Summer 5-1-2021

A Novel Technique for Sample Point Discovery and Its Use in a Proposed Broadcast Confusion Attack on High-Speed Controller Area Networks

Brendan Mulholland

A Novel Technique for Sample Point Discovery and Its Use in a Proposed Broadcast Confusion Attack on High-Speed Controller Area Networks

A Thesis

Submitted to the Faculty

of

Rose-Hulman Institute of Technology

by

Brendan Mulholland

In Partial Fulfillment of the Requirements for the Degree

of

Masters of Science in Electrical Engineering

May 2021

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Final Examination Report

Brendan Mulholland	Electrical Engineering
Name	Graduate Major

Thesis Title A Novel Technique for Sample Point Discovery and Its Use in a Proposed Broadcast Confusion Attack on High Speed Controller Area Networks

DATE OF EXAM: April 23, 2021

EXAMINATION COMMITTEE:

	Thesis Advisory Committee	Department
Thesis Advisor:	Zachary Estrada	ECE
	Jianjian Song	ECE
	Sid Stamm	CSSE

PASSED X

FAILED ___

ABSTRACT

Brendan Mulholland

M.S.E.E

Rose-Hulman Institute of Technology

May 2021

A Novel Technique for Sample Point Discovery and Its Use in a Proposed Broadcast Confusion Attack on High-Speed Controller Area Networks

Thesis Advisor: Dr. Zak Estrada

Over the last twenty-five years, the Controller Area Network, or CAN, has become ubiquitous in the automotive world as a communication network. That ubiquity is attributed to its high immunity to electrical interference and its resilience to data errors. CAN was designed to ensure data integrity during transmission and allow for multiple nodes to transmit information without a central device controlling that transmission. Given the ubiquity of CAN, much research has been performed to detect and protect against external intrusions on the network.

In this paper, I present a methodology for the measurement of key CAN timing parameters. With the detection and understanding of these parameters, I demonstrate a proof of concept attack, dubbed the Broadcast Confusion Attack, which allows for the data integrity of the network to be weakened. Evolutions of this attack could be performed without being detected by two of the three categories of CAN intrusion detection systems. In the evolutions of the

attack, devices could be completely overwritten by the attacker without any device (even the victim) knowing such an attack has occurred.

DEDICATION

To my Mom, for all the support over the years.

ACKNOWLEDGEMENT

Effort sponsored by the Air Force under MOU FA8750-15-3-6000. The U.S. Government is authorized to reproduce and distribute copies for Governmental purposes notwithstanding any copyright or other restrictive legends. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of the U.S. Air Force and/or U.S. Government.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACK	Acknowledgement
BRAM	Block Random Access Memory
CAN	Controller Area Network
CANH/CANL	Individual Wires for the CAN Differential Bus
CiA	CAN in Automation (User and Manufacturer Organization)
CRC	Cyclic Redundancy Check
DLC	Data Length Code
DUT	Device Under Test
ECU	Engine Control Unit
EOF	End of Frame
IF	Interframe
OSI	Open Systems Interconnection
RX	Receive
SOF	Start of Frame
SPS	Sample Point Subsystem
TDS	Temporal Delay Subsystem
TX	Transmit

LIST OF TERMINOLOGY

Dominant	Logical 0, ~ 3.5V = CANH, 1.5V = CANL
Recessive	Logical 1, ~ 2.5 V = CANH = CANL
Node	A device made up of a single CAN Controller
Tester	A system implementing the sample point subsystem and or the temporal delay subsystem

CHAPTER 1: INTRODUCTION

Controller Area Network, CAN, has become ubiquitous as a communication protocol because of its low cost and high reliability. Due to that ubiquity, CAN has become a target for many innovative attacks designed to exploit flaws in the fundamental design of the standard. Attacks range from ordinary denial of service to attacks such as the Bus-Off Attack[1], where network specifics of CAN are utilized to reduce the total integrity of data on the network.

Due to the fundamental layout of a CAN network, with all devices communicating on a shared bus, the raw signal on the bus is exploitable by playing a forced signal onto the bus. Networks that employ switches and routers use information contained in higher layer packets, taking the data contained in those packets and generating new higher layer packets. But with CAN, the raw signal on a shared bus is the information upon which all devices act. The raw signal provides the ground truth for information on the network, and controlling the raw signal allows an attacker to set the ground truth for the network.

The protocol designers of CAN designed a system that placed absolute priority on all devices on the network having a consistent view of the state of the system. This is key in a setting such as an electrically noisy automobile. For an attacker to modify the raw signal of the network, they would have to work around measures implemented by the protocol designers. In CAN, these measures are based upon sub-bit timings, the sample point, where the value of a bit is determined. Knowledge of the sub-bit timings of a network allows a base upon which an understanding of the entire bus can be built. It is with this, that I present the CAN Physical Timing Analysis Process, which is able to determine the sample points for all devices on a CAN network, and the Broadcast Confusion Attack, which is a proof of concept to show how this knowledge can be used to attack the CAN bus and reduce the overall trust in the network's data integrity.

1.1 Introduction to CAN

CAN is a multi-master serial bus communication protocol, with the standard first being published by Robert Bosch GmbH in 1986, formerly named the Automotive Serial Controller Area Network. While the network was originally designed for usage in the automotive world, it quickly expanded beyond the automotive world, being used for manufacturing communication and medical devices. CAN was officially standardized by the ISO in 1993 with the publication of the ISO 11898 family of documents. The 11898 standards would be further refined with six revisions and extensions to the standard over the following twenty-two years [2].

The importance of CAN in automobiles cannot be overstated. The first car company to utilize CAN for data transmission in the subsystems of one of its cars was Mercedes-Benz in 1991. CAN became the main automotive communication standard in 2008, when the U.S. Government mandated that the CAN network layer be employed for all communications to ensure environmental standards on new vehicles. As a result of this law, the network gained near ubiquity for all automobiles released over the next 13 years [3].

With CAN being a multi-master bus, every node is connected to every other node in the network, with every node being equal in importance to the other nodes. CAN does not establish any limitations on what, to whom, or when a device may communicate. As CAN was designed to be deployed in critical applications like automobiles, a failing device is designed to remove itself from the communication network, so that the devices which are still functioning correctly can communicate and continue their operation.

The CAN standards were designed with robustness and data integrity as their fundamental principles. As shown in [4], in an aggressive environment created by placing a high frequency arc welder two meters away from a CAN network, almost 12.25 GB of data were transferred with a bit error rate of $2.6 * 10^{-7}$, which is better by two orders of magnitude compared to Wi-Fi [5]. This sort of robustness at the physical side is paired with multiple error detection methodologies in the logical link control layer. Protections such as devices performing self-checks for transmitted data appearing on the bus; strict rules for stuff bit insertion; and a CRC-15 checksum with every frame help to ensure the data transmitted is the same as the data received for all devices.

1.2 CAN Intrusion Detection

As CAN has increased in ubiquity in the automotive world, many threats have been presented which threaten the network's ability to transmit data securely. With the open nature of CAN, and with federal law which requires the network be accessible through the OBD port, an equally large amount of work has been performed to secure the network. As shown in [6], there are around 14 attack surfaces in an automobile, many of them using CAN for communication. There, Young et al., showed that there are eight major intrusion detection systems, IDS, types.

An intrusion detection system is a set of systems that can determine the validity of a message on an open network through feature analysis. The eight IDS types can be further simplified into three groups based on the information utilized in their analysis. The first type of IDS utilized frame data analysis to validate the transmitter of the frame. These systems solely use the information in a CAN frame, such as data and the ID that transmitted the information. A node performing this technique would have the capability to identify if another node is transmitting using an ID registered to itself.

The second type of intrusion analysis is based on the timing of a message. These systems work by analyzing the frequency at which a given ID transmits a data frame on the CAN bus and reports when the frequency of communication has changed. This is indicative of attacks such as message injection and deletion attacks. These systems can use either statistical analysis or machine learning to report erroneous errors on the CAN bus. IDS utilizing timing and frame analysis are the most commonly used as they are relatively simple to implement, with most of the analysis requiring software additions instead of new hardware placed onto the CAN bus.

The third and final type of intrusion analysis uses ECU analog characteristics to authenticate a message. These systems employ anything from clock skew, clock drift, to voltage on the bus to correlate IDs to given hardware. When the analysis reports that there is a discrepancy occurring between the ID and the ECU transmitting the message, that could be a sign of an intrusion on the bus. These systems are the most theoretical out of the three, as they require both extensive neural networks and new hardware to be connected to the bus to perform such analysis. However, the increase in hardware does come with the benefit that, with enough training time and development, detection systems utilizing this methodology can detect attacks that were previously undetectable, like the Bus-Off attack[1, 6].

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 gives an overview and a discussion of the CAN 2.0b specification, with focus given to the aspects of the spec which are exploited to perform the analysis and attack. Chapter 3 provides details of the issues at hand and the procedure to perform the timing analysis, along with some background information of the test setups used throughout this paper. Chapter 4 provides in-depth details of the implementation of the sample point detection subsystem, along with experimentally captured data. Chapter 5 goes in depth into the implementation of the temporal delay subsystem, and as before, analysis of experimentally captured data. Chapter 6 delves into the Broadcast Confusion Attack, including how it’s implemented, its feasibility, and the effects it causes for the security of CAN. Finally, in Chapter 7, I will discuss areas in which the algorithm can be improved, the viability of this attack in the CAN FD protocol, and general conclusions that I have come to while working on my thesis.

CHAPTER 2: CAN SPECIFICATION

The most important CAN standards are those governed by the International Organization for Standards in documents ISO 11898-1 and 11898-2. The 11898-1 standard defines everything in the CAN controller module, but not the CAN transceiver. This is equivalent to the higher sublevels of the physical layer and the data link layers of a standard OSI networking stack. The 11898-2 standard specifies how CAN devices communicate with each other in high-speed applications of up to 1 Mbps. This applies to the lowest levels of the physical layer, and specifies voltages, slew rates, and transmission line qualities. The connection between certain functionalities and their corresponding OSI layers is given in Figure 1**.**

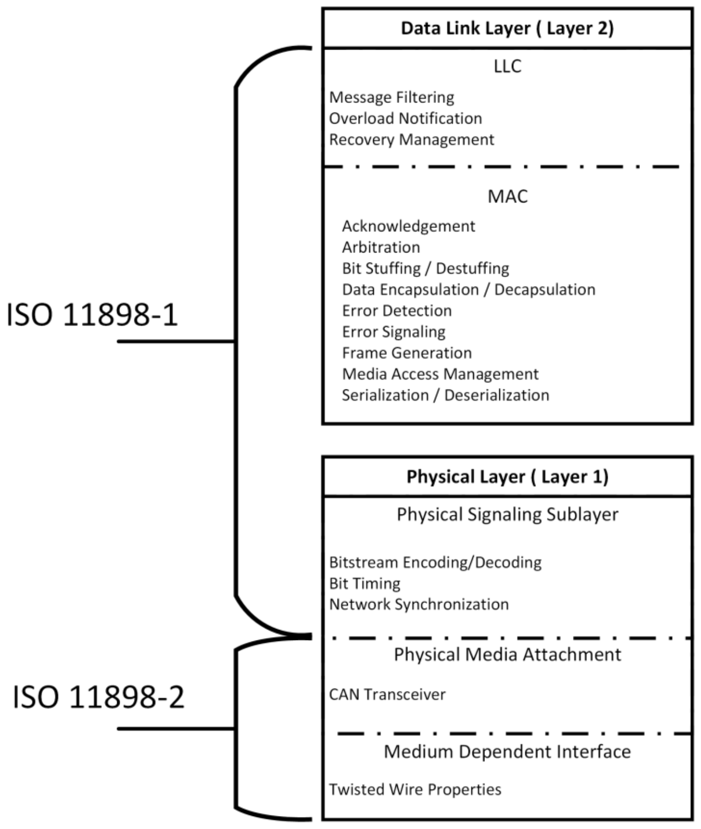


Figure 1 OSI Layout of Controller Area Network[2]

The third important CAN document is published not by the ISO, but instead through SAE in document J2284. This document discusses the standard procedure for ID allocation, and message format for use in automotive applications. This document, while important to automotive manufacturers, was not used in this thesis because it would constrain the network beyond what is required. J2284 is not universally utilized in every CAN network, and places assumptions on the network that may not be true outside of an automotive setting. Instead, focus will be placed on the information given in ISO 11898-1 which will be placed in the Controller subsection, and in ISO 11898-2 which is placed in the transceiver subsection. Any documents or standards higher than OSI Data Link Layer will not be raised in this thesis.

2.1 CAN Controller Specifications

2.1.1 CAN Frame

The CAN standard has five predefined frame structures, all with different functionality and meanings for the CAN controllers to decipher. Three of the frame types: the data frame; the extended data frame; and the remote request frame, are used in a properly functioning CAN bus. The other two frames, error and overload, signal an issue with either the bus or an individual node.

For the frames used in proper functioning, multiple shared header and footer fields are utilized. Below in Figure 2, the standard properly functioning CAN fields are shown. These fields have been aligned at the start of frame, or SOF, which is always present in all proper functioning frames, and signals the transition from the shared bus being idle to the bus being occupied. For all fields in a CAN frame, data is big-endian most significant bit first.

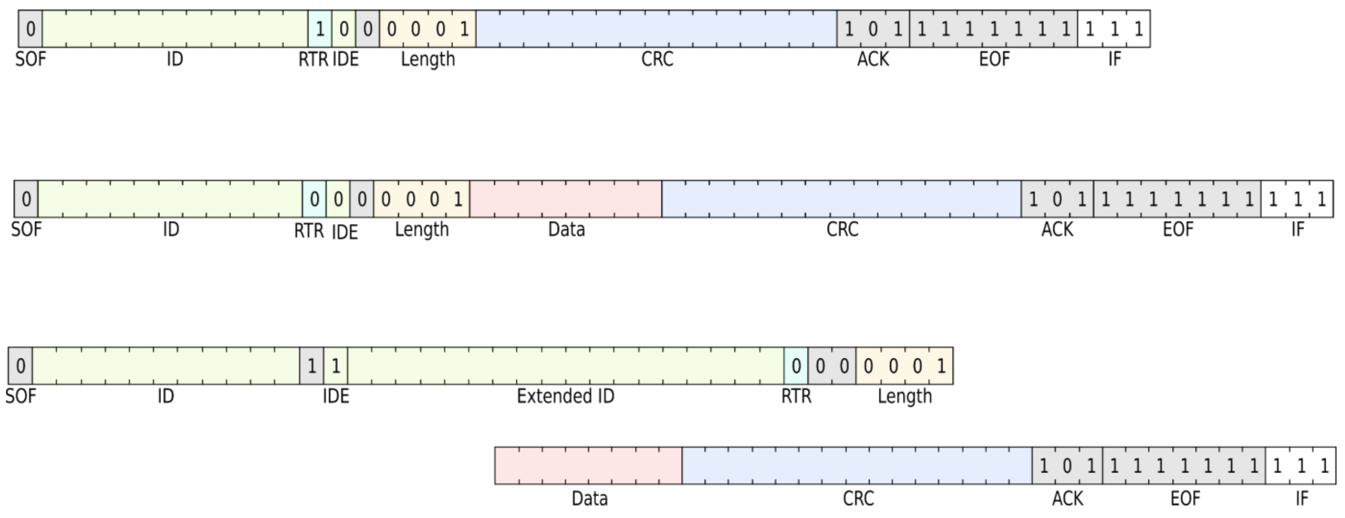


Figure 2 Comparison of CAN Data Frames

Top: Remote Request Frame, Middle: Standard Data Frame, Bottom: Extended Data Frame

The first section of interest in these frames shown in Figure 2 is the arbitration field, which includes the ID, RTR, and IDE sections. While the exact method of how arbitration is achieved is discussed later in this chapter, the fixed bits inside the arbitration field are the deciding factor as to which of the three normal operation frames the following bitstream will apply to. These fixed bits are placed surrounding an 11-bit ID field. This ID identifies the message priority but is not unique per device like a MAC is in ethernet. As shown in Figure 3**,** the arbitration field has three fixed bits of information about the frame.

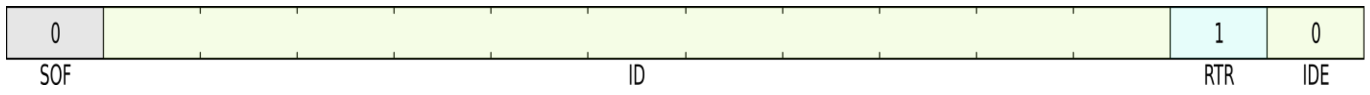


Figure 3 CAN Arbitration Field

The first, as mentioned before, is the SOF. The SOF is a single bit period dominant, logic low, signal. This bit is necessary for proper synchronization to the start point of a CAN frame. The next fixed bit is the remote transmission request, or RTR bit. When this bit is set as a recessive bit, logical high, the transmitting device is requesting that the CAN unit utilizing the given ID transmit some information. While remote transmission requests are still allowed by the standard, they have fallen out of general use due to the lack of definition in the standards [7].

In Figure 3, the final fixed bit is the ID Extension Bit (IDE) bit which signifies whether the message has an 18-bit ID extension on top of the 11-bit ID. The extended ID functionality is useful as it expands the total number of available ID bits, allowing for information about the sender and the data, along with information about the criticality of the message. ID extensions are used heavily in automotive applications as it is mandated by SAE J2284. For this thesis, out of the three frames used in proper functioning, only the standard data frame is considered. There is nothing stopping the handling of the other two frame types, however, this was an implementation decision to ease the possible avenues for design errors.

After the arbitration field is the control field. The control field for the standard data frame is given below in Figure 4. The fixed bit, r0, must always be dominant, as it was left unused to allow for future expandability to the CAN standard. This precedes the data length code, DLC, which are the last four bits of the field. The DLC informs the controller of the length in bytes of the data field. The data field for a CAN device is anywhere from 0 to 8 bytes. While 4 bits are allocated and some nonstandard CAN devices allow for larger transmission frames, 8 bytes is the largest allowed by the specification.

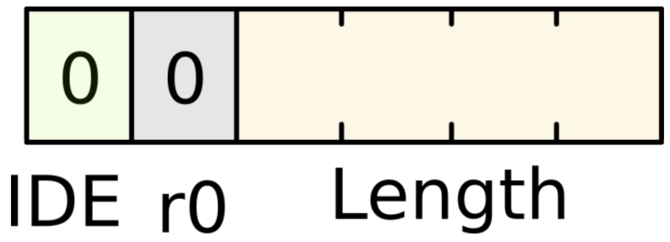


Figure 4 CAN Control Fields

After the given number of data fields, the CAN controller provides a constructed CRC field. The CRC field is a 15-bit checksum of all data transmitted from the SOF to the end of the data field. The CRC is a standard polynomial given in Equation 1. This function has a Hamming distance of six, allowing for the detection of all randomly distributed bit errors of up to five bits, and any burst errors of up to 15 bits [2].

$$\mathbf{x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1} \tag{1}$$

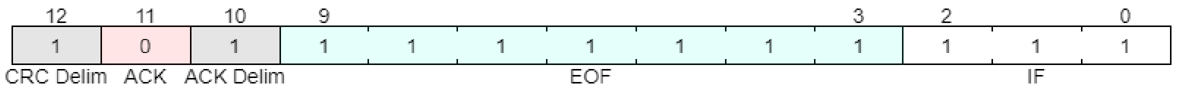


Figure 5 CAN Trailing Section

The final section of the standard data frame comprises the acknowledgment, ACK; end of frame, EOF; and the beginning of the interframe, IF, fields. This trailing section is shown in Figure 5. Note that only the ACK field does not have a fixed value. The ACK field is special in this way, as it is the only bit which during a non-contended data frame may be longer than a full bit period without any clock issues. This is because the transmitting device will transmit a recessive bit, and it is up to the receiving devices to transmit a dominant bit if the data received is valid. If the transmitting node does not detect a dominant signal by its sample point, the transmitting node will assume a transmission issue and signal an error. After the ACK bit, there is the ACK delimiter and the EOF fields. This adds up to a total of 8 sequential recessive bits. CAN devices can only transmit after 11 sequential recessive bits, so for convenience in modeling transmission rates and frame hierarchy, the first three bits of the IF are included as the trailer to the standard data frame.

If either during transmission or reception a message violates a list of given rules for CAN messages, a node will transmit an error frame. The transmission of an error frame is not dependent on having sole control over the data on the bus, and in some cases, is designed to cause transmission to cease. In Figure 6 the definition for an active error frame is given, and a passive error frame is given in Figure 7. More information about the difference between the two error frames, including why there are two and when each is used is given in §2.1.4 Error Detection, Signaling, and Confinement.

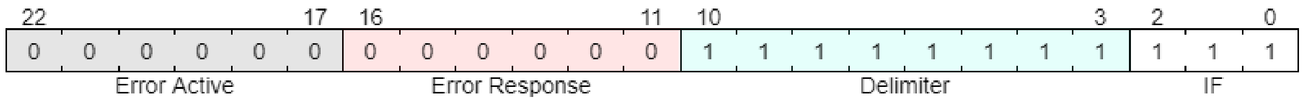


Figure 6 Active Error Frame Diagram

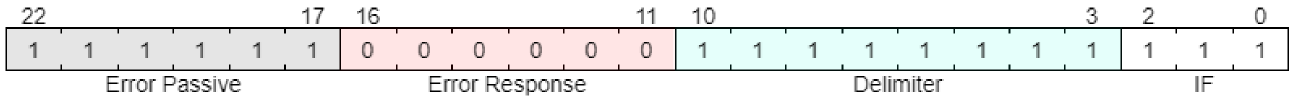


Figure 7 Passive Error Frame Diagram

The important point to note from Figures 6 and 7 is that the error frame is anywhere from 6 to 12 bits long without counting the delimiter and IF. These error frames are detected through a violation of CAN's bit-stuffing rules. In summary of the bit-stuffing rules, after five consecutive bits of the same polarity an opposite polarity bit is inserted into the bitstream to maintain clock synchronization. The transmission of six repeated bits will cause a stuffing error to always be detected. The active error frame ensures the reception of an error frame by all nodes on a bus, while a passive error frame only ensures reception when signaled by a node with arbitration.

The final frame is the overload frame. The overload frame is identical to the error frame shown in Figure 6. The key differences between the two in terms of response are given in the error section. The overload frame is transmitted during the IF and is differentiated from the error frame solely because an error cannot occur in the IF.

2.1.2 CAN Arbitration

During a transmission window, a transmitting node assumes that it is the only device transmitting on the bus, except when acknowledging a message and when it is in the arbitration phase. As CAN frames are transmitted on a shared bus, two nodes transmitting simultaneously is considered a collision and is normally an error due to the integrity of the data on the bus no longer being assured. However, during the arbitration phase, this destructive nature of collisions is embraced, as it is the foundation for determining the order of transmission.

This method is called priority-driven arbitration, which ensures that the highest priority message is the one placed on the bus first. The arbitration method utilizes the fact that a dominant signal on the CAN bus will always overwrite a recessive signal. If a node is transmitting a recessive bit during arbitration, and sees a dominant signal on the bus, the device will stop transmitting, and will enter receiving mode, as it has lost arbitration with another device.

This is shown in Figure 8, where two nodes are competing to send a message on the CAN bus. Here, both nodes are transmitting at the same time, and for ease of explanation, this is four bits into the IF. Contention between the ID's of two devices always starts at the SOF bit, which is always a dominant bit. Both devices will transmit a dominant bit to signal that a device is starting to transmit. After this, both devices will transmit the first bit of their ID, which in this figure is a dominant bit. Since both devices transmit and receive a dominant bit, they will both move on to the second bit of their IDs. Node A will transmit a dominant bit while Node B will transmit a recessive bit. Since the dominant bit will overwrite the recessive bit, Node B will not see the signal it transmitted on the bus and will determine it has lost the arbitration and will move into receiving mode. With no other devices arbitrating on the bus, Node A will transmit its remaining ID bits and go on to transmit its frame.

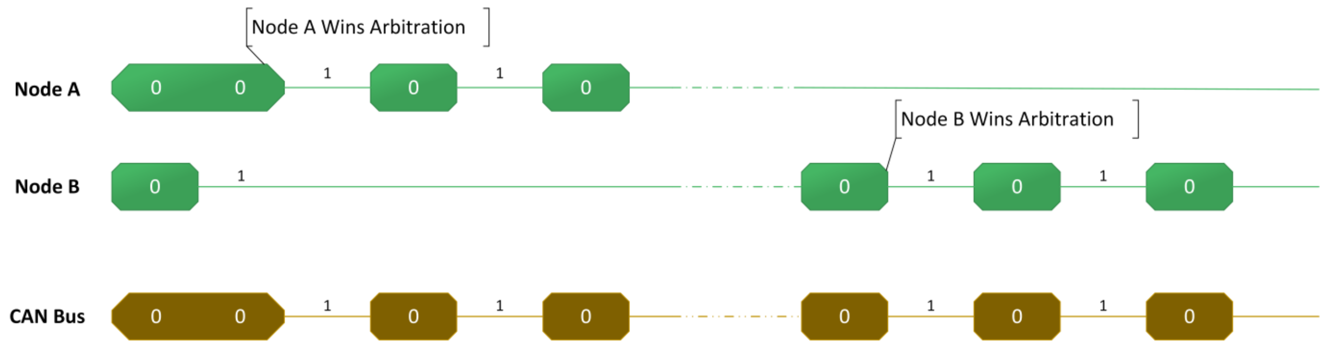


Figure 8 Two Nodes Arbitrating for Bus Control

The remote transmission request bit and identifier extension bits are included in the arbitration functionality. Due to data frames having a dominant bit in the remote request bit, a data frame will always have higher priority than a request frame. Along the same lines, as the identifier extension bit is always dominant in an 11-bit ID frame, the 11-bit ID frame will always have priority over an extended ID frame.

2.1.3 Clocking in CAN

CAN nodes do not share a common clock; therefore, they must synchronize themselves based on the only shared resources the nodes have, the signals on the bus. Nodes use certain bit transitions to self-correct their timing drift to the drift of the transmitting node. Every device is individually configured with a sub-bit timing interval called the time quanta. The minimal amount of time quanta per bit period is 8 time quanta, with the maximum being 25 time quanta. The CAN standard expects a transition to occur within a single time quantum, which is considered the beginning of a transmitted bit.

The time quanta are grouped into multiple segments for configuration of key parameters of a CAN node. These segments are the sync segment, the propagation delay segment, and phase segments 1 and 2. How a CAN bit is separated into individual timing intervals is shown in Figure 9. According to ISO standards, the sync segment must always be one time quantum long, the propagation segment and phase segment 1 can be any whole number of time quanta between one and eight. Phase segment 2's timing is dependent on the size of phase segment 1, where its value can be as low as one time quantum and as long as phase segment 1 time quanta count, or 2 time quanta if phase segment 1 was set to be only a single time quanta. Sync segment is the first segment within a bit time, where the aforementioned bit transition is expected to occur.

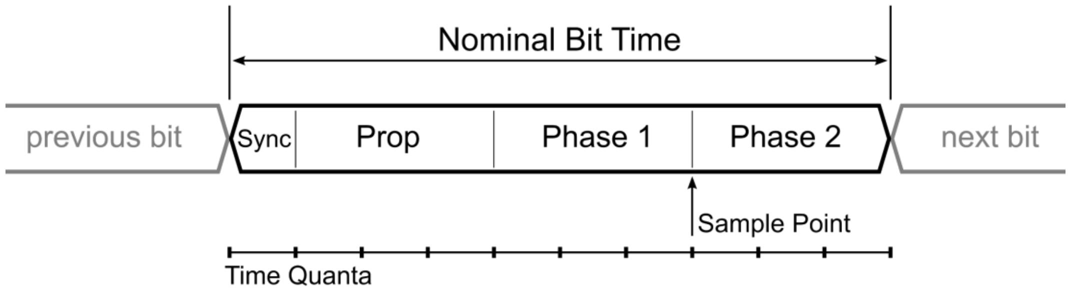


Figure 9: CAN Bit Timing Diagram

The propagation segment is required for proper arbitration to occur on the CAN bus. The length of time in seconds of this segment must be greater than or equal to twice the line delay of the CAN bus and the sum of the internal delays of the two slowest devices on the CAN bus. This is required for arbitration to correctly occur, as too low of a value in this segment could lead to data not reaching all nodes before a sample point, resulting in loss of system integrity [2].

Phase segments 1 and 2 come sequentially after the propagation segment. These are named as such because these segments can be lengthened or shortened due to resynchronization to align the bit with an edge outside of the sync segment. The transition between phase segment 1 and phase segment 2 is the point in time at which the bus level is read and interpreted as the value of the bit. This location is referred to as the sample point, and usually lies within 60%-90% from the beginning to end of a bit. The data read by the controller at this point is delayed from the current state of the bus by 10's to 100's of nanoseconds, due to the asynchronous handling of the signal through clock domain crossings, debouncing and receiver delay.

To maintain synchronization between multiple nodes on a bus, bit stuffing is used to mitigate long runs of a single polarity. Bit stuffing occurs at the transmission and reception of CAN data inside the controller. Frames are stuffed just prior to transmission, and are unstuffed before the submodules that require the bitstream. CAN demands that after the consecutive transmission of five homogenous bits a bit of opposite polarity must be transmitted, even if the bit that was to be transmitted is of opposite polarity from the consecutive transmission. This is shown in Figure 10.

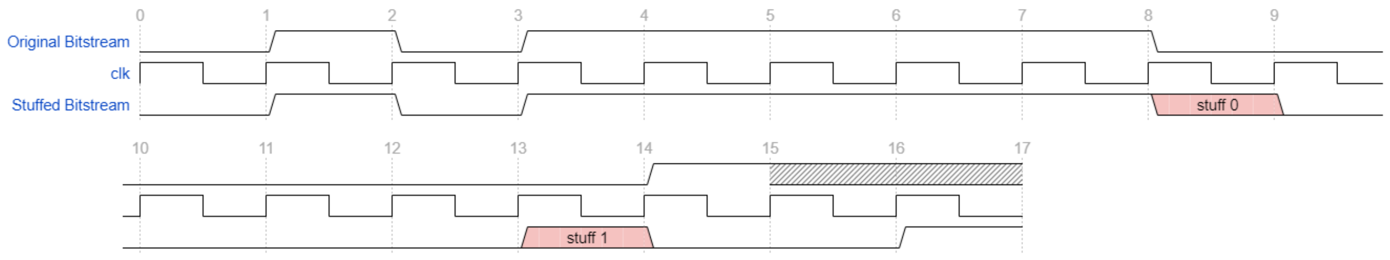


Figure 10 CAN Bit Stuffing Scheme

CAN allows bit stuffing to occur starting with the SOF all the way to the end of the CRC sequence. There is a wide range in which stuff bits occur. Since the insertion of stuff bits may cause further stuff bits to appear, CAN message may be up to 24 bits longer than expected [8]. Important to note is that the stuff bit counter is not incremented before the SOF or after the CRC and is kept at zero at these locations. Therefore, any bits that are not set to the expected values as given in the frame section after the CRC is treated as a frame error.

2.1.4 Error Detection, Signaling, and Confinement in CAN

Due to the importance placed on the priority of messages and the integrity of the data transmitted throughout the network, error handling is given great importance in the design of the network. Errors in the CAN network are broadcasted to all other nodes, and the entire network should enter an error state shortly after the error occurs. This is to speed up the time to retransmission as an invalid message received by a single node would cause the state of the system to be inconsistent across all nodes. If errors did not cause frame transmission to cease instantly, then the bus would be occupied by an invalid frame with all nodes having to discard the known invalid message.

ISO 11898-1 specifies five non-mutually exclusive error types that can be detected by a node. The first error is a bit error and occurs when a transmitting device does not detect the expected polarity on the bus during transmission of a bit. This error does not occur during arbitration; the ACK bit; or during the transmission of a passive error flag. The second error is an error in the stuffing protocol. This error is triggered when six consecutive bit times of constant polarity occur within a field within the frame in which bit stuffing can occur. The third error is a CRC error, caused by an error detected by the receiver's CRC calculation. The fourth error is a form error, which is when a fixed bit value in the CRC frame is not detected at the predetermined location. The final error type is the ACK error in which a transmitter does not detect a dominant signal in the ACK slot [2].

Using error states, a CAN node needs to have the ability to confine itself from the network if it is detecting too many errors on the bus. This is where the concept of error counting in CAN comes from. For each error, a counter, for either transmission or reception, is incremented by a set amount given in ISO 11898-1 §13.1.4.2 [2]. On average, the error counter is increased by eight on the detection of each error and decreased by one for the successful transmission or reception of a CAN frame. A node that either detects an error type or an error frame and is currently transmitting a CAN frame will increase its TX error counter by 8. A node that is not transmitting the current CAN frame will increase its RX error counter by 8.

The error counter value is directly utilized in the fault confinement logic. The fault confinement also determines how a device is to interact on the CAN bus, and what type of error frame will be sent by the node. There are three error modes specified in the standard, those being error-active, error-passive, and bus-off. Error-active is the default state of a CAN node and will partake in all bus activity. It is named error-active as it will send a dominant error frame, as shown in Figure 6. The next state is the error-passive state, named so because it transmits a recessive error frame as shown in Figure 7. A node in error-passive mode can partake in all bus activity that an error-active node partakes in but must wait eight bits into IF before attempting arbitration. The recessive error frame does not overwrite information currently on the bus and will only be detected by other nodes if the node sending the recessive error frame is the node that currently has arbitration on the bus. The final mode is bus-off. A node in bus-off mode neither receives nor transmits any frames onto the bus and effectively removes itself from the network.

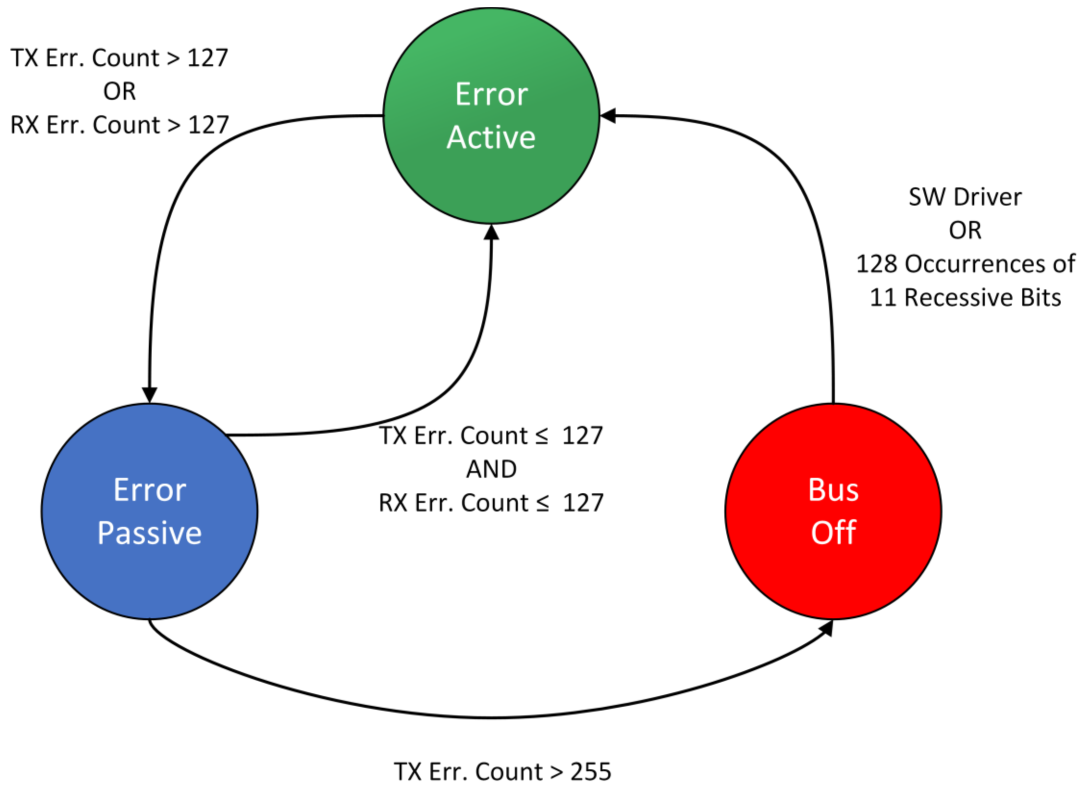


Figure 11 CAN Error State Machine

The values at both the TX and RX error counters provide the logic for determining which of these three modes the CAN device is in. As shown in Figure 11, an error-active node becomes error-passive if either the TX or RX error counters are greater than 127. The counters continue up to 255, where all values between 127 and 255 constitute the error-passive state. If both the RX error count and TX error count drop below 128, the node returns to error-active with its current RX and TX error counts. If, however, enough TX errors occur and the TX error count exceeds 255, then the node goes into bus-off mode. Upon reaching bus-off mode, the node must wait for 127 occurrences of a consecutive 11 recessive bit sequence on the bus. This can be either from the bus being inactive for that period of time, or from 127 frames of any kind being transmitted on the bus. Once either of those occurs, a CAN controller is allowed to return to transmission on the bus.

2.2 CAN Transceiver

CAN specifies two standards for physical layer signaling and topology, Low-Speed CAN and High-Speed CAN. Most CAN networks use the High-Speed standards, due to their ability to support all rated CAN bus speeds, while the Low-Speed CAN standard only allows for much lower speeds but with higher fault tolerance. ISO 11898-2, the High-Speed CAN standard, specifies a two-wire solution for CAN messages, with the differential voltage between the two wires producing the digital signals that the CAN Controller recognizes. These two wires are CANH and CANL. One key feature to note is that the standard does not require common grounds or voltages between nodes, only that the relative voltage from any node is not more than 4.5V offset from the expected voltage of 2.5V in respect to every other node’s ground. This standardized limit is lower than many transceivers are rated for, with some being rated for common mode voltages of up to 36V relative to ground [9].

The standards do not give the actual internal circuitry for a CAN transceiver, as that is up to the device manufacturer. The following portions of this section use information gained from the TI SNx5HVD251 Transceiver. This device was not employed during the implementation of this thesis but has similar specifications and abilities to the transceivers utilized.

2.2.1 Differential Driver

The driver of the CAN bus is controlled through a 5V CMOS digital input. The logic value placed on the digital input (TXD) will be the logical value driven onto the bus, so a '1' will drive a recessive bit, and a '0' will drive a dominant bit. To prevent a transceiver that is not connected to a controller from forcing a dominant bit on the bus, the input has an integrated pull-up resistor to VCC.

Most CAN devices also have an input to adjust the slope of the driver's output. This is useful for lowering the total EMI caused by a transmitting CAN node due to the lower interference with lower slew rates on the outputs to the bus. This is performed by connecting a resistor to the slew input to ground.

To drive the output, the digital signal goes into a differential output amplifier, which creates the voltages needed to drive the switching transistors. These transistors are both in series with a diode, which acts as a voltage buffer, providing an offset between the rail the transistor is connected to and the dominant voltage that will appear on the bus. When the device is outputting a recessive bit, the voltage on the bus is created by a simple voltage divider between the two rails. In the TI transceiver, the expected value for this resistance is around 9 kΩ.

2.2.2 Differential Receiver

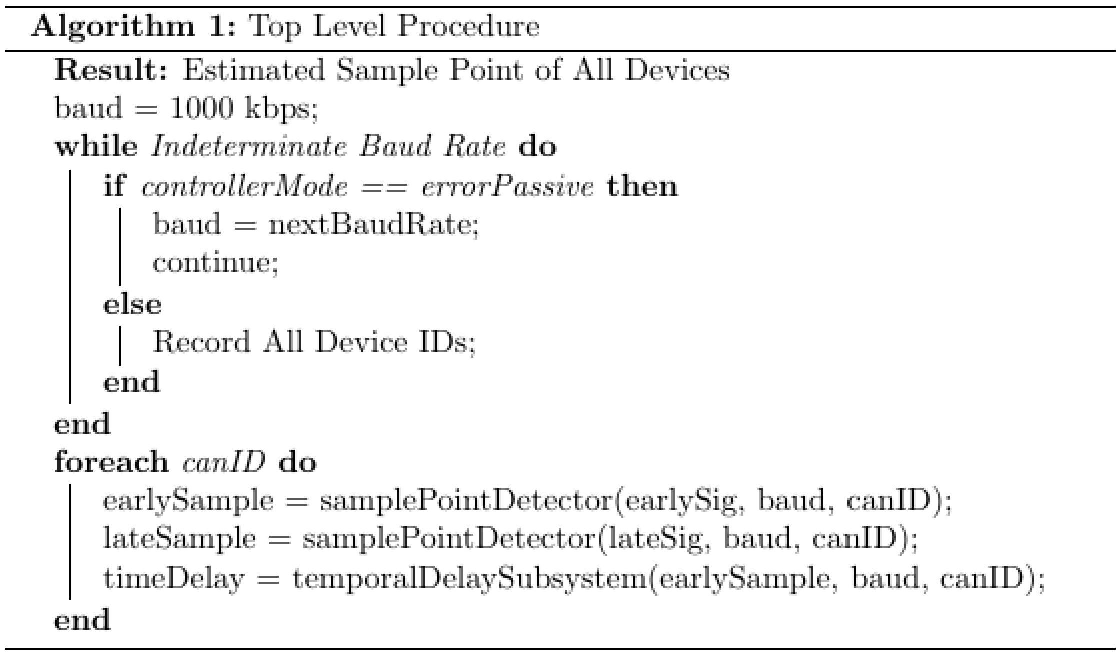
The CAN receiver uses a Schmitt triggered comparator. The receiver side of the chip also contains the recessive driver circuitry, with the voltage divider and receiver being separated by a ~45kΩ resistor from the bus pin. For this device, the comparator has a switching time of around

50ns, compared to the fastest CAN bit timing of 1us, for both positive and negative edges, with a comparator differential resistance of around 75kΩ.

CHAPTER 3: TIMING ANALYSIS PROCEDURE AND TEST SETUP

3.1 Procedure Overview

The procedure demonstrated in this paper is split into multiple sections due to the complexity of the subsystems. This procedure aims to describe a system that could be placed onto an arbitrary CAN network and produce timing analysis for the nodes connected to the network. With this in mind, Figure 12 describes the top-level procedure with references to the sample point and temporal delay subsystems.



```
Algorithm 1: Top Level Procedure
Result: Estimated Sample Point of All Devices
baud = 1000 kbps;
while Indeterminate Baud Rate do
    if controllerMode == errorPassive then
        baud = nextBaudRate;
        continue;
    else
        Record All Device IDs;
    end
end
foreach canID do
    earlySample = samplePointDetector(earlySig, baud, canID);
    lateSample = samplePointDetector(lateSig, baud, canID);
    timeDelay = temporalDelaySubsystem(earlySample, baud, canID);
end
```

Figure 12 Procedure Block for Top Level System

The first section of the procedure is the simplest in total scope. In it, a device is placed on the CAN bus in listen-only mode. The first goal for this section is to determine the bus speed at which all the CAN nodes are communicating. This tester should not take part in any transmission during this analysis, which is why listen-only mode is used. Listen-only mode is a commonly included mode in CAN devices in which the node will not output anything onto the bus but will still receive messages and errors. An incorrectly assigned baud rate will cause valid data on the bus to be incorrectly interpreted, leading to errors being detected by the device. A bus speed that allows for valid data to be interpreted from the bus is assumed to be the valid baud rate of the network.

The system should go through all common then uncommon baud rates, waiting for a minute to see if the error counter reaches error-passive through a standard CAN controller under the control of the tester. If the counter placed the device in error-passive mode, the system should reset and initialize its CAN controller at a new baud rate. If a minute passes and the device has not reached an error-passive state, the baud rate for which the tester's CAN controller has been configured should be the valid rate.

Once the valid baud rate has been determined, the system should listen and store a list of all CAN IDs viewed on the bus. This list of IDs will be sent to the sample point and temporal delay analyzers to parameterize those properties of each ID. This gathering process has no fixed period in which to run, as differences in baud rate, population on the bus, and time between a node transmitting are all unknown variables.

3.2 Physical Driver

A custom driver is necessary to force both dominant and recessive signals on the CAN bus, while also allowing the CAN bus to drive itself. This contrasts with a standard CAN driver as a recessive signal produced by these devices is not forced onto the bus but is treated as a default overwritable state. The driver needs to be controlled by a digital input while allowing arbitrary voltages to be selectively output from the driver. The driver also needs the ability to allow a high impedance to be selected to not affect the state of the bus. This leads to my choosing a four-input analog multiplexer for the selection between resistor circuits implementing the required voltages and high impedance [10].

3.2.1 Forcing a Dominant Signal

The driver forces a dominant signal using two voltage dividers, which attempts to force a constant 3.5V on CANH and 1.5V on CANL when selected.

3.2.2 Forcing a Recessive Signal

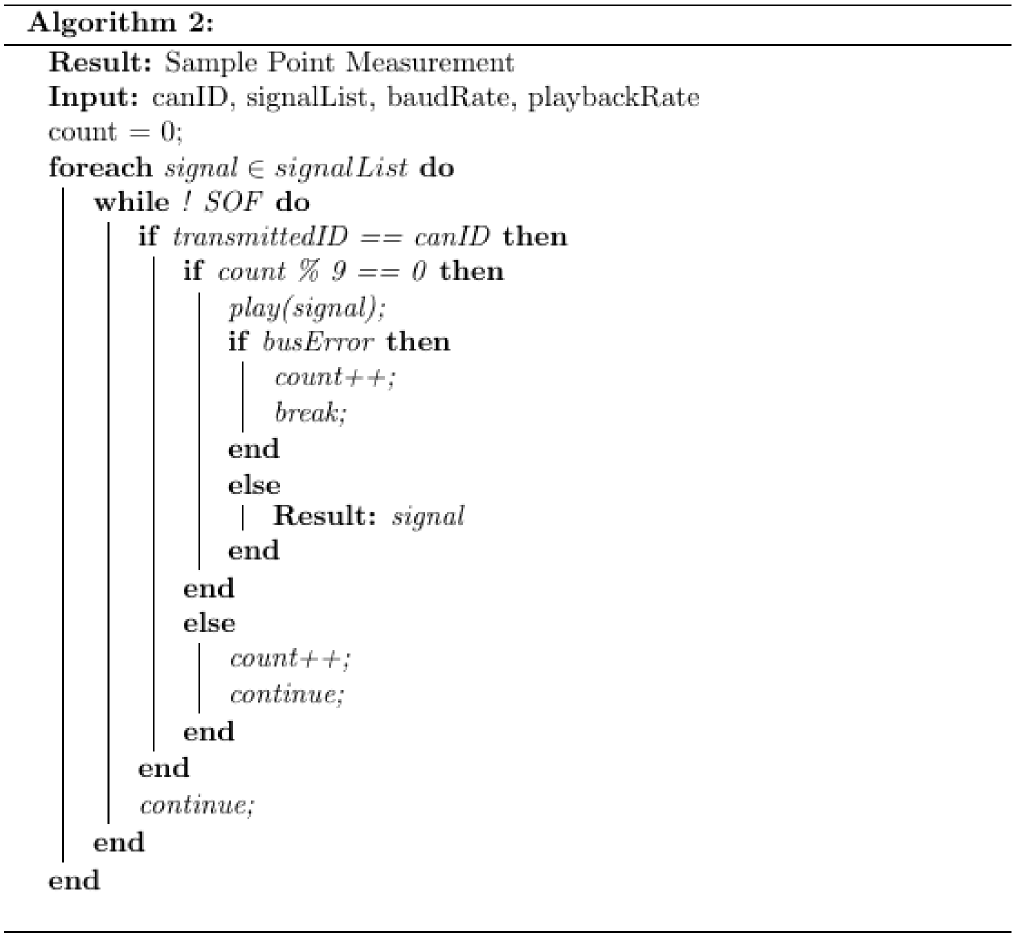
The recessive signal was more difficult to create than the dominant signal, as this had to be able to overwrite a dominant bit on the bus. Many CAN transceiver datasheets have bus limits on the number of devices to which the transceiver could drive a dominant signal. The TI SNx5HVD251 is only rated to transmit to 64 other devices [11], while the ISO standard is that 30 devices must be supported.

Because every node has some resistance between the two CAN lines, and resistance to ground and VCC, the parasitic losses end up appearing as parallel resistances to the load resistors, dropping their total resistance. This resistance drop leads to the transceiver becoming current limited, which could force the voltages of a dominant bit to be within the range to qualify as a recessive bit. Using the fact that a resistor between the CANH and CANL multiplexers would appear in parallel with the load resistors, this effect was created for a network with only two devices. A 20Ω resistor was chosen to maximize the voltage drop across the bus while keeping within the analog mux’s current limit. This resulted in a 75% drop in the load resistance of the network.

3.2.3 Forcing a High Impedance Signal

For the high impedance signal, the differential and internal resistance limits given by ISO 11898-2 were used to match the resistance of a recessive CAN transceiver closely. A resistance of 50kΩ was chosen to provide a differential resistance between CANH and CANL, and both CANH and CANL had a 25kΩ resistor to ground. These values were chosen due to their ease of acquisition and the values' large margin to the specification limits.

3.3 Sample Point Subsystem



```
Algorithm 2:
  Result: Sample Point Measurement
  Input: canID, signalList, baudRate, playbackRate
  count = 0;
  foreach signal ∈ signalList do
      while ! SOF do
          if transmittedID == canID then
              if count % 9 == 0 then
                  play(signal);
                  if busError then
                      count++;
                      break;
                  end
                  else
                      Result: signal
                  end
              end
              else
                  count++;
                  continue;
              end
          end
          continue;
      end
  end
```

Figure 13 Procedure for the Sample Point Subsystem

The sample point subsystem begins after being configured with a set of signals to playback, the rates at which data is being transmitted on the CAN bus, and a target CAN ID. The first stage is to listen on the bus for a SOF to occur. Once the SOF occurs, the ID of the transmitting device must be determined. Once the complete ID has been transmitted, if the ID does not match the target ID the frame currently being transmitted is of no interest and the subsystem will return to waiting for the SOF.

If the ID does match the target ID, a check must occur to ensure that the target does not enter a bus-off state due to the occurrence of too many errors. Therefore, a counter is kept, which counts how many times the target ID has been detected. This is incremented every time the target ID is detected. After the counter is incremented, if the value of the counter is not a multiple of nine, then the subsystem returns to waiting for the SOF.

If the ID does match and the counter is a multiple of nine, then a wait is applied until the frame is six bits into the data frame. The counter will wait for a multiple of nine to occur allowing the DUT to return to a normal state. The wait until six bits into the data frame minimizes the possibility that a stuff bit error will occur. After the wait, at the detection of a recessive state on the bus, a signal in the set of signals which with the subsystem was configured is played onto the CAN bus. The signal, either coming from the beginning aligned signal list or end aligned signal list, is referenced by an index into the signal list.

After the playback has concluded, the state of the bus is monitored for an error frame to be asserted over the current data frame. If an error frame is asserted, the overwrite signal was detected by the DUT. The index into the signal list is incremented, then returns to waiting for a SOF on the bus. However, if no error frame is asserted, the signal was not detected by the DUT, and one of the limits of the sample zone has been found. Upon reaching this, the subsystem is considered finished and returns the signal to the top-level procedure.

3.3.1 Detailed Explanation for Sample Point Subsystem

Knowledge of the sample point is key to the understanding of the internal configuration of a CAN node. The sample point is the location at which a CAN device reads a bit. By determining the location of the sample point, we know how the tested node responds to a signal produced by our system. This subsystem is run twice per CAN ID, unlike the temporal delay subsystem, because there is the issue of capacitance and switching speeds with any CMOS logic and communications bus. Due to the impossibility of creating a perfect square wave signal, the playback must be run more than once. Each stage of signal transition, from digital logic driving, differential signal driving, to the differential and logical reception, adds nonidealities that preclude a perfect square wave from existing. But as this is impossible, the two runs are needed to find a range at which a signal will be received by the DUT.

The sample point subsystem is run with two sets of signals, those beginning aligned and those end aligned. The beginning aligned signals are named, such as the set of signals is created to have a fixed start point, but with incrementally earlier endpoint. The signal is created utilizing enough discrete dominant samples to produce a one-bit long signal at a given sampling rate. Each successive beginning aligned signal has the last dominant sample in the previous signal replaced with a high impedance sample. This continues until there is only a single dominant sample in a signal. An example of this is shown in Figure 14.

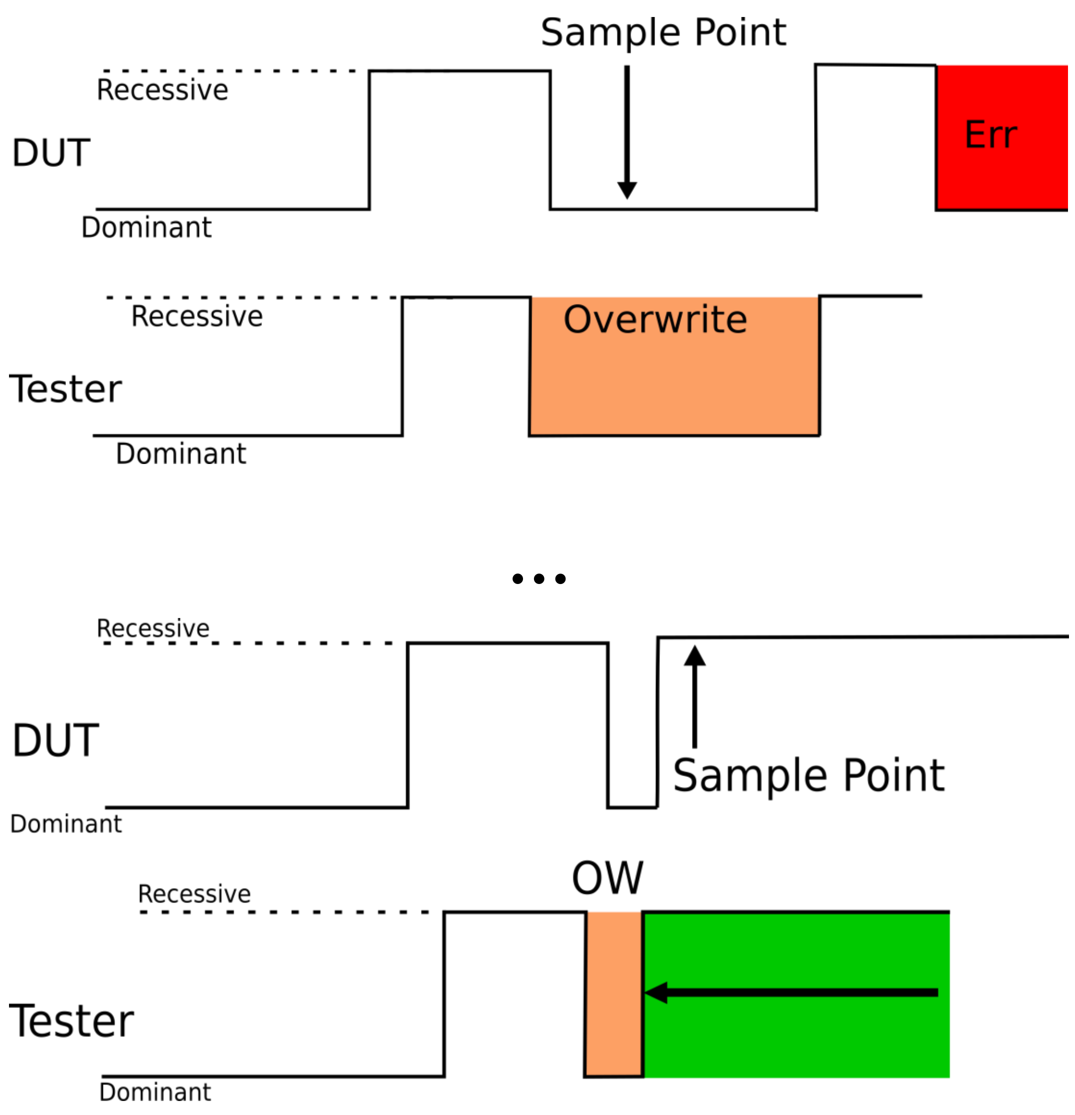


Figure 14 Diagram of Beginning Aligned Sample Point Detection

A similar process is performed for the end-aligned signals. Again, a signal made up of one-bit time worth of dominant samples is used. Unlike the beginning aligned signal, in the end-aligned signal, the first dominant sample is replaced by a high impedance signal for each successive signal in the list. This process continues until there is only a single dominant sample in a signal.

The beginning aligned signals find the last point in which a signal can change from a dominant to a recessive bit without the DUT detecting a dominant signal. The end-aligned signal finds the earliest point at which a dominant signal can overwrite a recessive signal without the DUT detecting the dominant signal. This creates a zone around which the sample point lies.

The idea of thinking about the sample point as a zone instead of an exact point comes from Ziermann's paper on CAN+, an experimental method for encoding a UART signal in a portion of the CAN bit where neither synchronization nor sampling occurs [12]. In this paper, a CAN bit is broken down into three sections, a synchronization zone, and gray zone, and a sample zone. In his paper, the sample zone extends beyond the sample point itself but does not encompass the end of the bit time [12]. Working with sample zones is much easier as there are many variables that could slightly change the relative sample point such, as voltage differences, heat, noise, clock drift and metastability when crossing a clock domain.

3.3.2 SPS Connection to ISO Standards

The method presented in this subsystem is similar to those put forward by the ISO in 11898-2 section 6.7 [9]. In this standard listing, a procedure for determining the delay in detecting an input signal transitioning from recessive to dominant and the delay in outputting a dominant to recessive signal of a DUT is given. The test setup is designed for a single device on a test bench. The ISO procedure starts with the DUT producing an SOF on the bus. Once the tester has detected the first recessive bit, the tester outputs a dominant signal at the expected end of the bit. The delay between detectioning the recessive bit and the output of the dominant signal is shortened each successive run until the DUT loses arbitration.

The overall outcomes of the two procedures are quite different. The ISO procedure is designed to produce the combined delay of the DUT both receiving and transmitting a dominant to recessive signal. The subsystem is designed to determine the sample point of the DUT relative to the round-trip signal propagation delay to the DUT. The ISO procedure treats a state in which the normal activity of the device, being arbitration, is interrupted as the final iteration of the procedure in which the value is determined. In the sample point subsystem, iterations occur with errors being produced until an error state is not observed. At that point where an error state does not occur, and the device continues to transmit data, the overwrite signal is considered outside the sample point range, and the length of the overwrite signal is stored.

While the sample point detection subsystem shares many similarities with the ISO procedure, this was found only after the subsystem had been designed. The ISO procedure should therefore be viewed not as a base upon which the sample point subsystem was developed, but instead as validation for its feasibility.

3.4 Temporal Delay Subsystem

```
Algorithm 3:
  Result: Temporal Delay Measurement
  Input: canID, baudRate, chosenID, DUT_samplePoint
  errorOutSig = generateErrorOut(baudRate);
  validSig = generateValid(baudRate, chosenID);
  badCRCSig = generateCRC(baudRate, chosenID);
  ACKSig = generateACKSig(baudRate);
  count = 0;
  delay = 3;
  delayIncrementer = 0;
  aloneFlag = false; while true do
      if Interframe then
          count++;
          if count == delay then
              if aloneFlag then
                  play(badCRCSig);
                  temporalDelay = recordDecodeBus();
                  return temporalDelay
              end
              else
                  play(validSig);
                  delayIncrementer++;
                  if delayIncrementer % 128 == 0 then
                      delay++;
                  end
              end
          end
          delay(1 Bit Time);
      end
      else
          if transmittedID == canID then
              delay(timeToACK);
              play(ACKSig);
              if ACK not Detected then
                  aloneFlag == true;
              end
          end
          else
              delay(6 Bit Times);
              play(errorOutSig);
          end
      end
  end
```

Figure 15 Procedure for Temporal Delay Subsystem

The subsystem is initialized with a target CAN ID; the sample point zone for the target; the baud rate of the network; a wait parameter initially set to the length of three bits; and an ID for the subsystem to use for communication with the network. The subsystem begins with the creation of four signals, those signals being the Error Out signal, the Valid signal, the Bad CRC signal, and the ACK signal. The Error Out signal is similar in function and design to the initial signals used in the sample point detection subsystem, being made up of one bit time of dominant polarity. When played over a recessive bit in a data frame, the Error Out will cause a transmit error for the device currently transmitting the recessive bit. The creation of this signal is only dependent on the system baud rate and the signal playback rate.

The Valid signal is comprised of the polarity needed to reproduce an entire CAN frame, including CRC, EOF sequence, and bit stuffs. Proper CAN functionality is key to this message, so the ACK and EOF sections of the CAN frame shall be high impedance, to allow for signaling in these sections from the other devices on the bus. This frame needs to use the chosen ID for communication. To maintain compliance with the CAN standard, the ID should not match any IDs viewed on the bus, while also having a high-priority encoding. This has the knock-on effect that the stuff bits and CRC must be calculated at runtime instead of being predefined. The length of this frame should be minimized to increase the total throughput of the CAN bus during the execution of the subsystem, as its reception will lower the RX error counts of all receiving nodes on the network. The generation of this signal is solely dependent on the system baud rate, the signal playback rate, and the choice of CAN ID.

The Bad CRC signal is the foundation for the temporal delay measurement. As such, the reasoning for the configuration of this signal will be covered in detail after the overview of the subsystem. To summarize, the Bad CRC signal begins with a fully valid CAN signal without any form errors initially. However, once in the CRC, the signal is manipulated to generate an invalid CRC, which will cause an error when calculated. The last 5 bits of the CRC shall be a sequence of pulses in the order of dominant, recessive, dominant, recessive, dominant, with each pulse lasting for one bit time. After this, the signal shall be made up of multiple high impedance signals to allow the DUT to communicate on the bus. This signal is dependent only on the signal playback rate and the system baud rate.

The final signal to be generated is the ACK signal. Its generation is dependent not only on the playback rate and the baud rate, but also on the sample point zone generated by the sample point algorithm for the CAN ID being treated as the target for the temporal delay subsystem. The signal shall be high impedance from the start of the bit time to five playback samples before the end of the sample point zone. At that point, the signal shall become dominant for the remainder of the bit period. By delaying the assertion of an ACK to the limit of when a signal could be registered by the DUT, the subsystem gains greater precision over the states of all non-DUT nodes connected to the bus.

After the generation of the signals, the subsystem must wait until the network enters IF. At the detection of the bus entering the IF, a count begins for the total number of bit periods that have elapsed. The count is compared against the wait parameter, and this process is the root of the subsystem’s decision tree. If the count exceeds the wait parameter, the subsystem will take control of the CAN bus and play its Valid signal. A separate count is taken for the number of times a valid signal has been played on the bus. This count is reset every time an Error Out signal is placed on the bus. Every time the valid counter reaches a multiple of 128, the wait parameter is increased by 4 to allow devices in error-passive mode to gain access to the bus. After this process, the subsystem, having placed a signal on the CAN bus, will wait for the network to enter IF.

At the root of the decision tree, if the elapsed bit counter is less than the wait parameter and a SOF is detected due to a device on the network attempting to communicate, the subsystem must determine if the device transmitting on the bus is the target device. This is done by reading the CAN ID transmitted and comparing it to the DUT's ID. If the ID does not match, the device transmitting must be removed from the bus. The only way to remove a device from the CAN bus is by incrementing the device's transmit error counter, which is done by using the Error Out signal. The subsystem waits for a recessive bit in the data segment of the frame and plays the Error Out signal, which causes the transmitting device to detect an overwrite error and increment its TX error counter. Upon playing the signal, the subsystem has completed this branch of the decision tree and waits for the network to enter IF.

However, if the CAN ID, once read, was found to match the target ID, the subsystem must allow for uninterrupted transmission of the frame and wait until the ACK for this frame to continue. At the beginning of the ACK field, the subsystem records, at a high temporal resolution recording rate, the signal on the bus until the point at which the dominant polarity of the generated ACK signal is output from the tester. The recorded signal is filtered to remove noise from the input and then is analyzed for a dominant signal. The filtering and analysis are treated as implementation details and vary based on the specifics of the network this subsystem is measuring and the specifics of the methodology of implementation. Filtering should remove all pulses whose length is less than the switching speed associated with the tester's differential comparator. If a dominant signal is reported, that means that there are still units other than the

DUT that are transmitting on the bus. The subsystem then returns to the standard, waiting for the IF before entering the decision tree.

If the analysis does not return a dominant signal, the subsystem has determined with a high degree of certainty that all devices on the bus have been forced to disconnect due to encountering too many errors, except for the DUT. After this determination, the subsystem waits for the IF to occur, then takes control of the bus at the earliest time allowed by the CAN standard. The subsystem then begins playing the Bad CRC signal onto the CAN Bus. Once the signal playback reaches the last five bits of the CRC where the pulse sequence was inserted, the subsystem begins recording the signal on the bus at a high temporal resolution recording rate. Once the ACK has been reached, the DUT will not assert an ACK, and will instead signal the CRC error after the ACK delimiter. The recording unit records through the assertion of the error signal, and after analyzing the recorded signal, returns the temporal delay from the device implementing the subsystem to the DUT.

3.4.1 Detailed Explanation for Temporal Delay Subsystem

The SPS produced the sample point zone. The sample point zone is offset by the temporal round trip delay between the tester and the DUT and by the nonideal response of the tester's driver and receiver. In a known system, such as an ideal testing environment, this can be calculated through bench testing, and substituted as a correction factor in the measurement. However, this platform assumes no knowledge about the makeup of this network and has to employ experimental processes to minimize the number of unknown variables.

Utilizing the temporal delay subsystem, a tester can determine the total round-trip delay between the tester and the DUT. When run on multiple nodes in a network, the relative offset of the sample point zone can be replaced by an absolute delay based on the tester's timing accuracy.

By modifying the relative delay into absolute delay, all nodes that have been run through the subsystem can have their sample point zones adjusted to the same epoch, the epoch being the logic and clock of the tester itself.

To correctly record the temporal delay measurement of a DUT, the nodes on the network must be in a state where the DUT is the only device that can use a dominant signal to report the CRC error. In his paper, Novak demonstrated a methodology for measuring the sample point of a single node. This methodology required the tester and a controlled CAN node for the detection of the sample point for a single DUT. Much of that procedure would fall apart with more than one device, which is mentioned in the paper [13]. Novak based his designs on the ISO 11898-2 test setup, having the test unit create the sample signal as a valid CAN frame and allow the error detection of the DUT to reveal the delay. This required the DUT to remain in an error-passive state, while the controlled CAN node would be disabled during transmissions from the tester, as it may accidentally signal the error, invalidating the measurement.

This subsystem does have the same requirement as [13] that the DUT be the only device to signal off the Bad CRC. However, we get around having a single device on the bus due to our Error Out signal. The nodes on the CAN bus that are not the DUT are quickly brought up to error-passive mode upon their first transmission through multiple playbacks of the Error Out Signal. The non-desired node, having triggered the playback of the Error Out Signal, will attempt arbitration at the earliest possibility with the message that had originally triggered the error out signal. The non-desired node will continue to trigger the error out signal through its transmissions until the increase in the non-desired node’s TX error count leads the node to enter into an error-passive state, where the node has to delay itself before transmission.

The fact that all the nodes that have transmitted before the DUT are in error-passive mode permits a false assumption that there are no more devices in error-active mode on the bus. This is not true, as other than keeping a count of the Error Out signals that have been used to overwrite every ID on the network, an implementation expensive task, there is no way to know for certain if there is more than one device in error-active mode from the tester's point of view. However, the implementation of the bus-off state does allow for the knowledge of bus occupancy.

As stated in Chapter 2, all CAN devices connected to the network will ACK a valid message, this is true for both error-active and error-passive nodes. However, a node in the bus-off state is considered disconnected from the network. This means those nodes do not take part in any network activity and do not ACK nor report errors. With this knowledge and with the knowledge of the relative sample point of the DUT, the system can determine if any devices responded with an ACK to a message sent by the DUT. If there is no response to the ACK, the DUT is the only device left on the bus. If there is a response, a node is still connected to the bus.

Extreme care needs to be taken to keep the DUT from exiting the error-active state. If the DUT was to enter error-passive mode, the device would respond with a passive error frame, from which no data can be discerned about its delay. The two methods for the DUT to enter error-passive mode are either having a failed frame transmission or receiving an error frame. Preventing the node from receiving too many error frames is difficult, as the process for all other nodes entering error-passive mode will lead to the DUT temporarily entering error-passive mode.

Due to the transmissive nature of CAN errors, the DUT could be forced into an error-passive state which would preclude the detection of the temporal delay. To mitigate this, the transmission of valid frames, must be able to bring the DUT from a full RX error-passive mode to standard error-active mode. To do this, 127 valid frames must be sent before it is possible for any error-passive nodes to transmit and be forced into bus-off mode. By doing so, the receive error counter for the DUT is dropped to zero from 127. This reasoning is also why the process of increasing an error-passive node to bus-off is performed over a much longer time period than the transition from error-active to error-passive. Care must be taken with the devices entering bus-off mode as the increase in received error count for the DUT when non-desired devices are transitioning to bus-off is the same as the initial error out sequence to force the non-desired devices to error-passive mode. If all devices were to enter bus-off with the DUT having entered error-passive mode and immediately transmitting a message, it would pass the sole device check, but would not produce an error frame to measure the delay.

Compared to minimizing the RX error count, minimizing the TX error count is much easier. The transmit error count, which is only increased when a DUT transmitted frame experiences an error state, is easy to prevent. The process of minimizing this for the DUT requires two rules to be followed: the DUT frame is never overwritten, and the DUT frame is always acknowledged. The second rule requires that only a part of the ACK bit for the DUT is recorded and analyzed, as the tester must still output an ACK for the DUT. By using the relative sample point measurement, the amount of information lost from prematurely ending the recording is minimized and should, with a properly configured CAN network, be more than enough for an accurate bus occupancy determination.

The usage and the makeup of the Bad CRC signal was guided by the goal of minimizing external factors in the final measurement of the delay. A CRC error was chosen over the other defined CAN error types since it is agnostic to the knowledge of the device’s sample point. This decision is counterintuitive for a setup where much is known about the sample point. However, all the other errors have quick responses to the error occurring, usually the next bit period. This quick response could lead to a mismatch between the ideal sample point of the controller, which is what the controller is configured to have, and the actual sample point of the controller due to synchronization and phase error.

The reasoning for utilizing a CRC error to measure the temporal delay instead of other CAN errors originates from the uniqueness in signaling of the CRC error compared to the signaling of all other errors. The CRC error is signaled three bit periods after the error is detected, and up to 18 bits after the error occurred. This is in comparison to all other error signaling which occurs the bit after the error is detected [2]. With such a long time frame, there are multiple opportunities to force a hard synchronization to minimize phase and time quanta offsets that could occur in a stuff bit error. The bit error cannot occur during our own transmission, and the stuff error requires a long period of time in which the nodes may become out of sync. That leaves the ACK error to which the transmitter is required to respond and the form error, which would leave the sample point issue from earlier.

The deliberate sequence of pulses at the end of the CRC is designed to provide multiple hard synchronizations for the DUT. The sequence also provides a unique view into the driver and receiver characteristics of the tester. By measuring the length of time that the recessive and dominant bits are detected, the effect of the nonideal components on the network can be factored into the temporal length of the signals. This permits a correction factor for the bus measured value to be found and applied without using bench testing.

3.4.2 TDS Constraints

This subsystem does have some constraints that conflict with features of CAN as laid out in ISO 11898-1. These constraints are directed towards the handling of a bus-off CAN controller and its recovery method. ISO 11898-1 13.1.4.4 on bus-off management states that a node in the bus-off state may become error-active after monitoring 128 occurrences of 11 consecutive recessive bits [2]. Such a recovery speed would severely limit the number of nodes this subsystem would work with to 16 nodes in the best case, much less in a realistic case.

The first constraint placed on the system is that all nodes on the network must be configured to ignore the automatic bus-off recovery method provided in the LLC and instead use driver bus-off recovery methods. The basis for this constraint comes from multiple sources, including the Linux Kernel [14], a major CAN system manufacturer [15], and by ISO 11898-1 itself. The standard states in the diagram for section 13.1.4 that bus-off can only be restored to error-active with the auto-recovery method and a user request. Along with this, the standard also states in the basic concepts of CAN, that a node shall only start recovery upon a user request.

The second constraint is that the network must allow bus-off recovery, but that the recovery period is greater than twice the transmission period of the least frequent device on the CAN bus. This constraint is required for nodes already in the bus-off state to remain there until all nodes other than the DUT are in the bus-off state. This constraint has its basis in simple recording theory, where a recording must occur above the Nyquist rate for all information to be retained. If devices recover faster than twice the slowest frequency, then the algorithm will never correctly resolve all temporal delay measurements.

3.4.3 TDS Connection to ISO Standards

As with the sample point subsystem, the procedure given for the temporal delay subsystem is also similar in nature to a measurement procedure given in 11898-2 section 6.7 [9]. The ISO procedure provides a method for determining the internal delay for a node using the delay at which a bit stuffing is signaled by the DUT using a known point created by the tester.

The ISO procedure employs the SOF bit as the known point and allows the bus to enter a recessive state. This recessive state continues until the DUT signals that it has determined that a bit stuffing error has occurred. The delay between the DUT signaling the error and the known time at which the error occurred is the recessive to dominant output delay of a node. This, along with the values from the calculation performed in the procedure described in the sample point algorithm, can be calculated to the internal delay of the node.

3.5 Experimental Test Setups

For the implementation of both subsystems, a set of test benches was made to provide nonideal scenarios with which the system could work. These systems were made with long transmission lines, multiple nodes, and non-standard sampling points. They were designed to test potential edge cases for the system that were viewed as possible sticking points in the procedure.

3.5.1 Testing Constraints

For the following implementation and test benches, some constraints had to be placed on the test setups. The first constraint was that devices would only utilize standard data frames and not extended ID CAN frames. Early in the design process, there were doubts as to whether the hardware for testing was able to handle extended CAN frames. With the inclusion of extended CAN frames breaking the ID detection logic already implemented, the choice was made to limit each node to utilize a standard-length CAN ID.

The next constraint on the system was to permit only one CAN ID to be allocated to each node. This is impractical and not done in the actual CAN implementations, where an ID functions closer to an IP/port pair than a MAC address. With real systems, a single node may communicate across dozens or hundreds of IDs depending on the system encoding scheme. Allowing for this in the implementation would have required the creation and usage of content addressable memory. This modification to the subsystem and implementation would be extensive, but worth it, and will be discussed in more detail in the conclusions.

The final constraint on the system is the restriction of stuff bits located in the data and CRC segments. The removal of stuff bits in CAN networks would be handled at the application layer, as bit stuffs will always be inserted where required in the LLC to maintain clock synchronization. If, however, data is encapsulated before being provided to the CAN controller, bit stuffing can be avoided within the data and CRC segments. This has been shown to reduce the magnitude of errors occurring on the network and reduce clock jitters as well. With this in mind, I chose the restriction to limit stuff bits to only the arbitration and DLC segments for all devices on the network.

3.5.2 Experiment 1 Setup

Experiment one utilized two CAN nodes, both placed 10m from the tester and 20m from each other. A figure demonstrating this setup is shown in Figure 16. The nodes were designed to handle the worst-case scenario for a network of this size. Both nodes used Arduinos communicating with MCP2515 CAN Controllers. The CAN Controllers for both DUTs used high jitter 8 MHz crystal oscillators for their clock signal. During testing, it was found that while the software for these devices was designed to work and configure the nodes for 1 Mbps transmission, the sample point values were out of spec and barely allowed the two nodes to communicate with each other. This occurs without intervention from the tester and is a limitation from the DUTs themselves. It proved an excellent test for the bus-off detection mechanism and the sample point detection subsystem. Any mistakes during implementation in either of the subsystems would be easy to detect during the data recording as these devices should have identical measurements to each other.

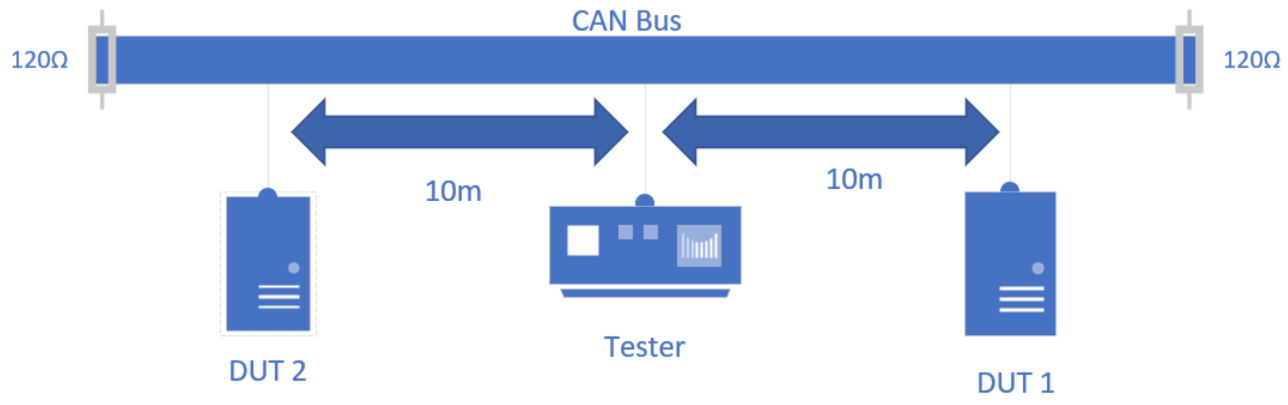


Figure 16 Design for Experiment 1 Test Bench

<u>3.5.3 Experiment 2 Setup</u>

Experiment two utilized three CAN nodes communicating at 500 kbps. These three CAN nodes were set up as shown in Figure 17. The test bench was designed to validate the 500 kbps speed on the tester. This demonstrates that the system can work at multiple bit speeds, not just the 1 Mbps speed for which the design was created. The nodes still use the same high jitter Arduino setups as before, but with half the required bit speed, they are better suited to the task. This also tested the ability to work with nodes on separate grounds, as all nodes were independently powered by separate household power adapters. As these power adapters did not use grounded connectors, and as each node was connected to outlets on separate breakers, these nodes had different dominant and recessive voltages in reference to the tester's ground. All the DUTs are an asymmetric distance from the tester to validate if a correlation between temporal delay measurement and distance can be determined by the data returned from the tester.

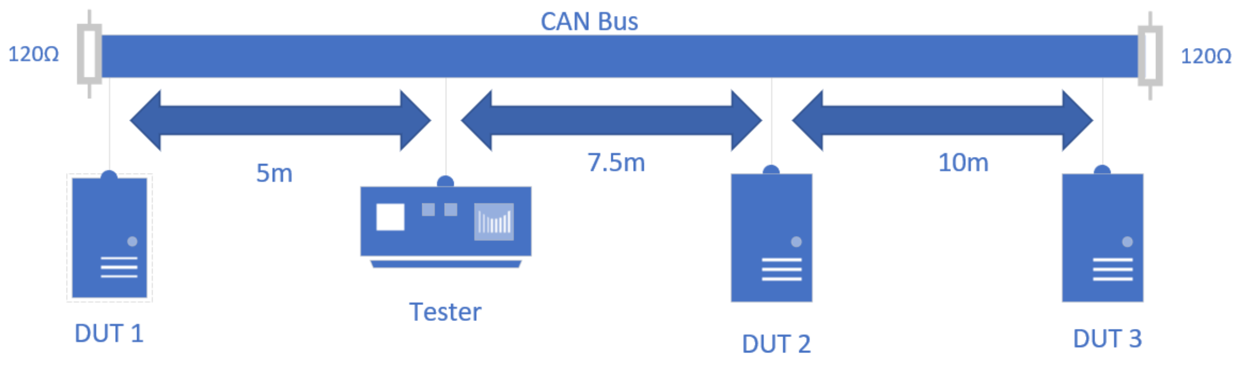


Figure 17 Design for Experiment 2 Test Bench

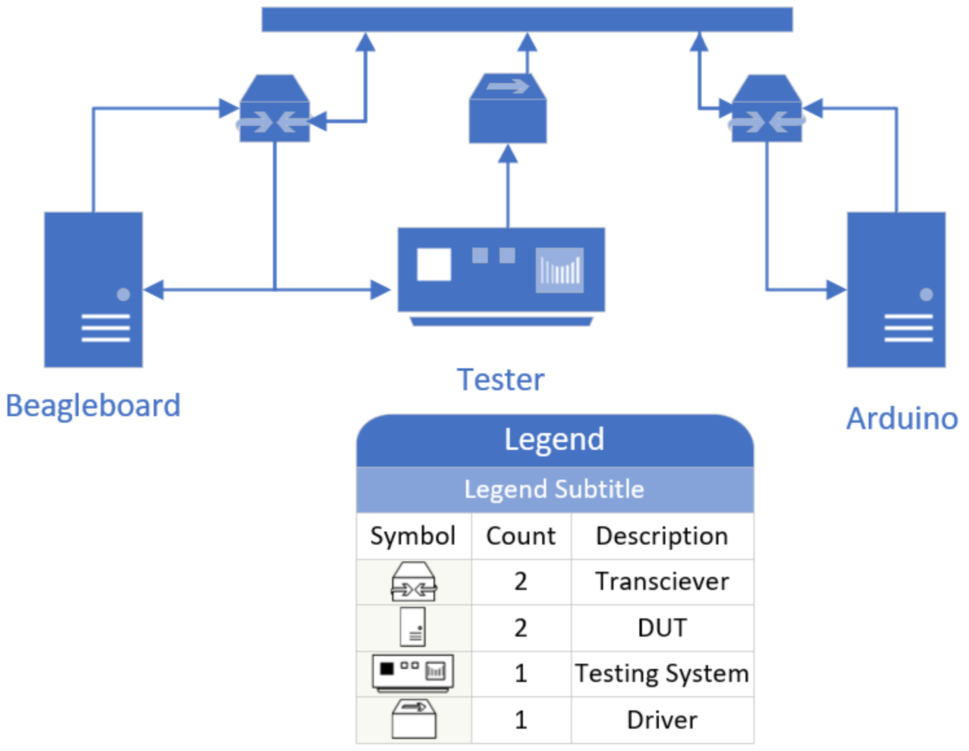
3.5.4 Experiment 3 Setup

Experiment three tests the linearity of the tester in response to bus length. Throughout earlier tests, oscilloscope readings showed that the bus behaved differently at extreme bus distances. Oddities such as greatly increased settling time and possible signal reflections were observed by placing an oscilloscope at the input to the tester's differential comparator. These oddities are caused by the bus becoming a transmission line at increasing distances, with capacitance between the lines and inductances leading to the wave reflections and less than optimal switching times. With this in mind, I wanted to view how the results from the bus would change with the distance from the tester to the DUT being the only variable.

A BeagleBone Black was used to communicate with the tester at 1 Mbps across the varying transmission line lengths. Seven distances: negligible; five meters; seven meters; seven and a half meters; 10 meters; 20 meters; and 30 meters were tested for ten iterations at each distance. The bus length was made up of precut ethernet wires, with CANH and CANL both being a twister pair. This wire was routed and placed in the test environment in such a way so as to minimize coupling with other sections of the wire.

<u>3.6 Verification Test Bench Setup</u>

The verification setup was designed to be the initial test setup for the system and would utilize bench testing to evaluate the accuracy of the measured data. The configuration of this setup is shown in Figure 18**.** The setup used two devices, a Beagleboard Black using a Bosch CAN IP in the SoC and an MCP 2551 transceiver, and an Arduino using an isolated MCP 2515 CAN Controller running at 16 MHz also using an MCP 2551 transceiver. The tester in this setup utilized the standard driver circuitry and the same CAN transceiver as the Beagle for reading data on the CAN bus. The use of a common transceiver reduced the number of bench tests and calculations required to calculate the percent error calculations.



Symbol	Count	Description
	2	Transciever
	2	DUT
	1	Testing System
	1	Driver

Figure 18 Design for Verification Test Bench

3.6.1 Physical Verification of Unit Delay

The physical verification of the input and output delays for each IC was performed using an arbitrary signal generator and a 5 GS/s oscilloscope with a 1 GHz bandwidth. The arbitrary waveform generator was set to generate a square wave pulse with a 5 ns rise time to maximize the measurement resolution. All measurements were taken at the 5 GS/s sampling rate and were scaled to maximize voltage accuracy without over ranging the oscilloscope inputs.

Measurements were taken at key points according to the datasheet for each device, except for the driver circuitry. In the table below are the measurement values for the four major transitions for the CAN transceiver. The transitions measured are the dominant to recessive driver delay; the recessive to dominant driver delay; the dominant to recessive receiver delay; and the recessive to dominant receiver delay. Each measurement has a matching waveform approximation with the points of measurement that are used in the table.

The driver was found to have a maximum switching time of 9.8 ns, and as this system can only measure at 10ns intervals, it was treated as a 10ns switching time.

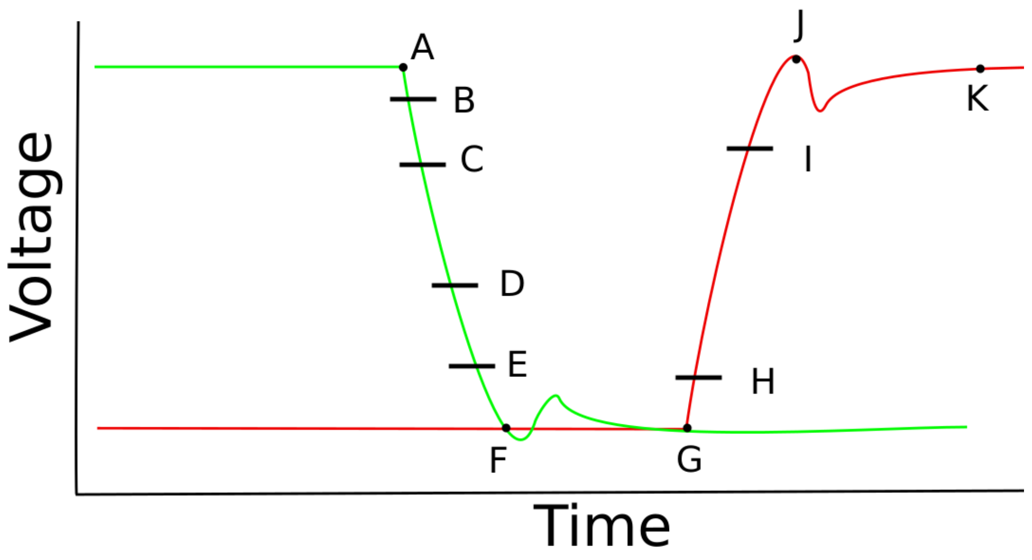


Figure 19 MCP2551 Receiver Recessive to Dominant Delay Waveform

Dark Waveform: Receiver Output. Bright Waveform: Bus Differential Voltage

Table 1 Values for Marked Waveform in Figure 19

Label	Description	Voltage	Time(ns)
A	Start of measurement	2.3V	0
B	90% Input Edge	2.05V	0.4
C	Input Minimum Dominant Voltage Threshold	890mV	4.32
D	Input Maximum Recessive Voltage Threshold	500mV	5.76
E	10% Input Edge	220mV	6.6
F	Input reaches 0V	0V	7.92
G	Output Voltage Rise Initial Point	10mV	64.6
H	Output Voltage V_{IL} Max Threshold	810mV	66.44
I	Output Voltage V_{IH} Min Threshold	2.01V	69.28
J	First Output Peak	3.03V	73.64

K	Output Settled	2.73V	106.8

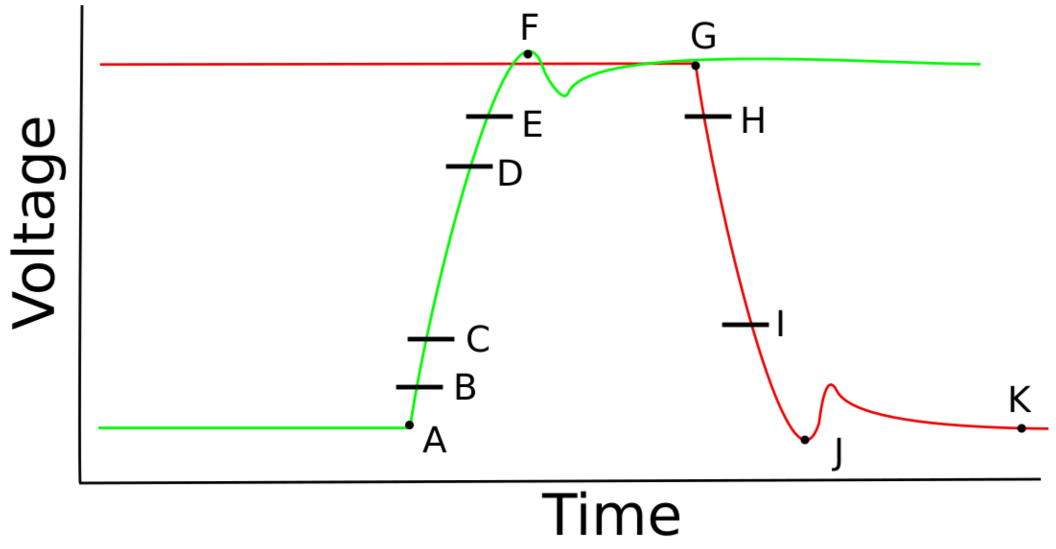


Figure 20 MCP2551 Receiver Dominant to Recessive Delay Waveform

Dark Waveform: Receiver Output. Bright Waveform: Bus Differential Voltage

Table 2 Values for Marked Waveform in Figure 20

Label	Description	Voltage	Time(ns)
A	Start of measurement	-400mV	0
B	10% Input Edge	0V	2
C	Input Maximum Recessive Voltage Threshold	540mV	3.4
D	Input Minimum Dominant Voltage Threshold	900mV	4.2
E	90% Input Edge	3.06V	7.6
F	Input Peak Voltage	4.96V	14
G	Output Voltage Drop Initial Point	2.56V	44.8
H	Output Voltage V_{IH} Min Threshold	2V	45.6

I	Output Voltage V_{IL} Max Threshold	820mV	48.2
J	First Output Peak	-340mV	55
K	Output Settled	20mV	86.4

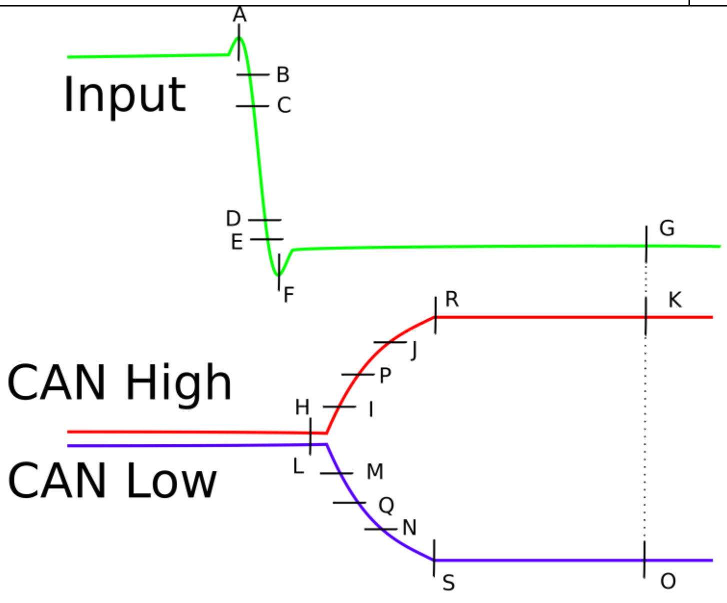


Figure 21 MCP2551 Dominant Driver Delay Waveform

Y-Axis: Voltage X-Axis: Time

Table 3 Values for Marked Waveform in Figure 21

Label	Description	Voltage	Time(ns)
A	Start of Measurement	3.56	0
B	Input Voltage Logic High Minimum	3.5	0.48
C	Input Voltage 90%	3.304	0.76
D	Input Voltage Logic Low Maximum	1.5	4.28
E	Input Voltage 10%	1.26	4.8
F	Input Voltage Lowest Peak	.52	8.48
G	Input Voltage at Stable Differential Bus	1.02	100.8

H	CANH Rise Initial Measurement	2.47	23.6
I	Transceiver CANH Minimum Dominant Voltage	2.76	30.4
J	Transceiver CANH Maximum Recessive Voltage	3	32.1
K	CANH Voltage at Stable Differential Bus	3.5	100.8
L	CANL Rise Initial Measurement	2.33	23.6
M	Transceiver CANL Maximum Dominant Voltage	2.25	33.5
N	Transceiver CANL Minimum Recessive Voltage	2	39.2
O	CANL Voltage at Stable Differential Bus	1.44	100.8
P	CANH Differential Recessive Maximum Voltage	2.65	28.44
Q	CANL Differential Recessive Maximum Voltage	2.52	28.44
R	CANH Differential Dominant Minimum Voltage	3.55	35.8
S	CANL Differential Dominant Minimum Voltage	2.06	35.8

3.6.2 TI SoC Datasheet Minima and Maxima

The last remaining piece of information required to obtain a full evaluation of the sub-bit timing of the BeagleBone was the delays inherent to the Bosch CAN IP. This proved difficult as this information is not presented in the operating system or within the BeagleBone or TI AM3358-Sierra SOC. The information from these datasheets was sufficient to make an educated guess as to internal delay structure. The datasheet revealed that the delay from the expected bit time was ±5ns for the upper and lower datasheet limits. Assuming the worst-case scenario, this would result in the internal delays for the SOC being equivalent to 10ns. The internal jitter from the SOC clock is rated at a maximum of ±1% with a tolerance of 50ppm. On the 24 MHz clock, this results in a total error of ±1.2 kHz [16].

3.6.3 Calculation of Expected Values for Temporal Delay

Below in Figure 22 is the timing path that a signal produced by the tester, in this case a Zynq Z70-20 FPGA, must take. Each delay is marked with a reference letter that denotes what causes the delay at the marked subpath.

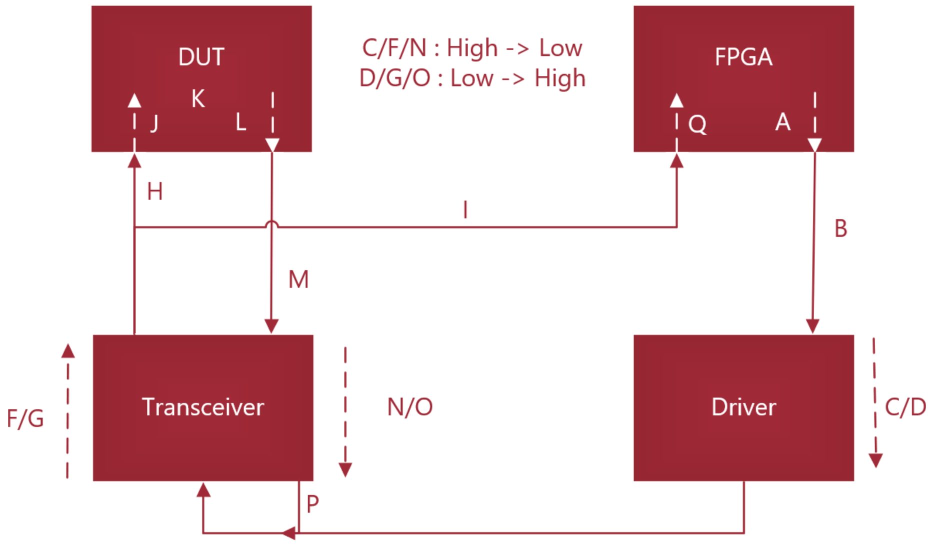


Figure 22 Labeled Signal Delay Path

Delay A is the delay for a signal being selected for playback to the signal appearing on the output of the FPGA due to internal routing delays. Delays C and D are the delay due to the driver switching from forcing a recessive to forcing a dominant signal on the bus, and the delay due to the driver switching from forcing a dominant to forcing a recessive signal on the bus, respectively. Delays F and G are respectively the delays in the RX line of the receiver switching from high to low due to a recessive signal on the CAN bus switching to a dominant signal, and the delay for the transition from low to high on the RX line due to the dominant signal becoming a recessive signal on the CAN bus.

Delay J is the internal routing delay inside the TI AM3358-Sierra SoC for the CAN RX line. Delay K is the internal delay error due to clock jitter in the TI SoC. Delay L is the internal routing delay inside the SoC for the CAN TX line. Delays N and O are the delays between switching from driving a recessive to a dominant state on the CAN bus and between switching from driving a dominant to recessive state on the CAN bus. Delay Q is the internal receiver delay for the FPGA, due to clock domain crossing, internal routing, and recording logic. Finally, Delays B, E, H, I, M, and P are all delays due to the propagation of a signal over wires.

There are three delay calculations that must be made for the three expected bus transitions in the verification experiment. The first is the delay from the tester asserting a recessive pulse on the CAN bus, shown in Equation 2**.** The second, shown in Equation 3, is the tester asserting a dominant pulse on the CAN bus as seen by the tester. The third is the delay for the tester asserting the recessive CRC delimiter as seen by the DUT and the delay for the assertion of the dominant signal by the DUT as seen by the FPGA. This is shown in Equation 4.

$$\tau_{RPulse} = \tau_A + \tau_B + \tau_D + \tau_E + \tau_G + \tau_I + \tau_Q \tag{2}$$

$$\tau_{DPulse} = \tau_A + \tau_B + \tau_C + \tau_E + \tau_F + \tau_I + \tau_Q \tag{3}$$

$$\tau_{RDUT} = \tau_A + \tau_B + \tau_D + \tau_E + \tau_G + \tau_H + \tau_J + \tau_K + \tau_L + \tau_M + \tau_N + \tau_P + \tau_F + \tau_I + \tau_Q \tag{4}$$

All of these delays are measured by the tester and used in the calculation of the approximation to τ_{RDUT}. This equation set can be simplified down utilizing knowledge of the setup. All wires in this setup are less than 15cm and can be assumed to have a total propagation delay of less than 5ns, which is ignored because it is less than our sampling rate. The result of this is shown in Equations 5, 6, and 7.

$$\tau_{RPulse} = \tau_A + \tau_D + \tau_G + \tau_Q \tag{5}$$

$$\tau_{DPulse} = \tau_A + \tau_C + \tau_F + \tau_Q \tag{6}$$

$$\tau_{RDUT} = \tau_A + \tau_D + \tau_G + \tau_J + \tau_K + \tau_L + \tau_N + \tau_F + \tau_Q \tag{7}$$

As shown in Equation 7, the delay for the DUT has components originating from the final dominant bit of the CRC, becoming a recessive bit for the EOF. These components are characterized by the difference between the expected recessive signal length and the measured signal length. This difference originates from the capacitance and inductance of the bus increasing the length of the previous dominant signal. The tester has already recorded the signal length of the recessive bits in the ending CRC pulse sequence and knows the expected length of these signals. By calculating the offset between the expected length of the recessive bits and the recorded length, τ_{RPulse} can be fixed to a measured value. While all variables in τ_{RPulse}, τ_{DPulse}, and τ_{RDUT} were calculated in bench testing, the delay values for the FPGA driver circuitry are heavily dependent on physical bus load. By determining these values at run time instead of at design time, the final calculation can be better approximated. This is shown in Equation 8.

$$\begin{gathered}\tau_{RPulse} = \tau_A + \tau_D + \tau_G + \tau_Q \\ \tau_{RPulse} = \tau_{Measured} - \tau_{Expected} \\ \tau_{RDUT} = \tau_{Expected} - \tau_{Measured} + \tau_J + \tau_K + \tau_L + \tau_N + \tau_F\end{gathered} \tag{8}$$

With the calculation of the delay of the DUT simplified to remove as much dependency from the bench test readings as possible, the estimated τ_{RDUT} can be calculated. The definitions for the τ values are given in Equation 9**.** While τ_F and τ_N can be described with the oscilloscope bench values, τ_J, τ_K, and τ_L are obtained from values given in the datasheet for the BeagleBone SoC. Due to the low jitter of the clock of the TI SoC, τ_K can be assumed to equal zero. The other internal SoC delays introduce some complexity. They were defined as a range of ±5ns from the perspective of the SoC at terms of arrival into the controller. As we are external to the system, a negative delay is impossible, so the range is fixed to have the lowest term as zero ns of delay. This brings the range to 0 to 10ns for both the input and the output signals combined. As this produces a range of values, the median of the range is chosen, 5ns, with an error term of ±5ns.

$$\begin{gathered}
\tau_K = 0ns \\
\tau_J = 5ns \pm 5ns \\
\tau_L = 5ns \pm 5ns \qquad (9) \\
\tau_F = Scope2_F - Scope2_D = 34.2ns \\
\tau_N = Scope3_S - Scope3_E = 31ns
\end{gathered}$$

For the verification testing, all communication is performed at a baud rate of 1 Mbps corresponding to an expected bit period of 1000ns. With the bit period and τ variables having their values fixed, the equation for the delay of the verification unit is simplified to the following calculation.

$$\tau_{RDUT} = 1000ns - \tau_{Mesured} + 75.2ns \pm 10ns \qquad (10)$$

CHAPTER 4: SAMPLE POINT SUBSYSTEM

The sample point subsystem is designed to brute force the sample point for a DUT on the CAN Bus through signal playback. The sample point subsystem was developed using hardware/software co-design principles. Using these principles, the data-intensive workloads were handled in software on the ARM core, while the timing-sensitive workloads were implemented in the FPGA fabric in SystemVerilog. On top of the timing-sensitive workloads, the FPGA fabric also requires some supporting hardware to permit communications with the ARM core. These principles also hold true for the implementation of the Temporal Delay subsystem shown in the next chapter.

This principle is based on those utilized for the CAERUS system [17]. This thesis started as an evolution of the CAERUS system with a complete redesign of the system from earlier implementations, providing the basis for the hybrid architecture. Unlike CAERUS, the tester implementation is designed to run without a controlling computer commanding the FPGA. These differences are due to the fundamental way CAERUS was designed to be primarily software with some Verilog modules for playback, while the implementation given in this paper places most of the work on the Verilog modules.

When designing an implementation using hardware/software co-design, low latency and parallelization are the two main driving principles. Throughout the implementation of this thesis, however, parallelization was never a concern for this algorithm. With the desire for accurate timing, a much faster out-of-order processor could be proposed as a better platform for development rather than a slower clocked FPGA. However, with this thesis, an accurate and deterministic clock and logic timing are paramount for repeatable measurements.

With an out-of-order processor, for example a standard x86_64 processor, standard operating clock speeds would be easily above 2.5 GHz. 2.5GHz results in at least 12 operations occurring on the processor as compared to an FPGA clocked at 200 MHz. These 12 operations, however, do not have deterministic timing. With any processor, a fetch to cache or memory results in a system delay. Even when writing directly in assembly, the timing of CPU instructions cannot be nailed down to the single clock precision like an FPGA can.

Using an FPGA, all calculations on received data can happen in parallel with control decisions, allowing complex decision trees to be traversed in far fewer than 12 desktop instructions. For the number of concurrent tasks being performed with high clock accuracy, a processor would need to have its cache fully deterministic at the assembly level and would require utilizing multiple cores for the implementation. With all of this in mind, the design decision to use an FPGA, with an ARM core providing large dataset creation was the clear choice in terms of accuracy of results and implementation effort.

4.1 FPGA HDL Implementation

The HDL was developed in a modular approach. A great deal of care was taken to create individual modules for all common and separate tasks. Some modules, like one-shot units, and clock dividers, are utilized heavily but will not be discussed in this paper due to their simplicity. Some common functionalities like counters were not made modularized due to the varying requirements for how stimuli and parameters of each counter.

The initialization of the subsystem is mainly handled through automatically generated AXI-4 Lite interfaces [18]. These AXI modules permit PGA registers to be mapped to the ARM memory space allowing for simple transfers of 32-bit messages between the two components. For large transfers, such as generated signals, direct memory addressing, and block memory are used. To the programmable logic, all signals are available in BRAM units with only a 2-clock cycle delay upon requesting data from the BRAM. Key aspects to note for the implementation are that all resets in the HDL are active low as per Xilinx recommendations. Along with this, all state machines are implemented using SystemVerilog enumerators to enable the synthesizer to optimize all state machine logic.

Multiple HDL modules were created to handle certain MAC level CAN controller functionalities. Modules were designed to signal the device entering the IF, the signal if an error message is transmitted on the bus, and to decode CAN IDs. These modules were controlled by a top-level state machine that determined the overall state for the sample point subsystem. This top-level state machine was assisted by a synchronization unit, which would keep the system in time with a transmitting node through mechanisms similar to those utilized in CAN Controllers. Without synchronization to the transmitting node, the high accuracy FPGA would eventually become desynchronized to the bus because all other nodes on the bus have less accurate clocks.

During design, care was taken to keep the HDL in a single clock domain. The asynchronous CAN signal enters the FPGA and goes through a dual flip flop clock domain synchronizer before being analyzed by the sample point subsystem. The dual flip flop design was used due to its ability to remove metastability from the input logic and to factor out any noise less than the setup time of the flip flops. In the rest of the system pulses are used to signal actions occurring at predetermined times which are slower than a clock cycle. The system was clocked at 5 ns period, with the fastest pulse speed possible for the system being generated at a 10ns period per pulse.

The playback unit in the implementation takes a sequence of 16 two-bit samples and produced these signals on the output of the FPGA at a configurable rate. The sequence of 16 signals is provided through the block memory with the signals being generated in the ARM core. The signals have three states, with one bit specifying whether the physical driver should be in high impedance mode, and the other bit specifying if the system should force a dominant or recessive bit on the bus.

4.2 ARM Software Implementation

The processor code’s main function is to produce the signals required for either beginning aligned playback or end aligned playback. These signals need to be generated at run time, because of the need for information regarding the baud rate of the bus. Besides the duty to generate the signals, the software must also configure the HDL system through memory-mapped registers and must also store the returned signal number in the CAN target data structure.

The algorithms used for the generation of the beginning and end aligned signals are designed to ensure proper byte and word alignment. While the signals themselves are represented as two bits, the C code necessitates that bytes are the smallest data size. The algorithm then has to translate these groups of four signals into groups of 16 signals, as the HDL requires full 32-bit words to be passed to the playback unit.

The algorithms were also designed to work around the shortcomings of the implementation of the playback unit. The playback unit will hold the last signal sent to the unit on the output of the device. This requires that the last signal sent from the device be a high impedance signal. If this last signal was not high impedance, then the FPGA driver would hold either a recessive or dominant bit on the bus, at a time at which the system is not intended to force the signal on the bus.

4.3 Experimental Data

4.3.1 Verification Test Bench

The verification experiment was run over four separate known sample points from the BeagleBone Black. The four sample points from the BeagleBone were chosen to give a snapshot of the typical sample range as defined by the CiA. These sample points were 62.5%, 75%, 83.3%, and 87.5% relative to the bit transition from the BeagleBone. Those times were validated by viewing the configured sample point value provided by the CAN drivers in the Linux kernel [14]. All four sample points were individually tested for 30 iterations for both the beginning and the end of the sample point zone. The results are shown in Figures 23 and 24. The expected values are based on a 200ns round trip delay found by measuring the Temporal Delay, in Chapter 5. The data in Figure 23 has an additional 80ns of delay attributed to internal playback delays.

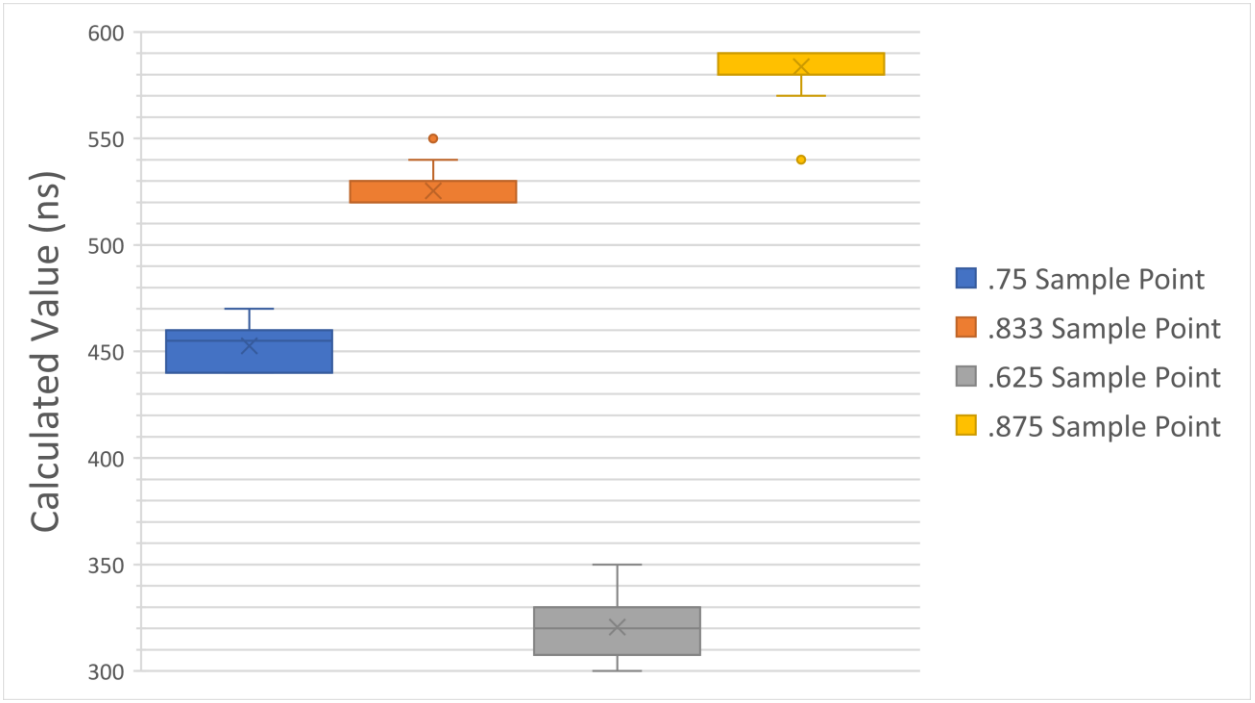


Figure 23 BeagleBone Black Beginning of Sample Point Zone Distribution

Expected Value .75: 470 ns, .833: 550ns, .625: 350ns, .875: 600ns

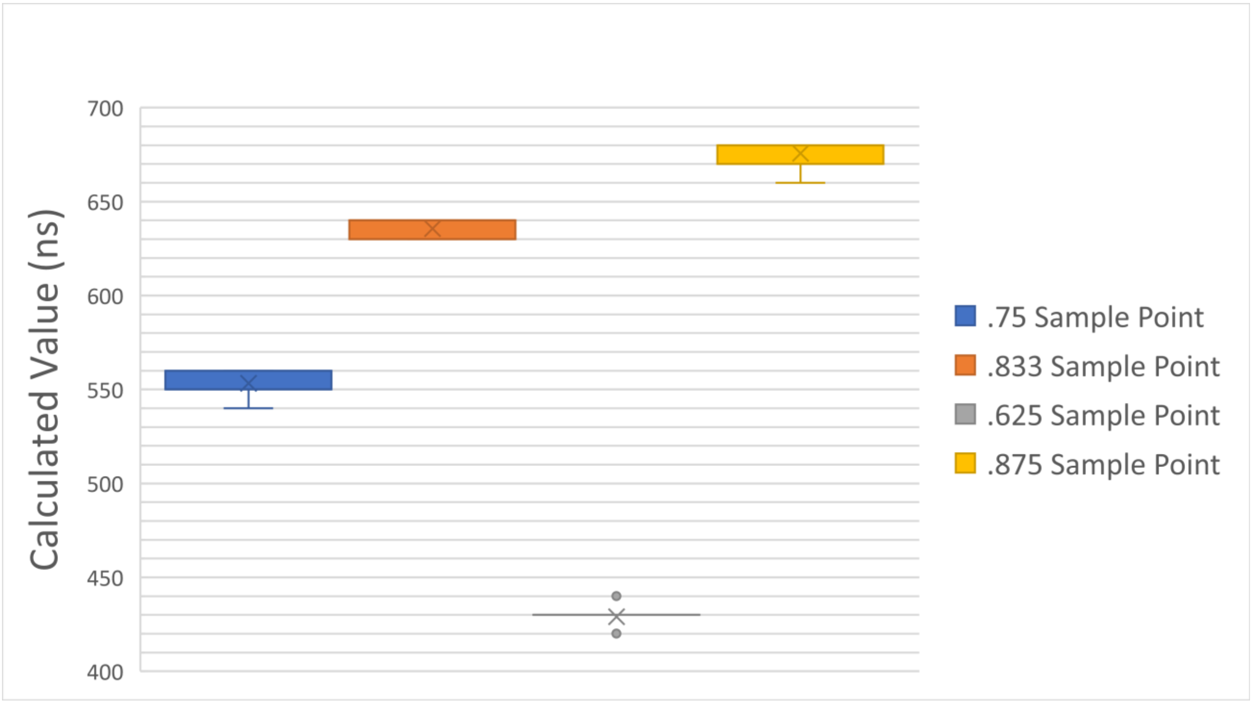


Figure 24 BeagleBone Black End of Sample Point Zone Distribution

Expected Value .75: 550 ns, .833: 630ns, .625: 420ns, .875: 670ns

For the beginning aligned signals each measurement except for the .625 sample point had a maximum standard deviation of 10.5ns, which is close to the sampling rate of 10ns. This standard deviation is equivalent to 1.05% of the bit period for the experiment and is considered excellent for a test in which values can differ based on environmental changes. These environmental changes could result in up to 30-40ns off offset, as the bench measurements of the DUT transceiver were found to have a third of the maximum delay per datasheet specs. These tests were performed over three hours per sample point, so for the deviation to be nearly a single sample is a testament to the consistency of the FPGA implementation.

For the end aligned signals, each measurement except for the .625 sample point had a maximum standard deviation of less than 7ns. The standard deviation being less than a sample shows how the recessive to dominant transition occurs in a much shorter time frame as compared to the dominant to recessive transition of the beginning aligned signal. The mean of these values closely align with a 200ns round trip delay from the tester to the DUT.

The beginning and end aligned sample point measurements for the .625 sample point range seem to be the odd ones out. This test was run twice to validate that an issue had not occurred with the initial experimental run that provided such strange behavior. For the beginning aligned data the standard deviation of 20ns, twice the standard deviation of the other sample points, and the much wider range of outliers is difficult to explain but could be caused by the Bosch CAN IP in the Beagle being unstable at this low of a sample point. The instability at low sample point values is possibly due to the sample point registering the edge as a delayed rise and synchronizing to this value [12].

In addition to the oddities of Figure 23, there is the shockingly consistent end-aligned data as shown in Figure 24. Only three different values were measured across the 30 experimental runs per configuration. This tight grouping of data results in a standard deviation of only 3.9ns, 1ns less than the next closest sample point. For this test to produce two diametrically opposed data sets is quite unexpected. However, I would place the beginning aligned signal as the more erroneous measurement, as the mean of the end aligned .625 measurement, 429ns, follows the pattern shown in the other end aligned measurements of being close to 200ns earlier than the known sample point.

4.3.2 Experiment 1 Results

The goal of the first experiment is to demonstrate the tester's measurement validity for nonideal systems. As such, two nonideal Arduino systems were used with a 10m wire connecting both devices to the tester circuitry as shown in Figure 16. By communicating at a speed of 1 Mbps and utilizing an 8 MHz crystal oscillator, the DUTs were running beyond rated datasheet limits. This was intentional, as these devices highlight the fact that software developed for these devices allowed for this rate, but the controllers would be running at an unvalidated rate. This would be the worst-case scenario for CAN transmission, both devices running beyond their stated limits, operating in undefined behavior. Luckily, the devices did work, and the FPGA produced the results graphed in Figures 25 and 26. The expected values are based on a 290ns round trip delay found in the Temporal Delay Data collection.

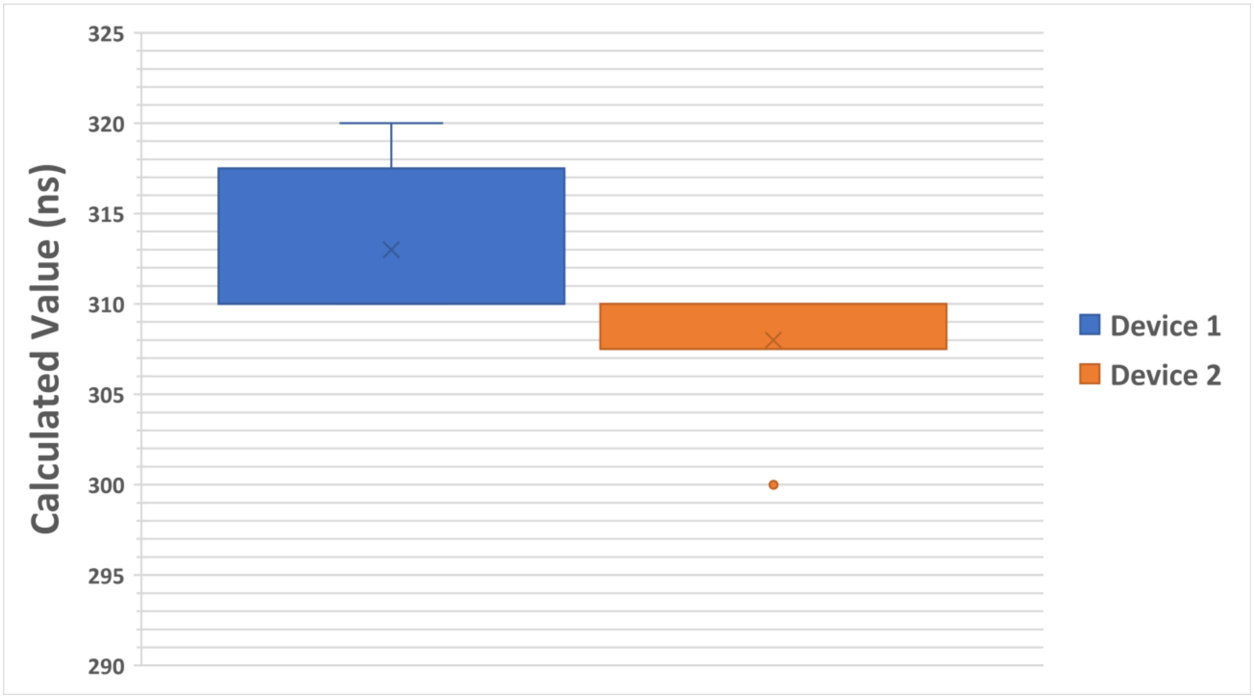


Figure 25 Experiment 1 Beginning of Sample Point Zone Distribution

Expected Values: Both Devices: 335ns

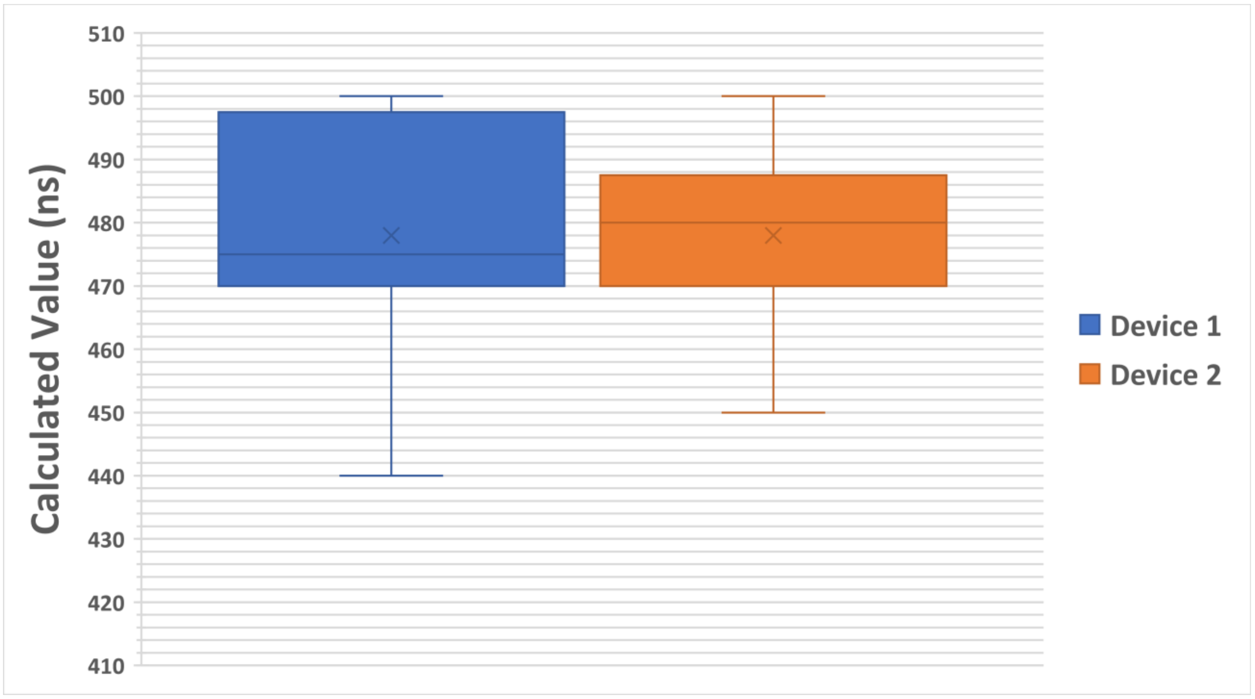


Figure 26 Experiment 1 End of Sample Point Zone Distribution

Expected Values: Both Devices: 425ns

With both DUTs being equidistant to the FPGA tester, any differences between the distribution of the two units is due to physical differences, with clock drift from the crystal, and nonidealities in the transceivers being the two main causes of difference. Figure 25 shows a tight grouping of measurements, with device one having a standard deviation of 4.5ns and device two having a standard deviation of 4ns. This close grouping shows how the transceiver on the Arduino was not heavily dependent on environmental factors.

The much higher grouping of the end-aligned signals in Figure 26 is evidence of the poorly configured Arduino, with a crystal unsuitable for such fast communication. Both devices show an apparent maximum of 500ns, which when combined with the data from the Temporal Delay subsection gives a relative sample point of 790ns. This is 4% of a nominal bit timing and could be due to the clock drift and jitter caused by the low-cost crystal and oscillator circuitry in the CAN controller.

4.3.3 Experiment 2 Results

The goal for experiment two was to validate the operation at lower bit rates than 1 Mbps. With the drop to 500 kbps, a bit would be made up of twice as many signal samples. This increase in samples produced measurements with twice the relative accuracy. This rise in accuracy is offset by the usage of the Arduino DUTs. At half the bit speed, the DUTs ran in a higher stability configuration, but were still left running with few configuration options. With these configuration options it was chosen to select two sample points for the devices on the network. Devices one and two would use a 75% sample point, while device three uses a 62.5% sample point. The measurements provided by the system are shown in Figures 27 and 28. The expected values are based on the measured temporal delay values for Experiment 2.

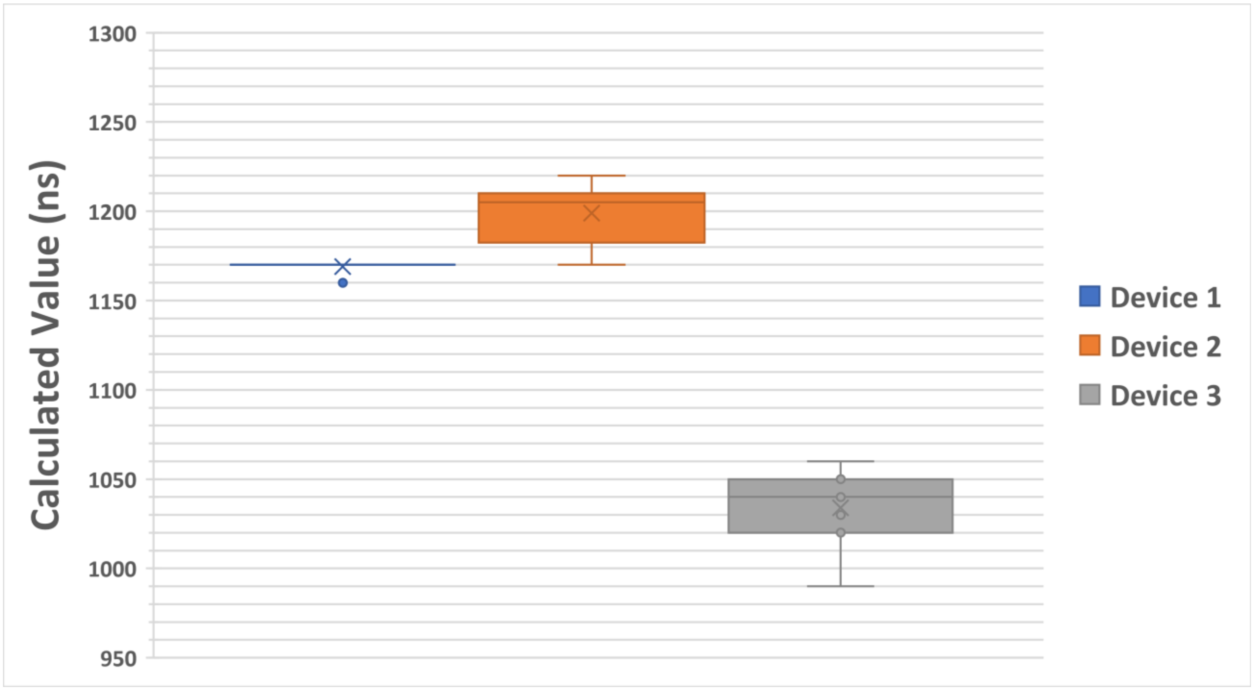


Figure 27 Experiment 2 Beginning of Sample Point Zone Distribution

Expected Values: Device 1: 1320ns, Device2: 1250ns, Device 3: 1000ns

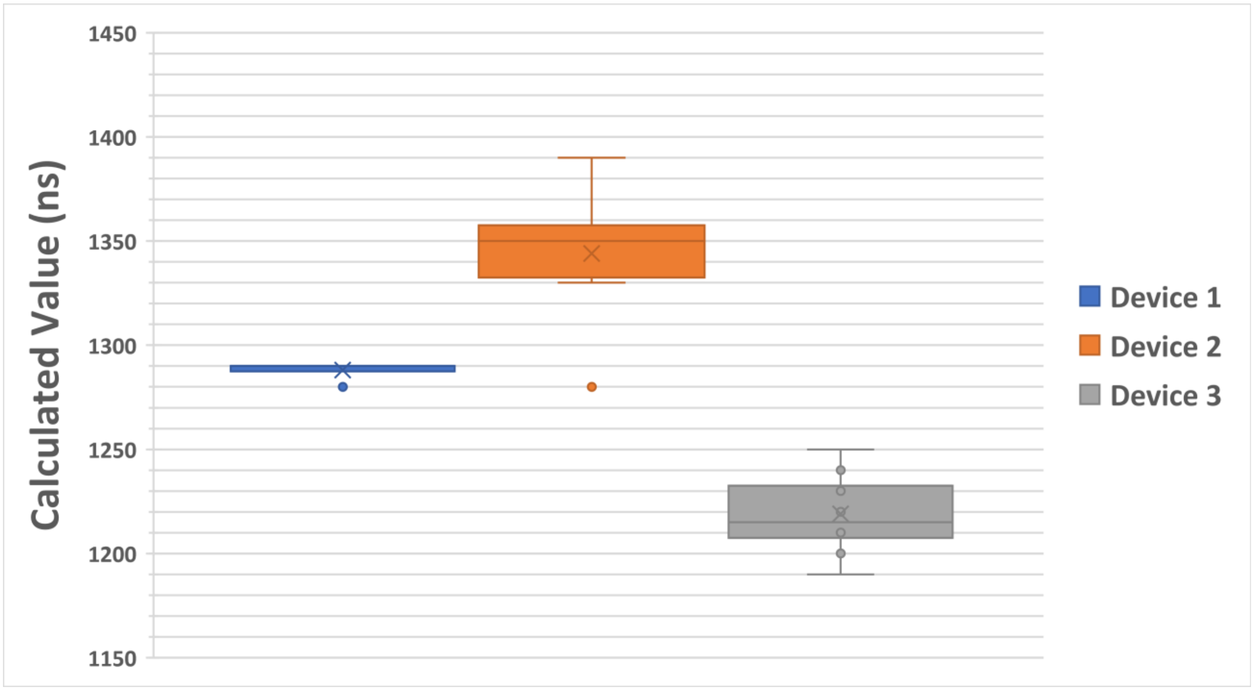


Figure 28 Experiment 2 End of Sample Point Zone Distribution

Expected Values: Device 1: 1395ns, Device2: 1325ns, Device 3: 1040ns

The data gathered from this experiment shows some unexpected deviations from the expected values in both Figure 27 and Figure 28. In the beginning distribution, Device 1 is almost 200ns earlier than the expected range. Such a difference is beyond the error caused by the transceiver but was verified using an oscilloscope. This could be the result of the controller being improperly configured for this device, but no physical phenomenon would produce such an offset from the expected value. Device 2 has a measurement much closer to the expected value and that offset could be caused by the nonideality of the transceiver. Device 3 is later than expected by 40ns, with this difference being due to synchronization. As the delay is later rather than earlier than expected it cannot be caused by transceiver issues and must be caused by controller issues. Device 3 is using a 62.5% sample point as required in the test plan, which as shown in the Verification experiment, could differ from the expected values due to the resynchronization window.

The data shown in Figure 28 again shows some unexpected deviations. Device one is only 125ns earlier than the expected range, much closer than in the beginning aligned signal. An offset of 125ns in the end aligned signal lines up perfectly with a synchronization jump having occurred in the measurement. The signal could have been forcibly resynchronized on the recessive to dominant edge caused by the tester. This would have caused a jump of one time quanta, which at this transmission speed would be 125ns. That would account for the offset in the measured sample point. Device 3 is 200ns earlier than expected. Such a delay is unexpected, and like the measurement for Device 1, must be caused by an issue in the controller. The length of the measurement was validated again using an oscilloscope, and its cause would be a misconfiguration of the controller.

4.3.4 Experiment 3 Results

Experiment three was designed to investigate the nonidealities involved in CAN communication with respect to distance. While performing experiment two, I noticed transmission line echoes in the analog component of the driver circuitry. These echoes appeared to increase the communication delay of the network, especially during any overwriting action. For a better understanding of the nonidealities of the bus when performing the overwrite action, I modified this experiment to investigate the correlation between bus length and sample point measurement. In Figures 29 and 30, I have a plot comparing the measured sample point zone to the distance between the tester and the DUT.

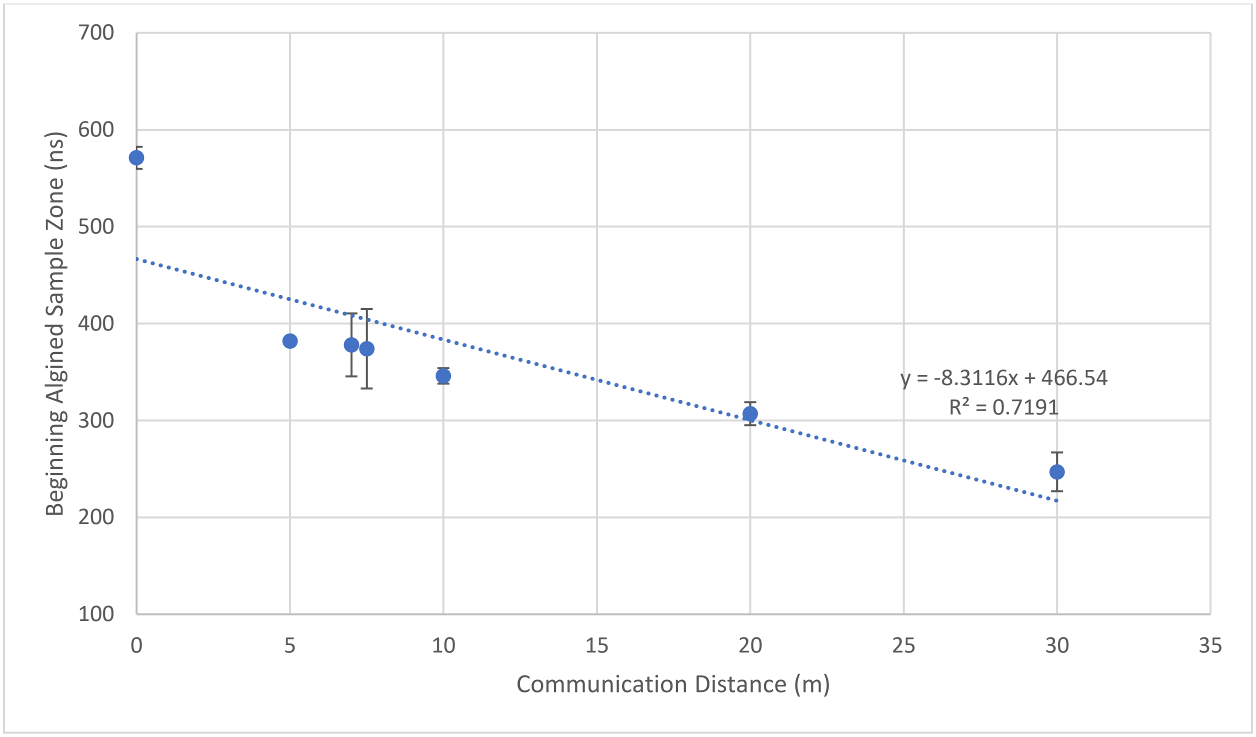


Figure 29 Experiment 3 Beginning of Sample Zone Distribution

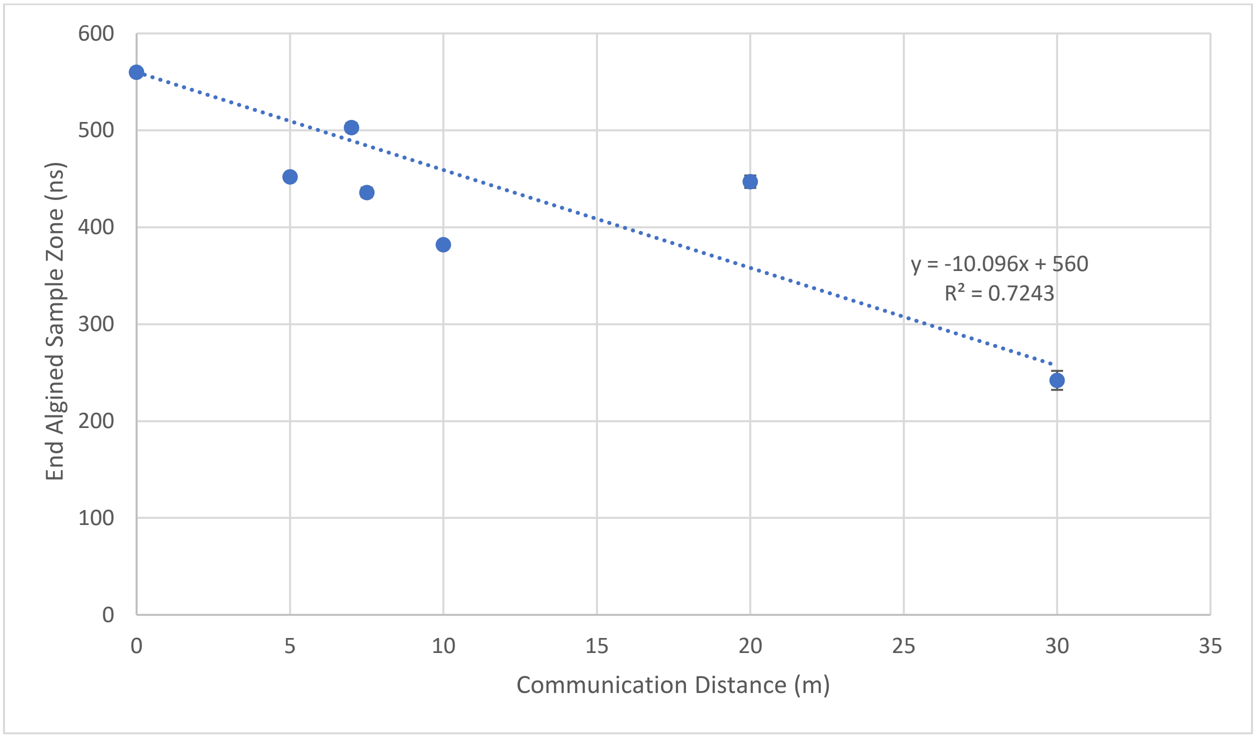


Figure 30 Experiment 3 End of Sample Zone Distribution

The data shown in Figure 29 shows a decrease in the beginning aligned signal that can best be described as roughly linear. The measurement at 0m is designed to show the sample point measurement when there is no propagation delay. All other measurements demonstrate how bus length affects the sample point measurement. The sample point measurement at 7.5m was found to be later than the measurement at 7m, which should not occur due to the fundamentals of propagation delay. This irregularity is most likely due to the node at 7.5m not being directly attached to a terminating resistor, unlike all other measurements. This difference could result in the voltage on the bus taking longer to settle at a dominant signal leading to a later than expected measurement. Apart from the 7.5m measurement, between 5m and 30m a decrease in early sample point of around 5ns/m is visible. This rate is important as it is the propagation delay of signals in the CAT5 wire used in this experiment.

The results in Figure 30 are less than ideal considering the quality provided from the data in Figure 29. Although it looks as if it is incorrect, the measurement for the 20m distance is most likely the most accurate measurement of the end-aligned signal. With the known propagation delay of the system, 5ns/m, the measurement at 20m being around 100ns less than the ideal case shows this measurement to be of higher quality than the other measurements. This is due to the termination of this node, which was placed next to the node for this test, instead of further down the transmission line like the 7m and 10m nodes.

One concerning aspect is the 30m node, which was also closely terminated. I believe this measurement suffered from its extreme length, not due to any issues in the BeagleBone Black, but with the Arduino. The Arduino was ACKing messages from the BeagleBone. The Arduino may have been configured to have its synchronization occur after the point at which the tester would overwrite the signal. This would possibly lead to the Arduino reporting bit stuffing errors if the tester overwrote a signal in a stuff bit. Such an occurrence would explain the large offset the 30m measurement has from the 20m measurement.

CHAPTER 5: TEMPORAL DELAY SUBSYSTEM

The temporal delay subsystem is designed to determine the temporal delay measurement between a DUT and the tester. This measurement summarizes all nonidealities that occur between a DUT signaling a bit on the bus to the DUT receiving a signal produced on the bus. The tester must manage the error states for all devices on the network simultaneously and perform high temporal accuracy recordings to correctly determine the temporal delay measurement.

5.1 FPGA HDL Implementation

The HDL for the temporal delay subsystem was made to be independent from the HDL of the sample point algorithm while still reusing code where possible. At synthesis time, the entire temporal delay subsystem is generated along with the entire sample point algorithm. Any shared modules between the two subsystems are synthesized separately to minimize potential issues during the experiment. Both units are disabled at the state machine level and are initialized and enabled at runtime by the ARM core.

In the TDS, multiple modules were reused from the SPS. Due to the modular design of these SystemVerilog modules, there was no need to modify modules that implemented CAN Controller features. The modules that did not have to be remade are the synchronization unit, the interframe detector, the error detector, the ID comparator, the playback unit, the clock divider, and the one-shot modules. This led to a much faster development cycle for this subsystem because of the presence of verified modules at the beginning of the design process for the TDS.

Because this unit requires knowledge of the status of the ACK bit, a module was created to decode the data length section and convert it into remaining bits in the CAN frame. This unit had to be aware of stuff bits occurring between the ID value and the data length value as a stuff bit frequently occurs here due to the large number of fixed zeroes.

The BRAM had to be updated for this unit to allow for the return of the recorded signal for calculating the error delay. Previously only reading from the BRAM was allowed, but in the temporal delay subsystem, both reading from and writing to BRAM is supported. As Vivado only has two input connections to the BRAM module, only a single BRAM connection could be made to the HDL. The other connection was used for communication with the ARM core. Having only a single BRAM connection required the implementation of a half-duplex BRAM controller.

An advanced recording unit with built-in signal storage and analysis was created to allow for both high accuracy recording and quick analysis. The recording unit permitted the bit value after going through clock domain crossing to be saved for HDL and CPU analysis. When determining if an ACK was sent by another node, the tester performs analysis on the recorded bus signal prior to the unit sending an ACK. If there is a run of 32 dominant samples within the unit at any point, or if there is sequence of six dominant samples in the last thirty-two samples, then it is determined that another node on the bus still has the ability to ACK and is not in the bus-off state. This analysis is performed in the HDL and saves clock cycles that would be required to send the signal to the ARM core for analysis.

5.2 ARM Software Implementation

Much of the processor code for this algorithm generates the signals for the HDL to playback. The method creatingthese signals involves numerous array writes that store the specific sequences for the signals into the RAM block allocated for the storage of signals to be played by the HDL. This process uses C macro functions to maximize code reuse in the signal generation, as hardcoding around 100 microseconds worth of signals with individual 10ns samples would create a sizeable surface in which errors could occur.

5.2.1 Configuration of HDL

The required signals for the algorithm are generated in the ARM core using C. Each signal has been hardcoded for ease of debugging. For both the 1 Mbps and 500 kbps runs, the HDL was set to record signals at 10ns per sample. The TDS was configured to play back the overwrite and ACK signals at 10ns per sample and the valid and invalid CRC signals at 250ns per sample. This allowed the resolution to be maintained for the high-speed actions on both bus bit rates while keeping the very long entire CAN frame signals at a small size in the BRAM.

5.2.2 Handling Returned Data

When handling the returned signal, multiple stages of analysis are taken to ensure that the measured delay value is valid. The recorded data includes the pulsing pattern in the CRC, which can be used to validate the current physical conditions of the bus in terms of irregularities from the expected bit periods the tester is playing onto the bus. Along with this information, the time between the last bit of the CRC and the error signal must always be greater than three times the bit period. If a value is measured at less than this length, the CPU will set a flag and rerun the test after a waiting period to get an accurate value. The same is true for values too long to be a valid recorded signal, which in this system is anything more than 500ns beyond the expected three-bit period.

5.3 Experimental Data

5.3.1 Verification Test Bench Results

The verification experiment was run for both the BeagleBone Black and the Arduino as described in the verification setup section. A single test run consisted of measuring the temporal delay to the BeagleBone, then measuring the temporal delay to the Arduino, with a delay of one minute between measurements to allow the network to return to normal operating conditions. The experiment consisted of 30 test runs, with their results given in Figures 31 and 32.

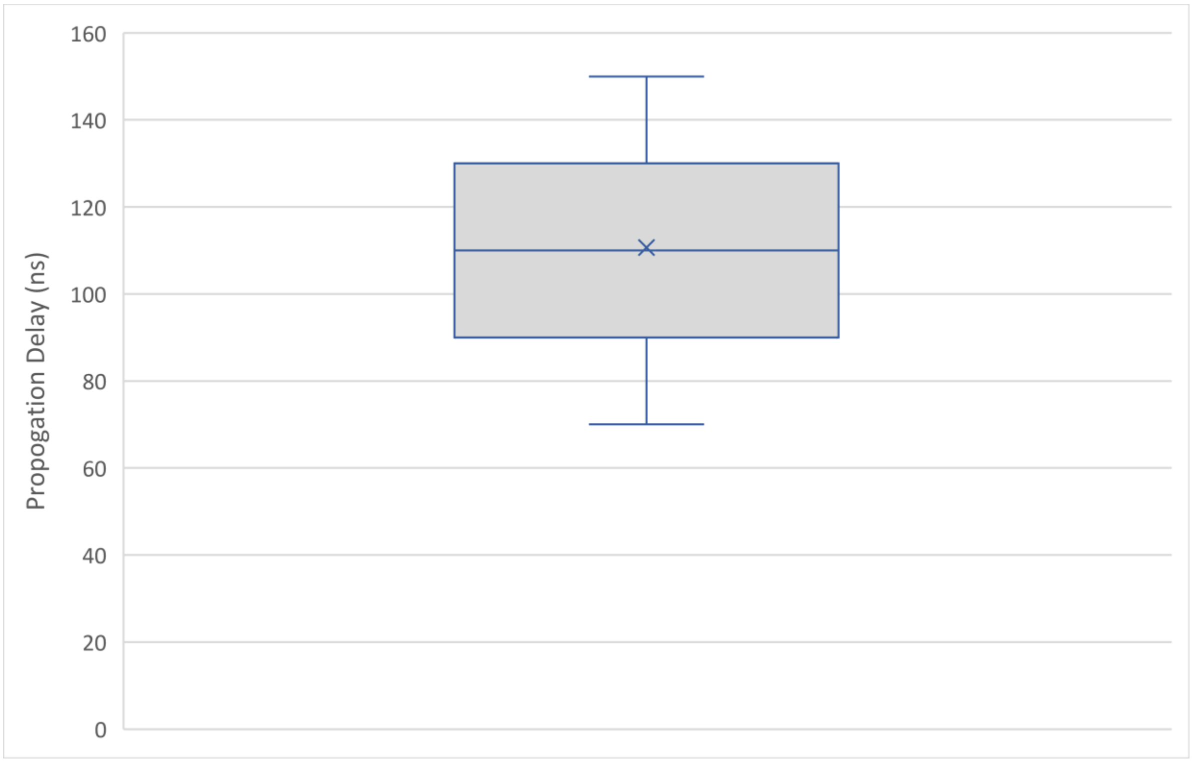


Figure 31 Arduino Calculated Propogation Delay Distribution

Estimated Value: ~100ns

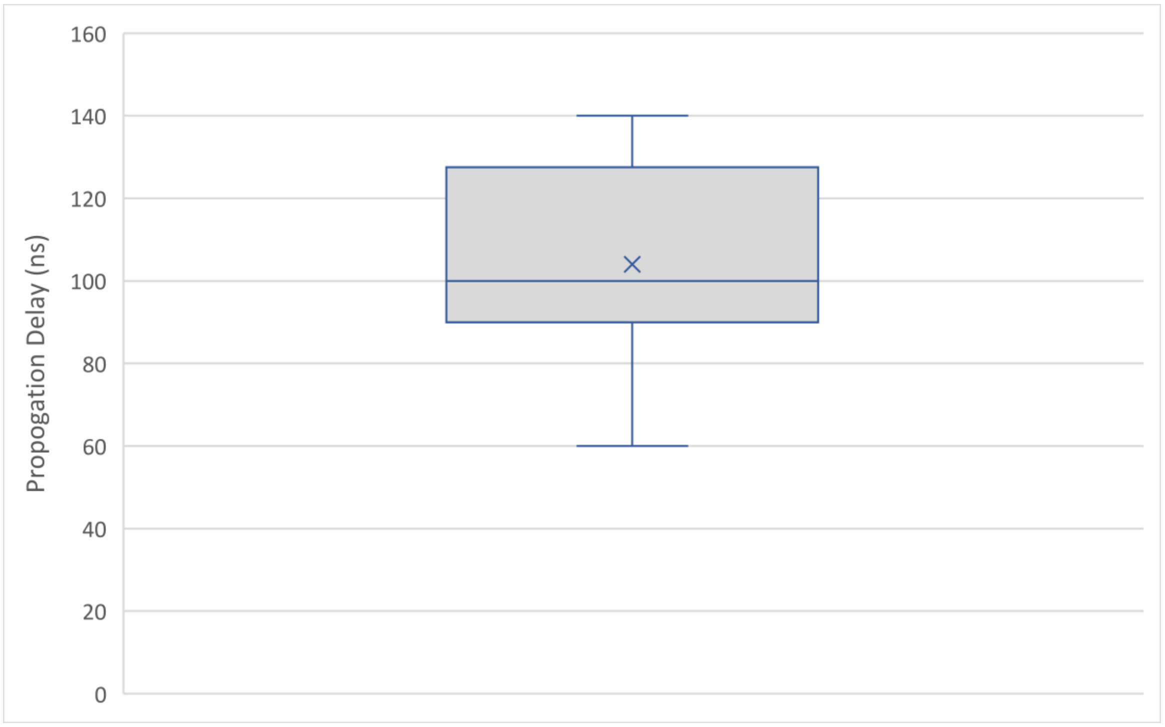


Figure 32 BeagleBone Calculated Propogation Delay Distribution

Estimated Value: 105.2ns

For the verification test bench, I expected that the data in Figures 31 and 32 should both be similar, with any differences being due to transceiver and controller idiosyncrasies. This was observed with the mean of the two measurements only being 7ns different from each other. The real use of this experiment is to validate Equation 10 in §3.6.3 Calculation of Expected Values for Temporal Delay. Using the measurement of the known length final and penultimate CRC bits, I was able to calculate $\tau_{Measured}$.

$$\tau_{Measured} = Length\ of\ Pennultimate\ Bit$$

$$\tau_{Measured} = 970ns$$

With this, the expected value for Figure 37 can be calculated.

$$\tau_{RDUT} = 1000ns - 970ns + 75.2ns \pm 10ns \qquad (11)$$

$$\tau_{RDUT} = 30ns + 75.2ns \pm 10ns$$

$$\tau_{RDUT} = 105.2ns \pm 10ns$$

The value of 105.2± 10ns is almost exactly equal to our calculated mean of 103.914ns. Our measured value can prove the validity of Equation 10. This measured value is used as the one-way temporal delay for measurements in the sample point zone to demonstrate their accuracy.

5.3.2 Experiment 1 Results

The temporal delay tests for experiment 1 were similar in scope to the tests for the sample point zone for experiment 1. A key feature for this test is the demonstration of the effect transmission line delay has on the temporal delay measurement. As these nodes are completely isolated from the DUT, the temporal delays capture not only the delays of the DUT, but also the line delay and the delay of the tester. The tester delay is slightly mitigated with the inclusion of the correction factor based on known vs. expected bit timing generated by the tester. This is shown in Figure 33.

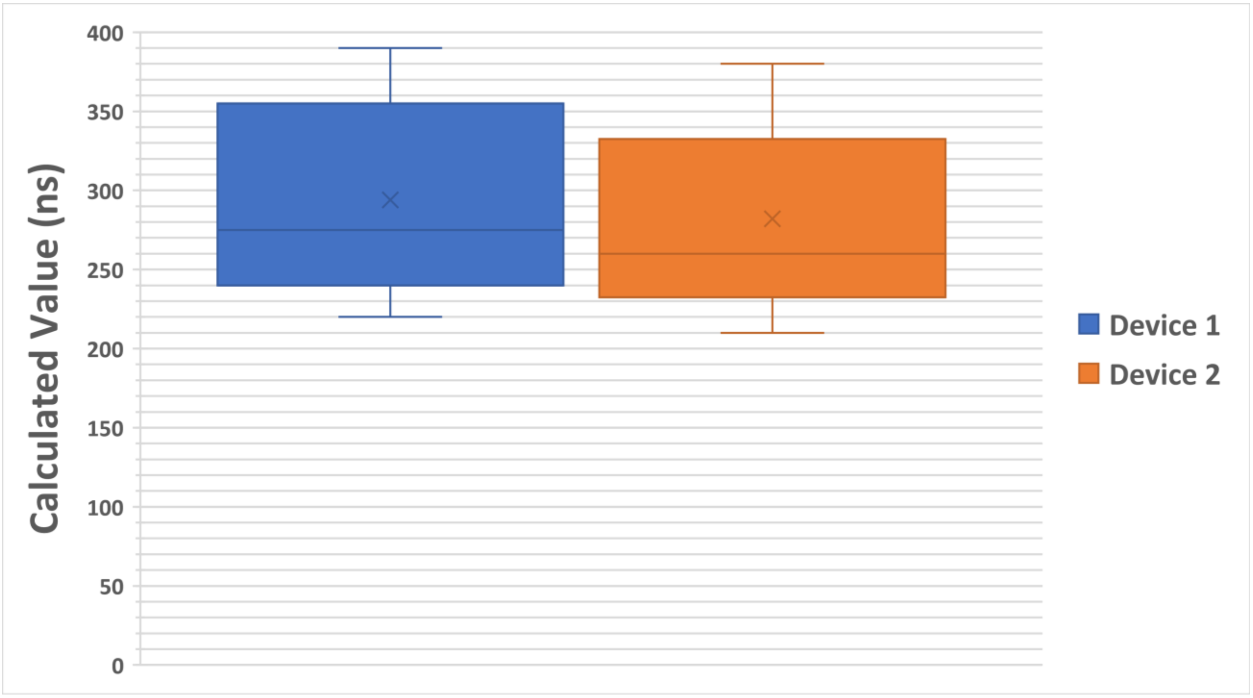


Figure 33 Experiment 1 Temporal Delay Measurements

Expected Values: 250ns< Measured < 350ns

These measurements show the large increase in temporal delay measurement once the nonidealities of a transmission line are taken into effect. For this test, two 10m lines were utilized. This gives each node a 50ns delay added upon its measurement. By factoring out this delay, we get around a 200ns measured temporal delay. Using the knowledge from the verification experiment that 100ns of transceiver delay is expected solely for IC delays, accounting for both the delays associated with the DUT and the tester, 200ns is a reasonable temporal delay measurement. Again, the slight differences in the measurements are most likely due to the individual transceivers having different parameters from each other, just like in the sample point experiments.

5.3.3 Experiment 2 Results

With the devices in experiment 2 running at half the speed of Experiment 1 and verification test benches, some modifications had to occur in the calculation of temporal delay. The main modification was changing the bit period from 100 samples to 200 samples while maintaining the sample rate. Unlike the sample point measurement, which had a doubling of resolution with the decrease in transmission speed, the temporal delay's accuracy is based on the recording rate. The measured values are shown in Figure 34.

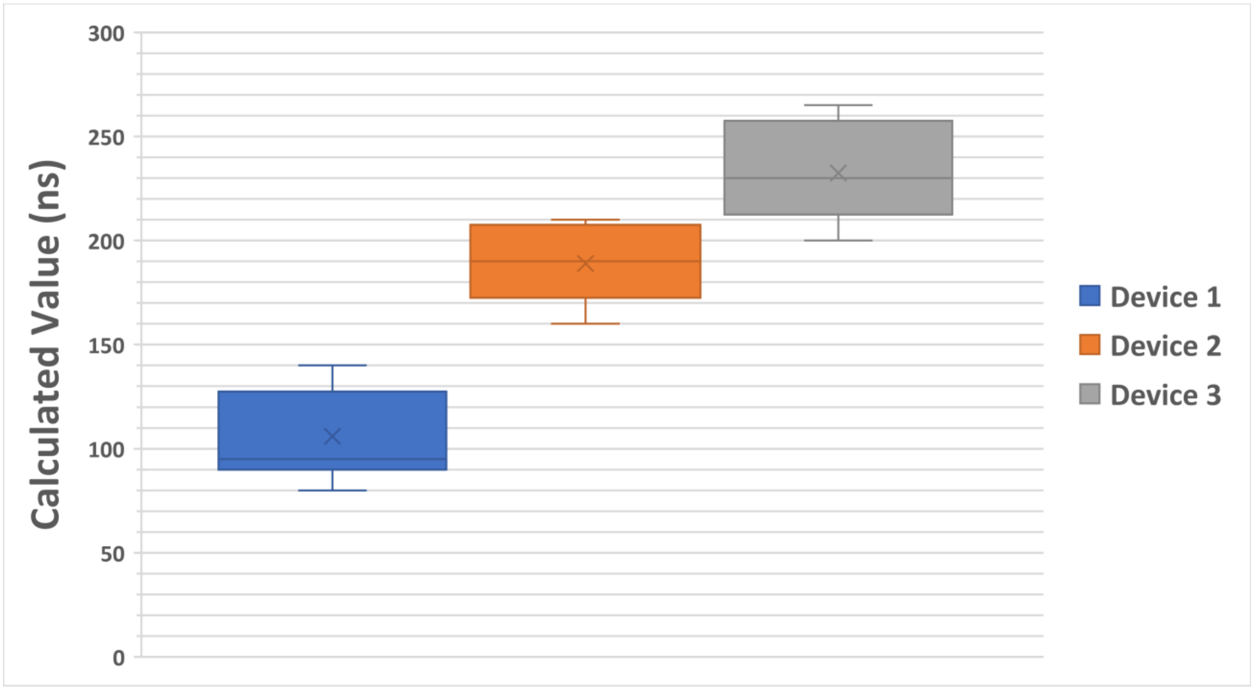


Figure 34 Experiment 2 Temporal Delay Measurements

As can be seen in Figure 34, there is a clear distinction between all nodes. Device 1, being 5 meters away from the tester, should have a 25ns delay. This would calculate down to a 75ns delay, which is possible if the device's transceiver was very well binned. The temporal delay associated with Device 2 is higher than would be expected for a 7.5m delay. I believe this is due to the node being unterminated at this unit, and is instead terminated at Device 3, 10 meters away. This would explain why the two devices have much closer temporal delay measurements even though device two is half the distance to the tester that Device 3 is. With the almost 20m delay between the tester and device 3, a temporal delay in the mid 100ns is expected. The inconsistency between expected temporal delay and measured temporal delay is possibly from transmission line capacitance, causing a settling time up to 100ns longer than expected. The data shown in Figure 34 was validated to be the temporal delay value through oscilloscope testing, validating the result produced by the tester.

5.3.4 Experiment 3 Results

Experiment three was focused on the linearity of the measurements. While the sample point zone measurement dealt with the controller which may have brought in extra error due to clock rates, the temporal delay measurement is geared towards the measurement of physical values rather than software-defined values. As for this experiment the DUT under all distances was constant, any inconsistency would be due to transmission line nonidealities and termination. This experiment was performed over 30 minutes for all runs, and therefore should not suffer greatly from environmental changes. The results are shown in Figure 35.

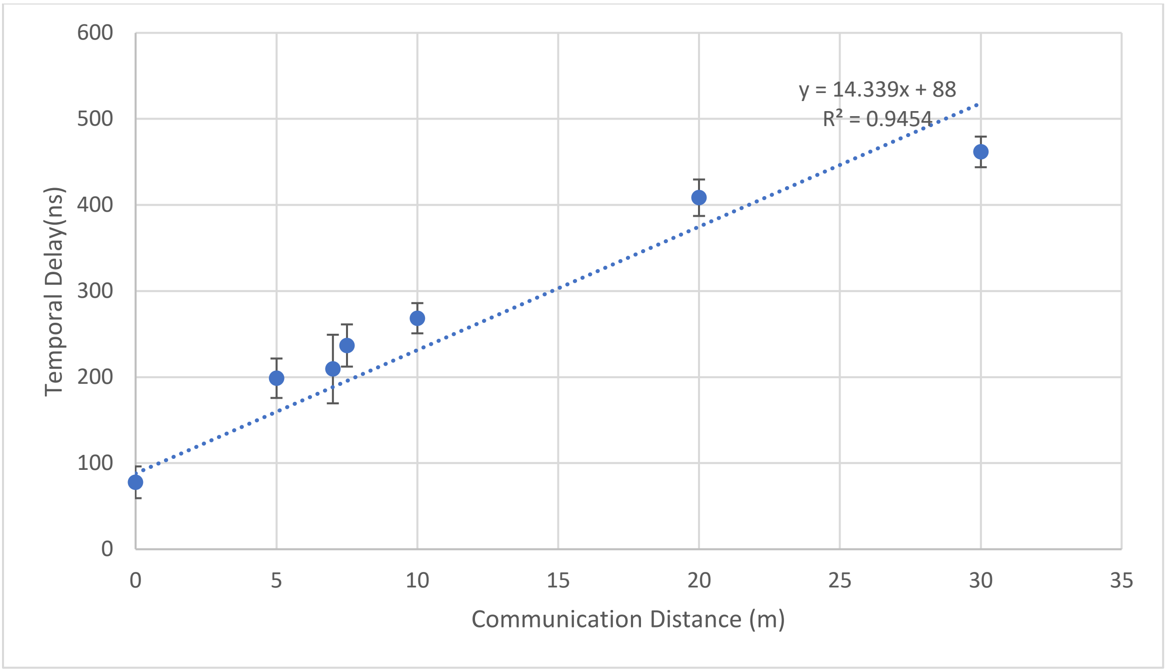


Figure 35 Experiment 3 Temporal Delay vs. Transmission Line Length

This measurement has a much closer linear correlation between the transmission line length and the temporal delay compared to the sample point measurements for this experiment. There are two groupings of signals, the 5-10m signals and the 20 and 30m signals. The 5-10m signals were tested using a simple bus configuration, with the second group tested using an extension on top of the bus to add the extra length. Inside each grouping, temporal delay seems to match with the expected rise more closely, with around 50ns separating 20m and 30m, and only 50ns separating 5m and 10m. I believe that any nonideal increase in temporal delay measurement could be from the different bus topologies. This experiment, therefore, shows that a temporal delay measurement is not just dependent on the physical bus length between the DUT and the tester, but also the total length of the bus.

CHAPTER 6: BROADCAST CONFUSION ATTACK AND SECURITY IMPLICATIONS

The analysis method presented in the previous chapters provides critical information for a new theoretical spoofing attack, dubbed the Broadcast Confusion Attack. This attack utilizes the information gathered on the temporal makeup of the bus to generate a CAN signal which the target of the attack cannot detect, but the other devices on the network can detect. Such an attack has the benefit of not alerting an intrusion detection system of a targeted node while still affecting the state of the overall system. In addition, I hypothesize that with enough development, a Targeted Broadcast Confusion Attack could be developed to overwrite only the payload of a targeted device, unlike the Bus-Off Attack [1].

The Broadcast Confusion Attack relies on knowledge of the timing parameters for the nodes on the attached bus. Once a network has been fully classified, the attack can commence. A target node is chosen, and a rouge message is crafted to spoof a valid message format for one of the target node’s IDs. The rouge message is created to avoid the sample point zone of the target node, while being seen by as many other nodes on the network as possible. This rouge message starts at the data segment of the target’s frame to maximize the possibility of the target’s ACK location matching with our own. If the ACK for the attack message does not line up with the targeted device, the transmission may be interpreted as an error by nodes on the network. This specific variant of the attack would be best described as a Targeted Broadcast Confusion Attack.

This attack is similar in scope to the attack presented in Cho and Shin [1]. That attack scenario, called the Bus-Off attack, utilizes the passive error response of a CAN node to produce an overwrite attack on the node. The Bus-Off attack ensures that a given node cannot transmit on the bus, and an attacking device can transmit in its place. One of the difficulties in the Bus-Off attack is the requirement for transmission synchronization as both the target and the attacker must transmit their SOF simultaneously. To perform this action, statistical models were utilized to find patterns in transmission before the target CAN ID. Such a requirement is not needed for the Targeted Broadcast Confusion attack as its playback synchronizes on the target's data segment, not the SOF segment. This difference allows the target node itself to gain arbitration on the bus, and thus not produce any timing differences that could be picked up by a timing analysis intrusion detection system.

6.1 Broadcast Confusion Attack Setup

A proof of concept for a standard Broadcast Confusion Attack was developed to demonstrate the attack's feasibility. The experiment is designed to show that two devices could receive different information on the state of the network from a single signal. This is the fundamental aspect of the Broadcast Confusion Attack. The ability to overwrite a signal and insert a new signal, which is required to perform the much more powerful Targeted Broadcast Confusion Attack, is not performed here due to the complexity of the creation of a driver that allows signals to be asserted by the tester at 100 MHz speeds while the target is trying to force the bus to either a dominant or recessive value. The current driver is not capable of this and fails to produce a forced recessive overwriting a dominant bit in adequate timing.

To demonstrate the viability of the Broadcast Confusion Attack, a setup using the ideal network, Figure 18 in §3.6 Verification Test Bench Setup, was used to minimize any variability in the network timing. On the tester, a signal was crafted in software to present two different remote information requests on the CAN bus depending on when the message was sampled. The device with the earlier sample point, the BeagleBone, would see a request to ID 0x09B request 4 bytes of information, and the later device, the Arduino, would see the request instead go to 0x523 for 8 bytes of information. Both ID values and data lengths requested were chosen arbitrarily but were ensured to have the same frame size. Remote requests only contain the ID and the data length, minimizing the number of samples needed to define a full frame. These two devices were configured to have highly different sample points, with the Arduino at a sample point of 85% and the BeagleBone at a sample point of 62.5%.

6.2 Broadcast Confusion Attack Outcome

The Broadcast Confusion Attack was performed over 5 minutes on a BeagleBone Black and over 2 minutes on an Arduino. The message played by the FPGA was not changed during this time, and both devices were present on the network at the same time. The Arduino ran for a shorter amount of time due to the slow write of output information from the Arduino to the recording computer.

The BeagleBone Black reported that it received 1,021,657 frames. Out of the over 1 million frames, only 45 frames were treated by the BeagleBone as an error and did not match the planned 0x09B ID for the device. The Arduino received 64,000 frames in its 2-minute run. The Arduino received the same number of frames as the BeagleBone, but the Arduino only reported 64,000 frames. This is most likely due to the internal registers of the MCP2515 on the Arduino overfilling due to the tester transmitting frames nearly constantly. Out of the 64,000 frames, 10 were reported as errors. A larger number of errors was expected for the Arduino, as the delay between signal switching from the tester to the sample point of the Arduino was much shorter than the delay for the BeagleBone. While both devices received messages other than the desired remote frame, neither device interpreted the data intended for the other device, with all erroneous messages being treated as standard CAN error frames. In total, out of the 1,085,657 frames interpreted by both devices, only 55 were erroneous, resulting in a 99.994% effectivity rate.

Overall, this experiment showed a great deal of promise of a successful Broadcast Confusion Attack. This attack was able to send two separate messages in the same frame. If a driver is developed that can overwrite data on the bus at fast enough speeds, the Targeted Broadcast Confusion Attack could become a serious threat to the security of CAN.

6.3 Security Implications

The Broadcast Confusion Attack has multiple security implications for CAN. With the standard Broadcast Confusion Attack signals could be crafted to send data updates only to devices which have no method of reporting the erroneous information. The security implications on CAN would be much greater if the attack could be developed enough to utilize overwrite methods to engage the attack on a frame transmitted by a single target device. The Targeted Broadcast Confusion Attack could bypass frame and timing analysis intrusion detection networks. As the attacker is overwriting the data from the target device and not presenting a CAN ID on the bus at any time, the target cannot detect the presence of an unauthorized use of its ID. Along with this, as the frequency of the attack is determined by the target node, a frequency-based intrusion detection analysis would not detect this attack type. This leaves only the physical parameter analysis intrusion detectors, which would be able to warn of this attack.

6.3.1 Loss of Data Integrity

The most serious consequence of this attack is the loss of data integrity on the network. CAN is designed to ensure data integrity. This means that all nodes receiving the same message is integral to the proper functionality of a CAN network. With this attack, nodes can receive a different message than is transmitted. This can lead to uncertainty over the true state of a network, which in a large metallic object traveling at highway speeds, could be life-threatening.

With this attack there can exist a discrepancy between the truth and the recorded truth and could result in blame falsely being placed on innocent parties, whether they be devices or persons. A frontal collision warning system having its messages being overwritten could result in the loss of safety-critical functionality, leading to an accident. At the investigation for the analysis, recorded data would show no such collision was ever detected, possibly misplacing the blame on one of the drivers.

6.3.2 Loss of Confidentiality

Controller Area Networks have historically used IDs as the sole means of authentication. As a safety-critical, and low-power network, authentication and encryption were considered too costly to require in CAN. Even to this day, production CAN buses are under no obligation to require authentication of a node's identity. Authentication is to be handled at the network level and higher. With the speed limitations of CAN, and with most cars having up to 60 nodes [1], network capacity is at a premium. The additional space that authentication methods would need to ensure the confidentiality of the bus would congest the network.

The Broadcast Confusion Attack cannot lead to a loss of confidentiality of the network on its own. This is because the attack has no foreknowledge of the application layer and working at layers below which confidentiality is ensured. If a loss of confidentiality is desired, it would need to be handled as an addition to the Broadcast Confusion Attack. An implementation could modify the generation of the overwrite data, where data that has been encrypted or formatted to a specification required by the higher layer implementation for a network is turned into the overwrite signal.

6.3.3 Loss of Availability

The CAN network has had many attacks on the availability of the network, with those being the most common form of attack against CAN. Attacks such as shorting CANH and CANL to ground can instantly disable an internal network and would require physical maintenance to undo. Through software, denial of service attacks have proven to exhaust the CAN network, leading to all devices lacking the ability to communicate [19].

The Targeted Broadcast Confusion Attack permits normal bus availability for all nodes except for the target node. The target node is effectively removed from transmitting on the bus and cannot update the system on its status through a given range of CAN IDs. The target would not know about its loss of availability unless this is assured through higher networking layers. For the standard Broadcast Confusion Attack, a node could be thought to be removed from receiving the data intended for other nodes, thereby reducing its availability to receive a set of information communicated on the bus.

CHAPTER 7: CONCLUSIONS

7.1 Improvements to the Timing Analysis Procedure

The current implementation of the analysis procedure works heavily under the assumption that each CAN device has only a single ID. This knowledge is incorrect throughout most implementations of CAN, and stems from the initial limitations on testing hardware. While much of the procedure can be changed to work on a per-node basis, this cannot be done without foreknowledge of a CAN network's makeup. There is no method of determining if two IDs are shared by a single node conclusively and thus would require configuration by the user. Testing each ID and comparing their temporal delay measurements would not be viable, as the process of attempting to disable all IDs except for the target ID would also cause the other IDs for the target node to be placed into bus-off mode.

To modify the implementation to work on a per node basis, a memory structure similar to a routing table would be required to be implemented on the FPGA. The structure would correlate a given ID to a node number, and then the node number would be the activator for the subsystems instead of the ID value. Such a design would considerably increase the complexity of the FPGA side, as the determination of remote frames and extended IDs would need to be considered for full CAN viability.

The node detection methodology could be augmented by a system similar to the methodology given in the CANvas [20]. The detection system in CANvas used ECU fingerprints to match frame IDs to their origination node. By performing the CANvas detection procedure prior to the SPS, the restriction on the number of IDs per node can be removed, and total analysis time reduced.

7.2 Viability of the Attack on CAN FD

In 2016, the ISO formalized the Controller Area Network Flexible Data-Rate, CAN FD, standard. CAN FD has multiple advantages over standard CAN, such as up to 64 bytes of data transfer per frame improved from the 8 bytes for CAN, and with data bandwidths of up to 8 Mbps compared to the 1Mbps of standard CAN. CAN FD has become more ubiquitous over the years, as the requirements for more data to be sent over the internal network of a car have increased.

The updated standard has many features that would actually improve the viability of both the algorithm and the theoretical attack. The first feature being that CAN FD does not support remotely requested data frames. This feature was not supported by the algorithm, and was not included in the standard as the CiA who helped create the first white papers for the CAN FD protocol had previously recommended against their usage [7]. The second feature is the design of an error state indicator in the header for the CAN frame. Every CAN FD node will communicate in a CAN frame if it is error-active or error-passive. This could lead to reduced complexity for the temporal delay algorithm if paired with content-addressable memory. This reduced complexity would lead to a reduction in the transmission rates of valid frames, lowering the bus utilization during the analysis period.

The most significant advantage with CAN FD for the timing analysis is the reworking of bit stuffing. Bit stuffing in CAN FD is still not able to be predetermined regarding the data segment. However, the total number of bit stuffs in a data segment is gray coded and given as a counter value before the CRC sequence. This rework goes along with the removal of nondeterministic bit stuffing in the CRC. In the CRC, every four bits are followed by a forced bit stuff. This leads to the CRC segment always being a fixed number of bits long in contrast to standard CAN, which can have its length changed based on the data segment.

That is not to say that a system implementing the procedure for CAN FD would not require a complete redesign. The increase in speeds during the data segment would require more analysis to determine the data transmission rate and would require a system with a faster clock for higher accuracy measurements. Aside from those design considerations, overall stability of the readings and the theoretical attack would increase due to fewer assumptions needing to be made about the network.

This thesis started with the parameterization of the CAN bus. It seemed to be an interesting topic which I believed would lead during its development to a hypothetical attack. I based my entire focus of the thesis for the first few months on the old idiom “knowledge is power.” I have always felt that there is never a thing as too much knowledge. So many CAN attacks are performed without in-depth knowledge of the CAN bus, focusing on information produced by the CAN controller. By diving into the knowledge inherent in the CAN communication I discovered the Broadcast Confusion Attack. A similar attack could be performed on any shared bus multi-master system, but CAN seems the most applicable. In the future, this attack could be improved with greater amounts of analysis on the network to paint a better picture of the network. Gaining as much knowledge of the system can only improve the feasibility and impact of attacks derived on the Broadcast Confusion Attack.

LIST OF REFERENCES

[1] K.-T. Cho and K. G. Shin, "Error Handling of In-vehicle Networks Makes Them Vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016 2016, BUSOFF: ACM, pp. 1044-1055, doi: 10.1145/2976749.2978302. [Online]. Available: https://dx.doi.org/10.1145/2976749.2978302

[2] *Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signaling*, 11898-1, ISO, 2003.

[3] *On-board diagnostics,* EPA 40 CFR 86.005-17 (h)(3), 2010.

[4] J. Ferreira, A. Oliveira, P. Fonseca, and J. A. Fonseca, "An experiment to assess bit error rate in CAN," in *Proceedings of 3rd International Workshop of Real-Time Networks (RTN2004)*, 2004, pp. 15-18.

[5] J. Yee and H. Pezeshki-Esfahani, "Understanding wireless LAN performance trade-offs," *Communication systems design,* vol. 11, pp. 32-35, 2002.

[6] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom, "Survey of Automotive Controller Area Network Intrusion Detection Systems," (in English), *IEEE Design & Test,* Article vol. 36, no. 6, pp. 48-55, Dec 2019, doi: 10.1109/mdat.2019.2899062.

[7] CiA, "CAN remote frames – Avoiding of usage," in "CANopen," Application Note 802, 2005.

[8] M. D. Natale, H. Zeng, P. Giusto, and A. Ghosal, "Worst-Case Time Analysis of CAN Messages," in *Understanding and Using the Controller Area Network Communication Protocol*: Springer New York, 2012, ch. Chapter 3, pp. 43-65.

[9] *Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit*, 11898-2, ISO, 2003.

[10] "Maxim MAX4618 Datasheet," [ONLINE].

[11] "TI SNx5HVD251 Datasheet," [ONLINE].

[12] T. Ziermann, S. Wildermann, and J. Teich, "CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16× higher data rates," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009: IEEE, pp. 1088-1093, doi: 10.1109/DATE.2009.5090826.

[13] J. Novák, "New measurement method of sample point position in controller area network nodes," (in English), *Measurement,* vol. 41, no. 3, pp. 300-306, Apr 2008, doi: 10.1016/j.measurement.2006.11.004.

[14] "SocketCAN - Controller Area Network," in *Linux Documentation* vol. Networking, L. Torvalds, Ed., ed. Kernel.org: Linux Foundation, 2020.

[15] O. Pfeiffer, A. Ayre, and C. Keydel, *Embedded networking with CAN and CANopen*, First ed. Copperhill Media, 2008. 2003.

[16] "TI AM336x Sitara Processors," [ONLINE].

[17] A. Seitz, A. Satar, B. Burke, and Z. Estrada, "CAERUS: Chronoscopic assessment engine for recovering undocumented specifications," 2018.

[18] S. S. Math and V. Math, "Design and Analysis of Xilinx Verified AMBA Bridge for SoC Systems," 2013.
[19] ICS Alert (ICS-ALERT-17-209-01) [Online] Available: us-cert.cisa.gov
[20] S. Kulandaivel, T. Goyal, A. K. Agrawal, and V. Sekar, "CANvas: fast and inexpensive automotive network mapping," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 389-405.

APPENDIX

All code for this thesis is available at: github.com/caerus-timing/CANTimingAnalysis