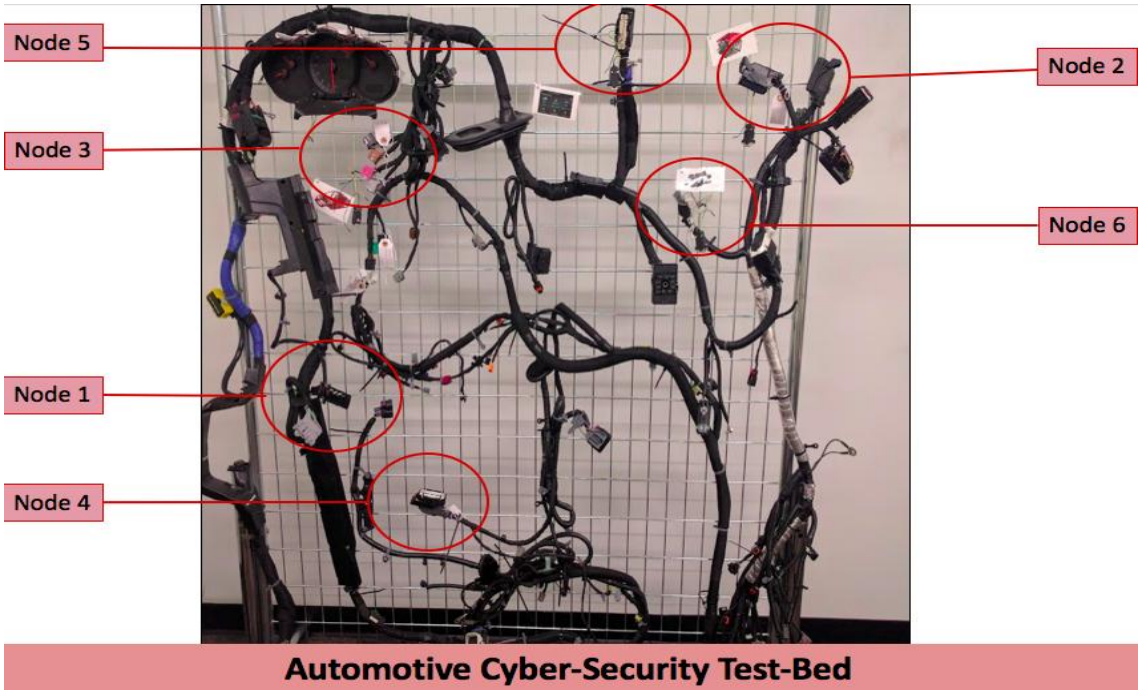

Figure 2.3: The CANbus from a 2003 Chevy Impala

When a node or ECU wants to send information, they use a special format that includes their identification number. If only one node talks at a time, the ID doesn't affect transmission. In

fast-paced, complex systems like modern cars, however, it is much more likely that many sensors will be trying to transmit messages simultaneously. In the CAN protocol, there is an arbitration period which picks out the person with the lowest ID (the closest node ID to zero). This node gets to send its message this round and everyone else can try again next round. This mechanism is what determines that changing the radio station is less important than hitting the brakes [19].

Table 2.1 demonstrates this method of prioritizing certain nodes over others - with three nodes, where the node with the lowest ID will be successfully received as the others drop out [19].

Table 2.1: CAN Message Priority

Identification (binary)	Identification (decimal)	Node in Network
0 1 0 0 0 1 1 0 1 1 1	567	Node 1
0 0 0 1 0 1 0 1 1 1 0	174	Node 2
0 0 0 1 1 1 0 0 0 1 0	226	Node 3
0 0 0 1 0 1 0 1 1 1 0	**174**	**Received Node**

In order to protect itself, CAN messages must follow a certain format. If they are not correctly framed, they will not be successfully transmitted. All packets sent go through a "sanity" check to ensure they do not have obvious errors. The cyclic redundancy check (CRC) at the end of a message ensures the message generated fits a certain mathematical formula from beginning to the end. If, when check time rolls around, the message does not abide by this formula, it means that something happened to it during transfer and the message is not in the expected format.

This being the case, the message is ruled as erroneous and is ignored. In the case of CAN, when an error is found by such detection systems like the CRC, it is as if the bus covers its ears and pretends the message was never sent in the first place - since after all, this message could be a

malicious attack. Whether or not the error is due to natural causes or malicious intervention, the location of this error and its impact on the intended receiver(s) is not necessarily known. This makes it safer and easier to simply throw away any messages perceived to be corrupted with an organic error rather than to try to fix the message.

2.5 Layout of a CANbus System

The CANbus itself is constructed of two lines and two terminators (small resistors). Due to the noisiness of moving, operating vehicles, simply reading the voltage off of one line is not enough. Any movement, vibration, heat, or impedances will distort the original signal. To remedy this, CAN sends all messages over both the CAN-Low and CAN-High line, one of which is the inverse of the other.

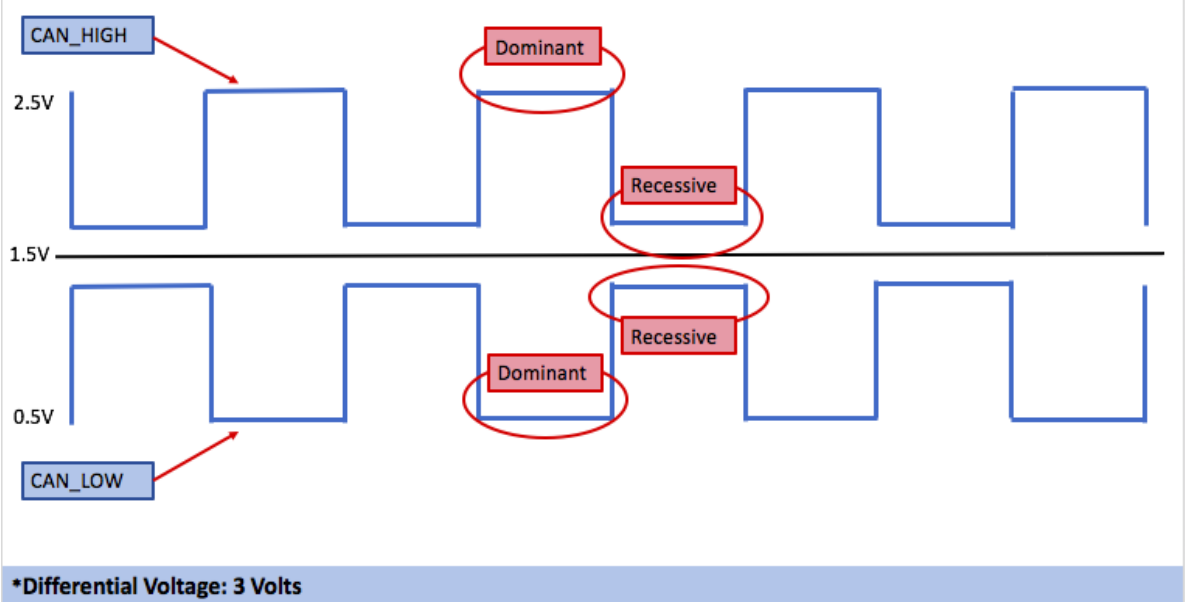

Figure 2.4: CAN Data Lines Dominant vs Recessive

This allows us to see a differential voltage between the two lines, greatly emphasizing the

difference between a 1 and 0. After all, the standard uses messages with binary bits [20].

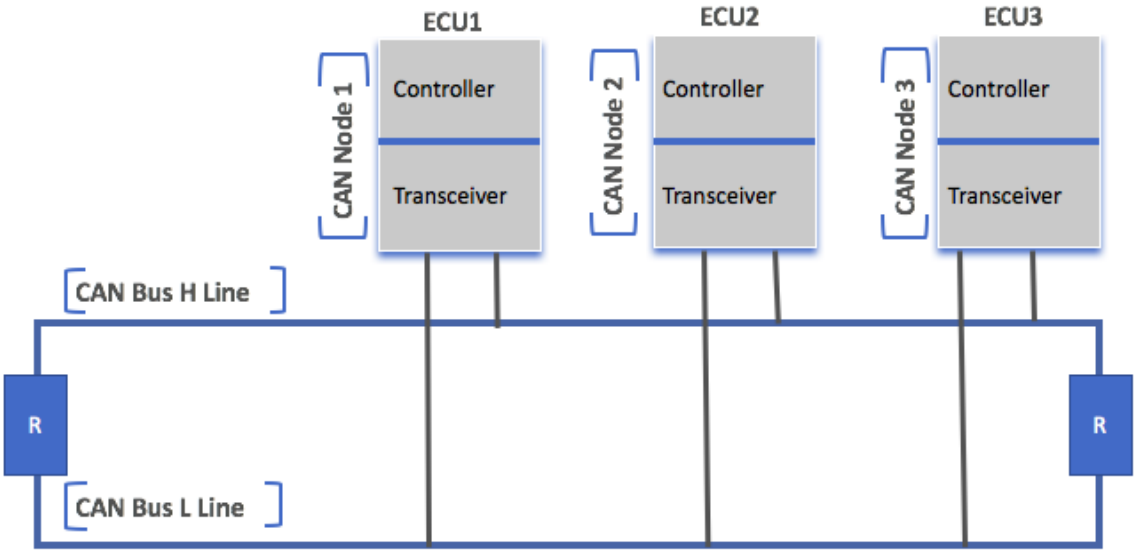

Figure 2.5 : CAN Architecture with bus lines and nodes

The CAN system consists of the following: at least one CAN node, two data bus lines, and two bus terminators (usually a 100 or 120 Ohm resistor on each end). The physical wires themselves can be any form of cable that can be terminated. In practice, however, shielded twisted pair is the most popular option due to its availability, dual-channeled nature, and low cost. Ethernet cables are also a frequent option, although the increased amount of cable required to send both channels is undesirable. In Figure 2.5, three ECUs are connected to the bus. In actuality, as many ECU's can be connected in a CAN system as are deemed necessary.[21] Among many possible ECUs to connect to the bus, we have the following which serve different functionalities, as shown in Table 2.2:

Table 2.2: Possible ECUs interconnected within a CANbus Network

Functionality	**ECUs**
Ensures peak performance from internal combustion engines via actuator control	Engine Control Unit
Controls shifting in automatic transmissions	Transmission Control Unit
Monitors rotational speed of each wheel	Anti-Lock Braking Electronic Control Unit
Actively prevents loss of traction	Traction Control System
Evaluates crashes and triggers appropriate safety measures based on severity.	Airbag Control Unit
Senses vehicle speed and steering torque to adjust how much to assist driver.	Power Steering Electronic Control Unit

Each node on the bus is capable of sending and receiving information, and in some cases, have their own sub-branch of sensors to convey information to. Each node, by ISO standards, must contain a CAN transceiver, CAN controller, and a CPU or microprocessor, though other additions may be made so long as they do not interfere with the node's timing. An image of the topology can be seen in Figure 2.6.

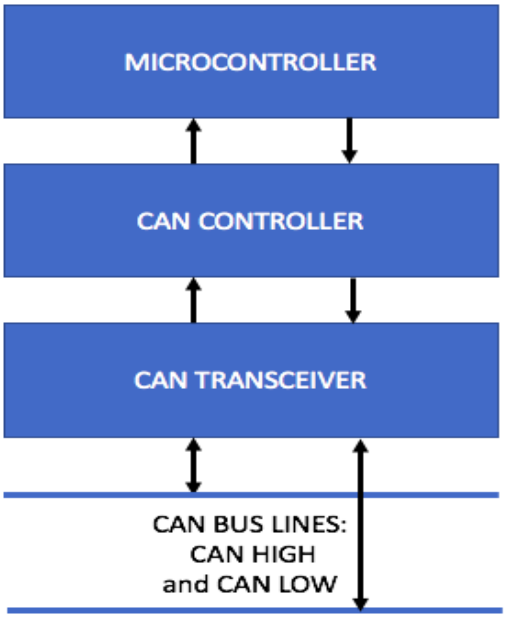

Figure 2.6 : CAN Node Composition

Their jobs are broken down as follows: The transceiver takes information off the bus and converts is to a readable format to the controller. When sending, it takes the controller's information and converts is to bus-readable format. The CAN controller takes the large, parallel chunk of data the transceiver provides and feeds it serially to the microcontroller. When sending, it takes the serial microcontroller's message and compiles it into a large, parallel chunk to give the transceiver. The microcontroller/CPU is the brain who reads messages and decided how to handle and respond to them [22].

In order for these nodes to communicate with one another in real time, and resolve who to listen to when multiple nodes send data, CAN has a very tightly regulated timing protocol. Each node must first become synchronized to the master clock. This allows for each node to be recognized and for arbitration to run more smoothly.

2.6 CAN with Flexible Data-rate

As technology advances, so do vehicles in terms of how many embedded networks they contain that need to transmit and receive messages. Instead of just the radio and TPMS, sensors have added at pedals and wheels, back-up cameras, remote start, Bluetooth connectivity and more. This gives vehicles greater functionality; however, it increases their complexity, which creates a need for the CANbus system to be able to transmit more data to compensate. CAN with Flexible Data-rate, also referred to as CAN FD, not only upgrades how large of messages can be sent but also improves the speed they can be sent at [23].

With both of these factors in play, throughput is greatly enhanced beyond classical CAN's capabilities. Compared to other networks (such as FlexRay), CANFD appears to be the most advantageous, as it has a similar physical layer and software to CAN [23]. The main difference in the hardware is in the transceiver, which needs to be able to adapt to speeds of up to 1Mbps.

CANFD is different from CAN in that it can have a payload as high as 64 bytes as

opposed to CAN's quite small 8 bytes. This reduces the need to split messages into chunks and also decreases the ratio of overhead to payload, making each message more efficient [23].

Table 2.3 : Characteristics of CAN FD and Vanilla CAN

Vanilla CAN	CAN FD	Characteristics
8 Bytes	64 Bytes	Max Data Payload
1 Mb/s	10 Mb/s	Max Data Rate
15	17-21	CRC Bits

To combat the increase in transmission time but maintain proper arbitration key to CAN's functionality, they also employed a second improvement. Arbitration is a delicate procedure which is heavily reliant on a small frame window, since speeding up the entire packet was unfeasible. However, once arbitration was handled, the payload itself can be send at a much higher rate, making the time the message as a whole shorter. The Flexible Data-Rate refers to exactly this function, found in the packet itself. After the payload has transmitted, the footer of the packet (containing the CRC and ACK listener) returns back to its previous 1Mpbs speed.

As visualized in Figure 2.7, each frame has a header and footer (in grey) before and after the data itself (both shades of blue). Both lines are the same frame, but the data portion of the first is sped up, shrinking it in time to what is shown in the second line. No data is lost, but the overall length of each frame sent this way is reduced [17].

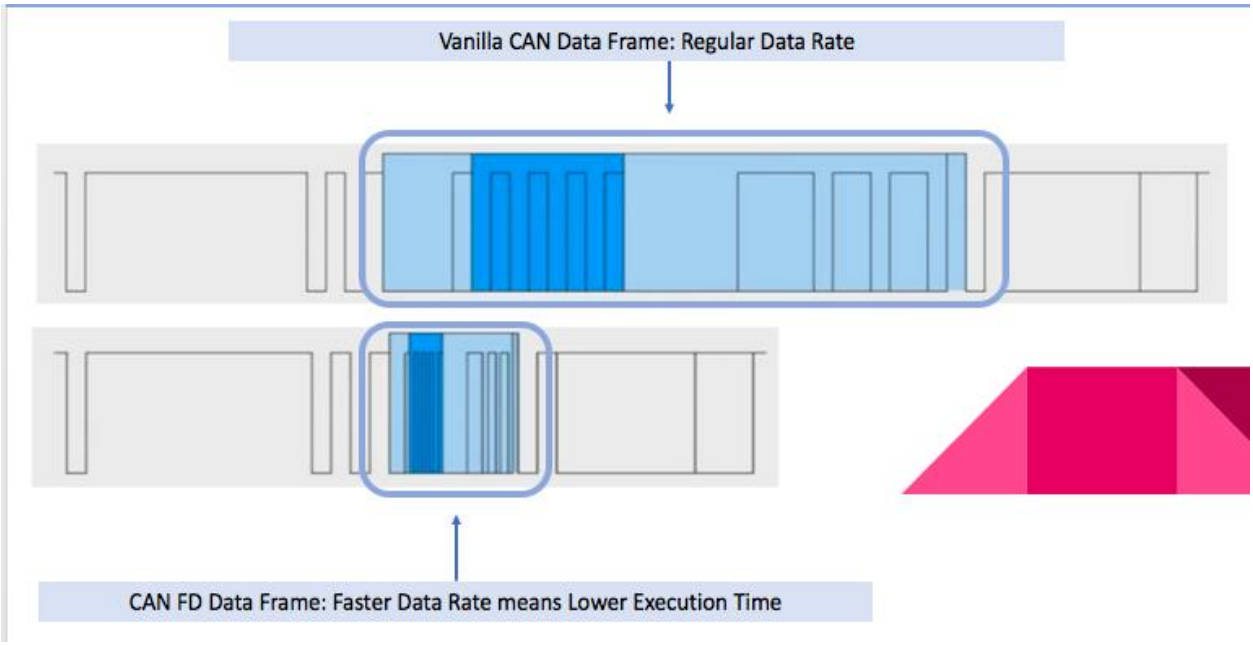

Figure 2.7 : CAN FD Data Compression

The message format in CAN FD has several differences from its CAN counterpart. The format specifications and additions are present in the Extended Data Length (EDL), r1 and r0, Bit Rate Switch (BRS), Error State Indicator (ESI), Data Length Code (DLC), and the Cyclic Redundancy Check (CRC). The message frame format is displayed in Figure 2.8.

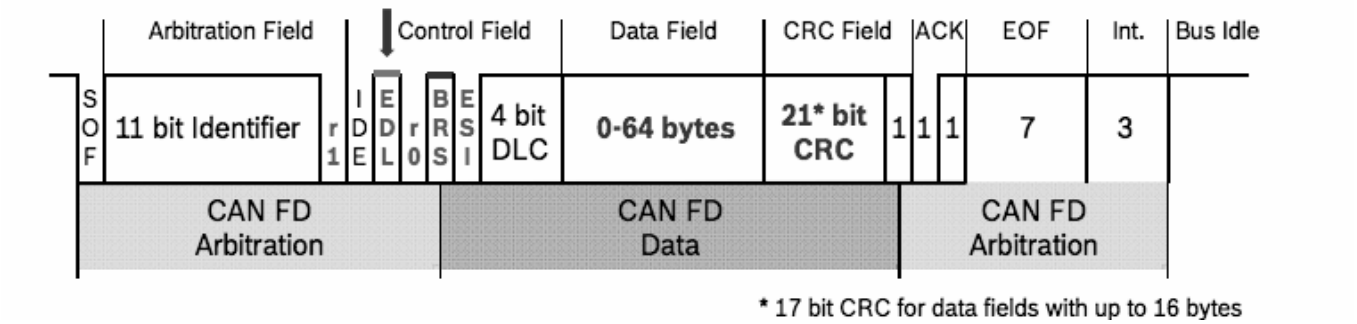

Figure 2.8: CAN FD Message Frame Format

EDL is the reserved bit after either the IDE or the RTR bit in a CAN frame, which is

recessive when transmitted. R1 allows protocol expansion and r0 synchronizes again before the bit-rate switch. Both are transmitted dominant. When dominant a BRS transmission means the bit-rate in the data phase is the same as the arbitration phase, else, this signifies a faster bit-rate for the data phase. Furthermore, ESI when dominant signifies error active mode and when recessive signifies error passive mode. DLC values specify the data lengths of 12, 16, 20, 24, 32, 48, and 64 bytes. Finally, the CRC length depends on the length of the DLC and EDL. The CRC is 15-bits for CAN messages and either 17 bits or 21 bits for CAN FD [23].

2.7 M-ary Amplitude Shift Keying

When working with electricity and digital signals, there is always a waveform which sends the data. In CAN, information is sent in binary, or a wave of 0s and 1s. Each 0 or 1 is one bit of information, however, a stream, 100110 could be split into 6 1 bit values (0-1) or 3 2bit values (0-3, where 0 = 00b, 1 = 01b, 2 = 10b, and 3 = 11b). The purpose of this is to transmit more data at once, sending 212 instead of 100110.

With M-ary Amplitude Shift Keying (M-ASK), we simply tell the signal to send at one of 2^n (where for this case, n=2) levels, instead of binary's 2^1 levels. Now, rather than send 1 bit per symbol, we're sending 2 bits per symbol, thus doubling throughput. The more bits encoded into a single symbol, the less symbols needed to send the same data, and the more efficient each send cycle becomes. This technique is known as multi-level (M-ary) communication [24].

The risk incurred by utilizing this technique lies in the fact our bandwidth does not change, and symbols will become closer together as more must fit in the same "hallway." The ability to distinguish between symbols is paramount, as a misinterpreted symbol represents many misinterpreted bits. The

retention of reliability dictates how many symbols, how many voltage levels, can be used. Nevertheless, so long as the ability to reliably distinguish voltage levels as the intended symbols, it is possible to decode the analog to binary, and vice versa. Some benefits that M-ASK offers are presented in Table 2.4 [24]:

Table 2.4: Pros and Cons of Amplitude Modulation Shift Keying

Disadvantages of M-ASK	**Benefits of M-ASK**
Sensitive to atmospheric noise, distortion and propagation	Modulation and demodulation is inexpensive
Requires a lot of Bandwidth	Implementation of transmitter is relatively simple
Inefficient Power Use	Current in transmitter is smaller than in a Frequency Shift Keying (FSK) transmitter
	ASK requires less bandwidth than FSK

A good visual representation of 4-ary ASK is provided in Figure 2.9. Note that time 4-ary ASK takes half the time as the binary message to convey the same information. To double 4-ary ASK's rate, we would have to use 8-ary ASK, and to double that once more, 16-ary ASK (where 8 and 16 are the number of levels the message could be sent at, each incorporating M=2^n bits per symbol.)

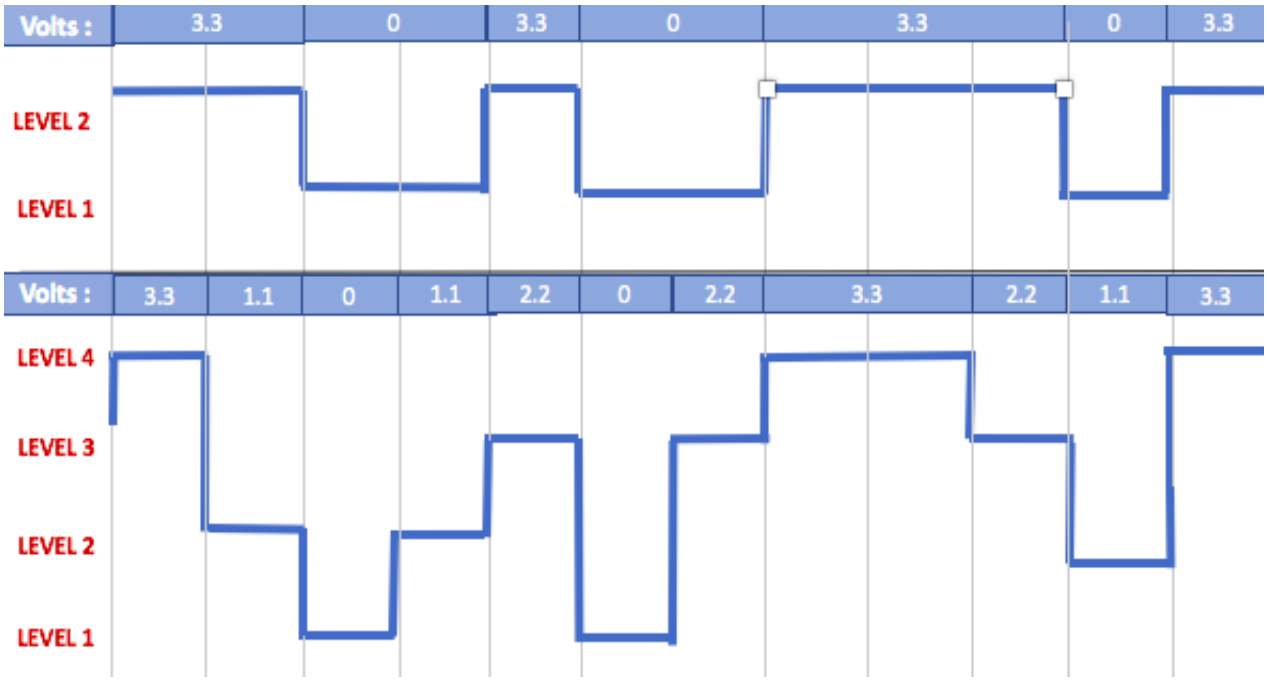

Figure 2.9: Visualizing Four Level Amplitude Shift Keying

2.8 Transient Analysis

In this project, the signal impedances and other physical events within thc bus will affect the integrity of the CAN node signals within the system, jeopardizing the M-ASK modulation process. For this reason, transient analysis is vital to determining the impact on the signals within the physical bus, and make predictions as well as recommendations to mitigate any issues that might occur during the implementation phase.

In electrical engineering a transient response is defined as the response of a system to a change from equilibrium. The transient response is not necessarily tied to "on/off" events, as induced for example by turning a switch, but to any event that affects the equilibrium of the system. The AC signal can be classified as one of three types of damped: UnderDamped, OverDamped and Critically Damped which are shown in Figure 2.8. Similarly, a transient function has several properties: Rise Time, OverShoot, Settling Time,Delay Time, Peak Time and last but not least Steady State Error. Transient analysis is the analysis of

AC circuit in order to determine how non behaved signals affect the circuit and how they impact the signal and calculate that transient function's properties. This analysis is rooted in differential equations and is characterized by the damping ratio (is less than 1, equal to 1, or greater than 1) [25].

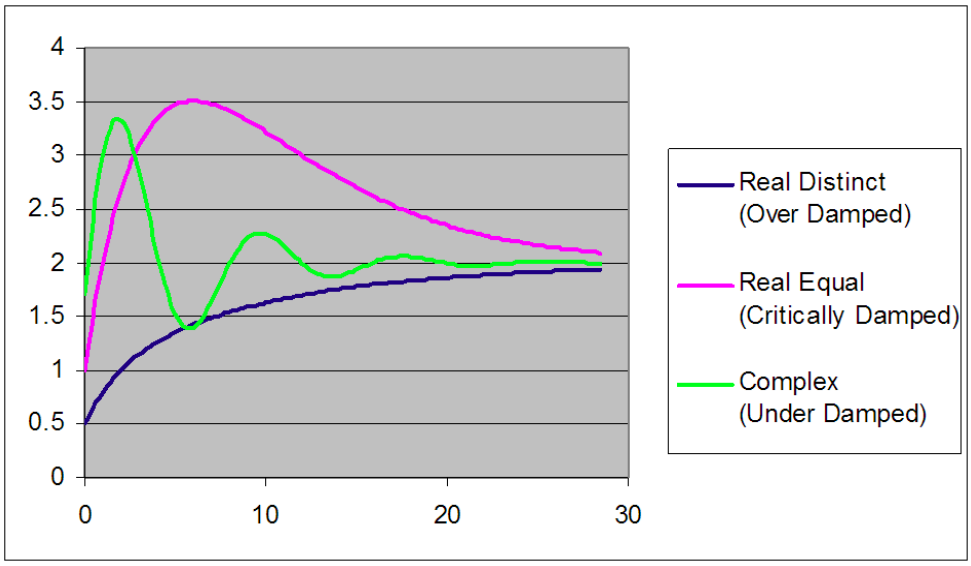

Figure 2.10: Transient Response Behaviors

Of course the ideal digital signal would take the form of a 0 to 1 square wave, however, in most scenarios this signal will not be a perfect. With that being said, the question becomes, which of the three damping types does the CANbus signal embody.

2.8 Chapter Summary

To summarize, CANbus protocol is complex and has strict timing requirements. While CAN and CAN FD are similar, understanding when and where the data-rate increases is key to implementing CAN FD. M-ASK can allow us to increase throughput many fold, so long as there is enough of a voltage bandwidth to support it, however due to sensitivity issues, 2-ary ASK is used.

Chapter 3: The MQP Journey

3.1 Expected Timeline

Various project topics were considered at the beginning of A term 2016. With the options being configuring CAN nodes to work on CAN FD, and developing an algorithm that monitors the state of car sensors. The topic was selected at the end of A term, so that after break, research could begin. B term's objective was to create a plan of attack as well as begin the literary review section of the report.

In C term the development began and the group separated to do their individual parts. This development was expected to take up most of the term (five weeks), then once development phase ended there would be an integration of discrete parts and testing period wherein any bugs moving from digital to analogue would be ironed out. The testing phase was anticipated to only last three weeks and end once the term finished. Once measurements were taken the final phase of the project would be to put final outcomes and conclusions in the report and prepare for the presentation.

3.2 The Initial Proposal

The MQP was created in the Fall 2015 when two team members decided to get the group together to do a project that focused on vehicular security, specifically in finding and exploiting any security holes so that a plan could be devised to close said vulnerabilities. With this interest, they found the most suitable project advisor. Together after discussions on what the focus of the project should be, a decision was made to concentrate on CANbus hacking. As Spring project registration came closer, a new member joined and the project got the name "CANbus Vehicular Security."

3.3 The First Stumbling Stone

With the beginning of a new school year, the group of three met, got organized, and began research which they summarized in a shared Google Drive. This research included how to test various attack vectors. Suddenly, one of the team members decided to change schools, and thusly had to drop out of the MQP, leaving a team of two behind.

From the beginning, there was concern that three members would not be enough to complete a project as complex as this. The loss of a member created an even greater concern and amongst the panic, another member split from the group to seek an alternative MQP project.

With the team down to one member and MQP teams already solidly formed, they could either fight to join an MQP half way through A-term or take in another ECE student who also needed an MQP and would be able to join in B-term. The two decided to join up and remained on the vehicular security project.

3.4 A Second Approach

Before the team changed, a colleague gave the team a possible direction for the MQP and showed the team their base platform and how to use it. The suggestion was to stuff bits of extra information into CAN messages between clock cycles.

This idea, dubbed "extreme bit stuffing," left the group a tad skeptical, but the colleague's conviction led the team to consider it. Still, the project direction was not taken up by the team of three as they much preferred the more direct aspect of attack vector analysis.

When the initial team exploded, the previous plan (which they wanted four people for) had

to be scrapped. This lead the remaining two members to return to the colleague who pitched the bit-stuffing project idea, which they decide to go with. This required serious research into the inner workings of the CANbus and CANbus FD nodes, as well as the CAN protocol as a whole, which took the remainder of A term and majority of B-term.

With focus on catching up, the project direction was not questioned until the majority of the research on both CAN and CAN FD was completed. As it turned out, not only was CAN's bit stuffing method something completely different from the pitched idea, but also it turned out that the "extreme bit stuffing" presented to us was not possible to implement in digital logic. Thusly, this idea had to be scrapped as well.

3.5 The Third and Fourth Project Plans

Having sunk a non-negligible amount of time into research on CAN transceivers and nodes, the group felt it needed to continue on a similar path. During the search for a topic relating with CAN FD, the newer communication network technology that is soon to be the next automotive standard. During this research, it was discovered that in a CANbus, FD messages are only heard by FD nodes and plain CAN messages only by CAN 2.0 (nicknamed “vanilla”) nodes. Based on this, the team decided to make a "translator" between CAN FD and normal CAN messages, which would allow the nodes to hear nodes from another generation as well as their own.

Truncating CAN FD messages to original format seemed easier, since it would only require taking out the extra framing information and running a shorter CRC. After a week looking into that research topic, it was decided not worthy of pursuit, due to the realization that cutting the longer message into parts, sending it as multiple messages at a time (which would occupy the bus,

blocking other from using it), and cherry picking what original nodes could use would be difficult.

The fourth project idea was much the same, but reversed. Vanilla nodes can be reframed to fit the FD frame layout, sent as one short message, and given a longer CRC. This was much more intuitive, since newer networks with FD nodes only needed to retrofit older nodes in. While both could already coexist on a network, they would have only sent messages between nodes of their generation, and not across generations.

This would have been a very useful endeavor since a CAN-FD system is predicted to replace vanilla CAN to become the main communication network. When this were to occur, not all functionalities performed by vanilla nodes may be updated to FD nodes yet. If a company were to require an older node in a primarily FD network, this project would allow them to keep it in the network and communicate with them anyhow, saving the company time and money. This project idea seemed the best since the initial group split, and definitely worth looking further into.

Work and research continued for several weeks, including communication with Kvaser, a company known for their CAN products, until research was interrupted. It was found that despite CAN FD not being public yet, a company had already beat us to it by two months. Unfortunately, this called for the termination of plan four and set us back to square one. This occurred approximately three weeks before the end of B-term.

3.6 The Fifth Plan and Our Tallest Hurdle

To recap, the gears shifted from performing attacks and rating system security to infeasible data stuffing, to translating messages in one direction, then the other. The team was under the impression that with the colleague's showing and tutorial of related work, and the partnership

between Worcester Polytechnic Institute and the company that permitted us to use the CANbus harness, the team would be granted access to use some of the colleague's work as a foundation to build upon. Specifically, wishing to use the working CANbus node created in a previous project.

After scrapping idea four half way through the year-long project, it was extremely difficult to think of where else to go. With so much time dedicated to researching the CANbus it was too late to switch platforms.

The fifth and final idea, loving dubbed "Limbo," was to take a CAN FD node and then encode the messages sent by it with M-ary ASK. In order to accomplish this, CANbus base code was needed, as the implementation would have to occur within the node to be effective.

There are only three ways to obtain code for CAN nodes, as the property was proprietary. You could buy the license for the IP, which would cost six figures and be far out of the allotted MQP budget and price range. You could also scour the Internet to find the singular open source project to build upon or you could write it all by hand. Writing it yourself entails a full year's effort of reverse engineering (which the colleague who guided us dutifully underwent as a graduate thesis).

While the team was given some aid by being permitted to use the CAN harness, not having access to a base node set us back significantly. To even tackle the core of the MQP, integrating M-ASK from the inside, we need to somehow obtain this code. With limited time remaining, the only feasible option was to find and integrate open source code - an effort beginning approximately two and a half weeks before winter break.

This begins the period the Verilog writer refers to as "Limbo." As we knew going in that there was simply not enough time to start a vanilla node from scratch and then also update it to FD, we looked to the singular piece of code we could find free of charge. Hosted on OpenCores, this project was created by Igor Mohor in 2002 and was last updated in 2004. It promised both online and in code comments that it was a

fully functioning CAN node, including arbitration, error checking, error active/passive modes, and CRC generation [26]. Unfortunately this was not the case and a significant portion of time was spent attempting to make this code work.

From before winter break to late February, this code was added to, had sections rewritten, and amassed over 20 local, saved copies. Not once did the testbench simulation reflect any change. Switching from ISE Design Suite to Vivado to ModelSim did not change the simulation. Changing the testbench to only run certain tasks did not change it, nor did completely rewriting the 2,000+ line testbench. Completely removing it and giving it the task to idle did not even change it - something which frankly shouldn't have been possible.

Yet, neither careful planning nor head-first charges into this brick wall of a code showed any signs of progress. Let us restate that this is the base, the foundation, that we needed to even start work on the real project. The MQP assumed a functioning CAN base prior to work and as of February 22nd, most the way into C-term, arrived, the team called a meeting, sat down, and proposed a different approach.

The problem of being unable to obtain a fully functional CAN node, let alone FD node, in code form would plague any team put on the MQP. Until someone either put in the year's effort into making a node for WPI or purchased the code (costing 6 figures+), it would simply not be possible to work with a node directly. Nodes bundle in their own transceiver and receiver, meaning that the M-ASK encoding scheme would have to be placed within the node, between the microcontroller and transceiver, to properly affect throughput.

3.7 The Sixth (and Final) Method of Approach

Since the largest obstacle in MQP iteration five was the node IP itself being out of our reach, the sixth MQP kept the same goal of M-ASK encoding but instead made a smaller base to work on rather than

the full base with arbitration. While non-ideal, time was limited and reverse engineering a fully functioning CAN node is a year-long endeavor.

Instead of focusing on fully functional CAN nodes, it was decided to cut our losses and make some nodes which could send and receive CAN messages that were properly framed, but would not generate or check CRCs, nor trigger errors (which would terminate transmission and possibly put the nodes into error active or error passive states.) They were dubbed “base2CAN” nodes playing both on their function and their binary nature

Since work was primarily involved with the messages and the physical medium they went through, worry about arbitration or CRCs wasn’t as crucial. So long as the concept could be proven with correctly framed CAN messages, it follows that it would hold when dealing with true CAN nodes. With the ability to provide proof of concept as well as possibly placing it on the physical bus in time, the MQP could still be fulfilled with only eight weeks before project presentation day. That the project took its final form so late into the year highlights the need for this chapter. It has indeed been a mercurial beast of a project.

3.8 Chapter Summary

To summarize, the direction of the MQP has shifted many times, however, the team was able to overcome the drawbacks associated with this. While communication issues left the final design without a 100% working base to launch from, base2CAN was created to solve this issue, allowing forward progress.

Chapter 4: Design Implementation

4.0 Introduction

In the following chapter the discussion will transition to what equipment and tools were available for this project, and which were utilized. Also there were two different modules that this project consisted of, the MASK-ing and the Sounding, both of which were done separately and have their own design sections dedicated to them. Finally, the integration process and results are discussed after both modules, the MASK-ed base4can signal and the CAN nodes are put together

4.1 Hardware and Software Utilized

This project required the use of various hardware and software, some of which were necessary for the sounding of the CAN harness, others which were necessary in creation and M-ASK modulation of a base2CAN message. The following two tables displays all the vital components that were involved in the execution of the MQP.

Table 4.1: Hardware Utilized in Sounding the Physical Bus

Hardware Purpose	**Hardware**
A physical CANbus that can be measured and analyzed	CANbus harness
Digital and Analog Logic Analyzer	Saleae Logic Pro 8
Sends an impulse signal to a CAN node	Function Generator
Allow the signal to bridge through one node on the harness to another	2 x Raspberry Pies 3 Model B with CAN Shields
Used to connect multiple CAN nodes' high and low pins to the CAN Shield and Function Generator	10 x 28 Gauge Wires and a Breadboard

Table 4.2: Software Utilized in Sounding the Physical Bus

Software Purpose	**Software**
Acquires and analyzes the data collected by the Logic Pro 8	MATLAB
Displays digital and analog signals at various nodes	Logic (by Saleae)

Table 4.3: Hardware Utilized in Creation of base2CAN and M-ASK modulator

Software Purpose	**Software**	**Hardware Purpose**	**Hardware**
Used to create M-ASK & base4CAN messages	Verilog	Two simple ADC and DAC peripherals - one for each FPGA	2 x Pmod ADC1, 2 x Pmod DAC1
		Implement code which allows the sending and receiving of CAN messages (both for “base2CAN” and for “base4CAN” nodes.)	2 x Nexys3-Spartan6 FPGAs

4.2 Design Methodology: Creating base2CAN

The development of base2CAN modules quickly was underway, three iterations deep by the end of the first week. The base2CAN modules were to perform three functions as reasonably close to CAN as possible. For this to be accomplished, the Register-Transfer Level of digital logic abstraction (RTL design) was used.

Functionality was split between a datapath and controller. The datapath is the muscular structure of the circuit - performing the tasks assigned to it. The controller uses state machine logic to determine what steps the circuit should occur next. When both are interconnected, the controller watches for flags and sends the appropriate signals to the datapath. The datapath then executes its operations based off of these control signals and updates flags that tell the controller its current status. A good visual of this weaving is in Figure 4.1.

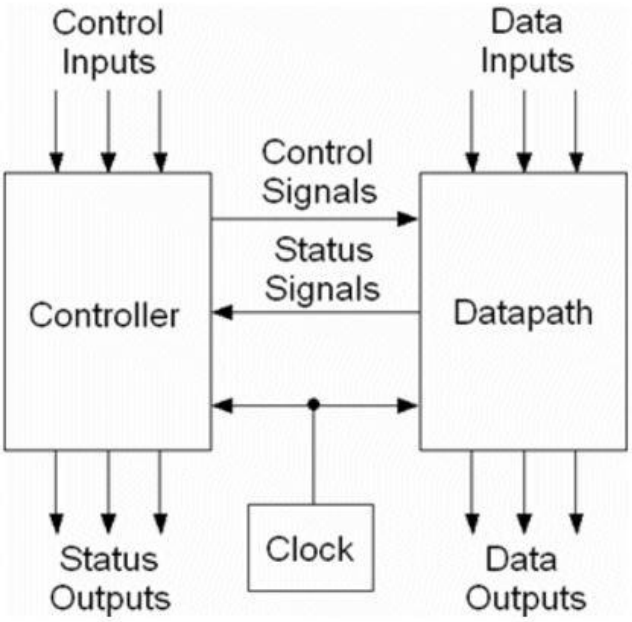

Figure 4.1: Configuration of datapath and controller in RTL design

Both the datapath and controller run off of the same clock, and both can each have inputs and outputs, though neither must. base2CAN's controller diagram is as Figure 4.2 shows. Each circle represents a state, and each line and transition between states.

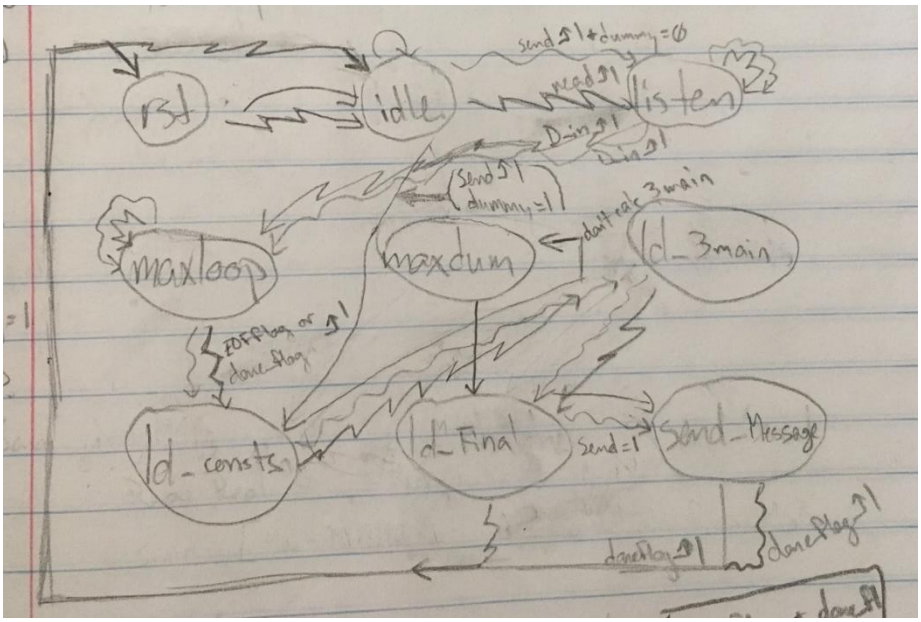

Figure 4.2: A state diagram of the base2CAN controller

In Figure 4.2 you see three different types of lines, one straight, one wavy, and one jagged. This is because base2CAN has three modes of operation: sending a predefined message (straight lines), receiving and storing a message (jagged lines), and receiving, performing operations on, and sending a modified message (wavy lines). Split into different diagrams, the state machine diagram is as shown in Figure 4.3.

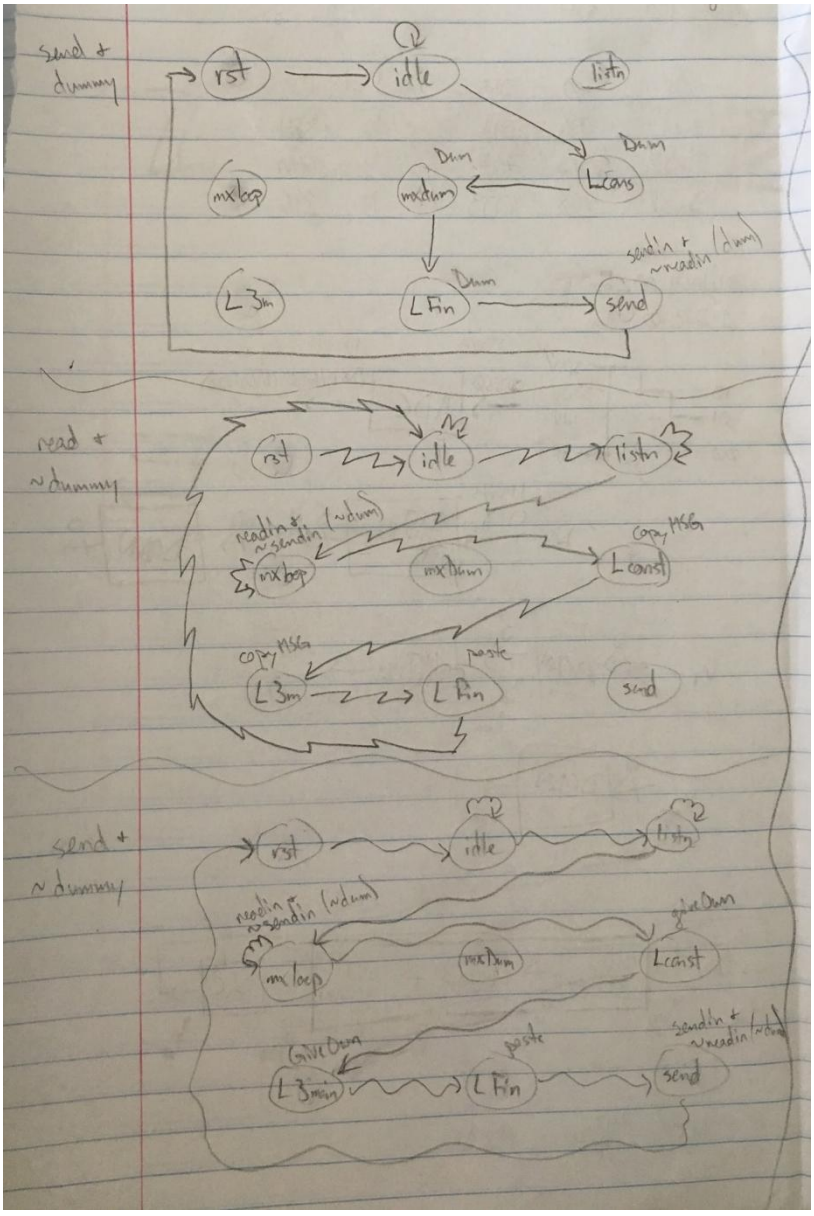

Figure 4.3: A separated diagram of each function

The first function sends a predefined message where some parts are loaded with predefined constants before the rest are calculated, the node frames up the sections into proper format, and sends it serially on two outputs (the normal on CAN high and its inverse on CAN low). This is to make sure that, first and foremost, it frames up information correctly and can send without issue.

The second function base2CAN performs is to read an incoming message and store it. This ensures that when put on the bus it can successfully read incoming data and store it for processing. If this were a more genuine node, this function would also check the message for errors as it comes in before storing it, via frame checking and bitstuffing logic. Nonetheless, it is still imperative to confirm the message was received correctly. This will be especially true when shifting from M-ASK and back is added in.

The third function it will be able to perform is to read in an incoming message, react to it, and send a response. For our purposes, it will take in the initial message, edit the data payload, and send it back out on the bus (using the new sender's ID and a new CRC.)

First the design will be designed, simulated, debugged, and refined solely virtually. This is the advantage of Verilog and other HDLs - physical access to the bus is not required to refine the design. Figure 4.4 shows the receiving, modifying, and sending of a message, as per function three of the base2CAN node which here is divided into two parts due to length.

Figure 4.4: Read, perform operation on, and send message

The Serial_out signal is another node sending information, while D_in shows the base2CAN node receiving said data. D_outHi and D_outLo are, as they sound the two CANbus data lines, Lo being a reciprocal of Hi; these are the signals that would be output on the CANbus. readin and sendin are control signals from the controller telling us (through LEDs) what the whole system is currently doing. readin is high only when D_in has data to read and goes low when complete, the conversion happens, then sendin goes high while sending the new message on D_outHi and D_outLo. (Note: readin and sendin are purposely misspelled because reading and sending are both ANSI standard Verilog constructs and thus cannot be used.)

With this, the functionality of base2CAN has been proven a success in reverse engineering of the CAN protocol base. Message can now be sent, received, and heard, modified, and sent in binary.

4.3 Design Module - Sounding the Bus

In order to expect the measured base4CAN at the receiver be the same as the message at the transmitter, the physical bus has to be assessed. The objective in sounding the CANbus was to determine

how much impedance exists between each node and how that affected the CAN signal. For example, hypothetically, if there is a loss in voltage between node1 and node2 from 1.5V to 1.15V, when M-ASK encoding some part of the message may be averaged down. Thus, impedances due to reflections within the bus may be disruptive in that it messes with the encoding and decoding of the message, and corrupts the data which could be detrimental with a vehicle.

Already, there are two 120 ohm terminating resistors used that were selected when standardizing the CAN protocol to match the characteristic impedance(resistance that occurs in AC signals) and eliminate it. This helps prevent signal distortion or uncharacteristic resonance behavior. However, probabilistically, not all impedance can be removed with just two termination resistors. The goal in sounding the bus is to determine the difference between the ideal and actual voltage measured, and analyze the resonance behavior associated. Depending on the results, if the impedance turns out to have a greater effect on the bus then acceptable for the MASK-ing protocol to work, measures would be taken to diminish it.

To sound the bus, it was necessary to familiarize oneself with the physical CAN harness, and the pinout of each of the nodes. The K43 connector for example, shown in Figure 4.5, is one of the smallest connectors, containing only 10 pins.

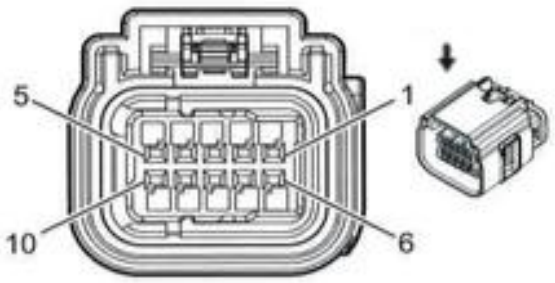

Figure 4.5: Example CAN Node Connector [27]

This connector can be seen in the automotive test-bed, which is shown below. The physical harness that contains all the connectors to which select CAN ECUs are attached is shown in Figure 4.6. The connection are made between the connectors via unshielded twisted pair cables[28]. The connector falls in the group of connectors that belong in node 3 shown in Figure 4.6. The K43 in a vehicle serves as the Power Steering Module. The test-bed that was accessible for this project contained six nodes as shown in the figure. The way the harness was was set up was that it was strapped onto a metal fence using zip ties and clustered together into various sections. (Using a conductive fence, may not have been the best idea due to the fact that we were applying current to the connectors). See the figures below for the entire test-bed setup and close ups of several connectors.

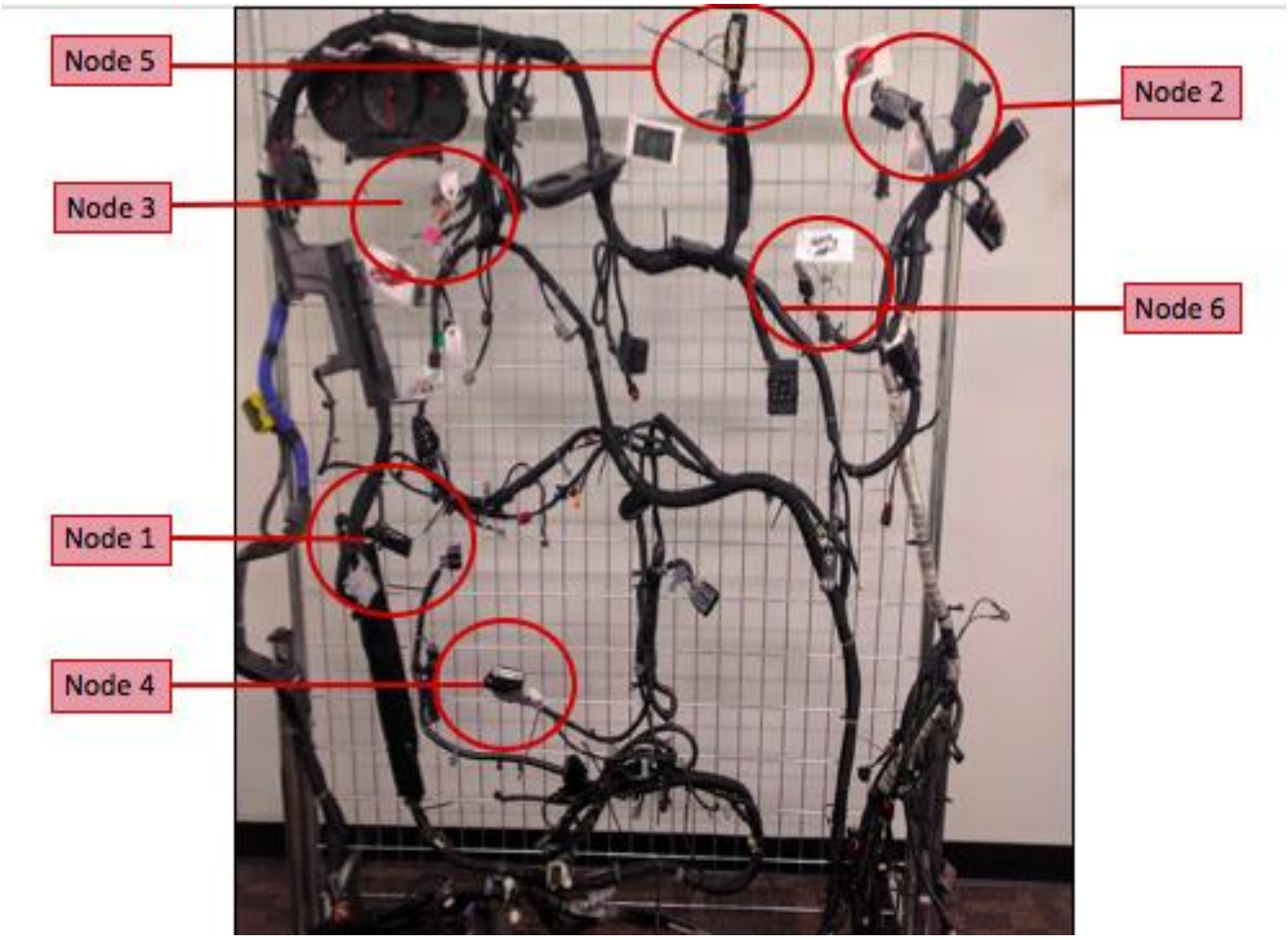

Figure 4.6: CAN Harness with 6 Nodes

Figure 4.7: Close up of CAN Nodes available on the CAN Harness

To collect data from the harness, specific pins within each node had to be measured. To determine which pins of the connectors were important it was necessary to look under the specific module (in this case K38 Power Steering Control Module) and check for the pin with the desired functionality then check what pin number corresponds to it. In some cases the connector consists of > 60 pins so finding the particular pin on the physical connector could be time consuming. Figure 4.7 is shown an example of the Power Steering Control Module pinout. The two pins used for sounding the node include the CAN low and CAN high pins which are encircled in Figure 4.6 [27].

K43 Power Steering Control Module X2 (NJ2)

Pin	Size	Color	Circuit	Function	Terminal Type ID	Option
1	0.5	D-BU/YE	6105	High Speed GMLAN Serial Data (+) (2)	I	-
2	0.5	WH	6106	High Speed GMLAN Serial Data (-) (2)	I	-
3	0.5	D-BU	2500	High Speed GMLAN Serial Data (+) (1)	I	-
4	0.5	WH	2501	High Speed GMLAN Serial Data (-) (1)	I	-
5	0.5	WH/D-BU	5986	Serial Data Communication Enable	I	-
6	0.5	D-BU/YE	6105	High Speed GMLAN Serial Data (+) (2)	I	-
7	0.5	WH	6106	High Speed GMLAN Serial Data (-) (2)	I	-
8	0.5	D-BU	2500	High Speed GMLAN Serial Data (+) (1)	I	-
9	0.5	WH	2501	High Speed GMLAN Serial Data (-) (1)	I	-
10	-	-	-	Not Occupied	-	-

Figure 4.8: K43 Connector PinOut

Various attempts were taken in gathering data from the physical harness. The hardware and software needed for this part are provided in Table 4.2. The first flawed attempt to sound the bus was to interconnect the CAN high and CAN low signals of various CAN nodes of the harness together without the CAN shield. This was wrong and gave inaccurate results, as there needed to be a CAN shield with terminating resistors to serve as the starting and ending nodes on the bus.

To acquire measurements various methods were attempted and utilized. The first was to use the oscilloscope, however, the scope provided was an old Tektronix model "TD2010" and did not have a USB only an RS232, so for getting the data, python pyvisa library was seen as a good option. The MAC used for collecting data only had a USB so to transfer data and RS232 to USB was purchased and the software NI-VISA was used to troubleshoot and setup the initial connection. This software however, was not recognizing a device at the COM and was timing out or running indefinitely (regardless of the timeout time specified). It was concluded that the software and the library were only compatible with straight USB or RS232 connections. Following this discovery, an alternative idea for gathering data from the bus was to use an Arduino Mini. This would work because the measurements would be taken straight from the Arduino to the bus, by connecting the Arduino analog pins to to the CAN node pins and running code that

collected and wrote this data to a serial monitor and serial plotter. The only issue with this method was that there were a very limited number of analog pins, so there would be a limit to how many nodes can be measured at a time. Also, in those attempts the analog data was mapped to numbers 0 to 255 which made the results harder to interpret. That being said another option was in using the Saleae Logic Analyzer [29]. This device was a good alternative because it had software compatible with MACs, with all the parts included, along with a high sample rate and eight leads for an increased measuring capacity.

Table 4.4: Benefits and Setbacks to Using Various Analytic Equipment in this Project

Logic Analyzer: Saleae Logic Pro	**Microprocessor: Arduino Mini 05**	**Oscilloscope 2: Tektronix TDS 2004C**	**Oscilloscope 1: Tektronix TDS 210**	
Many test leads, user friendly, simple to transfer data	Most sensitive data collection, easy to transfer data	Has USB port, Transient Behavior clear	Transient Behavior clear	**Pros**
No grids, signal averaged, large data size- takes long to transfer	Limited number of analog pins, data maps to range	Few leads, simpler to transfer data but still not automatic	Few leads, difficult to transfer data, Older: no USB port	**Cons**

The second attempt was to setup the circuit on a breadboard with the function generator connected to one of nodes on the Raspberry PI's CAN shield. The nodes were connected in series on the breadboard with both raspberry pi nodes, with the resistors, serving as the termination nodes. The function generator settings were as follows : the signal amplitude was set to approximately 3.3V, the signal generated was a square wave, the frequency selected was low, approximately 100 Hz.

Once the setup was complete, the Saleae Logic Pro 8 was connected, with each positive lead (eight possible leads) connected to points of interest to measure ex. lead_1 to the function generator, lead_2 to K-83 CAN High pin, etc. It turns out after Sounding the CAN node Harness that no major reflections occurred within the bus at a slow rate.

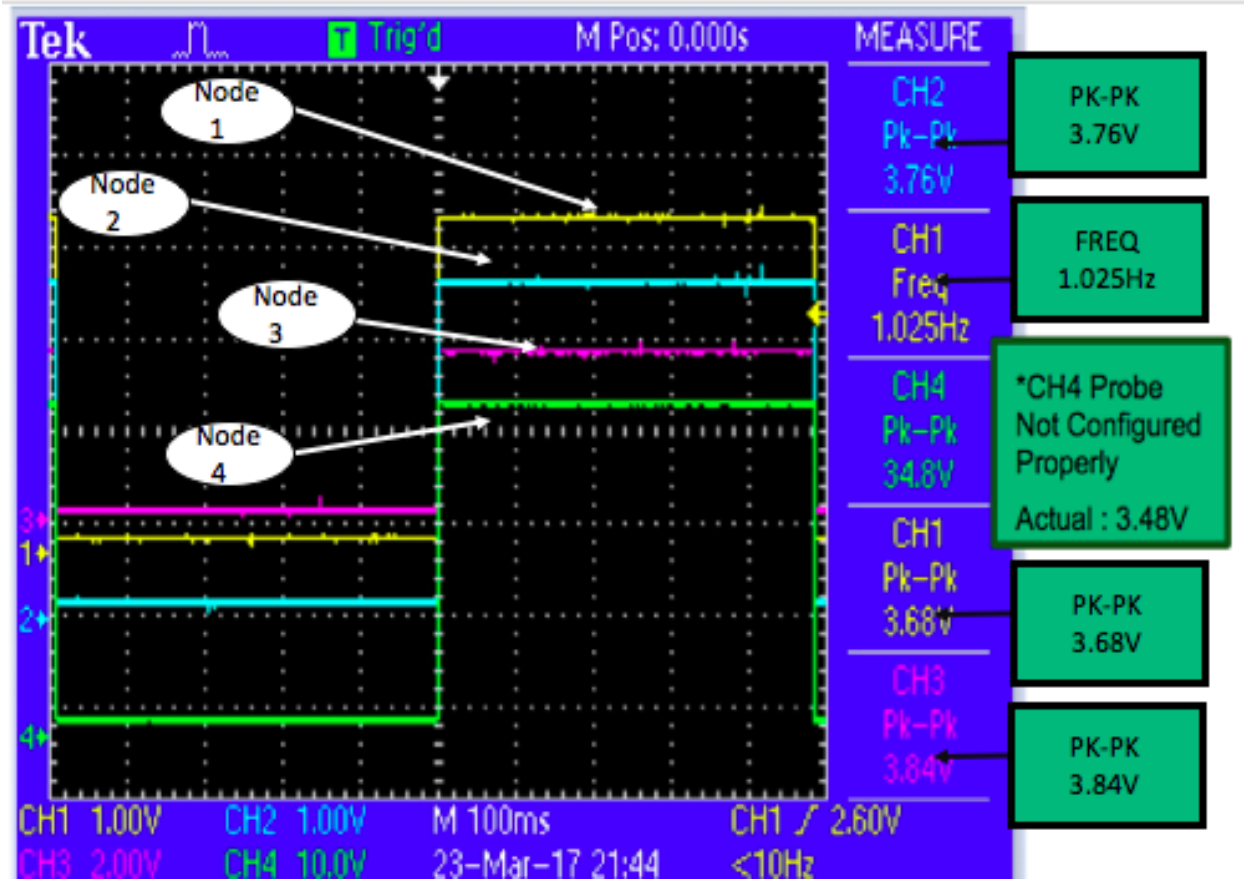

Figure 4.9: Signals Generated from four different CAN H nodes

As you can see in this case the Pk to Pk between CH1 through CH4 range only by about 0.1V. This is not a significant difference and characterizing the nodes based solely on this would likely give inaccurate results. This, however, could be hypothetically be a way to better secure this network. The plot below shows the same nodes and their transient behavior.

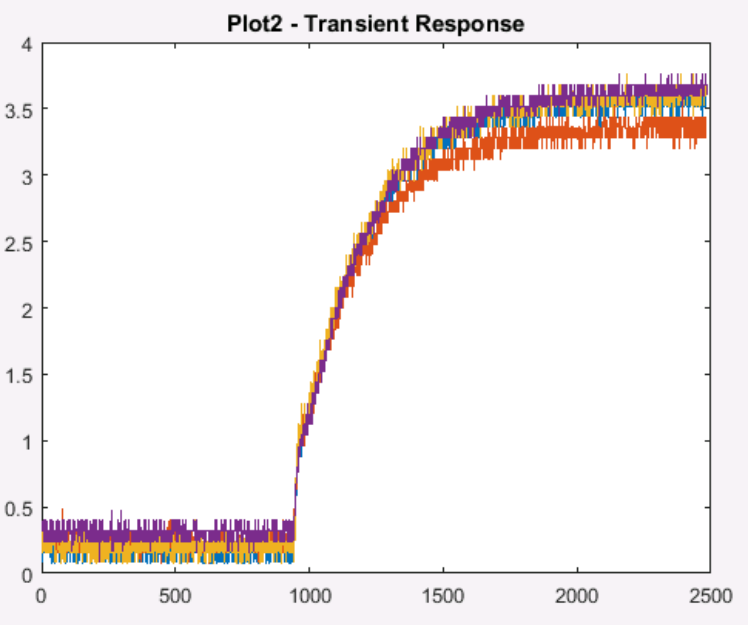

Figure 4.10: Transient Behavior for four CAN nodes

The transient behavior is overdamped and for various nodes, this behavior is almost indistinguishable. Research indicated the CAN node signal transient behavior would be most affected by interference and reflections when several conditions are not met including:

- Proper Resistors Not Used for CANbus termination
- Material For Twisted-pair Line is Not Uniform
- Material For Connectors Has Large Impedance
- Branch Length Does Not Follow Specifications [Table B1]

The main conditions that help regulate the impedance within the CANbus include the following, which are shown above: resistors (which have to match the characteristic impedance of the bus) the material used for connectivity between nodes and bus, the material the connectors are created with and the

length of the bus [30]. Out of the aforementioned, the termination was the only physical characteristic that could be tweaked, through varying resistor sizes or varying the termination method [31].

4.4 Design Module - Transceiver Setup and Procedure to Test M-ASK Encoding Theory

Before the MASK modulation is implemented and combined with base2CAN, the theory can be tested and observed using a power supply and an oscilloscope.

Process: with a CAN Transceiver given in Figure 4.8 connected to the CAN Shield nodes on the Raspberry PI, a Power Supply input as specified, and receiver connected to the oscilloscope the voltage the digital CAN output can be monitored as the input voltage of the signal increments.

As demonstrated from the TI DataSheet shown on the right side of Figure 4.8, the signal will be received as 0 (Dominant) when the voltage difference is 2V between CAN L and CAN H. Also, the digital signal will be 1 (Recessive) at approximately 2.3V [32].

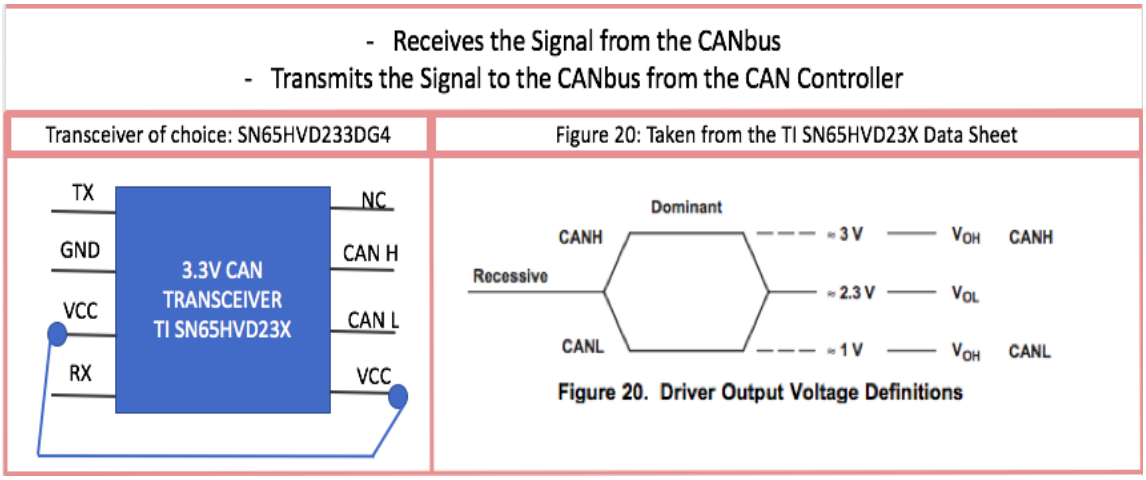

Figure 4.11: Incorporating the CAN Transceiver to Drive the CAN Signal [32]

In this case, to eliminate extra complications, only CAN H will be used for our measurements.This will mean that the signal change from recessive to dominant will happen at > 4V. While this cannot be

produced by the FPGA due to the voltage restrictions, it can be tested using a power supply. With the industry transceiver in use the signal should be measured to change from recessive to dominant when the supply voltage exceeds 4V. One can when trying to measure voltage levels using only an industry transceiver, is that the only signal levels that are recognized are 0 and 1. Shown below is the setup done when using the TI standard 3.3V CAN transceiver.

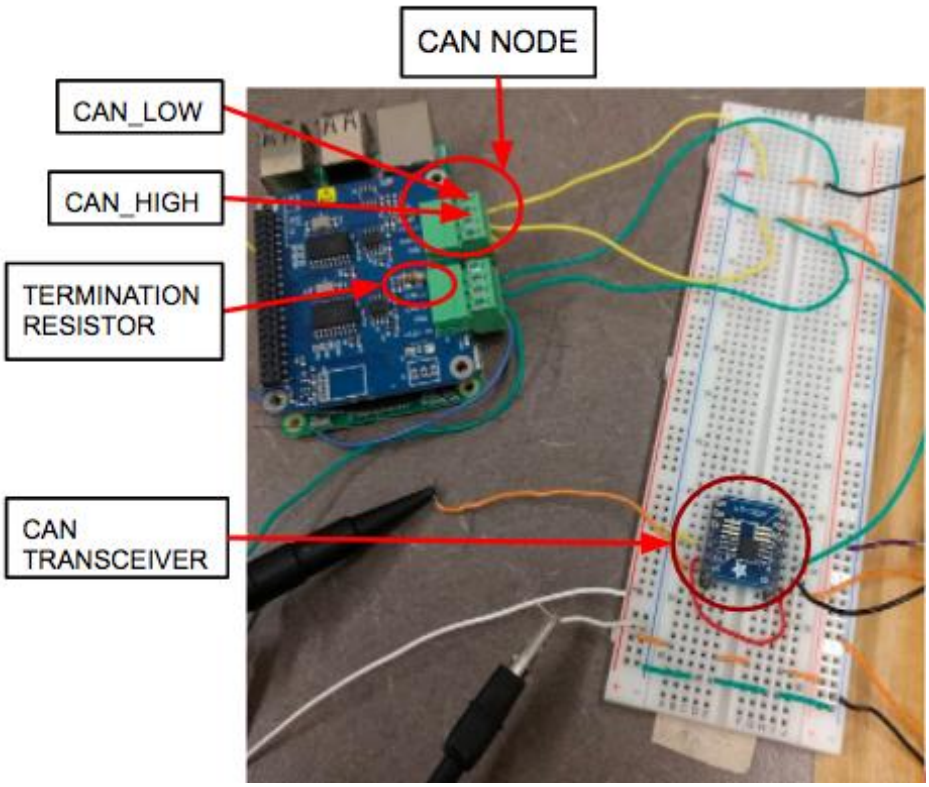

Figure 4.12: Sounding / Integration Setup

Thusly, even when you have a difference in voltage of around 8V, the SN65HVD23X transceiver circuitry is custom to detect only two voltage levels. Ideally, a custom analog transceiver could be created to detect more than two voltage levels. Once the transceiver is connected to the bus and the supply is increased the receiver should distinctly display 0, 1 and +1 levels. Without the controller needing to be connected, the MASK modulation could be proven to work on CAN nodes.

4.5 Design Methodology: M-ASK and base4CAN development

Despite working on a wired bus with binary signals, a very analog approach was taken, encoding payload using wireless communications techniques. While a bit unconventional, encoding the binary signal with 4-ary ASK would provide twice the throughput of the binary signal. M-ary ASK is one of the schemes with the highest promise, especially considering the short time-frame of the project. It requires less hardware and computation time than match filtering would have if implemented, requiring an additional FPGA and FD receiver per symbol.

Rather than send a 0, 1, 1, and 0, in four clock cycles, you can send 01 and 10 with 4-ary ASK, as demonstrated in Figure 4.13 below.

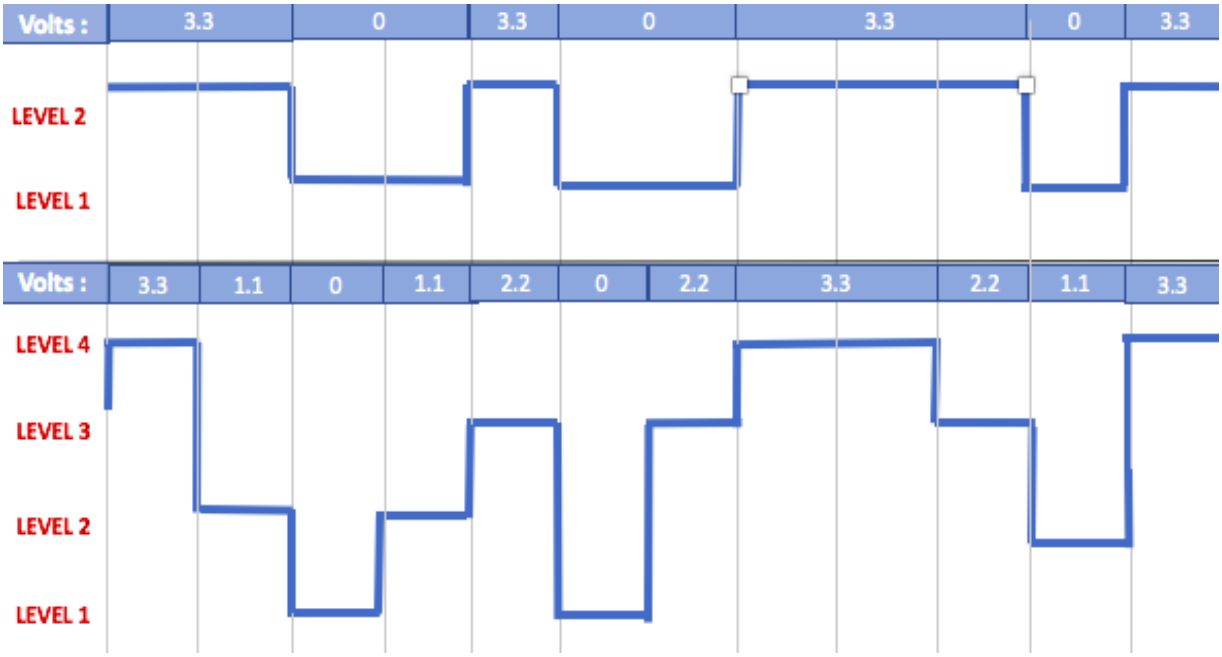

Figure 4.13: 2-ary ASK (binary) versus 4-ary ASK signal voltages

While 6-ary or 8-ary ASK would have tripled or quadrupled the original CAN output, this got far too close to level blurring, or being unable to tell the difference between each step of voltage. Should two voltage levels be confused due to closedness and/or noise, the message would be distorted and decrease

signal integrity. For these reasons and that our allotted voltage range was quite small, 4-ary ASK was chosen.

To encode messages in M-ASK, it was first thought necessary to have two MSP430 microcontrollers per FPGA node. As FPGAs are logic 1 or 0 and are not capable of in-betweens, the multiple levels of voltages required for M-ASK implementation would need to be produced elsewhere. This led to the simple solution of turning the digital output of the base4CAN node into an analog signal before placing it on the bus. This would send it through a microcontroller and gain the ability of adjusting the voltage output on CAN high and CAN low. An analog to digital conversion would undo the process via a second microcontroller before reaching the base4CAN node again.

While the MSP430 would not double the throughput as originally intended, it would offer a solid base to incorporate into a follow-up MQP. For the throughput to be positively modified, the M-ASK transformations would need to occur inside the node (between the microcontroller and transceiver) before transmission. That is, instead of the node sending binary signals, it sends 4-ary out of the node's own transceiver. Sticking a M-ASK module onto the end means it would have to wait to get two binary outputs from the CAN module regardless and negate the change to throughput.

However, it quickly became clear that this hurdle might be overcome if, rather than two MSP430s per node, a ADC and DAC peripheral be used. There exist standalone, low-power ADC and DAC modules that can plug directly into the Nexys 3 Spartan 6 FPGA, as seen in Figure 4.14. Due to their direct I/O, rather than serial wired connection, not only would space and resources be saved, but also the synchronization of separate boards, each with their own clock, would not have to take place and less wires would be connected to the physical bus.

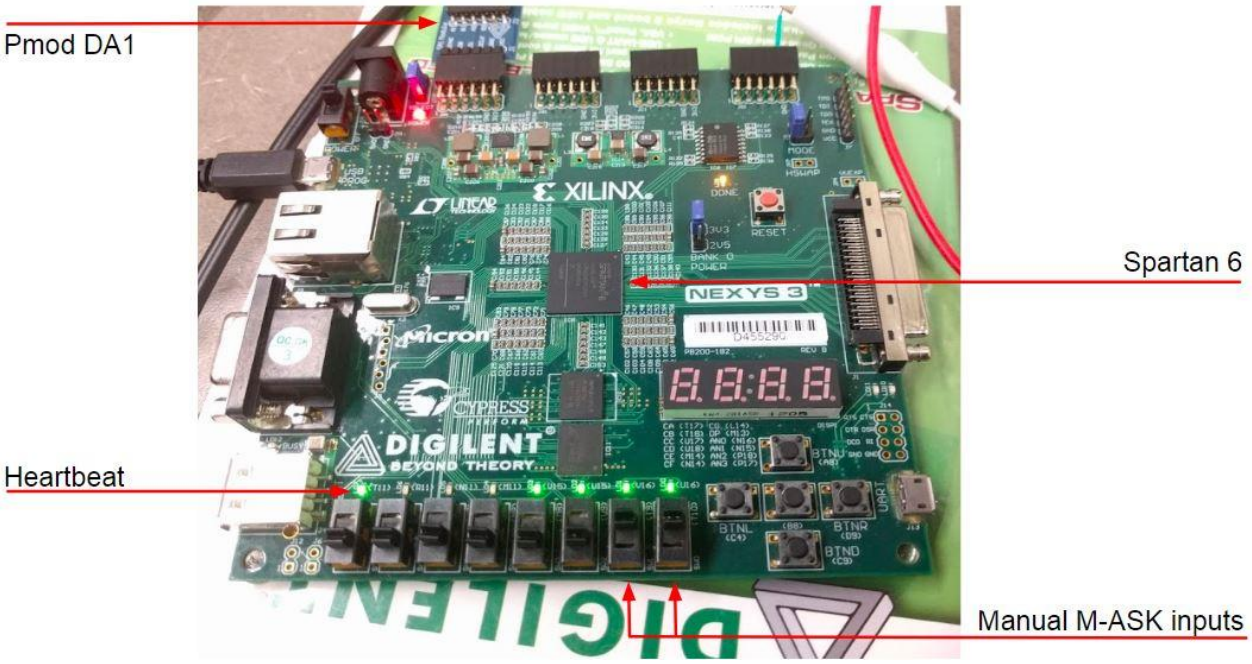

Figure 4.14: The Nexys 3 Spartan 6 FPGA with Pmod DA1 peripheral (AD1 not shown.)

In order to test the PMod AD1 and DA1, isolation testing was done. Before connecting M-ASK encoding to the base2CAN system, we must first test M-ASK on its own. Figure 4.14 shows the board setup with peripheral locations. Figure 4.15 below shows the successful operation of 4-ary ASK. To accomplish this without connecting to a CAN message, instead uses two input switches on the FPGA. As shown below in yellow, 4-ary ASK on its own is operational - 0V levels being hidden behind the blue ground line.

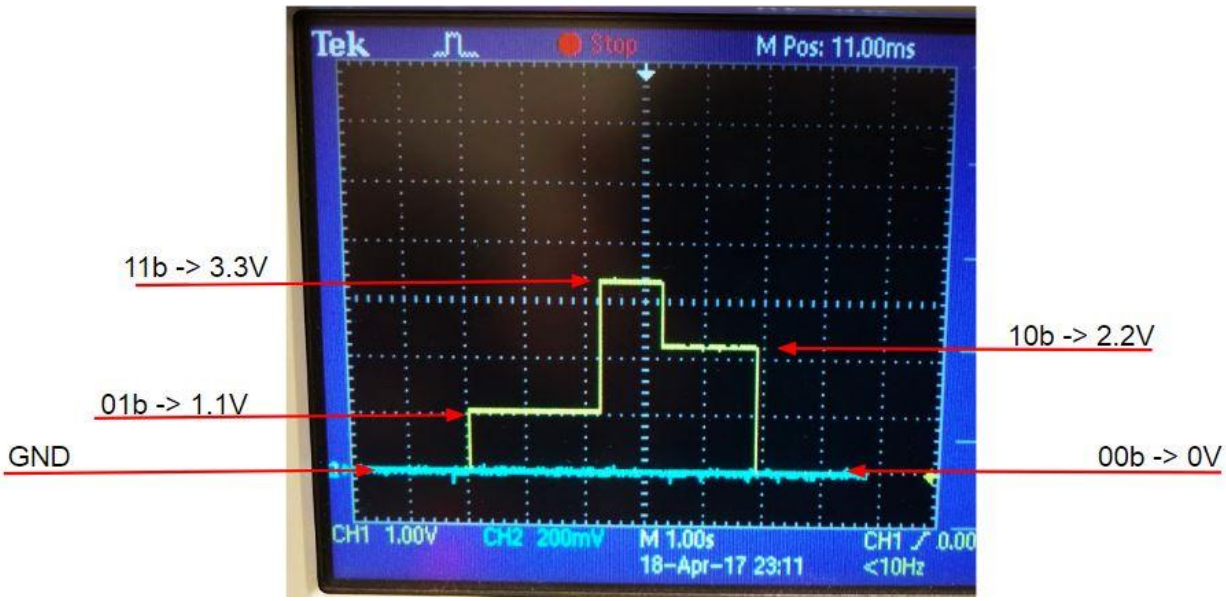

Figure 4.15: MASKed Digital Waveform

When testing Verilog in simulation, the code was not calibrated to the real bus since the virtual bus does not experience any losses. Regardless of the coding scheme, bus-specific fine tuning to the physical bus was necessary to overcome losses within the different nodes of the bus. Due to the high quantity of branches and sensors, impedance losses were surely a stumbling block to overcome when calibrating the base4CAN code to work on the physical bus. To implement the finished product on the harness, it was necessary to sound the bus, determine the losses between nodes and create filters that minimize this loss.

When it came time to integrate base2CAN with 4-ary ASK, creating base4CAN, a change or hierarchy was needed. To give a sense of the magnitude of hardware description language involved in the final version of base4CAN, we have Figure 4.16 below as well as the majority of appendix A. Inputs and outputs now must each be conducted at 2 bits per clock cycle, needing a restructuring of the inner workings.

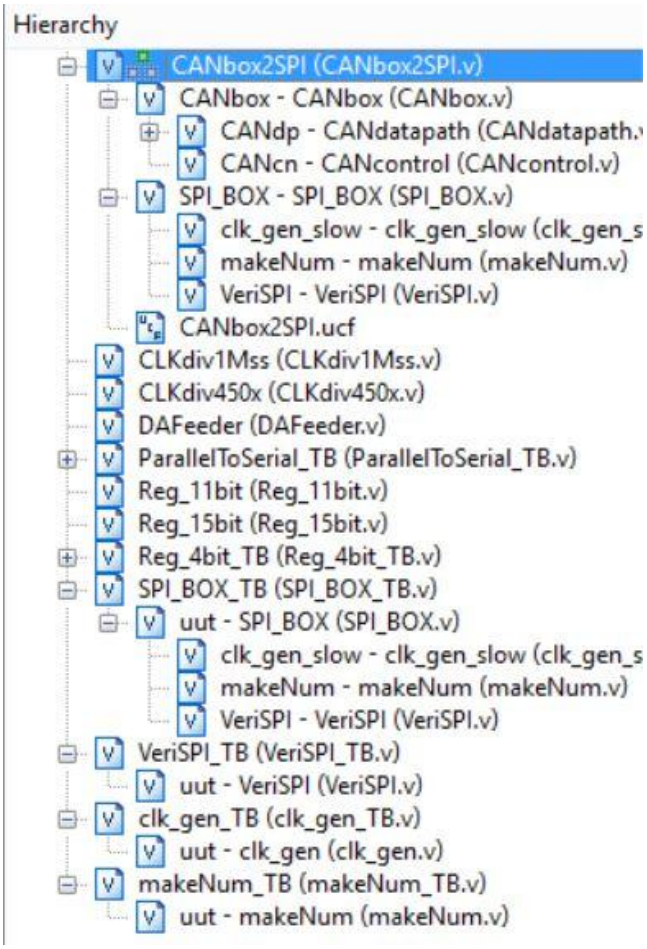

Figure 4.16: CANbox2SPI is the code that drives base4CAN

That being said, the ADC conversion process integrated successfully, being able to send a full CAN message at 0, 1.1, 2.2, and 3.3V -ASK, as shown in Figure 4.17 below. Figure 4.17 is not controlled by external switches like Figure 4.15, but internally by a true base4CAN message.

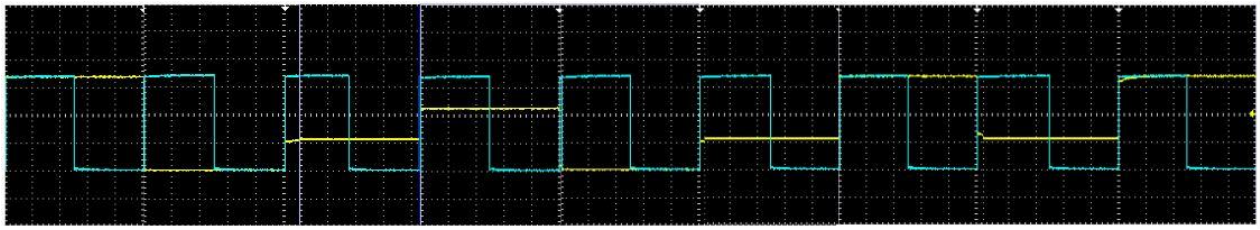

Figure 4.17: Oscilloscope view of base4CAN in action.

4.6 Summary

Through hardships, we have developed, implemented, and integrated our custom base2CAN and base4CAN nodes. M-ASK modulation proved effective at doubling throughput, as shown through Figure 4.17, though a current amplifier is needed to prove physical bus communication. Nodes on the CANbus have been sounded and analyzed for impedance mismatches and reflection distortions.

Chapter 5: Results

5.1 Expected Results

The expectations were to successfully implement M-ASK modulation on existing test nodes. Before implementing the M-ASK modulation digitally (within a Verilog environment), the idea was to prove the concept of M-ASK modulation. If there was sufficient time a way to prove the concept was through creating a custom made analog transceiver. Another task that would be necessary was sounding the CAN harness and if needed, creating a filter that minimizes signal loss, so that this modulated signal would not be wrongly encoded. The M-ASK modulation method would be tested within Verilog but would then be implemented on the physical bus. It was expected that this would create a faster throughput when implemented properly. Ultimately, encoding and decoding on the physical CANbus is a very useful proof of concept.

base4CANnodes would to be synthesized and simulated in Verilog and prove sending and receiving capabilities. Microcontrollers would then simulate the M-ASK modulation process, and implementation onto physical boards would begin. Once proved in discrete parts, the assembly could come together and be placed on the physical CANbus itself.

The expectations were that if a TI transceiver was used you would not be able to see two levels. If the custom transceiver was used, once the signal is sent through a CAN node, the received signal when measured would have multiple signal levels (at least three for proof of concept) 0, 1 (the original) and 1+ (the extra signal level that corresponds to the dominant signal), all of which would be compatible with the bus. The functionality depend on transceiver setup.

5.2 Obtained Results

Ultimately, the individual parts of the projects were completed. These were: sounding the bus, creating a base2CAN, transforming it into a base4CAN. The integration of the DAC digital output to the CAN shield on the Raspberry Pi was more challenging, as the output signal with the configuration was not exactly what was expected. The reasons seemed unclear initially. Below is the oscilloscope image of the output measured after the integration.

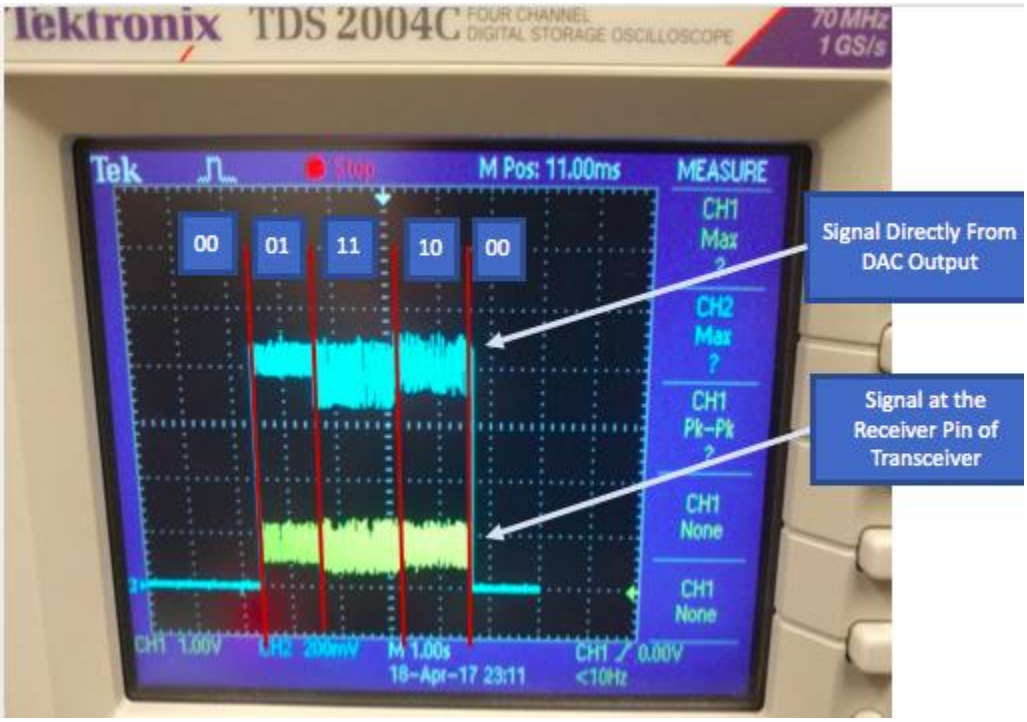

Figure 5.1: Green DAC output, Yellow Rx line

The signal itself seemed very noisy despite the use of two 120 ohm termination resistors to cancel out the characteristic resistance. This, is caused by many factors, mentioned before, including materials used for the test-bed and length of bus lines. A solution that we came up with but were not able to implement due to time restrictions included implementing split termination instead of the standard

termination. This method filters high frequency from the bus lines by acting like a low pass filter [31]. In this case the capacitor and the resistor form an RL circuit which filtrates unwanted frequencies that can be calculated using the formula: $f = \frac{1}{2\pi RC}$ [33]. This would be more effective in removing noise than linear components such as resistors.

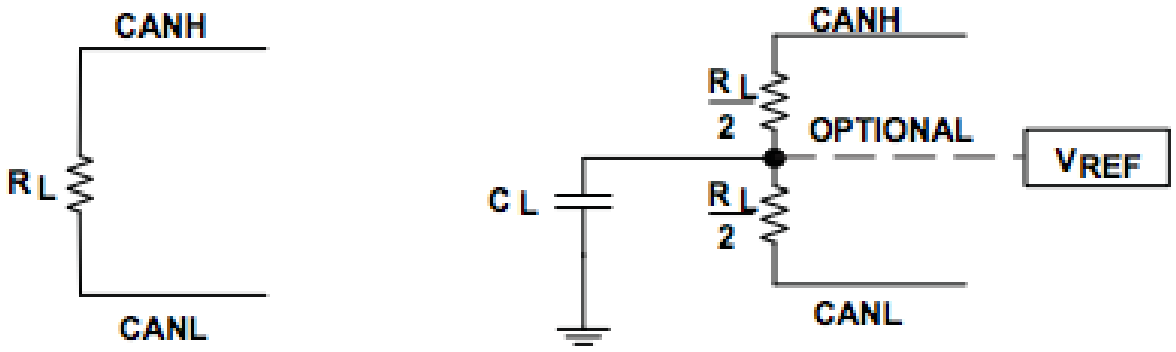

Figure 5.2 : Split Termination -
decreases noise by removing high frequencies with the addition of a Capacitor between termination resistors which serves as a low pass filter [31]

Another issue during integration included the signal not being very distinguishable regardless of the switch setting. The root of the problems seems to be the current produced by the DAC is minimal, and does not drive the CAN H bus line. The solution to this, would be to boost the current, which can be done with the addition of an amplifier stage between the DAC and the CAN H bus line.

5.3 Hardships While Creating M-ASK Adapter and base4CAN

The base2module, although easy to create a simple message to send, was made more complex, providing a true starting point for future teams to launch off on. Multiple registers make up the various sections of frames, such as the DLC, control bits, and data field, each of which' values are computed in

different states. Flags both internal to the datapath and tied to it from the controller take part in assigning tasks, and while certainly not up to protocol, provides some foundation.

The state machine logic and control signals needed to be added to and shifted around in the earlier stages of the process. An originally four state flow was fleshed out into nine, detailing features which changed depending on which mode the node was operating in. While simulation and debugging took a normal amount of time and implementation to the Nexys 3 was simple, it was found a previously lent Nexys 2 was unable to support the module fully.

As mentioned previously, the original plan was to implement M-ASK within the standard compatible nodes, these were not able to be attained.Most of this trial is detailed in section six of chapter 3, ending in the base4CAN module with two microcontrollers before and after the FPGA's inputs and outputs. While possible, this was inefficient, clunky, and better handled by the Pmod ADC1 and DAC1 modules.

Thereon, the integration with the Impala's CANbus and viewing messages on the bus was the final hurdle to clear.

5.4 Hardships While Sounding the Bus and Integration

A major problem with sounding the bus was the setup, which was very long and repetitive, as multiple trials that came with fixing mistakes and the use of unique testing equipment. For example, initially, measuring the nodes was done using an oscilloscope- and since there were only two channel probes, every time a different node was measured the connections had to be changed which required checking for specific pins.

Another problem was that the impedance was not very high and thus, the transient response was the not as distinct for each node as was expected. Ultimately, it turned out that the reflections at the rate

measured did not result in a significant loss in the bus, and for our purposes it could be considered negligible.

In integration between the base4CAN and the setup to the CAN shield hardship occurred in that the measured digital wave output and that there was a lot of noise despite taking measures. After troubleshooting, we came up with several possible solutions.

5.5 Summary

While each individual section of the project was completed and proven operational, there was not enough time in the final stretch to iron out all issues with integration, although ideas of how to do this were suggested. Furthermore, the impedance sounding was not as conclusive as expected. However, the base for a true CAN node has been created, both sending one bit per cycle and two bits per cycle. M-ary ASK was also proven operational and capable of throughput scaling when paired with base4CAN. Ultimately, we have a basis future projects can use when conducting similar research.

Conclusion and Future Developments

To conclude, our group has successfully reverse engineered the CANbus protocol into a base2CAN foundation, as well as a M-ASK implementation. These have both been integrated into the final product, base4CAN, able to send messages at four voltage levels. Thusly messages can now be sent at the original speed of CAN 2.0 and at a doubled throughput via base4CAN . While we were not able to accomplish all the goals we had, the following we did accomplish:

- **Accomplishment 1**: Created base2CAN, base4CAN
- **Accomplishment 2**: Sounded CANbus harness
- **Accomplishment 3**: Integrated base4CAN, CANbus

This research, designs, implementations, demonstrated could be extended in future work. With the developments we have provided, multiple projects could be birthed. Some future developments that we propose for those continuing with this research include:

- **Project Idea 1**: Designing a to-spec CANnode.
- **Project Idea 2:** Conduct nodal analysis to identify location-specific impedances.

For our project, one big disadvantage was that we did not have a working CAN node to use to start us off with our implementation, we had to create a basic one to work with. The first project suggestion is that a team creates a CAN node that future projects can use for development and research purposes. This node would mimic a node that satisfy all specifications. including arbitration and error checking. All together, this would not be a simple task.

Project idea two consists of finding another way to identify location specific impedances caused by reflections within the bus and with the intention to characterize nodes. This could prove useful for

increasing security measures within the bus. Also, signal differentiation within the MASK-ed signal could be improved(such as base4CAN) in a later project.

Bibliography

[1] P. Bigelow, "A 14-year-old hacker caught the auto industry by surprise", *Autoblog*, 2015. [Online]. Available: http://www.autoblog.com/2015/02/18/14-year-old-hacker- caught-industry-by-surprise-featured/. [Accessed: 04- Apr- 2017].

[2] "Automotive Cyber Security: An IET/KTN Thought Leadership Review of risk perspectives for connected vehicles", The Institution of Engineering and Technology, 2016. PDF Available: http://www.theiet.org/sectors/transport/documents/automotive -cs.cfm

[3] "Controller Area Network (CAN Bus)", *ប ច្ចេក វិទ្យារ ថ យ ន្តទំនើប*, 2017. [Online]. Available: https://autodockhmer.net/2016/05/16/controller-area-network-can-bus/. [Accessed: 27- Apr- 2017].

[4] B. Borowicz, "CAN FD - The Next (Big) Fast Thing", *Grid Connect*, 2017. [Online]. Available: http://gridconnect.com/blog/tag/canbus/. [Accessed: 22- Apr- 2017].

[5] "Everything about the CAN bus or Controller Area Network", *Canbus.us*, 2017. [Online]. Available: http://www.canbus.us/. [Accessed: 04- Apr- 2017].

[6] "History of CAN", *Canopen.us*, 2017. [Online]. Available: http://canopen.us/ home/history-of-can. [Accessed: 04- Apr- 2017].

[7] "OBD-II Background Information", *Obdii.com*, 2017. [Online]. Available: http://www.obdii.com/background.html. [Accessed: 04- Apr- 2017].

[8] O. Esparza, W. Leichtfried and F. Gonzalez, "Transitioning Applications from CAN 2.0 to CAN FD", *EEcatalog.com*, 2016. [Online]. Available: http://eecatalog.com/ automotive/2016 /04/22/transitioning-applications-from-can-2-0-to-can-fd/. [Accessed: 04- Apr- 2017].

[9] A. Arnold and S. Piscitelli, *TPMS Receiver Hacking*, 1st ed. Worcester: WPI, 2015.

[10] J. Kluser, CAN Based Protocols in Avionics, 1st ed. Vector, 2012.

[11] Raj, "Research", *Carnegie Mellon University*, 2017. [Online]. Available: https://users.ece.cmu.edu/~raj/research.html. [Accessed: 27- Apr- 2017].

[12] J. Gambatese and J. Louis, "Improving Safety on Highway Work-zones by Real-time Tracking of Operation and Equipment Status", Pacific NW Transportation Consortium, 2017. [Online]. Available: https://depts.washington.edu/pactrans/research/projects/improving-safety - on-highway-work-zones-by-real-time-tracking-of-operation-and-equipment-status/. [Accessed: 27- Apr- 2017].

[13] A. Latcher and MITRE Corporation, "Autonomy & Transportation: Addressing Cyber-Resiliency Challenges", NIST Information Security and Privacy Advisory Board, 2015.
//chapter2
[14] CAN in Automation, "History of the CAN technology", *Can-cia.org*, 2017. [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-history/. [Accessed: 04- Apr- 2017].

[15] "CANopen - Applications", *Canopen.us*, 2017. [Online]. Available:http://www.canopen.us/applications. [Accessed: 08- Apr- 2017].

[16] "CANaerospace - Designed to fly.", *Stockflightsystems.com*, 2017. [Online]. Available: http://www.stockflightsystems.com/canaerospace.html. [Accessed: 08- Apr- 2017].

[17] "Understanding SDS, DeviceNet Protocol and Can Kingdom", *Kvaser*, 2017. [Online]. Available: https://www.kvaser.com/sds-devicenet-can-kingdom/. [Accessed: 08- Apr- 2017].

[18] "Antilock braking systems (ABS) - Active safety features", *Brainonboard.ca*, 2017. [Online]. Available: http://brainonboard.ca/safety_features/active_safety_features_ abs.php. [Accessed: 08- Apr- 2017].

[19] “Sistemas de Comunicação CAN FD Modelamento por Software e Análise Tempora”, *Teses.usp.br*, 2014.[Online- Master Dissertation].Available:http://www.teses.usp.br/ teses/disponiveis/3/3140/tde-06082015-111553/pt-br.php. [Accessed: 07-Sept- 2016].

[20] Mucevski, Kiril. "Automotive CAN Bus System Explained". *www.linkedin.com*. 2015. [Online]. Available:https://www.linkedin.com/pulse/automotive-can-bus-system- explained-kiril-mucevski.[Accessed: 22-Sept-2016].

[21] “CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus”, *Esat.kuleuven.be*, 2011.[Online- Article]. Available:https://www.esat.kuleuven.be/cosic/publications/article-2086.pdf. [Accessed: 13-Oct-2016].

[22] Di Natale, Marco. "Understanding And Using The Controller Area Network". *Springer.com*. 2017.[Online]. Available:http://www.springer.com/us/book/9781461403135. [Accessed: 21-Nov-2016].

[23]"Understanding CAN with Flexible Data-Rate (CAN FD)", *National Instruments*, 2014. [Online]. Available:http://www.ni.com/white-paper/52288/en/. [Accessed: 11-Nov- 2016].

[24] Chapal, Md. "Amplitude Shift Keying (ASK) Modulation".*Technoeverywhere. blogspot.com*.2017.[Online].Available:http://technoeverywhere.blogspot.com/2011/05/amplitude-shift-key-ask-modulation.html. [Accessed: 12-Oct-2016].

[25] "Transient State And Steady State Response Of Control System". *Electrical4u.com*. 2017. [Online].Available:https://www.electrical4u.com/transient-state-and-steady-state-response-of-control-system/. [Accessed:16-Jan-2017].
//chapter3
[26] I. Mohor, "CAN Protocol Controller", Opencores.org, 2002. [Online]. Available: https://opencores.org/project,can. [Accessed: 27- Apr- 2017].
//chapter4
[27]"CANbus Connector Part Information", *Wiring Systems and Power Management Manual*, 2007.[Book]. [Accessed: 19-Sept- 2016].

[28]"Canopen Network CAN Bus Cabling Guide". *Copleycontrols.com*.[Online]. Available:http://www.copleycontrols.com/Motion/pdf/CAN-Bus.pdf. [Accessed: 15 -Mar- 2017].

[29]"Saleae Logic. The Logic Analyzer You'll Love To Use.". *Saleae.com*. [Online]. Available:https://www.saleae.com/. [Accessed: 14-Feb- 2017].

[30] Wang, Dafang et al. "Research On Reflection Of CAN Signal In Transmission Line". *Ieeexplore.ieee.org*. 2008. [Online]. Available:http://ieeexplore.ieee.org/document/4456406/. [Accessed:19-Jan-2016].

[31] "Controller Area Network Physical Layer Requirements". *www.ti.com*. 2008. [Online]. Available:http://www.ti.com/lit/an/slla270/slla270.pdf. [Accessed: 25-Feb- 2017].

[32] "Sn65hvd23x 3.3-V CAN Bus Transceivers". *www.ti.com*. 2015. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn65hvd233.pdf. [Accessed: 14-Feb-2017].

[33] "Low-Pass Filters | Filters | Electronics Textbook". *Allaboutcircuits.com*. [Online]. Available: https://www.allaboutcircuits.com/textbook/alternating-current/chpt-8/low-pass-filters/.[Accessed: 6-Nov-2016].

[34] "What Is The Maximum Cable Length For A CAN Bus?". *Digital.ni.com*. 2016. [Online]. Available:http://digital.ni.com/public.nsf/allkb/D5DD09186EBBFA128625795A000FC025. [Accessed: 13-Feb-2017].

Appendices

Appendix A: Verilog Code & MATLAB Commands

```
filename7 = 'R:/MQP_Data/Transient_Behavior/F0009CH3.csv';
Trial11_CH3 = importdata(filename7,',',15);

filename8 = 'R:/MQP_Data/Transient_Behavior/F0009CH4.csv';
Trial11_CH4 = importdata(filename8,',',15);

%all_Voltages=[];
ch1Voltages_11= Trial11_CH1.data(:, 3);
ch2Voltages_11= Trial11_CH2.data(:, 3);
ch3Voltages_11= Trial11_CH3.data(:, 3);
ch4Voltages_11= Trial11_CH4.data(:, 3)/10;

%plot
%Time applies to all figures
time1=0:1:2484;

%As you can see from the plots(Plot1 and Plot2) the node voltages plotted are very similar.
plot(time1,ch1Voltages_11,time1,ch2Voltages_11,time1,ch3Voltages_11,time1,ch4Voltages_11);
title('Plot1 - Transient: Channel1 vs Channel2 vs Channel3 vs Channel4, Take1');

figure;
plot(time1,ch1Voltages_12,time1,ch2Voltages_12,time1,ch3Voltages_12,time1,ch4Voltages_12);
title('Plot2 - Transient Response');

% Just how colse are Plot 1 CAN volatages? The difference will be evaluaded below
%Diff between channels 1-4 wherse CH1 is the basis of the comparison  since it is
%connected straight to the Function Generator and has most ideal voltage
plot1Diff1(:,1) = ch1Voltages_11- ch2Voltages_11;
plot1Diff1(:,2) = ch1Voltages_11- ch3Voltages_11;
plot1Diff1(:,3) = ch1Voltages_11- ch4Voltages_11;
plot1Mean= mean(plot1Diff1);
```

Figure A1: Matlab Snippet for Importing, Plotting and Comparing Various Nodes' Data_H Signal

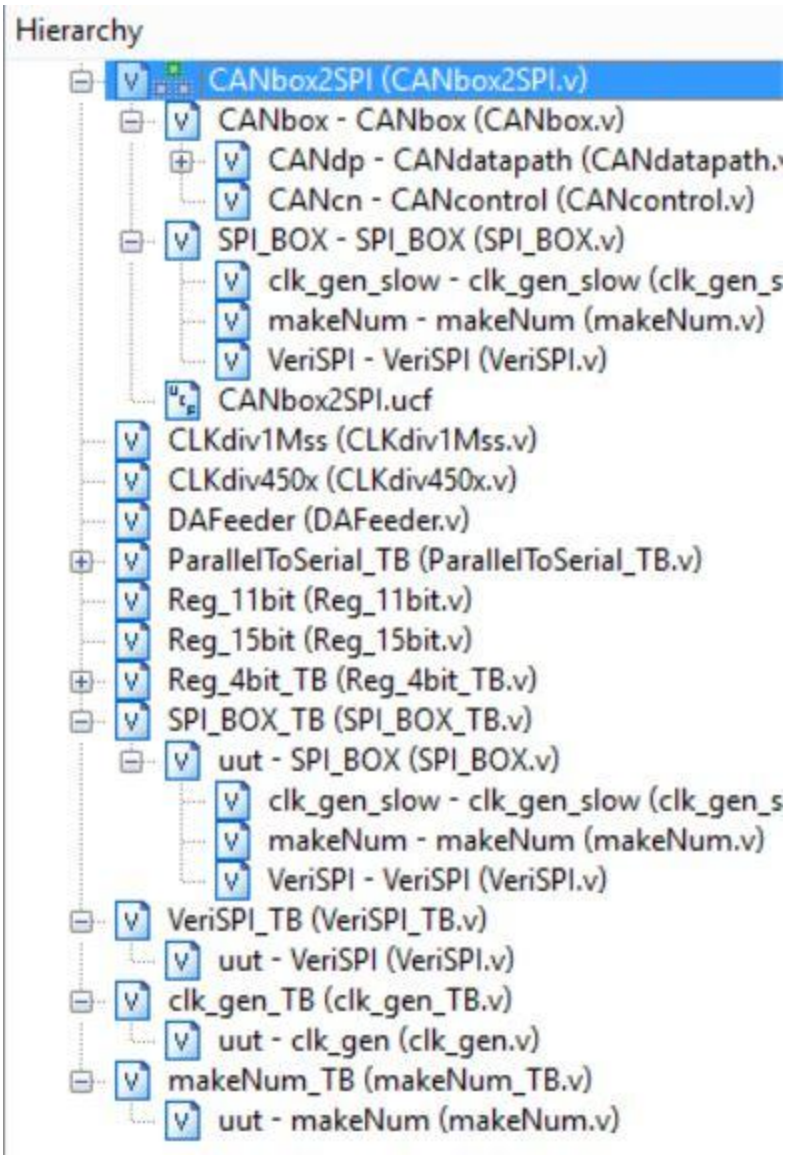

Figure A2: An overall hierarchy of modules and submodules

```
21  module CANbox2SPI(
22        input clk_100MHz, RST, D_in, read, send, dummy,
23        output mosi, sckOut, csn,
24        output [1:0] D_outHi,
25        output readin, sendin, done_flag, idle_flag,
26        output [3:0] nums
27      );
28
29     wire [1:0] D_outHi, D_outLo;
30     wire [3:0] nums;
31
32
33  CANbox  CANbox( D_in, read, send, dummy, clk_50Hz, RST, D_outHi, D_outLo,
34                    readin, sendin, done_flag, idle_flag, CLK2 );
35
36  SPI_BOX SPI_BOX( clk_100MHz, RST, D_outHi,
37                    clk_1Hz, clk_800Hz, clk_50Hz, clk_50Hzb, nums, mosi, sck, csn );
38
39  ODDR2 #(
40        .DDR_ALIGNMENT("NONE"), // Sets output alignment to "NONE", "C0" or "C1"
41        .INIT(1'b0),     // Sets initial state of the Q output to 1'b0 or 1'b1
42        .SRTYPE("SYNC") // Specifies "SYNC" or "ASYNC" set/reset
43     )
44     ODDR2_inst (
45        .Q(sckOut),      // 1-bit DDR output data
46        .C0(sck),  // 1-bit clock input
47        .C1(~sck), // 1-bit clock input
48        .CE(1'b1), // 1-bit clock enable input
49        .D0(1'b0), // 1-bit data input (associated with C0)
50        .D1(1'b1), // 1-bit data input (associated with C1)
51        .R(RST),    // 1-bit reset input
52        .S(1'b0)    // 1-bit set input
53     );
54
55  endmodule
```

Figure A3: The CANbox2SPI module, containing the CANbox and SPI_BOX

```
21  module CANbox(
22     input D_in, read, send, dummy, CLK, RST,
23     output [1:0] D_outHi, D_outLo,
24     output readin, sendin, done_flag, idle_flag, CLK2
25      );
26
27  wire EOF_flag, send_Serial, ld_consts, ld_ID, ld_DLC, ld_CRC, CLK, CLKi;
28  wire rst_consts, rst_ID, rst_DLC, rst_CRC;
29  wire ld_MaxData, dumld_MaxData, ld_MaxFrame, dumld_MaxFrame;
30  wire rst_MaxData, rst_MaxFrame, ld_FinalFram;
31  wire inc_countD_in, inc_countD_out, rst_countD_in, rst_countD_out;
32
33
34  //Datapath and controller are instantiated below and tied together with the wires above
35  CANdatapath CANdp(D_in, read, send, dummy, CLK, RST, readin, sendin, send_Serial, ld_consts, ld_ID, ld_DLC, ld_CRC,
36                    rst_consts, rst_ID, rst_DLC, rst_CRC, ld_MaxData, dumld_MaxData, ld_MaxFrame, dumld_MaxFrame,
37                    rst_MaxData,  rst_MaxFrame, ld_FinalFram, inc_countD_in, inc_countD_out, rst_countD_in, rst_countD_out,
38                    D_outHi, D_outLo, done_flag, EOF_flag);
39
40  CANcontrol  CANcn(D_in, read, send, dummy, CLK, RST, done_flag, EOF_flag, readin, sendin, send_Serial, ld_consts, ld_ID, ld_DLC, ld_CRC,
41                    rst_consts, rst_ID, rst_DLC, rst_CRC, ld_MaxData, dumld_MaxData, ld_MaxFrame, dumld_MaxFrame,
42                    rst_MaxData,  rst_MaxFrame, ld_FinalFram, inc_countD_in, inc_countD_out, rst_countD_in, rst_countD_out, idle_flag);
43
44
45  endmodule
```

Figure A4: The CANbox outer assembly; our base2CAN node

```
37  clk_gen_slow clk_gen_slow (
38        .clk_100MHz(clk_100MHz),
39        .clk_800Hz(clk_800Hz), //1.25ms  x16
40        .clk_50Hz(clk_50Hz),   //20ms    x1
41        .clk_50Hzb(clk_50Hzb),
42        .clk_16Hz(clk_16Hz),   //62.5ms  (x16)
43        .clk_1Hz(clk_1Hz)      //1000ms  (x1)
44     );
45
46
47  makeNum makeNum (
48        .CLK(sck), //50Hz
49        .RST(RST),
50        .SW(SW),
51        .NumOut(NumOut)
52     );
53  assign nums = NumOut [3:0];
54
55
56  VeriSPI VeriSPI (
57        .clk_16MHz(clk_800Hz), //actually 800Hz
58        .clk_1MHz(clk_50Hz),   //actually 50Hz
59        .clk_1MHzb(clk_50Hzb),
60        .number(NumOut),
61        .csn(csn),
62        .mosi(mosi),
63        .sck(sck),
64        .load(load),
65        .enable(enable)
66     );
```

Figure A5: The wiring of submodules within the SPI_BOX

```
124        default: begin
125           readin=1'b0;   sendin=1'b0;    send_Serial=1'b0;   idle_flag=1'b0;
126           ld_consts=1'b0;     ld_ID=1'b0;     ld_DLC=1'b0;   ld_CRC=1'b0;
127           rst_consts=1'b0;    rst_ID=1'b0;    rst_DLC=1'b0;  rst_CRC=1'b0;
128           ld_MaxData=1'b0;     dumld_MaxData=1'b0;   rst_MaxData=1'b0;
129           ld_MaxFrame=1'b0;    dumld_MaxFrame=1'b0;  rst_MaxFrame=1'b0;   ld_FinalFram=1'b0;
130           inc_countD_in=1'b0;  inc_countD_out=1'b0;  rst_countD_in=1'b0;  rst_countD_out=1'b0;
131        end
132     endcase
133  end
134
135
136  //Next-state logic
137  always @(posedge CLK) begin
138     case( state )
139        reset:   state <= idle;
140        idle:    state <= ((send & dummy) ? load_consts :
141                           (read & ~dummy) ? listen :
142                            (send & ~dummy) ? listen : idle);
143        listen:  state <= (~D_in) ? maxFrameLoop : listen;            //b/c SOF is a 0
144        maxFrameLoop:  state <= (done_flag | EOF_flag) ? load_consts : maxFrameLoop;
145        maxFrameDummy: state <= load_Final;
146        load_consts:   state <= (send & dummy) ? maxFrameDummy : load_3main;
147        load_3main:    state <= load_Final;
148        load_Final:    state <= (send) ? send_Message : reset;
149        send_Message:  state <= (~done_flag) ? send_Message : reset;
150        default: state <= reset;
151
152     endcase
153  end
```

Figure A6: An example of the CAN control module, including signals and logic

```
// These constants won't change.  They're used to give names
// to the pins used:
const int analogOutPin1 = A2;  // Analog inputData1 pin
const int analogOutPin2 = A3; // Analog outputData1 pin
//const int analogOutPin2 = A2; // Analog outputData2 pin

int inputData1 = 0;         // input value read from port
int outputData1 = 0;         //output value at (+CAN dataline) read from port
//int outputData2 = 0;        //output value at (-CAN dataline) read from port

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  // read the analog in value:
  inputData1 = analogRead(analogOutPin1);
  outputData1 = analogRead(analogOutPin2);
  //outputData2 = analogRead(analogOutPin3);
  // change the analog out value:
  analogWrite(inputData1,outputData1);//

  // print the results to the serial monitor:
  Serial.print(""); //K83 input psotive GMLAN dataline
  Serial.print(inputData1);
  Serial.print("\t  "); //K38 positive GMLAN dataline
  Serial.println(outputData1);  //negative GMLAN dataline
  Serial.print("\t ");
  //Serial.println(outputData2);
```

Figure A7: Arduino Mini Code- CAN node Measurement Tool

Appendix B: Relevant Figures and Tables

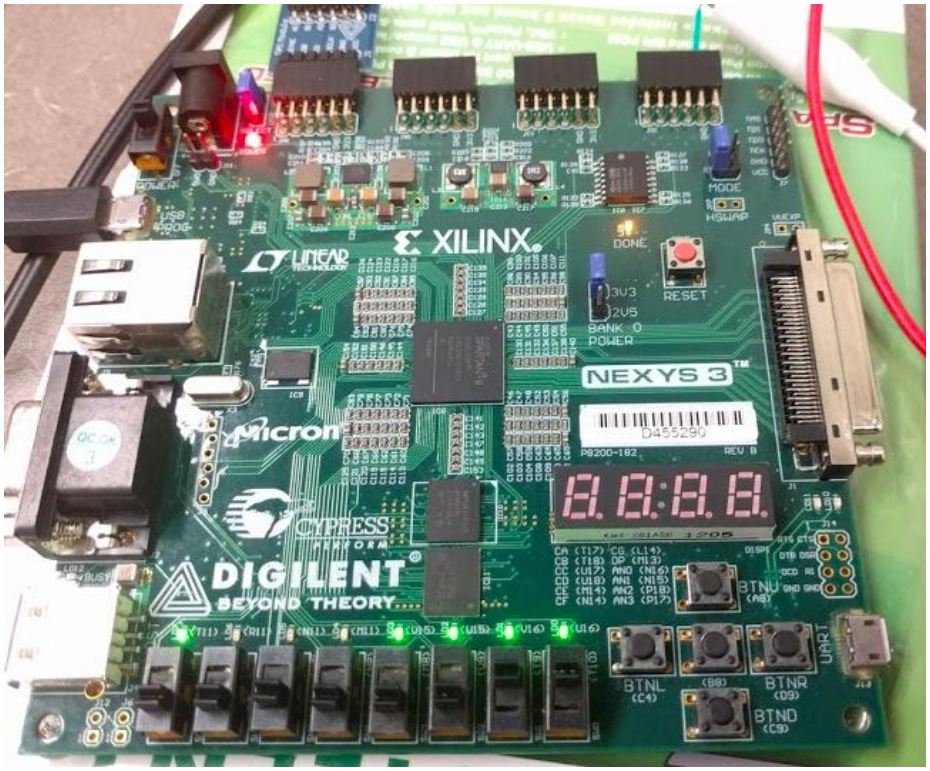

Figure B1: FPGA- Digilent NEXYS3 Xilinx Board

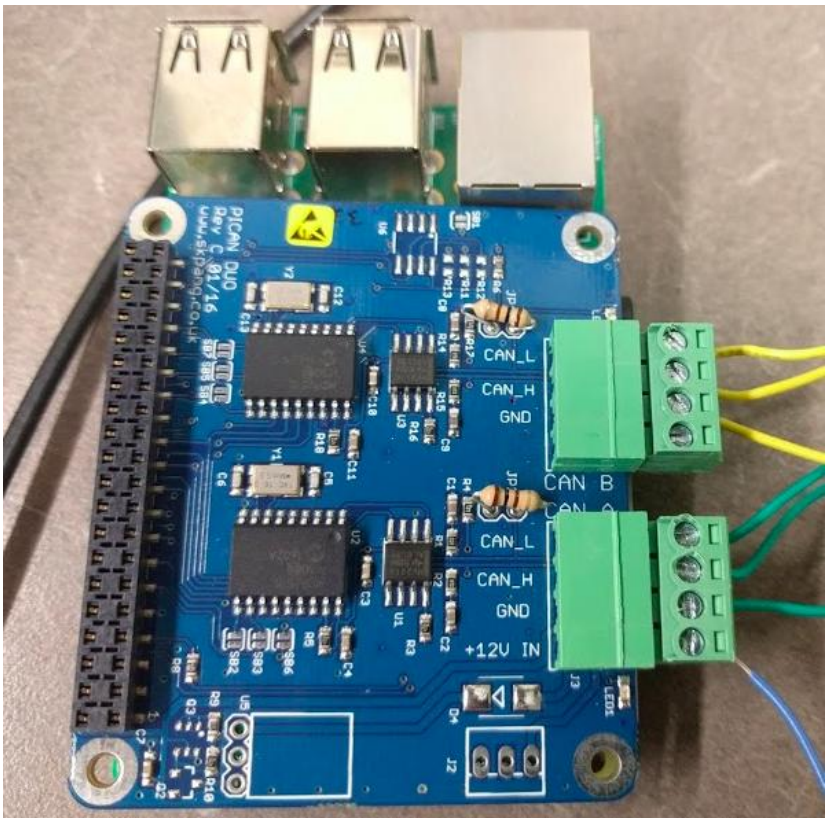

Figure B2: Raspberry Pi 3 Model B with CAN Shield Connected

Table B1: Length Specifications for a CANbus Network [34]

Bus Speed	Bus Length (L)	Cable Stub Length (l)	Node Distance (d)
1 Mbit/Sec	40 meters (131 feet)	0.3 meters (1 foot)	40 meters (131.2 feet)
500 kbits/Sec	100 meters (328 feet)	0.3 meters (1 foot)	100 meters (328 feet)
100 kbits/Sec	500 meters (1640 feet)	0.3 meters (1 foot)	500 meters (1640 feet)
50 kbits/Sec	1000 meters (3280 feet)	0.3 meters (1 foot)	1000 meters (3280 feet)