Intrusion Detection using Bit Timing Characteristics for CAN Bus

Chitvan K. Patel

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

June 17, 2019
Blacksburg, Virginia

Keywords: Intrusion Detection, CAN Bus, TDC, Machine Learning, KNN, Neural Networks

Intrusion Detection using Bit Timing Characteristics for CAN Bus

Chitvan K. Patel

Academic Abstract

In today's world, most automobiles use Controller Area Network (CAN) bus for communication between various Electronic Control Units (ECUs), also called nodes on the CAN bus. Each ECU on the CAN bus is a microcontroller that sends a unique identifier used for node identification. It is possible to spoof node A by sending the same identifier through node B and thereby control node A. Thus, a hacker can control the steering using the car's internal lights and render it ineffective or misuse them. In order to combat this, we try to fingerprint each node by identifying its identifier's unique bit timing characteristics. To that extent, bit timing characteristics used are the Time of Flight (TOF) intervals between successive rising edges of identifier bits, for an ECU. Similarly, other characteristics such as TOF between successive falling edges of the CAN bus node identifier can also be used for node classification.

In order to measure these TOFs, we use a device called Time-to-Digital Convertor, which essentially triggers a ring oscillator to measure time values between rising/falling edges of a signal, to the order of picosecond accuracy. These timing values are used as features into the K-nearest neighbors (KNN) classifier algorithm. Once the classifier is trained, it can be used to predict a new timing value into a particular node category, which if different from the expected category is a sign of compromise or intrusion. It is seen that we achieve 95% accuracy of correctly predicting the compromised node under simulation tests. Thereafter, the thesis deals with experimentally predicting an intrusion in the CAN bus system utilizing EPOS Studio CAN bus position controller for Maxon motors. The clock timings being extremely accurate leads to the conclusion that employment of better statistical techniques for node characterization is needed for intrusion detection, which is outside the scope of this work.

Intrusion Detection using Bit Timing Characteristics for CAN Bus

Chitvan K. Patel

General Audience Abstract

In today’s world, most automobiles use Controller Area Network (CAN) bus for communication between various Electronic Control Units (ECUs), also called nodes on the CAN bus. These nodes can range from car headlights, radio, doors, internal lights to brakes, steering, throttle and much more. Each node on the CAN bus is a microcontroller which controls its proper operation. This also means that if a node is compromised using external hardware or a piece of software, it could be quite risky. Thus, a hacker can control the steering using the car’s internal lights and render it ineffective or misuse them. In order to combat this, we try to fingerprint each node by identifying its unique time domain characteristics. These characteristics can be the Time of Flight (TOF) measurement values between successive rising or falling edges of a node’s unique identifier, using an instrument called a Time-to-Digital convertor. Furthermore, these TOF values are used as features for the K-nearest neighbor (KNN) classifier machine learning algorithm, which uniquely identifies signals coming from any of the fingerprinted nodes, thereby raising a flag if a message comes from an unidentified node. In addition, experimental data is obtained for node identifiers on the CAN bus, in digital form, and passed into a neural network (NN) for training the classifier. We achieve an 95% and 70% prediction accuracy for the KNN and NN classifiers respectively.

Acknowledgments

Transforming from a core mechanical engineer to an engineer i.e. problem solver, has been a rough ride, only aided by some supportive people and a great institute. To this point I would like to thank all the factors who have contributed in transforming my personality to only become better and more responsible each day at VT. I would like to thank Dr. Wicks for all his guidance and patience, bearing with me throughout the 2 years that I have been associated with the Mechatronics Lab. He has always challenged me to take up tasks outside my comfort zone and dig deeper to strengthen my knowledge in new areas of engineering. He has been a major factor in making me an engineer and channeling my thought process towards problem solving. His efforts towards discussing various topics even outside engineering and continuous motivation has boosted my intellect. Without his continuous support my time at Virginia Tech would not have been so great and eye-opening.

I am very thankful to my committee members Dr. Southward, who has been very helpful with his conceptual suggestions for TDC and review of the research proposal, along with Dr. Asbeck, who supported the inception of this thesis as part of the mechatronics project. A special shout-out to all my lab members namely Matt Spicer, Joshua Moser, Clinton Burns, Bhavi Bharat, Phil Repisky, Adam Lowery for their constant support, motivation, guidance while bearing with all my stupid questions be it technical or non-technical, and especially Tim Pierce, without whose electrical knowledge it wouldn't have been possible to debug the experimental apparatus. In addition, I would like to thank my parents, who gave me the confidence that life is all about facing the challenges and surviving through thick and thin. Last but not the least, I am very grateful to God because of whom I made it so far in life and hopefully continue to progress and prosper ahead.

Contents

Chapter 1 ... 1
1.1 Introduction ... 1
1.2 Motivation ... 1
1.3 Hacking a CAN bus ... 2
1.4 Types of Attacks ... 5
1.5 Existing Intrusion Detection Systems ... 5
1.6 Approach ... 10
1.7 Thesis layout ... 11
Chapter 2 ... 12
... 12
2.1 Theory
2.2 CAN Bus Theory ... 12
2.2.1 Introduction ... 12
2.2.2 Applications ... 13
2.2.3 Advantages and Bus Design ... 14
2.2.4 Example CAN messages with PCAN Viewer: ... 21
2.3 Time to Digital Convertors ... 21
2.3.1 Introduction ... 21
2.3.2 Application ... 22
2.3.3 TDC architecture ... 23
2.3.4 Errors in TDC ... 27
2.3.5 Survey of available TDCs ... 27
2.3.6 Choice of TDC ... 28
2.3.7 TDC 7201 ... 29
2.3.8 TDC7201-ZAX-EVM ... 39
2.4 Machine Learning Techniques ... 42
2.4.1 K-Nearest Neighbor Algorithm ... 42
2.4.2 Neural Networks ... 46
Chapter 3 ... 50
3.1 Process ... 50
3.2 What is meant by intrusion in CAN bus? (revisited) ... 50
3.3 Equipment ... 53

3.4 Wiring Diagram and Data Acquisition 55
3.5 Thesis goal revisited: (Process description) 62
Chapter 4 64
4.1 Experimental Design 64
4.2 Functionality of TDC to be verified 64
4.3 Simulating CAN identifier signal using microcontroller and checking for differences in relative timing values for fingerprinting nodes 68
4.4 Collecting actual CAN bus identifier bit timing data with the TDC EVM 72
4.5 Neural Network trained on the Identifier data from Picoscope 78
4.6 Modifications to the TDC EVM 80
Chapter 5 84
5.1 Results and Discussions 84
5.2 Verification of TDC functionality 84
5.3 CAN identifier signal simulated on MSP432 using TimerA module 85
5.4 KNN applied to CAN bus identifier bit timing data 88
5.5 Neural Network Results for model trained on raw identifier data 90
5.6 Actual CAN bus identifier bit timings 96
Chapter 6 97
6.1 Conclusions and Future Work 97
6.2 Intrusion detection based on timing measurement 97
6.3 Fingerprinting applied to CAN bus setup 98
6.4 Future Work 99
6.5 Final Remarks 101
References 103
Appendix A 112

List of Figures

Figure 1-1: Hacking attacks possible in CAN bus namely Fabrication attack, Suspension attack and Masquerade attack 5
Figure 2-1: DB9 CAN Connector 16
Figure 2-2: DB9 Connector Pin Out Diagram 16
Figure 2-3: CAN node consisting of CAN controller and CAN transceiver on the CAN bus with terminating resistance at the ends 17
Figure 2-4: Signal from CAN controller (TTL) converted by the CAN transceiver to match CAN bus protocol 17
Figure 2-5: CAN frame transmitted by any node on the CAN bus 18
Figure 2-6: Bit stuffing in CAN bus frame for bit rate synchronization of CAN transceivers 19
Figure 2-7: Bit arbitration phase showing node 3 to be transmitting its identifier, gaining priority over nodes 1 and 2 based on their CAN IDs 20
Figure 2-8: PCAN view showing CAN bus nodes' identifiers and corresponding messages 22
Figure 2-9: Time to digital convertor realization with counter mechanism 23
Figure 2-10: Tapped Delay line having propagation delay of τ every flip-flop to measure time interval between start and stop pulse 24
Figure 2-11: Delayed start signal from tapped delay line mechanism 25
Figure 2-12: TDC using a Vernier oscillator mechanism 26
Figure 2-13: TDC 7201 Block Diagram showing internal circuit elements 29
Figure 2-14:TDC 7201 Pin Layout 30
Figure 2-15: Standard Deviation in time measurements vs reference clock frequency 31
Figure 2-16: Measurement mode 1 working to calculate TOFs 31
Figure 2-17: Measurement mode 2 working to calculate TOFs 32
Figure 2-18: Multi-cycle Averaging for measuring time values from 250ps under very low power consumption mode and noise reduction 34
Figure 2-19: Result of multi-cycle averaging for time of flight values 34
Figure 2-20: Example of clock jitter accumulating with increase in number of clock cycles passed 37

Figure 2-21: Minimum signal parameters for correct working of TDC7201 38
Figure 2-22: TDC evaluation module 39
Figure 2-23: TDC EVM GUI configuration window 40
Figure 2-24: TDC EVM one shot measurement values for time of flight recording.................. 40
Figure 2-25: TDC EVM graph window displaying the Start-Stop1 timing values captured........ 41
Figure 2-26: TDC EVM schematic for board 41
Figure 2-27: Classification example for data point(green) using 3 neighbors and 5 neighbors ... 43
Figure 2-28: Training error for KNN algorithm vs ‘k’ value 44
Figure 2-29: Validation error for KNN algorithm vs ‘k’ value 45
Figure 2-30: Multi-layer perceptron network with an input layer, 2 hidden layers and an output layer... 47
Figure 2-31: Simplest example of neural network with 3 input neurons and 1 output neuron whose value is between 0 and 1 47
Figure 2-32: Single class neural network with 1 hidden layer.. 48
Figure 3-1: Node identifiers before intrusion 50
Figure 3-2: Node identifiers after intrusion 51
Figure 3-3: Expected histogram for bit timing values from 2 nodes A and B sending CAN messages with same identifier... 52
Figure 3-4: Actual test setup 53
Figure 3-5: CAN bus setup wiring diagram showing 3 nodes and TDC connected to acquire the differential data from the system for each node active at a given time 54
Figure 3-6: Molex 4-pin CAN connector for tapping into the bus 55
Figure 3-7: Wiring diagram for full setup.. 56
Figure 3-8: Initial setup of TDC using GUI.. 57
Figure 3-9: Configurations for edge polarity and number of stops for TDC 58
Figure 3-10: TDC measuring 19873 ns with a resolution of picoseconds............................ 59
Figure 3-11: Example CAN signal from EPOS controller using Picoscope confirming to the CANopen protocol 61
Figure 3-12: Edge polarity depiction from CAN signal for TDC start and stop pulses and expected bit timing.. 63
Figure 4-1: Physical connections to the TDC for start, stop and trigger pulses 65

Figure 4-2: Tektronix AFG3102 function generator connections to/from the TDC 65
Figure 4-3: PWM signals for start and stop pulse for TDC from signal generator captured using an oscilloscope, having a delay of 19us.. 65
Figure 4-4: TDC GUI with setup configuration 66
Figure 4-5: GUI graph for time of flight values from function generator 67
Figure 4-6: Histogram obtained for time interval between 2 PWM signals from signal generator using TDC, delay between 2 pulses is 19us.. 68
Figure 4-7: CAN bus identifier signal split into start and stop pulses 70
Figure 4-8: PWM signals from TimerA of MSP432 captured using an oscilloscope 70
Figure 4-9: TDC GUI showing time of flight values for PWMs generated 71
Figure 4-10: Histogram showing time intervals for PWMs observed from 6 different MSP432s using the TDC 72
Figure 4-11: DIP switch configuration to program the identifier with ID 0d73........................... 73
Figure 4-12: CAN bus signal from Node 1 with ID 0d73 (0x49) captured in Picoscope............. 73
Figure 4-13: CAN differential signal between CAN H and CAN L measured using Picoscope without the TDC connected to the CAN bus setup for acquiring timing data 74
Figure 4-14: CAN differential signal with the TDC connected to the bus captured using Picoscope 75
Figure 4-15: Zoomed in view of the identifier signal with bits laid out and expected start and stop pulses by the TDC.. 76
Figure 4-16: Time interval between start-stop1 plotted as histogram (for training dataset) 77
Figure 4-17: Time interval between start-stop1 plotted as histogram (for test dataset) 77
Figure 4-18: CAN signal for identifier ID73 using Picoscope showing different Picoscope parameters for data acquisition 78
Figure 4-19: Identifier data extracted from the CAN signal using MATLAB showing 32 datasets plotted over each other.. 79
Figure 4-20: TDC EVM schematic showing 50-ohm resistor between the TDC input pin and ground 81
Figure 4-21: CAN bus signal with TDC connected (after removing 50-ohm resistor) as measured from Picoscope.. 81

Figure 4-22:Identifier signal with bits laid out and expected start and stop pulses by the TDC with timing values for multiple start-stops 82
Figure 4-23: Histogram for time interval between start-stop1... 83
Figure 4-24: Histogram for time interval between start-stop2... 83
Figure 5-1: Histogram fitted with normal distribution for TDC functionality verification experimental data obtained in Section 4.2 85
Figure 5-2: Test data for label 1834 overlaid with training data histogram 87
Figure 5-3: Test data for label 2580 overlaid with training data histogram 87
Figure 5-4: Test data for label 6526 overlaid with training data histogram 88
Figure 5-5: Combined histogram depicting training and test data for actual CAN bus setup 89
Figure 5-6: Sigmoid function for neuron activation in neural network 94
Figure 5-7: Rectified Linear Unit (ReLu) activation function for neurons 94
Figure 5-8: Performance of different optimization functions in presence of saddle points.......... 95
Figure 6-1: Conditioned CAN H signal to be used with TDC... 100
Figure 6-2: Conditioned CAN H and CAN L signals to be used as the start and stop pulse for the TDC.. 100
Figure A.1: Histogram for Time of Flight values from simulation experiment 112
Figure A.2: Test case 1834 histogram for time of flight values overlaid on simulation histograms 113
Figure A.3: Test case 6526 histogram for time of flight values overlaid on simulation histograms 115
Figure A.4: Test case 2580 histogram for time of flight values overlaid on simulation histograms 116

List of Tables

Table 2-1: Survey of available TDCs 28
Table 2-2: Important parameters of TDC7201 38
Table 3-1: EPOS position controller configuration parameters... 60
Table 4-1: Important TDC configuration parameters 66
Table 4-2: Important signal characteristics to meet the requirements of TDC functioning 69
Table 5-1: table showing mean, standard deviation and skewness of the data for TDC functionality verification experiment in Section 4.2 84
Table 5-2: loss results for trained KNN model on data from Section 4.3 86
Table 5-3: Test case and corresponding accuracy of KNN trained on data from Section 4.3...... 86
Table 5-4: Loss and Accuracy of KNN classifier for actual CAN bus identifier data 89
Table 5-5: Neural Network performance statistics 92
Table A.1: Mean and standard deviation values for histograms from simulation experiment... 112
Table A.2: Mean and standard deviation for test case 1834 data 114
Table A.3: H values for test case 1834 under hypothesis test 114
Table A.4: p-value for test case 1834 under hypothesis test.. 114
Table A.5: Confidence Interval for test case 1834 under hypothesis test.................................. 114
Table A.6: Mean and standard deviation for test case 6526 data 115
Table A.7: H values, P-values and Confidence Interval for test case 6526 under hypothesis test 115
Table A.8: Mean and standard deviation for test case 2580 data 116
Table A.9: H values, P-values and Confidence Interval for test case 2580 under hypothesis test 116

Chapter 1

1.1 Introduction

In this chapter we will briefly go through the motivation for this thesis, challenges proposed by intrusion in automotive CAN bus, hacking mechanisms in a CAN bus, existing solutions to determine hacks, our approach to detect intrusions and the thesis layout. The goal of this chapter is to give an overview of the different topics related to CAN bus intrusions and what existing steps have been taken to either detect or prevent the hacks.

1.2 Motivation

There was a growing interest to develop automobiles with embedded electronic systems and replace the mechanical parts since the 70s. Today's vehicles have a high number of electronic modules operating through high speed digital processors and communicating over network services. An increasing use of Wi-Fi and Bluetooth services in vehicle has been seen for infotainment and telematics control units. These electronic modules communicate over the Controller Area Network (CAN) protocol. CAN is a multi-master bus protocol which allows every module to broadcast its ID and data in the form of a CAN packet. Modern cars are getting equipped with a higher number of electronic control units (ECUs) communicating over CAN and with the advent of autonomous vehicle technology we also see V2X communication meaning vehicle-to-vehicle and vehicle-to-infrastructure communication. This opens up a series of potential threats that can make way into the vehicle's security system and perform malicious activities.

Safety of both personnel and vehicle is an increasing concern. As the vehicles start getting more and more complex with increasing functionality to be achieved and different modules being incorporated, ensuring safe operation of each module is a challenging task. There have been an increasing number of attempts to hack a module on the CAN bus, several of which have succeeded. The pioneers at demonstrating the gravity of an automotive intrusion were Miller and Valasek, who not only demonstrated how to reverse engineer a car's CAN bus system but also gained control of the Park Assist Module of the 2014 Jeep Cherokee. They also pointed out several remote attack points such as Wi-Fi, USB and Bluetooth network entry points, to which a potential hacker could gain access and thereby render some modules on the bus ineffective, thus initiating a form of attack. Say if the intruder gains access to the CAN controller module on the bus and renders the brakes ineffective through software, it could be a serious hazard for both passengers as well as surrounding environment. It is also possible for the hacker to go one step further, by not only rendering a module ineffective, but to gain control of another module such as the steering wheel, override human inputs and control the vehicle heading using software.

This is a matter of great concern from the cyber-security viewpoint as we move towards an era of electric and autonomous vehicles which will have an increasing number of software communication between different modules and hence an increased number of entry points into the vehicle making it easier for a hacker to remotely control vehicle modules. We aim to propose a method for detecting such hacks/intrusion by way of fingerprinting each module on the bus.

1.3 Hacking a CAN bus

CAN bus hacking has become very popular recently as more and more automobile manufacturers have equipped the vehicle to communicate inter-module using the CAN protocol along with the advent of autonomous vehicles which rely on the drive-by-wire framework. One of the first results on hacking a CAN bus that I came across on the internet was on hacking a Ford Fusion for programmable control of the A/C temperature [1]. The blog clearly explained how to reverse engineer the car's CAN bus and use SocketCAN framework integrated with ROS to identify the correct CAN bus frame, and data to be sent for temperature control using Kvaser hardware. There is an entire handbook available [2] online for free, which demonstrates the breadth of reverse engineering a CAN bus using diagnostic and data logging tools, and thereafter hack an ECU.

There are numerous references online which demonstrate the ease of spoofing a CAN bus node using simple off-the-shelf hardware products. Eric Evenchick developed a $60 piece of embedded hardware which connects to the OBD-II port and computer to gain access to the bus, eliminating the need for sophisticated and unaffordable hardware for hobbyist hackers [3]. Kristoffer Smith on the other hand demonstrated the use of a standard ELM327-based OBD-II scan tool (off-the-shelf product worth $25) to interface with the car's bus, conduct a series of step-by-step tests to validate the interface and then control the car's steering wheel using simple look-up tables created by the hacker himself [4]. Even Arduino users can now carry out hacks using a CAN shield equipped to Arduino Uno with integrated example code in the Arduino IDE, and a step-by-step Instructables guide accompanied by visual descriptions of data logging, analyzing and resending the data [5].

A point to ponder about is that if we have so many open source references for hacking a CAN bus and controlling the car's essential modules like steering, brake, throttle, etc. at the level of a hobbyist, imagine what an experienced hacker could do at this point! To demonstrate the vulnerability to such hackers, who could initiate a software attack instead of a physically connected hardware attack, the pioneer research into car hacking began at the University of Washington and University of California San Diego in 2010 [6]. Researchers at these universities showed that they could remotely control vehicle's (2009 Chevy Malibu) locks and brakes using Bluetooth and cellular connections to the car [7]. They highlighted that modern vehicles provided call services to the driver using cellular connection to the car which made it easier for hackers to track the car's location and unlock doors, remotely. They also pointed out attacks through network ports for car maintenance and internal CD players.

In 2012, DARPA funded the researchers Valasek and Miller to conduct research in this domain of intruding a CAN bus and their work showed that it was possible to steer and brake using a laptop connected to the vehicle [3]. They released several documents of their work on exploring the different remote attack points in a vehicle and proposed some mitigation strategies (later discussed in this thesis) [8] and [9]. They have gone to show the different remote endpoints available to hackers for attempting an intrusions and their attack range such as passive anti-theft system, tire pressure monitoring system, remote keyless entry which were all short range attacks ranging from 10 cm to 200 cm, in contrast to Bluetooth, telematics, cellular Wi-Fi, internet apps, etc. which have a high attack range up to few miles and are very high risk access points providing hackers a great advantage for remote hacks. They exploited these attack points for different automobiles like the 2010 Ford Escape, 2010 Toyota Prius, 2014 Jeep Cherokee and several other vehicles.

Miller and Valasek went on to describe the attack mechanism steps needed. Firstly, an attacker would have to access the communication network either remotely or physically. Secondly, after gaining the access to CAN bus, desired messages would have to be created and injected on the bus to control steering, brakes, etc. Lastly, in order to create the appropriate messages, a hacker would have to reverse engineer the CAN messages [8]. They also demonstrated how to reverse engineer the CAN messages and inject them appropriately into the bus to control throttle, engine, etc. [9]. They gained access to the OBD-II port on the vehicle, monitored the different ECU messages, decoded those messages, calculated the correct checksum for the messages and then began to inject their own messages. The diagnostic session messages were also very critical to help them debug the CAN messages and understand the checksum calculation algorithms. All these have inspired various companies and government to also conduct research into vehicle intrusion and inspired hackers to contribute in this area [10] and [3].

1.4 Types of Attacks

Three types of attack can be found in literature [66], namely Fabrication attack, Suspension attack and Masquerade attack as shown in Figure 1.1.

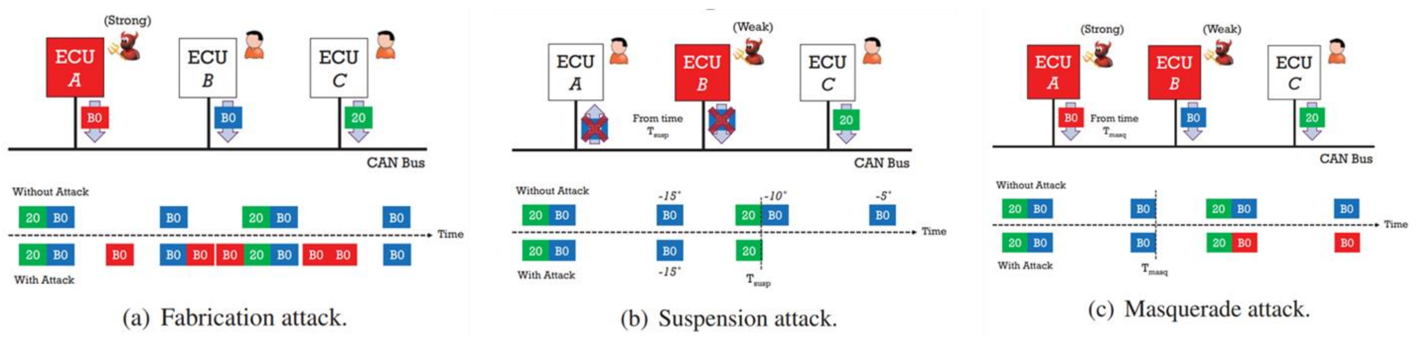


Figure 1.1: Hacking attacks possible in CAN bus namely Fabrication attack, Suspension attack and Masquerade attack

In the fabrication attack, ECU B sending a CAN message with identifier B0 is hacked by ECU A who also sends a CAN message with identifier B0. Since both the nodes are active on the bus and the ECU A sends messages a-periodically, it can be seen that after the attack, the message frequency of the identifier B0 has increased. During the suspension attack, it can be seen that ECU A suspends the ECU B from sending messages over the bus and the message frequency of the identifier B0 recues to 0 post attack. Lastly, in the masquerade attack, ECU A attacks ECU B while sending messages with identifier B0 at the same frequency as ECU B. This denotes that ECU B remains in suspension and data sent by ECU B is never transmitted over the bus.

1.5 Existing Intrusion Detection Systems

An intrusion to a computer system means safe working of the system being compromised. To guard against this an Intrusion Detection System (IDS) can be made which is essentially a device or piece of software to monitor the system from unintended activity or protocol violations [11]. There are 3 main approaches to classify existing IDS into signature-based detection, anomaly-based detection and reputation-based detection [11]. Each of these will be described in some details. The reader is however encouraged to read more about these if it sparks their curiosity from the reference papers mentioned.

Signature based detection systems analyze the network frame packets and perform a check against existing signature attacks. This however limits their ability to work only for pre-determined or pre-configured attack patterns and it is almost impossible to detect new anomalies, where no pattern is available [12]. To overcome the above drawback, anomaly-based IDS rely on machine learning approach to detect unknown attacks by creating a model and training it for desired behaviors. Any deviation from expected behavior will signal an anomaly though there is a possibility of classifying normal behavior as an anomaly commonly termed as false positives. Hyper parameter tuning for such models can be performed for achieving better accuracies and reducing false positive occurrences such as better feature selection, regularizing your data, etc. [13]. Reputation based detection goes a step further than the other methods by directly assigning a reputation i.e. good or bad or none to a node in communication based on particular characteristic exhibited by the node. So for example if a node is within your local network it will be given a positive reputation contrary to a node which is trying to communicate remotely to your system [14]. This forces an intruder to gain access to your communication protocol, identify the unique characteristic of the node to be hacked which is the identifier value in case of the CAN bus packet, duplicate that identifier from the node a hacker has accessed remotely/physically and then send the desired data to control a particular trait of the vehicle once he reverse engineers the bus message frame. Now our thesis aims to focus on how to distinguish nodes on a CAN bus based on identifier characteristics such as identifier bit timings.

Miller and Valasek in their paper [8] gave some defense mechanisms towards intrusion attacks for the CAN bus. They suggested to have a layered defense mechanism, which would end up making each stage of the attack more difficult to be performed. Firstly, they suggested to secure the remote attack surfaces such as Bluetooth and cellular modems. However, they also pointed out that complete lock down of such services is not possible in today's world rather just improving security by introducing security patches every now and then. Secondly, they pointed out that it is possible to make it harder for a hacker to inject messages on the CAN bus once they gain access to an ECU with various examples. They said that a Bluetooth stack may never be required to send CAN messages since Bluetooth communication is not over CAN bus and similarly telematics and other units running on Android and Linux Kernel have capability to avoid injecting CAN bus messages even if compromised, thus adding an additional layer of security. Lastly, they suggested to verify CAN messages using unique encryption keys, making it even more troublesome for the intruded ECU to send messages on the bus. They further pointed out in their paper [9] to monitor the rate of messages on the CAN bus. Since a compromised ECU must send messages at a higher rate than normal, the total number of messages per unit time will increase in case of an attack in order for the vehicle to take note of them.

Hoppe and Kiltz also discussed about attack scenarios with increased message frequency and low level CAN communication structure for identifying intrusions [15]. They pointed out that each CAN message is an electrical signal generated by a CAN controller on the bus, and that each CAN controller being from a different manufacturer would have a unique chip characteristic which could be identified by observing the electrical signals more closely. Some of the suggested features of these chips could be stability of voltage amplitude, shape of the clock edges, propagation delays and signal attenuation due to varying wire lengths. These features could serve as signatures for each ECU on the bus and monitored to detect anomaly if a node is in fact compromised. If the signal properties of a node would change, it would be certainly under attack. Although this was not the main focus of their paper, it was a worthwhile suggestion.

Taylor et al [16] developed an algorithm to predict intrusion based on increased message frequency from a compromised ECU. They measure the time intervals between successive message packets, using a sliding window (Hamming), and compare this time to a historical packet timing, obtained heuristically. This information is later used in a single class support vector machine to classify anomalies with a high accuracy rate. The features that they use for the Support Vector Machine (SVM) are number of packets in a window, mean and variance of packet time differences and mean and variance of Hamming differences for every node ID on the bus under consideration. They use windows of varying time intervals from 50 milliseconds to 1 second, which is in general the frequency of messages for most nodes on the bus. Additionally, they show results with different message injection rates varying from 0.2 seconds to 0.5 seconds exploring the limitations of their algorithm. Their results showed that the false positive rates were high if they used a simple t-test statistic, but decreased as they used the SVM because of additional features. They also ignored the Node ID on the bus and showed that the Hamming distance of data packets was not reliable for classification.

Song et al exploited the same aspect of increased message frequency of the compromised node and proposed an algorithm for intrusion detection robust for 3 kinds of injection attacks and 0 false positives [17]. Their algorithm detects the message from an ID and checks the last time of message arrival. If that time does not match the regular message frequency, then they increase the score of their measured metric by 1. Once this metric crosses a known threshold they term the node to be under attack. The limitation to this algorithm is that it is extremely simple and their assumptions about threshold may vary from vehicle to vehicle.

Matt Spicer from VT [18] on the other hand proposed an algorithm to identify each node on the bus using its unique noise properties. He performed statistical noise modelling and extracted useful features of the noise for each ECU, passing those in a neural network to fingerprint them and detect anomalies, with prediction accuracies up to 99.9%. He also compared performances of neural networks to support vector machines with the same feature set and showed them to be almost the same. This approach was more frequency domain based for signals than time domain and computationally extremely expensive on account of the hyperparameter tuning that was performed.

Vasistha, a graduate student from TAMU, [19] proposed a different type of layered approach to detect anomalies. Firstly, he proposed sensor fusion from multiple data sources to improve the integrity of your data. Secondly, he sought to verify if the nodes were indeed transmitting as per their arbitration scheme i.e. priority of identifiers instead of being spammed arbitrarily. Lastly, he proposed an algorithm to calculate bus utilization based on statistical techniques to identify timing behavior anomalies. Though his approach was different and elegant it led to increased latencies because of applying statistical techniques for verification with not a large amount of decrease in the false positive alarms.

Lokman et al [20] constructed a baseline classifier to predict CAN data based on their CAN ID using Convolutional Neural Networks (CNN). In effect they check the CAN ID based on the CAN data and see if the actual ID matches the predicted ID to detect an intrusion. This is because during an intrusion, although the CAN ID would be same but the hacker would send different messages to achieve desired output from the node. These messages, if did not correspond to be deemed as normal for a particular ID, will be termed as an anomaly and hence a malicious attack. In order to use the CAN data as inputs to the CNN, the researchers use a technique called word embedding. Hyper-parameter tuning for the CNN model is performed using the Taguchi method. The open source CAN dataset from KIA SOUL is utilized to train the CNN and detect intrusion. This work is interesting as it makes use of the complete packet information in contrast to our method where we propose to just utilize the ID information. Computationally their method is more expensive and needs much more tuning but achieves a good accuracy of 96%. The reader is encouraged to look into the reference for further details into CNN, word embedding and sequence classification.

1.6 Approach

Matt Spicer from VT proposed a solution to intrusion detection using frequency domain techniques by analyzing noise properties of the signals on the CAN bus and fingerprinting each module on the CAN bus based on its unique signal noise characteristics. We try to achieve a similar process for intrusion detection but using bit timing characteristics of the pulses on a CAN bus and thereby fingerprint each node.

Essentially, we measure the time interval between the identifier bits of a node on the CAN bus. These time of flight measurements are used to characterize each node. Timing values are a direct representation of the node's clock. Although each node sends a message that has a unique identifier, during hacking, one would have to send the same identifier using a different node. This means that a different clock would be driving the bus operation, although the baud rate would be same. Given that each clock would have some manufacturing inconsistencies and tolerance levels, these can be observed in the time of flight values between identifier bits. Thus, we can characterize each node using timing values and then apply a classification technique for identifying any node on the CAN bus. If a node, such as infotainment unit, sends data to control throttle we can detect that it’s a different node trying to control the bus and deem it a case of intrusion.

To this extent we use a measurement instrument called the Time-to-Digital Convertor which gives us time of flight values in order of picoseconds, between successive pulses essentially similar to the identifier of a node on the bus which is just a bunch of logical binary values. The classifying approach selected was the K-Nearest Neighbor (KNN) classifier to characterize the node and then identify intrusion.

1.7 Thesis layout

The thesis chapters are outlined as follows. In chapter 2 we discuss the theory of CAN bus, time-to-digital convertors and machine learning techniques of relevance to this work. Chapter 3 discusses the actual CAN bus hardware on which the testing is performed for intrusion detection, wiring diagram and testing process describing our approach. Chapter 4 deals with the design of experiments conducted to exploit the validity of our approach. It takes the reader through a series of 5 experiments to understand functionality of our time-to-digital convertor, simulation experiment on microcontrollers, actual application to a CAN bus setup, implementation of machine learning techniques like k-nearest neighbor and neural networks for classification and intrusion detection, and lastly some modifications to the time-to-digital convertor for further examination. Chapter 5 discusses the results from the experiments performed in the previous chapter and Chapter 6 lays out the conclusion from our results and limitations of our approach along with some future work experiments to be conducted for completeness to our approach.

Chapter 2

2.1 Theory

In this chapter we will go over the necessary background to get a better understanding of the topics that this thesis deals with. First, we start with CAN bus theory, next we describe time to digital convertors and lastly, we go over machine learning approaches used in this work like the k-nearest neighbor algorithm and neural networks.

2.2 CAN Bus Theory

2.2.1 Introduction

CAN bus or Controller Area Network bus is a communication system that allows multiple microcontrollers to communicate with one another in the absence of a host computer or high level operating system [21]. It was originally developed for automotive applications by Robert Bosch GmbH in 1983 [22], eliminating the need for every module in a vehicle to be attached to every other module, for which it would be in communication with. This reduced the wiring needs to a large extent and also the complexity of communication software. The protocol became official at SAE in 1986 and Intel developed the 1st CAN communication chip in 1987, while Mercedes-Benz W140 was the pioneer vehicle to have implemented the bus protocol in it [23] and [24]. Bosch released the CAN 2.0 protocol in 1991 which had 2 parts to it. The 1st part was for using the bus with a 11-bit node identifier specification allowing up to 2^11 nodes to be used on the bus at a time, while 2nd part of the protocol could be used with a 29-bit node identifier allowing up to 2^29 nodes to be used on the bus [25]. International Organization for Standardization (ISO) released several CAN standards in 1993 describing the data link layer for data transmission standards and physical layer describing the hardware specifications for achieving proper inter-module communication using the bus protocols [26] and [27]. These standards are namely ISO 11898-1, 11898-2 and 11898-3 and any manufacturer may purchase these from ISO to integrate the bus protocol in their vehicle as per their requirements. In addition, these standards also provide specification for using high-speed CAN bus and low-speed/fault tolerant CAN bus [28]. The basic difference between the 2 buses being that the high speed operates at baud rates ranging from 40kbits/s to 1Mbits/s and the low speed bus operates up to 125kbits/s. Furthermore, these buses have different terminating needs i.e. high-speed bus devices have end-to-end termination of 120 ohms, allowing simple cable connection, whereas devices on low speed bus each have their own termination allowing the nodes to operate even after a wiring failure on the bus due to individual terminations. Lastly, in 2012 Bosch further released a new standard CAN FD 1.0 which allowed for flexible data lengths in a CAN frame as well as higher bit rates post node arbitration. This was a direct result of the automotive manufacturers demands for more accurate and real-time data including error messages, ranging from applications for defense, automation, autonomy, underwater and medical devices [29]. As CAN protocol is used on the On-board diagnostics II port on a vehicle for tracking errors and debugging issues, or even reverse engineering the bus has become very popular in today’s world, and its security is especially at risk, for a hacker could gain access to the bus and take control of the engine, brakes, steering, etc. which could be very fatal for humans and surroundings.

2.2.2 Applications

CAN bus protocol has now become a standard for most automotive applications given the advantages it offers in terms of wiring reductions, electrical interference shielding, noise immunity and impedance matching for avoiding signal reflections, high speed data transmission, error checking, diagnostics and multi-master communication without the need of a host computer. The major applications which use CAN bus protocol for communication are:

- Automobiles
- Aerial vehicles and navigation equipment sensors
- Industrial automation
- Medical equipment

In particular, we will look into automobile control units which use the CAN bus for inter-module communication. Some of the important modules are engine, transmission, airbags, anti-lock brakes, cruise control, power steering, windows, doors, audio systems, battery systems for hybrid electric cars, etc. Not only do these modules have to communicate with each other for controlled operation of the vehicle but with autonomous application, they also have to take data from sensors equipped on the car and control relevant actuators which also needs to be communicated. To give a simple example, seat belt sensors communicate the status of the seat belts over CAN bus to disengage the parking brake automatically upon car movement. Other examples include auto shut off of the engine when the engine temperature sensor records a value over some threshold and communicates it over the CAN bus, parking sensors communicate proximity data over CAN bus to Driver Assist Systems (DAS) for detecting lane departure, steer-by-wire and brake-by-wire technologies also take input from different sensors over the CAN bus and make appropriate decisions for collision avoidance, etc. Recently even robotic systems use CAN bus to communicate between different control modules within a robot.

2.2.3 Advantages and Bus Design

CAN bus is a multi-master serial communication bus protocol for inter-module communication between different Electronic Communication Units (ECUs) on a vehicle also called nodes. Each node consists of 3 parts namely, a CPU, a CAN controller and a transceiver. All the nodes are connected to just 2 wires namely the CAN High (CAN H) and CAN Low (CAN L) wires. The wires terminate at either end with 120-ohm resistance. They are in a twisted pair format providing magnetic field shielding effect, since current flows through the wires in the opposite directions cancelling the magnetic field generated, thereby providing good noise immunity. Noise is a direct result of coupling of electric and magnetic fields and thus a balanced line with current flowing in opposite direction and magnetic field cancellation is a huge advantage that the CAN bus offers. The terminating resistance at either end of the bus helps prevent signal reflections and incorrect signal readings due to impedance mismatching. As water always flows from higher altitude to a lower altitude, similarly signals get reflected from higher impedances to lower impedances causing an error in the recorded values.

Furthermore, CAN uses a differential signal because the CAN H and CAN L wires are biased at a 2.5V value and then a logical low is recorded when the CAN H line goes from 2.5V to about 4V and similarly the CAN L line goes from 2.5V to about 1V. A logical high on the contrary is when both the lines remain at 2.5V. This differential signaling leads to an improved noise immunity because of common mode rejection. The noise from external sources gets subtracted in case of differential signals improving the signal stability and increases the reliability of data [30] and [31]. Balanced lines also offer the advantage of longer cable lengths for same signal strength as they reduce the amount of noise per unit distance. These are some of the advantages that CAN bus offers in terms of electronic circuitry and hence has become popular in the automotive market as well as other areas.

Wiring: CAN bus is a 4-wire system with the wires being, CAN V+ for power, GND for ground, CAN+/CAN H for CAN High and CAN –/CAN L for CAN Low. These wires are mechanically integrated into a 9-pin D-sub type male connector often called a DB-9 connector cable for CAN which is connected to every CAN controller node on the bus [25]. The DB-9 connector is shown in Figure 2.1[32] and its pin diagram is shown in Figure 2.2 [33].

Figure 2.1: DB9 CAN Connector

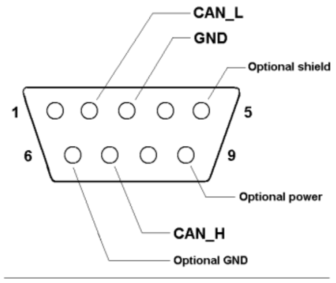


Figure 2.2: DB9 Connector Pin Out Diagram

Node: Every node on the bus consists of 2 parts ([21], [34]):

- **CAN Controller:** consist of a microcontroller which interprets incoming data and decides on the data to be transmitted. The data is in the form of TTL between 0V to 5V.
- **CAN Transceiver:** converts the data from the host microcontroller to CAN logic of differential voltage signal as required by the CAN bus standard.

Figure 2.3 shows a typical CAN controller and the bus in general. Figure 2.4 shows the signal from the CAN controller to the CAN transceiver [34]. As mentioned above, the controller’s TTL signal needs to converted into the CAN differential voltage level, with the dominant bit (i.e. logic level high) being when V_{diff} is 0V and similarly the recessive bit (i.e. logic level low) being when V_{diff}

is about 2V, which can also be seen in the Figure 2.4. Commonly the devices connected to CAN bus can be different sensors, actuators and control devices interfaced through a CAN controller.

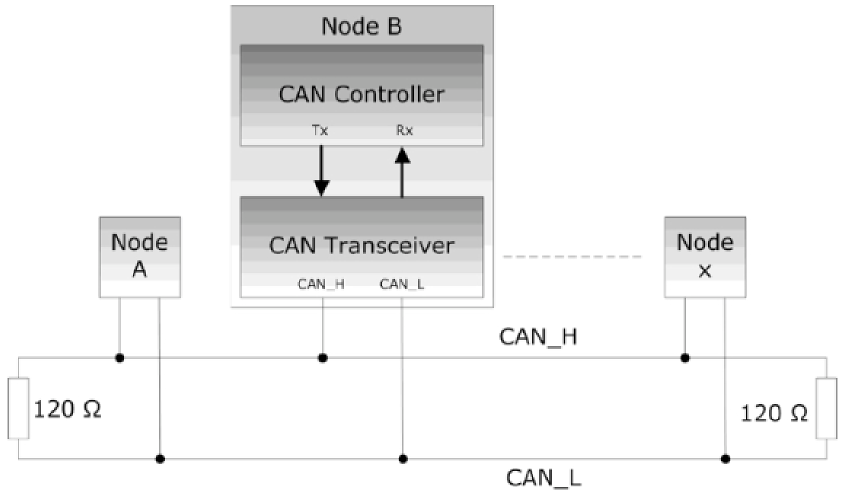


Figure 2.3: CAN node consisting of CAN controller and CAN transceiver on the CAN bus with terminating resistance at the ends

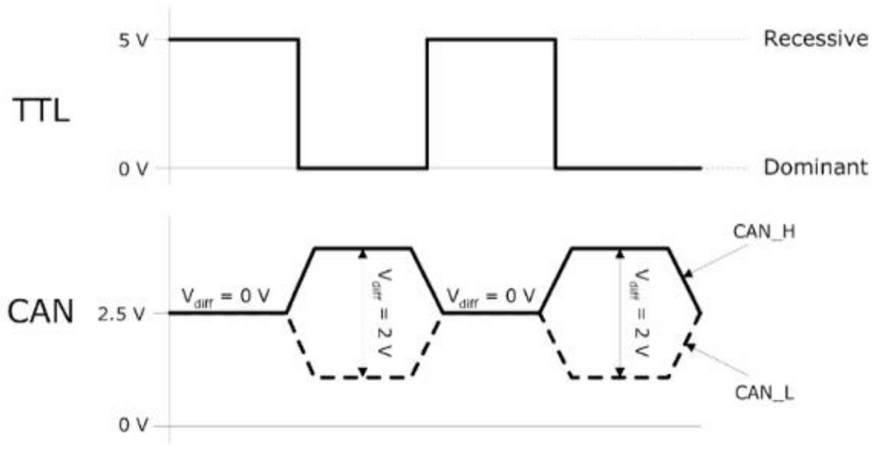


Figure 2.4: Signal from CAN controller (TTL) converted by the CAN transceiver to match CAN bus protocol

Frame: [35] A CAN bus frame consists of the following parts:

- **Start of Frame (SOF)** – 1 bit, logic level high at the start of every frame

- **Node ID** – unique 11-bit or 29-bit value which determines order in which a node can broadcast on the bus
- **Remote Transmission Request (RTR)** – 1 bit, logic level high, used only when data is needed from another node for diagnostic reasons
- **Identifier Extension bit (IDE)** – 1 bit, logic level high
- **Reserved bit (R)** – 1-bit value, logic level high, cannot be accessed
- **Data Length Code (DLC)** – 4-bit value, tells how many data bytes are present in the frame
- **Data** – maximum of 64 bits in chunks of 8 bits/1 byte
- **Cyclic Redundancy Check (CRC)** – contains the checksum for error diagnostic purposes
- **Acknowledgement (ACK)** – 1-bit value, logic high if the data was received successfully by a node else logic low if an error occurred during transmission
- **End of Frame (EOF) -** 7-bit value, all bits are logic low to denote frame end
- **Interframe Space (IFS)** – minimum 7-bit value, all logic low, allows the bus to have some bits between end of one frame and start of a new frame for consistency purposes

Figure 2.5 shows a typical CAN frame

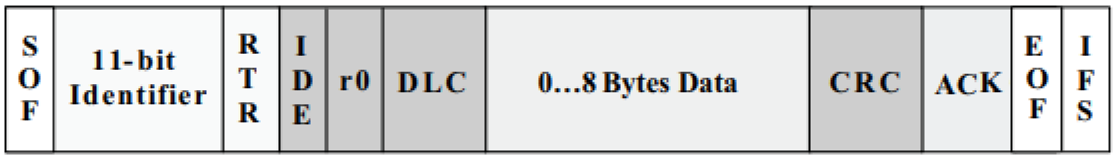


Figure 2.5: CAN frame transmitted by any node on the CAN bus

An important point to note here is that CAN is a message-based protocol, and it is the message frame that carries the identifier with it. Every node transmits messages having unique identifiers, which in turn designates the nodes to be representative of that identifier. In essence we may assume that the node transmitting a certain message type has a unique identifier latched to it. This is an important piece of information when dealing with intrusion in CAN bus.

Bit Stuffing: Since the CAN bus protocol also allows for synchronization of the bit rates for the transmitter and receiver, it implements something called bit stuffing [36]. It means that when we have 5 consecutive bits of same logic, a bit of opposite polarity is added at the end to allow for bit rate synchronization. This added bit although received by the receiving node, is however not parsed during data check, thereby avoiding any erroneous interpretation for the data. Figure 2.6 shows an example bit stuffing in CAN bus [37].

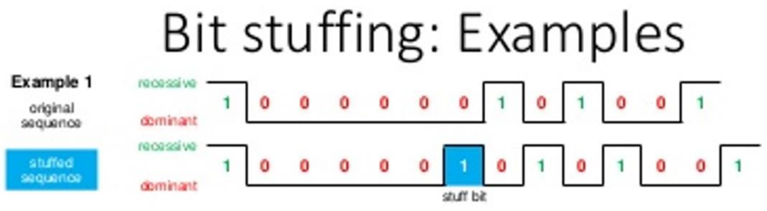


Figure 2.6: Bit stuffing in CAN bus frame for bit rate synchronization of CAN transceivers

Bit Arbitration for data transmission: Every node on the CAN bus is connected by 2 wires and CAN is a multi-master bus with each node trying to broadcast its message over the 2 wires. Now how does the CAN bus know which node should be allowed to transmit data in comparison to another, when both are trying to broadcast their messages at the same time. To overcome this issue, the bus is designed to perform something called a bit arbitration.

Every message broadcasted from a node is allocated an ID called a CAN ID. For example, the steering wheel node will broadcast a message on the bus showing the current steering angle of the wheel. This message will always have the same ID or identifier, which means that every node receiving a message with this identifier knows that the steering wheel is updating its data on the bus. The CAN IDs are assigned as per the priority of the message they carry, so the electronic brake node may have a higher priority over the seatbelt node. We also mentioned before that the identifier can be up to 11-bit or 29-bit depending on the CAN standard followed.

Next when 2 nodes try to broadcast their message at the same time, all the node identifier bits are checked one by one and compared against each other by using a logical AND function. This means that the bus would see a value of 1 if all the bits under check are 1 else the bus would see a 0. This means that if any node outputs 0, the bus will see a 0 implying that a node having a 0 bit in the identifier in comparison to a 1 bit will gain priority over the other node. Thus, once the nodes under arbitration phase see the bus outputting a bit other than its own bit, senses that there is a message that needs to be sent at a higher priority than itself and therefore automatically stops its transmission. The node that ends up sending its entire identifier over the bus automatically wins arbitration phase and continues to broadcast the rest of its data [38]. A dominant bit (i.e. logical high or 0) always gains priority over a recessive bit (i.e. logical low or 1). Figure 2.7 shows an arbitration process. In this case node 3 wins' arbitration and broadcasts its message whereas node 1 and 2 are now listeners.

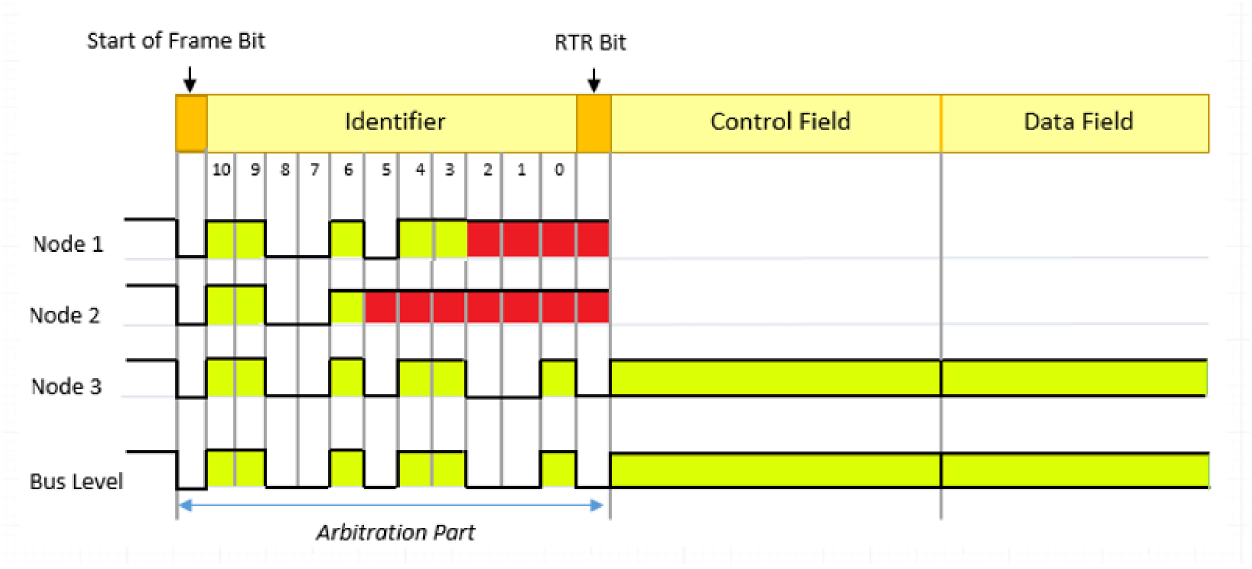


Figure 2.7: Bit arbitration phase showing node 3 to be transmitting its identifier, gaining priority over nodes 1 and 2 based on their CAN IDs

Frame types: There are 4 types of frames on the CAN bus:

- **Data frame:** simply contains the data to be transmitted, as simple as steering wheel angle, etc. This data may be stored in memory or used by another node for making a decision.
- **Remote frame:** frame to request data from a specific node, very powerful for diagnostic applications, used by car repairmen.
- **Error frame:** usually transmitted if a receiving node detects an error in the data value or something which is not part of protocol, such as an overflow from 8-bit value for the data.
- **Overload frame:** introduces a delay between RTR frame or data frames.

These frames can be extremely useful while trying to determine errors on the bus as well as reverse engineer the CAN bus [9].

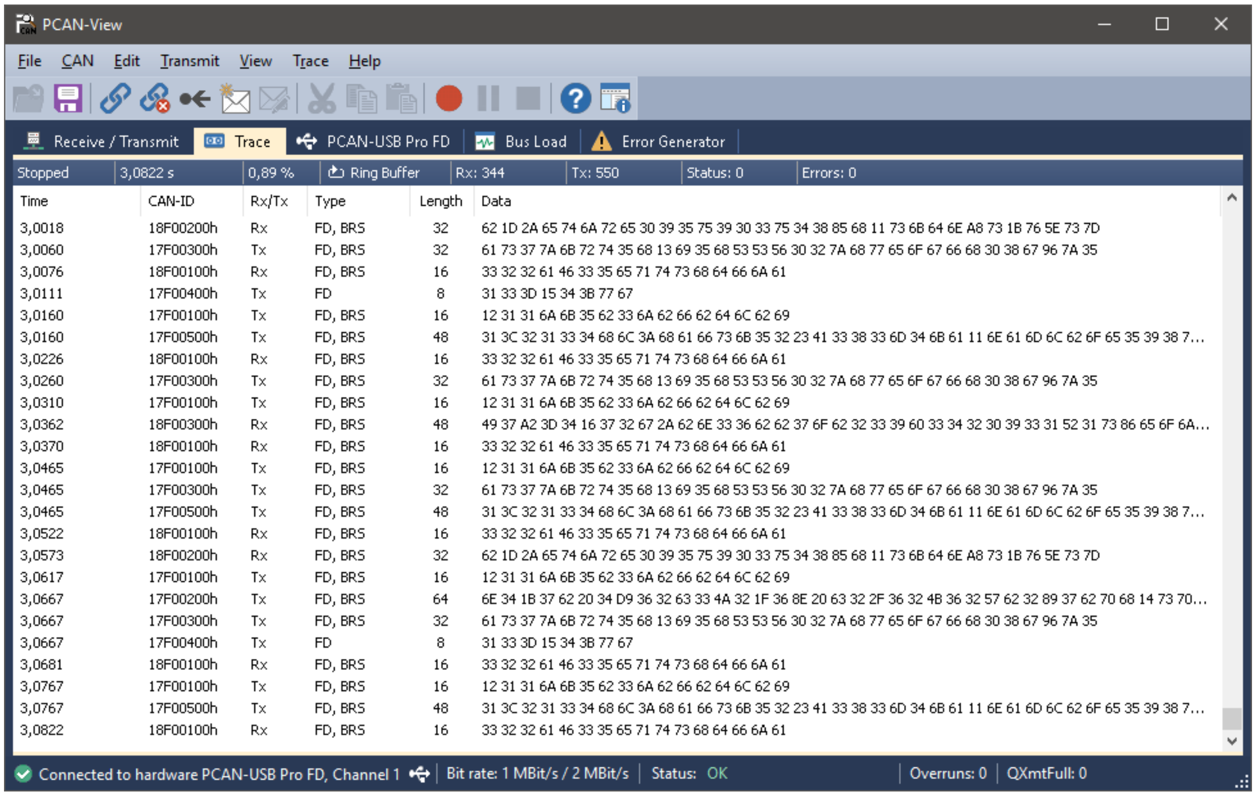
2.2.4 Example CAN messages with PCAN Viewer:

PCAN viewer is piece of software that works with PCAN PCIe CAN card, in other words, PCAN PCIe is the CAN controller node that can be integrated with a sensor for communication over CAN bus [39]. Figure 2.8 shows the different CAN nodes on the bus and their corresponding data transmissions. It can be seen that each node has a 29-bit identifier and transmits 32 bits of data using the CAN FD protocol.

2.3 Time to Digital Convertors

2.3.1 Introduction

The wikipedia definition of a time-to-digital convertor (TDC) says that a TDC is an instrument to provide a digital representation of the time of occurrence of an event [40]. In addition, the most common use of a time-to-digital convertor is to measure the time difference between the occurrence of 2 events. It essentially functions as a stopwatch. For our case also, we want to use the TDC to get the time interval between 2 signals. The most widely used application of a TDC is in laser range finding, when a start pulse is generated as the laser is sent and a stop pulse is generated when the laser returns from a target [41]. Once the signal’s rising or falling edge crosses a predetermined threshold, the measurements are made.



Stopped	3,0822 s	0,89 %	Ring Buffer	Rx: 344	Tx: 550	Status: 0	Errors: 0

Time	CAN-ID	Rx/Tx	Type	Length	Data
3,0018	18F00200h	Rx	FD, BRS	32	62 1D 2A 65 74 6A 72 65 30 39 35 75 39 30 33 75 34 38 85 68 11 73 6B 64 6E A8 73 1B 76 5E 73 7D
3,0060	17F00300h	Tx	FD, BRS	32	61 73 37 7A 6B 72 74 35 68 13 69 35 68 53 53 56 30 32 7A 68 77 65 6F 67 66 68 30 38 67 96 7A 35
3,0076	18F00100h	Rx	FD, BRS	16	33 32 32 61 46 33 35 65 71 74 73 68 64 66 6A 61
3,0111	17F00400h	Tx	FD	8	31 33 3D 15 34 3B 77 67
3,0160	17F00100h	Tx	FD, BRS	16	12 31 31 6A 6B 35 62 33 6A 62 66 62 64 6C 62 69
3,0160	17F00500h	Tx	FD, BRS	48	31 3C 32 31 33 34 68 6C 3A 68 61 66 73 6B 35 32 23 41 33 38 33 6D 34 6B 61 11 6E 61 6D 6C 62 6F 65 35 39 38 7...
3,0226	18F00100h	Rx	FD, BRS	16	33 32 32 61 46 33 35 65 71 74 73 68 64 66 6A 61
3,0260	17F00300h	Tx	FD, BRS	32	61 73 37 7A 6B 72 74 35 68 13 69 35 68 53 53 56 30 32 7A 68 77 65 6F 67 66 68 30 38 67 96 7A 35
3,0310	17F00100h	Tx	FD, BRS	16	12 31 31 6A 6B 35 62 33 6A 62 66 62 64 6C 62 69
3,0362	18F00300h	Rx	FD, BRS	48	49 37 A2 3D 34 16 37 32 67 2A 62 6E 33 36 62 62 37 6F 62 32 33 39 60 33 34 32 30 39 33 31 52 31 73 86 65 6F 6A...
3,0370	18F00100h	Rx	FD, BRS	16	33 32 32 61 46 33 35 65 71 74 73 68 64 66 6A 61
3,0465	17F00100h	Tx	FD, BRS	16	12 31 31 6A 6B 35 62 33 6A 62 66 62 64 6C 62 69
3,0465	17F00300h	Tx	FD, BRS	32	61 73 37 7A 6B 72 74 35 68 13 69 35 68 53 53 56 30 32 7A 68 77 65 6F 67 66 68 30 38 67 96 7A 35
3,0465	17F00500h	Tx	FD, BRS	48	31 3C 32 31 33 34 68 6C 3A 68 61 66 73 6B 35 32 23 41 33 38 33 6D 34 6B 61 11 6E 61 6D 6C 62 6F 65 35 39 38 7...
3,0522	18F00100h	Rx	FD, BRS	16	33 32 32 61 46 33 35 65 71 74 73 68 64 66 6A 61
3,0573	18F00200h	Rx	FD, BRS	32	62 1D 2A 65 74 6A 72 65 30 39 35 75 39 30 33 75 34 38 85 68 11 73 6B 64 6E A8 73 1B 76 5E 73 7D
3,0617	17F00100h	Tx	FD, BRS	16	12 31 31 6A 6B 35 62 33 6A 62 66 62 64 6C 62 69
3,0667	17F00200h	Tx	FD, BRS	64	6E 34 1B 37 62 20 34 D9 36 32 63 33 4A 32 1F 36 8E 20 63 32 2F 36 32 4B 36 32 57 62 32 89 37 62 70 68 14 73 70...
3,0667	17F00300h	Tx	FD, BRS	32	61 73 37 7A 6B 72 74 35 68 13 69 35 68 53 53 56 30 32 7A 68 77 65 6F 67 66 68 30 38 67 96 7A 35
3,0667	17F00400h	Tx	FD	8	31 33 3D 15 34 3B 77 67
3,0681	18F00100h	Rx	FD, BRS	16	33 32 32 61 46 33 35 65 71 74 73 68 64 66 6A 61
3,0767	17F00100h	Tx	FD, BRS	16	12 31 31 6A 6B 35 62 33 6A 62 66 62 64 6C 62 69
3,0767	17F00500h	Tx	FD, BRS	48	31 3C 32 31 33 34 68 6C 3A 68 61 66 73 6B 35 32 23 41 33 38 33 6D 34 6B 61 11 6E 61 6D 6C 62 6F 65 35 39 38 7...
3,0822	18F00100h	Rx	FD, BRS	16	33 32 32 61 46 33 35 65 71 74 73 68 64 66 6A 61

Connected to hardware PCAN-USB Pro FD, Channel 1 | Bit rate: 1 MBit/s / 2 MBit/s | Status: OK | Overruns: 0 | QXmtFull: 0

Figure 2.8: PCAN view showing CAN bus nodes’ identifiers and corresponding messages

2.3.2 Application

TDCs find a huge application in range finding as mentioned above. They are very popularly used for LiDAR (light detection and ranging), drones and robotics, ADAS systems for automobiles, collision avoidance systems, flow meters, IC testing, high energy physics experiments and high-speed data transfer applications [41]. One of the unique features of TDC is that it can very accurately measure the time intervals even to the order of picosecond resolution, which makes it very useful for above applications where time measurements are extremely critical to the order of picoseconds and for some cases even up to femtoseconds.

2.3.3 TDC architecture

Coarse Counter mechanism: The simplest form of a time-to-digital convertor is a stopwatch or a simple counter mechanism. The time difference between the rising or falling edges of the start and stop pulse are measured using a high frequency clock. The time difference will be an integer multiple of the reference clock period, and the counter value denotes number of clock cycles passed between the start and stop pulses at any point of time during the measurement phase. However, this limits the resolution of our TDC counter to a single clock period as any time interval between a clock pulse will go unnoticed. To achieve higher resolution, we would need a faster clock frequency however the fast frequency of the oscillator could impact the performance of the clock. The mechanism to realize this architecture is shown in Figure 2.9. [42]

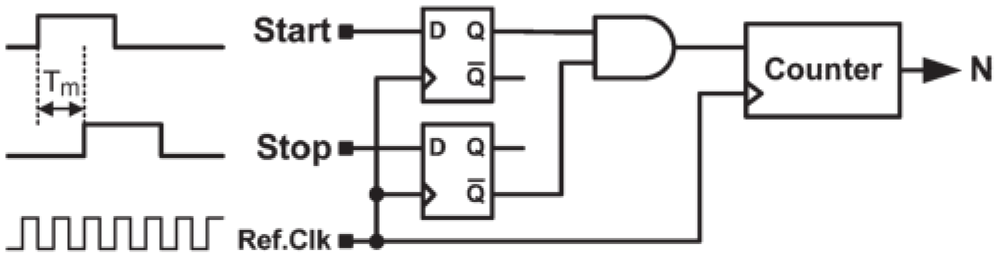


Figure 2.9: Time to digital convertor realization with counter mechanism

An important thing to note here is the use of AND gate which enables counter when the start and stop pulses are logically different. The maximum error possible with this method is nearly 1 clock cycle, if the start pulse is received just after the clock pulse begins and if the stop pulse is received just before a new clock cycle.

Finer Counter mechanism: As seen from above we definitely need a higher resolution to measure time values at higher orders of accuracy. This can be achieved in several ways however at the cost of measuring much smaller time intervals. One of the methods is to use a tapped delay line, which consists of several D-flip-flops each with a delay buffer of very specific delay time τ. The start signal is delayed by time τ every time it travels through a flip-flop and the output of the flip-flop is measured and recorded as the signal traverses through it. The stop signal is latched to all the flip-flops but does not get delayed. The flip-flop outputs are measured until the delayed start signal matches with the un-delayed stop signal. Thus, the number of flip-flops which give an output correspond to the number of delays and hence the time difference between the measured signals. The drawback to this mechanism is that firstly the resolution is again limited to the delay time of each flip-flop and secondly because we have only a finite number of flip-flops the total range of time that can be measured is very limited. An example of the tapped delay line circuit is shown in Figure 2.10 and Figure 2.11 [43] [44]

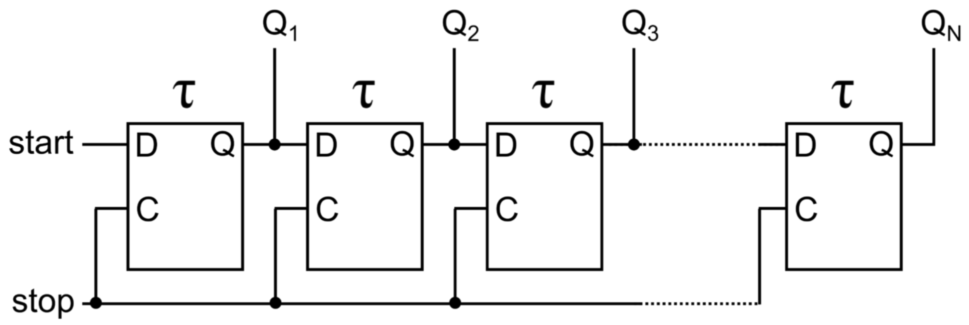


Figure 2.10: Tapped Delay line having propagation delay of τ every flip-flop to measure time interval between start and stop pulse

$$\Delta T = n\,\tau$$

The other methods include flash tdc, Vernier oscillator, cyclic pulse shrinking tdc ([42], [45], [46], [47]). We will be briefly explaining these methods; however, the reader is encouraged to explore

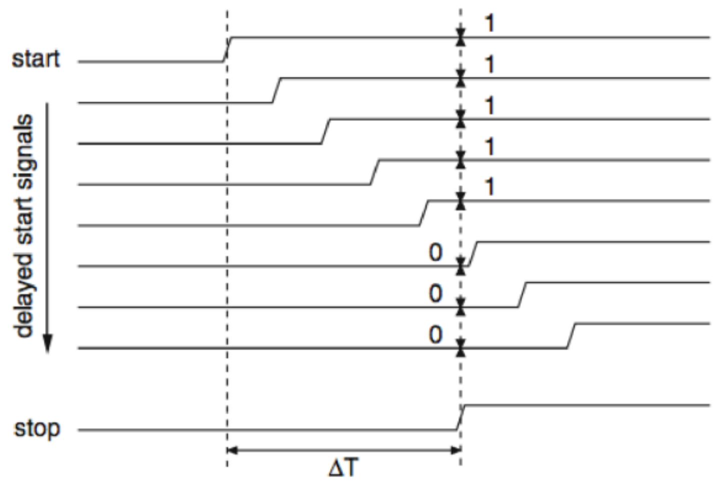


Figure 2.11: Delayed start signal from tapped delay line mechanism

them more in details if needed. The flash TDC is very similar to the tapped delay line except the start pulse is compared to a reference clock instead of the stop pulse. It suffers the same drawback as the tapped delay line where resolution is limited to τ. Flash TDCs find applications in on-chip timing measurement systems. On the other hand, Vernier oscillators use ring oscillators coupled to the start and stop pulse to achieve much higher resolution. However, they are much slower as compared to the flash tdc. As the 2 ring oscillators are having a slight difference in their delay time, this leads to a higher resolution.

$$\Delta T = n.(\tau_1 - \tau_2)$$

Figure 2.12 shows a typical Vernier Oscillator TDC

Ring Oscillators in general are composed of an odd number of NOT gates in series each having a specific time delay [48]. The output of the ring oscillator is fed back into the input creating a square wave of time period 2 times the total delay of the oscillator. It is sufficient to understand that ring oscillators are nothing but a clock counter with a very high frequency or very small time period which is used to measure the time delay for our purpose. The specific cycle time delay of the ring oscillator used for our application is 55ps, as mentioned in the later part of the thesis in Section 2.3.7.

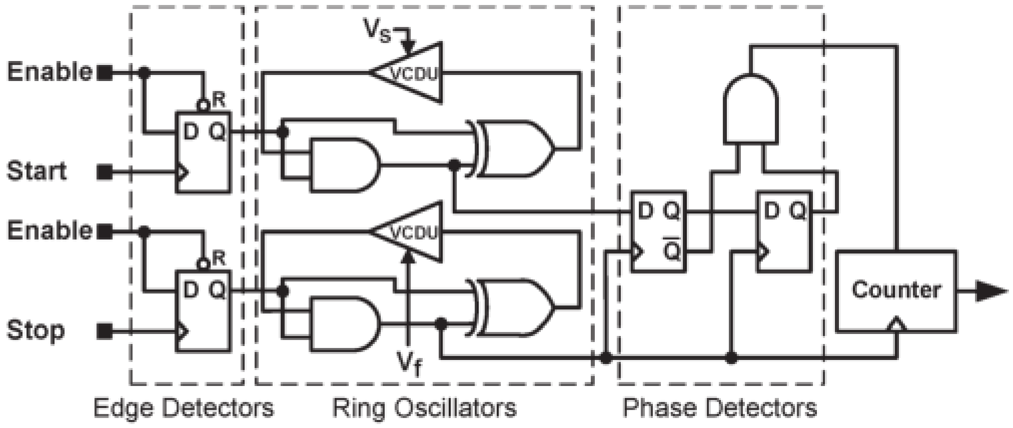


Figure 2.12: TDC using a Vernier oscillator mechanism

Hybrid Mechanism: This mechanism takes the best of both worlds for coarse and fine counters and achieves a higher resolution with longer time measurement range [49]. It uses both a coarse counter for long time interval measurement and a finer counter for measuring time between start pulse and next clock cycle as well as stop pulse and next clock cycle. With these 3 measurements, the time interval between start and stop pulse can be calculated with a higher resolution and a longer interval. A synchronizer is used to determine the nearest clock edge to the incoming pulses. This approach is commonly known as the Nutt method [50]. The reader can get a better descriptive understanding of this concept from Figure 2.17.

$\Delta T = n.\tau + T_{start} - T_{stop}$, where T_{start} is the nearest clock cycle from start pulse and T_{stop} is nearest clock cycle from stop pulse

It has been proved that a resolution in the range of picoseconds can be achieved with this method.

2.3.4 Errors in TDC

Although we can achieve a high level of precision in time measurement, the TDC is subject to certain kinds of errors [40]. Firstly, at high reference clock frequencies, clock performance can be impacted and clock jitter can affect the resolution of the TDC. Secondly, offset errors (non-zero readings measured at time T=0) may be present which can be removed. Thirdly, there may be some nonlinearity in the TDC measurement, which can be found by taking a large number of readings from a Gaussian signal and plotting the data to check for nonlinearities. TDC accuracy may be subject to temperature changes. Due to asynchronicity in the fine and coarse counter, calibration may have to be performed to achieve consistent measurements. Lastly, external sources can contribute to noise in the instrument which can be removed using robust estimation techniques [51]. We will not be going into a lot of details for these but rather try to use a TDC which can have certain compensation techniques in-built to overcome these errors as much as possible.

2.3.5 Survey of available TDCs

Sr. No	**TDC**	**Range**	**Resolution**	**Current**	**Voltage**	**Temperature**	**Interface**	**Number of channels**	**Applications**	**Cost**
1	Max35101	8 ms	20 ps	10 uA	2.3-3.6 V	-40 to 85 ºC	SPI	Single stop	Ultrasonic heat, water and gas meter	$3.94
2	TDC 7200	12ns – 8ms	55 ps	0.5 uA	2-3.6V	-40 to 85 ºC	SPI	5 stops	Flow meter, TOF LiDAR, level sensing	$2.08
3	TDC 7201	0.25ns-8ms	55ps	2.7mA	2-3.6V	-40 to 85 ºC	SPI	10 stops	Range finder, LiDAR, Drones, ADAS	$3.15
4	THS788	0-7s	8ps	204mA	3.3V	-	SPI	4 stops	Radar, medical imaging, mass spectroscopy	$227.43

5	TDC-GPX2	0-16s	20ps	136mA	3.3V	-	SPI	4 stops	Laser range finding, particle physics	$43.04
6	TDC-GP22	700ns-4ms	Up to 20ps	0.5 uA	2.5-3.6V	-40 to 125 ºC	SPI	3 stops	Ultrasonic flow meter	$2.96
7	TDC-GPX	65us	40ps	39mA	2.3-3.6V	Max 125 ºC	SPI	2 stops		$167.39

Table 2-1: Survey of available TDCs

The prices for above equipment are from Mouser website as of 3rd May, 2019.

2.3.6 Choice of TDC

Since the objective was to get a piece of equipment which was not only cost effective but also supported variable voltage input with less power consumption and reasonable accuracy for CAN bus applications, where time of flight values are in the range of several microseconds, it was decided to move forward with the TDC7201 chip. The ultimate aim is to deploy the circuit as a standalone system integrated with every CAN controller for intrusion detection purposes.

Additionally, we found that the TDC7201 module offers some advantages over the other options. Firstly, there are 2 built in TDC chips each with 5 stop pulses which can be measured from a single start pulse allowing us to measure up to 10 stop pulses simultaneously. It has a built-in self-calibration technique allowing a high resolution of 55 picoseconds, while compensating the drift over time and temperature. Since the system needs to be deployed, battery life was a concern for us which the TDC handles by having an Autonomous Multi-Cycle Averaging mode allowing for a low power consumption unit. It remains in sleep mode until an interrupt is generated for new measurement. It allows for flexible voltage supply from 2-3.6V which is available on most microcontrollers. Lastly to save us from the trouble of doing the coding for SPI, TI has an evaluation module made which has all the firmware built-in for accessing registers and performing calibrations. The TDC-7201-ZAX EVM is a PCB package booster pack for the MSP430 microcontroller with the TDC chip integrated on it.

2.3.7 TDC 7201

- *Block Diagram:* The functional block diagram of the TDC as shown in Figure 2.13 [41].

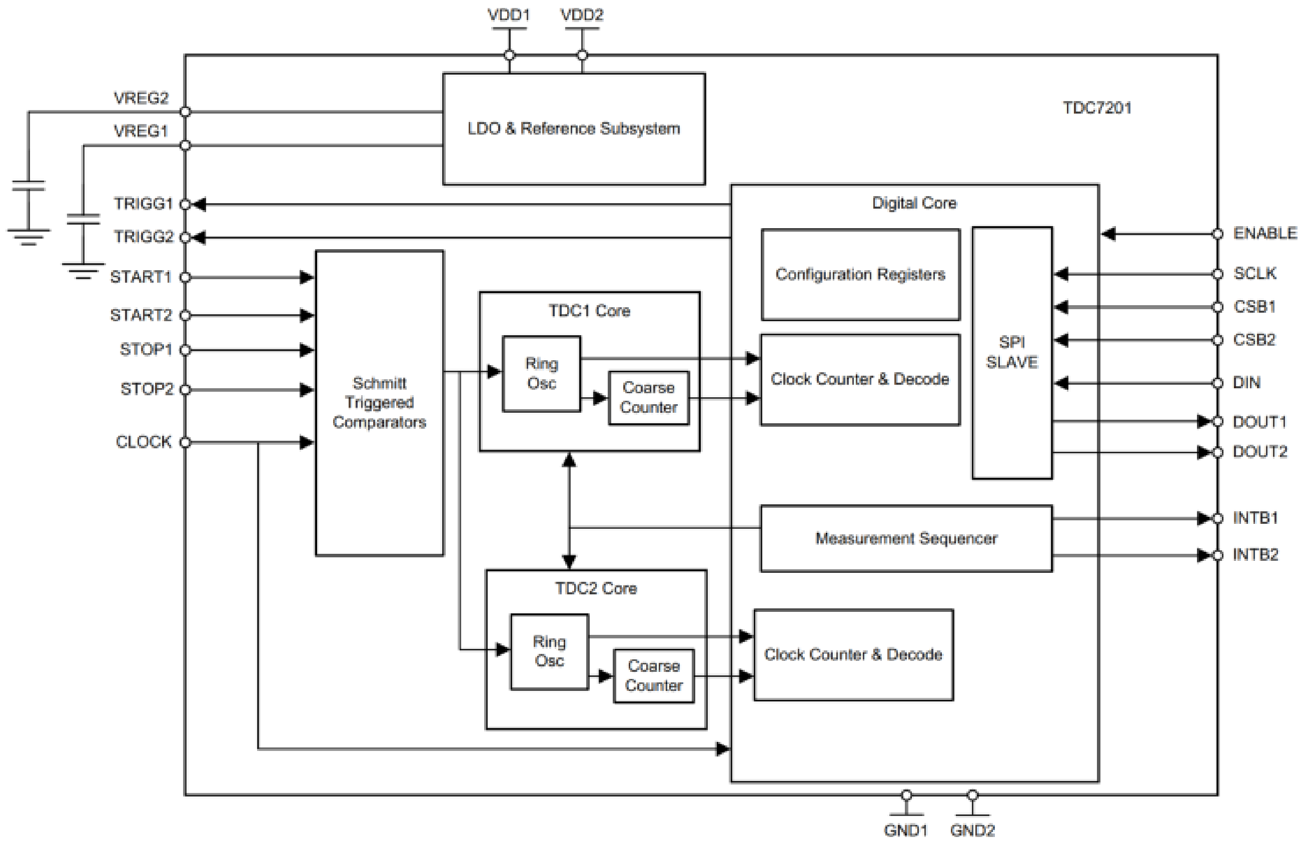
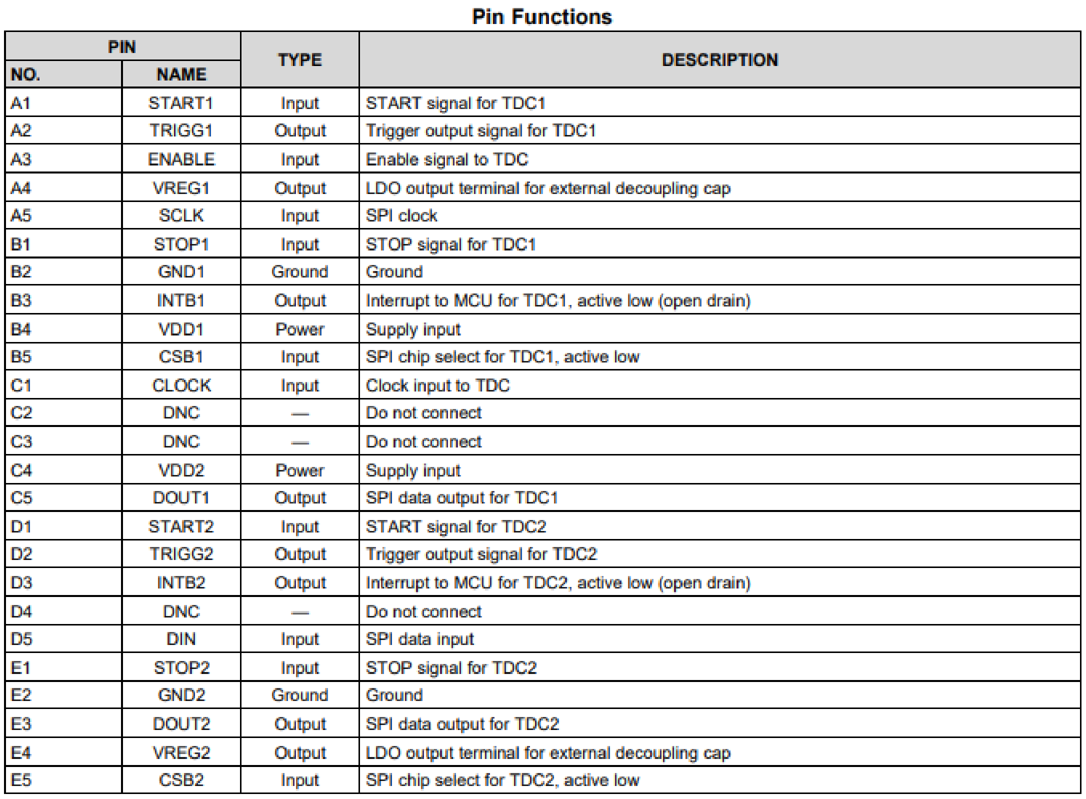


Figure 2.13: TDC 7201 Block Diagram showing internal circuit elements

The pin descriptions are shown in Figure 2.14.

It can be seen in the functional block diagram that VDD1 and VDD2 are power supply inputs to the TDC chip and Vreg1 and Vreg2 are the regulated outputs after attaching a decoupling capacitor. Then we can see that the start and stop signals go into the Schmitt trigger comparator along with the reference clock signal. The output of the trigger comparator goes to the ring oscillators in each of the TDC chips. If the time interval between the start and stop pulses is less than 2000ns, then the it is best to operate the TDC in measurement mode 1 in which the ring oscillator keeps running and the coarse counter keeps a check on the number of ring oscillator



Pin Functions

PIN		TYPE	DESCRIPTION
NO.	NAME		
A1	START1	Input	START signal for TDC1
A2	TRIGG1	Output	Trigger output signal for TDC1
A3	ENABLE	Input	Enable signal to TDC
A4	VREG1	Output	LDO output terminal for external decoupling cap
A5	SCLK	Input	SPI clock
B1	STOP1	Input	STOP signal for TDC1
B2	GND1	Ground	Ground
B3	INTB1	Output	Interrupt to MCU for TDC1, active low (open drain)
B4	VDD1	Power	Supply input
B5	CSB1	Input	SPI chip select for TDC1, active low
C1	CLOCK	Input	Clock input to TDC
C2	DNC	—	Do not connect
C3	DNC	—	Do not connect
C4	VDD2	Power	Supply input
C5	DOUT1	Output	SPI data output for TDC1
D1	START2	Input	START signal for TDC2
D2	TRIGG2	Output	Trigger output signal for TDC2
D3	INTB2	Output	Interrupt to MCU for TDC2, active low (open drain)
D4	DNC	—	Do not connect
D5	DIN	Input	SPI data input
E1	STOP2	Input	STOP signal for TDC2
E2	GND2	Ground	Ground
E3	DOUT2	Output	SPI data output for TDC2
E4	VREG2	Output	LDO output terminal for external decoupling cap
E5	CSB2	Input	SPI chip select for TDC2, active low

Figure 2.14:TDC 7201 Pin Layout

cycles passed. Each cycle has a specific delay of nearly 55ps. This value is then passed to the digital core which has an Arithmetic Logic Unit (ALU) performing the calculations for the time measurement values. If, however, the time interval to be measured is greater than 2000ns then the TDC should run in measurement mode 2 in which the ring oscillator checks for the time between start pulse and next reference clock cycle, the clock counter keeps track of number of clock cycles passed from the start pulse and then again, the ring oscillator checks the time from the stop pulse to the next clock cycle as seen in Figure 2.17. Using these 3 measurement values, the digital core makes a calculation of the total time between the start and the stop pulse.

The digital core contains all the configuration registers for setting up the TDC, clock counter for checking the integer number of reference clock cycles happening between the start and stop pulses, SPI registers for chip select, clock, data in and data out registers, and finally a sequencer to generate interrupt once the measurement is complete. The digital core is also responsible for sending a trigger signal to the target for sending the start pulse to the TDC. The reference clock frequency can go up to 16 Mhz. The higher the reference clock frequency more accurate are the TDC readings, with the standard deviation in the time measurement decreasing as the clock frequency increases. Figure 2.15 shows the variation in standard deviation with clock frequency.

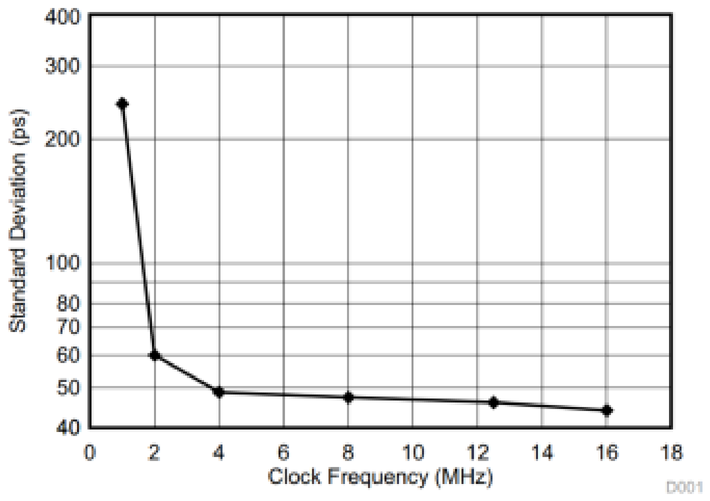


Figure 2.15: Standard Deviation in time measurements vs reference clock frequency

- *Measurement Mode 1:* The measurement mode 1 is shown in Figure 2.16.

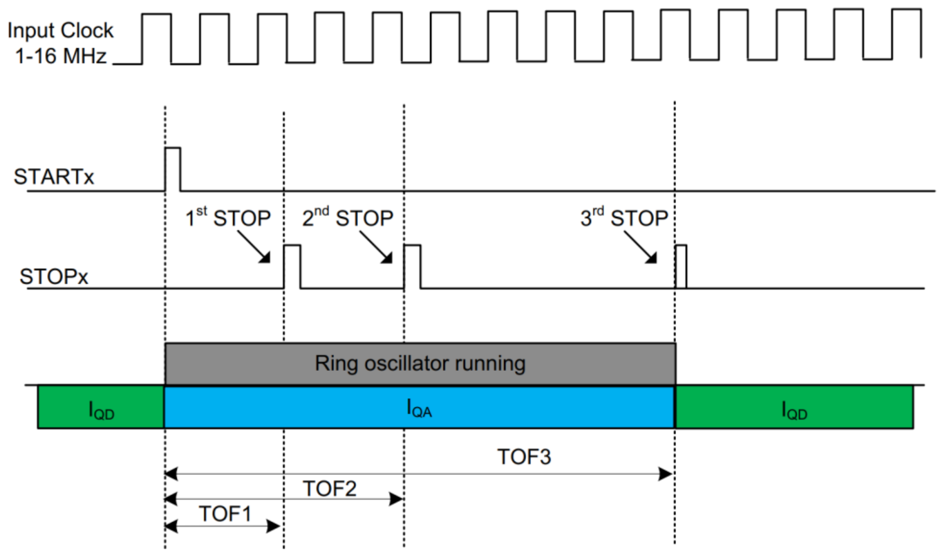


Figure 2.16: Measurement mode 1 working to calculate TOFs

It can be seen that multiple stop pulse can be measured in this mode however, all measurements use the ring oscillator fine counter only for measurement and since the counter value is limited to a 23-bit number, it can only go up to 400 us. However, TI recommends using measurement mode 1 only up to 2000ns else the standard deviation in the results will be higher than 200 ps. One of the forum question on TI also mentioned that measurement mode 1 should be used for time intervals only up to 500ns else the ring oscillator would heat up leading to an increased jitter and therefore increased standard deviation [52].

We conducted some tests to verify this and realized that the counter value can only go to a maximum of 22-bit and not 23-bit, limiting the measurement time to 230 us. However, there is a high standard deviation in agreement with TI. The datasheet also shows how to take the register values and calculate the resolution achieved and time interval between the pulses. The reader may refer the datasheet for sample calculations which are fairly straightforward to understand.

- *Measurement Mode 2:* The measurement mode 2 is shown in the Figure 2.17.

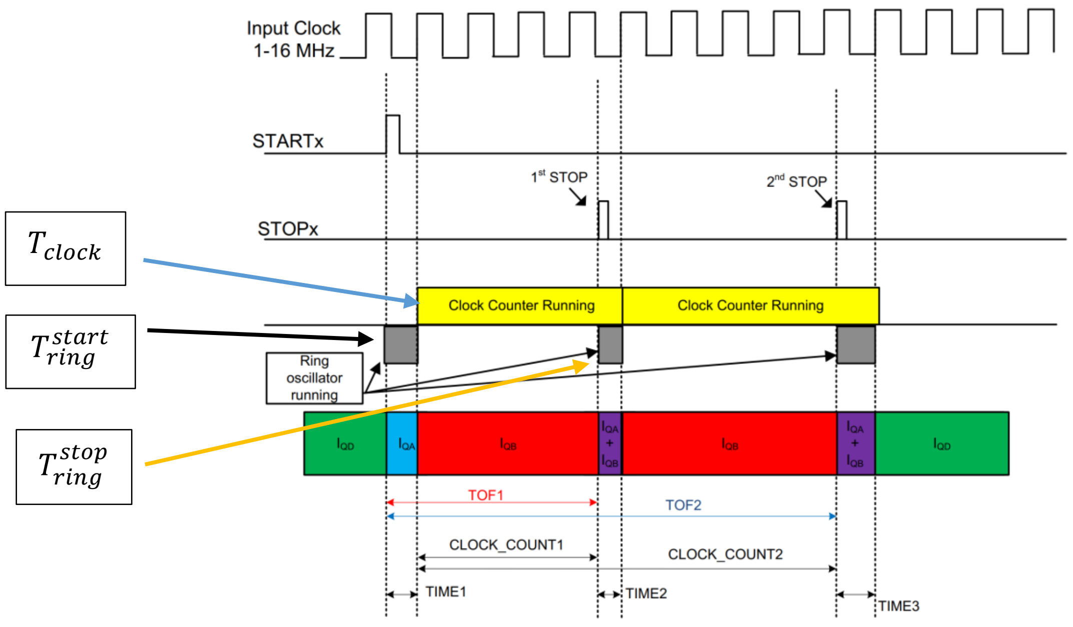


Figure 2.17: Measurement mode 2 working to calculate TOFs

It can be seen that the ring oscillator runs from the start pulse to the next clock pulse, after which the clock starts counting, and the ring oscillator again runs from the stop pulse to the next clock pulse.

$\Delta T = T_{ring}^{start} + T_{clock} - T_{ring}^{stop}$, where T_{ring}^{start}, T_{clock} and T_{ring}^{stop} are shown in the figure above

- *Calibration:*

The TDC has an accuracy of 28 ps in general however, this value can change due to temperature, system noise, and other characteristics. There can be offset errors too depending on device's internal delays. In order to update the LSB or resolution, the TDC performs internal calibration to compensate for these errors.

The TDC performs 2 calibrations consuming a total of 2 measurement cycles of the reference clock. During any time of flight measurement cycle, the 1st calibration is essentially measure of a single clock period of the external or reference clock. The 2nd calibration is the number of reference clock period set for calibration in the configuration register by the user. These values are stored in the calibration registers and the datasheet shows how to perform calculations with these values to calculate the actual resolution for time of flight measurement. The TDC performs automatic calibration after the start and before the completion of every measurement, unless any counter overflow is recorded which terminates the calibration.

- *Multi-cycle Averaging:*

This feature is mostly used to optimize power consumption of the target MCU as well as reducing noise in the measurements giving a better accuracy. This feature is very useful during operation in combined measurement mode to measure extremely low time of flight values starting from 250ps. This will allow for noise averaging and since the MCU stays in sleep mode until the TDC performs averaging, allowing for reduced power consumption, after which the TDC sends an interrupt to the MCU to start data processing. Figure 2.18 shows concept of multi-cycle averaging and Figure 2.19 shows it being applied for combined mode operation [41].

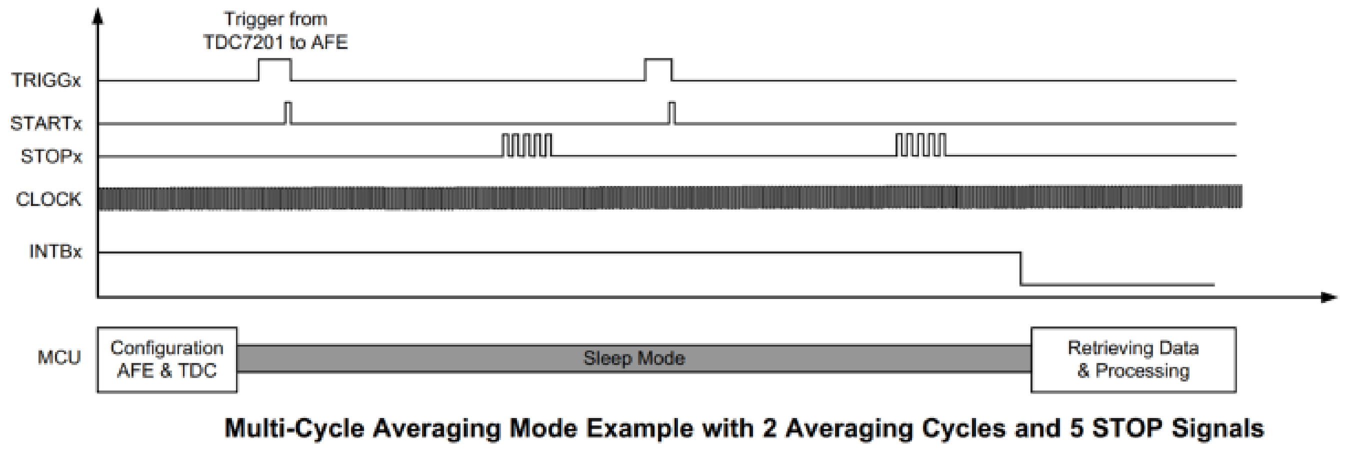

Trigger from TDC7201 to AFE
TRIGGx
STARTx
STOPx
CLOCK
INTBx
MCU
Configuration AFE & TDC
Sleep Mode
Retrieving Data & Processing
Multi-Cycle Averaging Mode Example with 2 Averaging Cycles and 5 STOP Signals

Figure 2.18: Multi-cycle Averaging for measuring time values from 250ps under very low power consumption mode and noise reduction

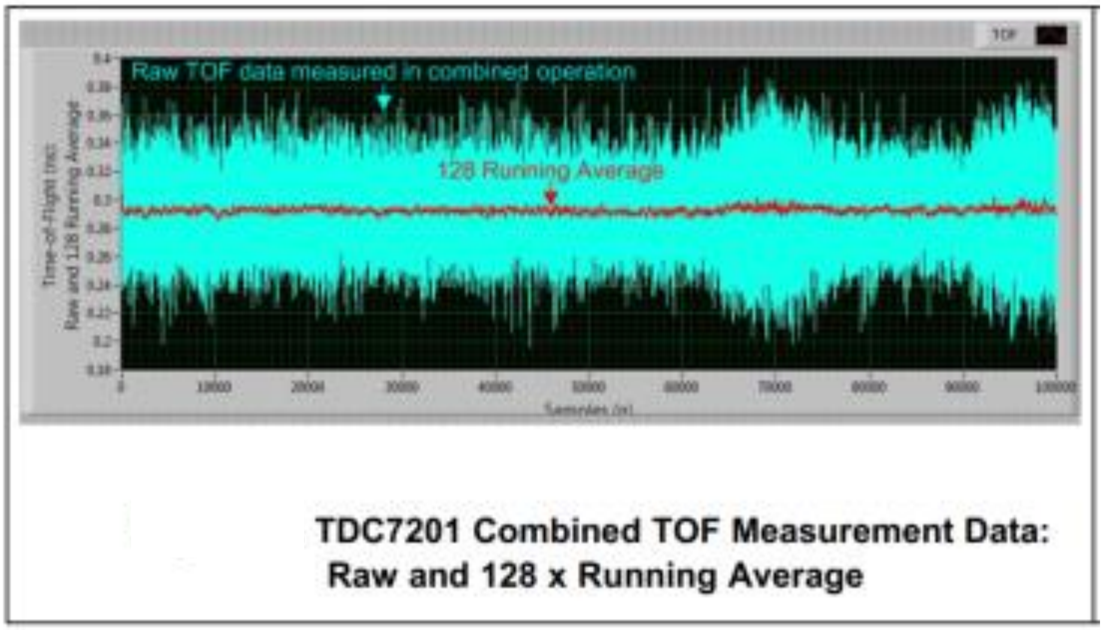


Figure 2.19: Result of multi-cycle averaging for time of flight values

- *Measurement Sequence:*

There are 10 steps in the measurement sequence.

1. Once the TDC is powered up, the Enable pin needs to go low and then a low to high transition is required for correct power up sequence.
2. MCU requests TDC for new measurement via SPI
3. The start new measurement bit is set in the config register after which the TDC generates a trigger signal on the Trig pin allowing the front-end sensor to start its own measurement sequence, for example sending the laser pulse for starting measurement.
4. Once the trigger is sent, the Start pin on the TDC is enabled, waiting to receive the start pulse.
5. Once the start pulse arrives, the trig pin is reset by the TDC.
6. The clock counter starts counting from the next rising edge of the reference clock signal from the arrival of start pulse.
7. The TDC waits for the stop pulse and it can record up to 5 stop signals.
8. Once the last stop signal arrives, the TDC will send an interrupt to the MCU to perform processing of available data. The INTB pin is low during data processing and goes back to high when the MCU asks the TDC to start a new measurement.
9. Once the MCU fetches the results from the TDC registers, it can ask to start a new measurement by setting the start new measurement bit via SPI. The enable pin does not need to go low again.
10. If the enable pin goes low for setting the TDC in low power mode due to a long-time measurement then once the enable pin goes high, the config settings would have gone to default.

- *Clock Accuracy and Jitter:*

Texas Instruments says that two parameters critical in time of flight applications are clock accuracy and clock jitter [41].

Clock accuracy is the number of pulses that a clock can be offset from its nominal value during operation. It is calculated in ppm (pulses per million). If an 8MHz clock has an accuracy or error of 20ppm then the frequency range can be calculated as follows:

8 MHz + 8 MHz * 20/10^6 = 8.00016 MHz and 8 MHz - 8 MHz * 20/10^6 = 7.99984 MHz

Thus, range is [7.99984 MHz, 8.00016 MHz] for an 8 MHz clock with 20ppm accuracy.

This inaccuracy is directly translated to time measurement as well. If the same clock having 20ppm error measures a time of 10us then the error in that measurement is as follows:

10us * 20/10^6 = 0.2 ns

Clock jitter on the other hand contributes more in increasing uncertainty in the measurement than the accuracy, i.e. proportional to standard deviation; whereas, accuracy is proportional to mean/bias. The jitter is a function of clock cycles and uncertainty due to jitter accumulates with every cycle.

Clock jitter uncertainty = $\sqrt{n} * \ \theta\_jitter$, where n is the number of clock cycles and Θ_jitter is cycle-to-cycle jitter of clock

So, if we want to measure 50us using an 8 MHz clock then n = 400 clock cycles. Assume jitter = 10ps, then measurement uncertainty = 20 * 10 ps = 200 ps for measuring 50 us time interval.

If we perform ‘m’ averaging cycles for a measurement then the jitter reduces by the order of $\sqrt{m}$.

Example of clock jitter in principle is shown in Figure 2.20.

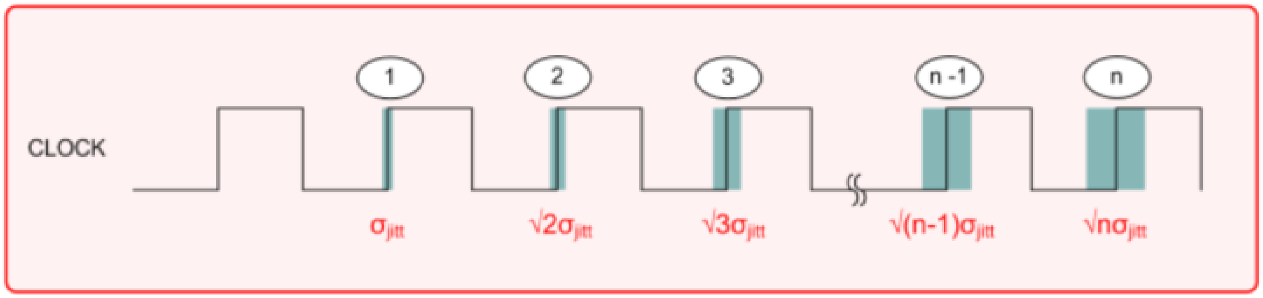


Figure 2.20: Example of clock jitter accumulating with increase in number of clock cycles passed

We can safely avoid going into much details of the errors posed by reference clock accuracy and jitter, since our TDC has built-in internal compensation techniques to correct for these errors as mentioned in Section 2.3.6.

Important TDC parameters:

Supply Voltage	2 – 3.6 V
Differential voltage between any 2 terminals	3.9V
Input current at any pin	-5 to 5 mA
Ambient Temperature	-40 to 125°C
Voltage input high	0.7*V_{supply} - V_{supply}
Voltage input Low	0 - 0.3*V_{supply}
Frequency of reference clock	1-16MHz
Min time between start and stop signal	12 ns (measurement mode 1)
Min time between start and stop signal	2*t_{clock} (2*125ns = 250ns for 8MHz clock)
Max time between start and stop signal	(2^{16}-2) * t_{clock} (8ms for 8MHz clock)

Max rise/fall time for start/stop signal	1ns
Time from trig to start signal	5ns
LSB/resolution for single shot measurement	55ps
Accuracy for 100us time interval	28ps (bias)
Standard deviation for 100us time interval	35ps (std dev)
Min pulse width for start and stop signal	10ns
Range	250ns-8ms

Table 2-2: Important parameters of TDC7201

Figure 2.21 shows (minimum value of) signals expected by the TDC7201 for correct working.

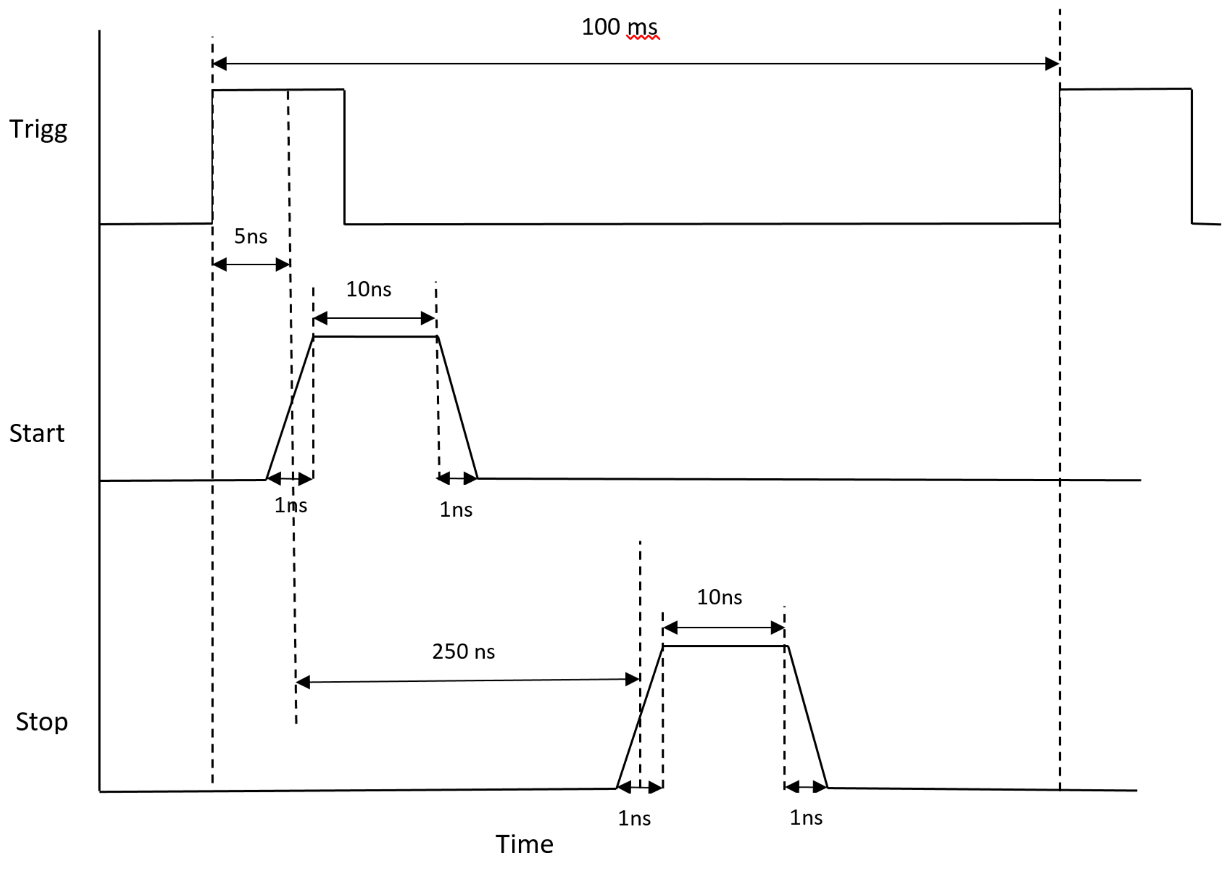


Figure 2.21: Minimum signal parameters for correct working of TDC7201

2.3.8 TDC7201-ZAX-EVM

The TDC EVM is an evaluation module build to serve as a booster pack for the MSP430 which behaves as the host microcontroller for acquiring and processing the data from the TDC using SPI communication. The advantage of using this EVM is that TI has created a GUI with this module, to be used for manipulating the registers as per our needs eliminating the SPI intensive register level code for accessing each register and keeping track of bit timings. We can simply access the GUI and tune the TDC parameters as per our needs. All one has to do is to connect the TDC to the MSP and update the MSP firmware from TI's website which will directly flash the necessary code to the board for working with the TDC and evaluating its performance. Figure 2.22 shows the EVM board.

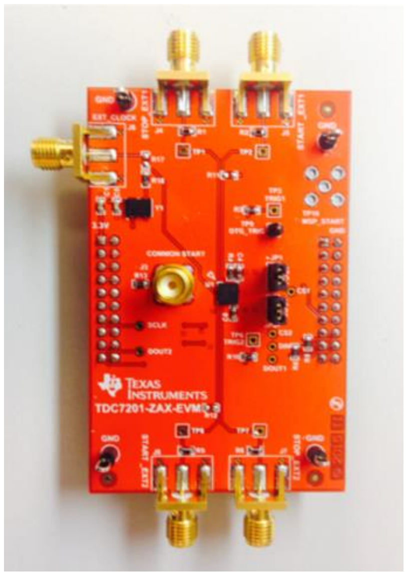
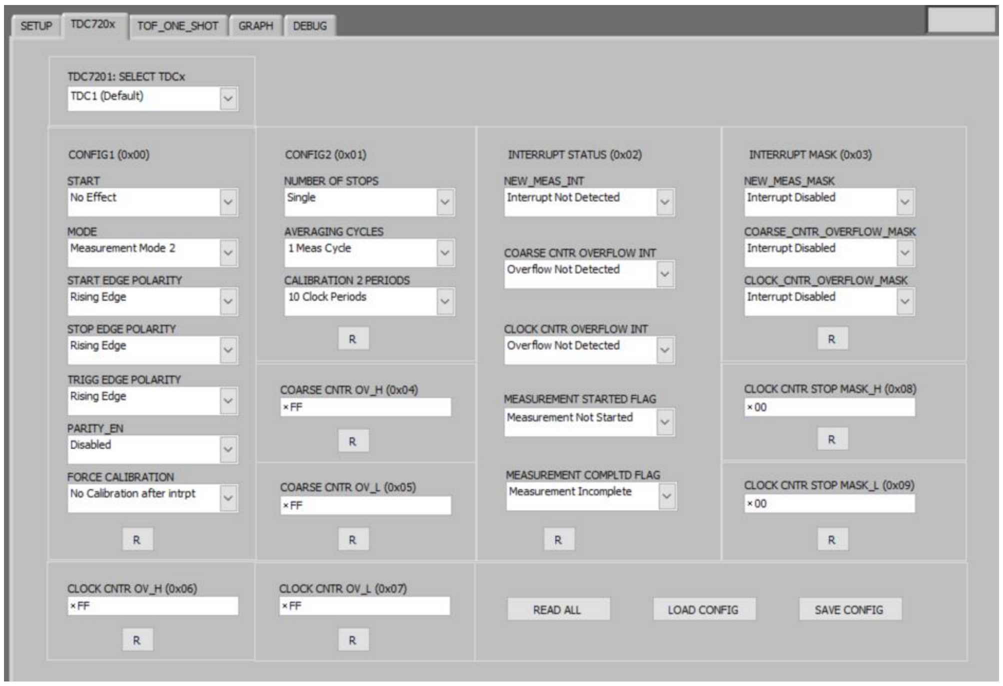


Figure 2.22: TDC evaluation module

Additionally, we just need to use 2 BNC-to-SMA cables for connections to the start and stop pins, a micro-USB to power the MSP430 from a laptop/PC and we must have the GUI installed on the host system. Few pictures of the GUI are shown in Figure 2.23, Figure 2.24 and Figure 2.25.

Note these figures have been adopted form the TDC EVM manual.



TDC7201: SELECT TDCx: TDC1 (Default)

CONFIG1 (0x00)
START: No Effect
MODE: Measurement Mode 2
START EDGE POLARITY: Rising Edge
STOP EDGE POLARITY: Rising Edge
TRIGG EDGE POLARITY: Rising Edge
PARITY_EN: Disabled
FORCE CALIBRATION: No Calibration after intrpt

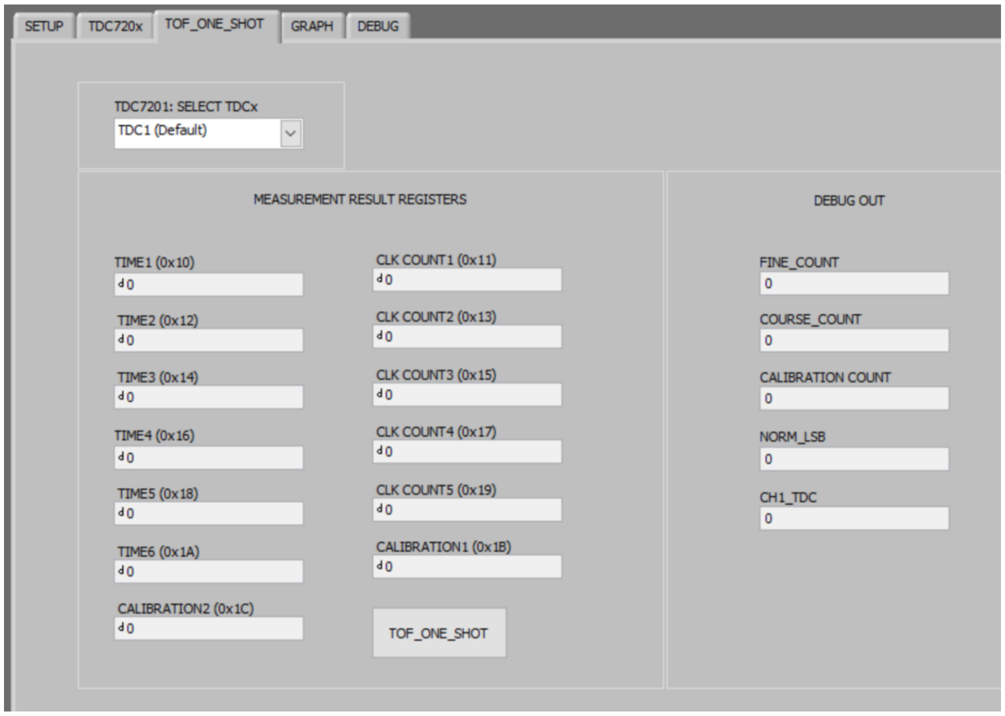
CONFIG2 (0x01)
NUMBER OF STOPS: Single
AVERAGING CYCLES: 1 Meas Cycle
CALIBRATION 2 PERIODS: 10 Clock Periods

INTERRUPT STATUS (0x02)
NEW_MEAS_INT: Interrupt Not Detected
COARSE CNTR OVERFLOW INT: Overflow Not Detected
CLOCK CNTR OVERFLOW INT: Overflow Not Detected
MEASUREMENT STARTED FLAG: Measurement Not Started
MEASUREMENT COMPLTD FLAG: Measurement Incomplete

INTERRUPT MASK (0x03)
NEW_MEAS_MASK: Interrupt Disabled
COARSE_CNTR_OVERFLOW_MASK: Interrupt Disabled
CLOCK_CNTR_OVERFLOW_MASK: Interrupt Disabled

COARSE CNTR OV_H (0x04): x FF
COARSE CNTR OV_L (0x05): x FF
CLOCK CNTR STOP MASK_H (0x08): x 00
CLOCK CNTR STOP MASK_L (0x09): x 00
CLOCK CNTR OV_H (0x06): x FF
CLOCK CNTR OV_L (0x07): x FF

Figure 2.23: TDC EVM GUI configuration window

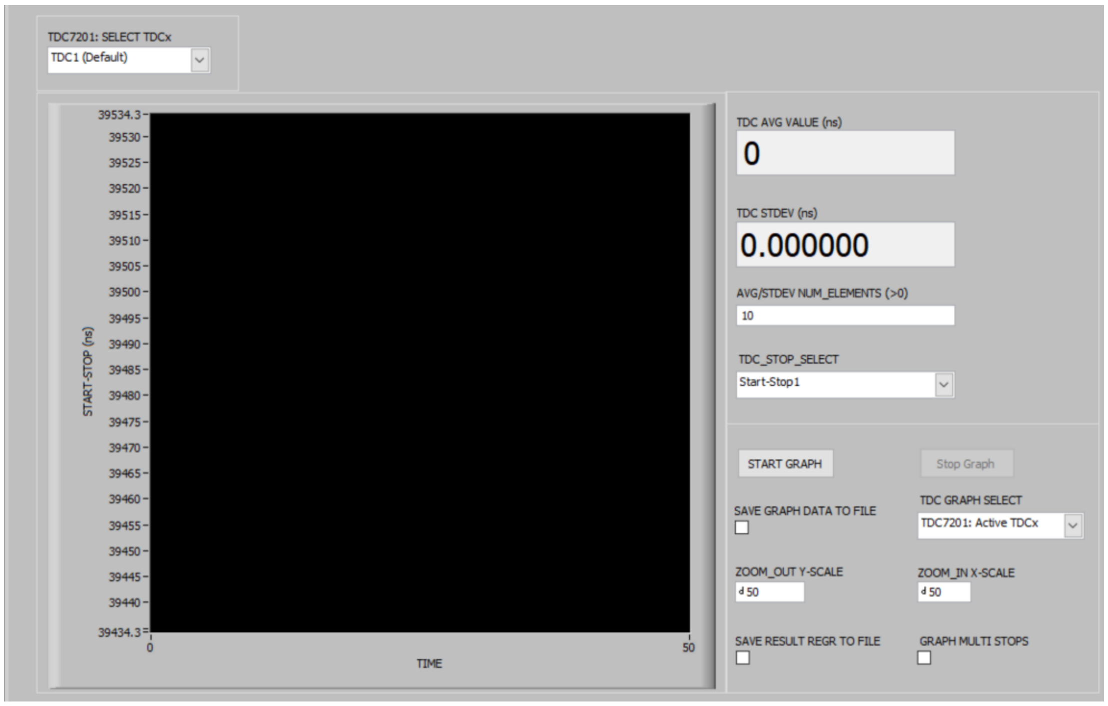


TDC7201: SELECT TDCx: TDC1 (Default)

MEASUREMENT RESULT REGISTERS
TIME1 (0x10): d 0
TIME2 (0x12): d 0
TIME3 (0x14): d 0
TIME4 (0x16): d 0
TIME5 (0x18): d 0
TIME6 (0x1A): d 0
CALIBRATION2 (0x1C): d 0
CLK COUNT1 (0x11): d 0
CLK COUNT2 (0x13): d 0
CLK COUNT3 (0x15): d 0
CLK COUNT4 (0x17): d 0
CLK COUNT5 (0x19): d 0
CALIBRATION1 (0x1B): d 0

DEBUG OUT
FINE_COUNT: 0
COURSE_COUNT: 0
CALIBRATION COUNT: 0
NORM_LSB: 0
CH1_TDC: 0

Figure 2.24: TDC EVM one shot measurement values for time of flight recording



TDC7201: SELECT TDCx: TDC1 (Default)

TDC AVG VALUE (ns): 0

TDC STDEV (ns): 0.000000

AVG/STDEV NUM_ELEMENTS (>0): 10

TDC_STOP_SELECT: Start-Stop1

SAVE GRAPH DATA TO FILE: ☐

TDC GRAPH SELECT: TDC7201: Active TDCx

ZOOM_OUT Y-SCALE: 50

ZOOM_IN X-SCALE: 50

SAVE RESULT REGR TO FILE: ☐

GRAPH MULTI STOPS: ☐

Figure 2.25: TDC EVM graph window displaying the Start-Stop1 timing values captured

The schematic of the EVM is shown in Figure 2.26.

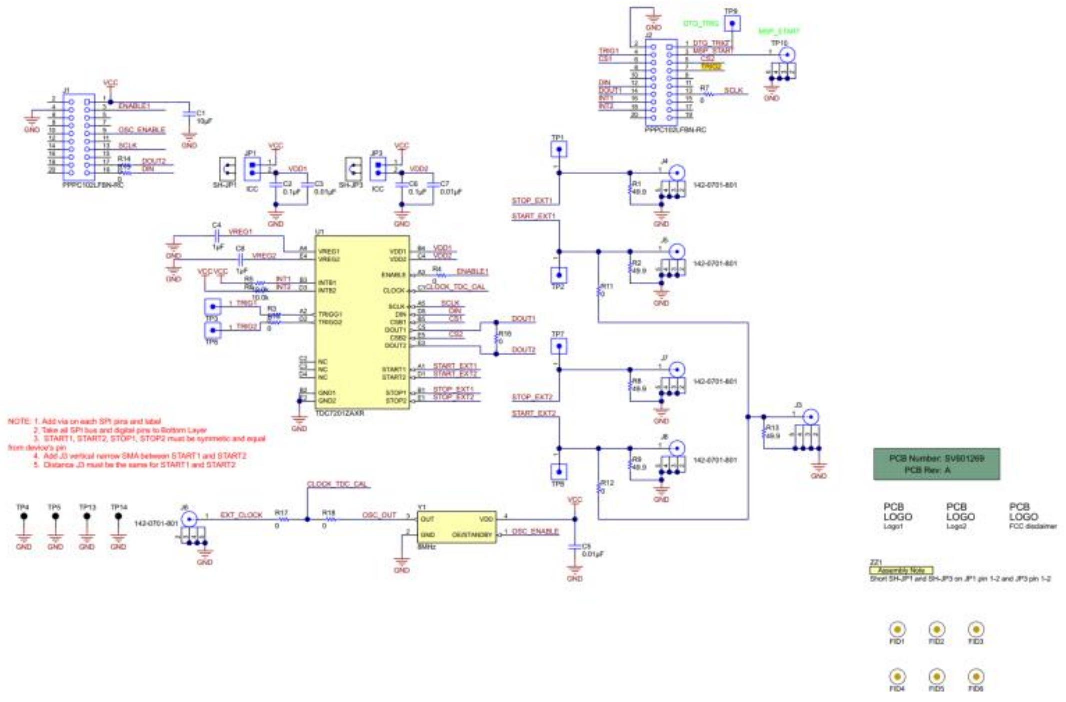


Figure 2.26: TDC EVM schematic for board

2.4 Machine Learning Techniques

2.4.1 K-Nearest Neighbor Algorithm

K-nearest neighbor is a very popular algorithm in pattern recognition, used for supervised machine learning problems, especially classification. In simple words, you have clusters of data available which have already been classified and now you get a new data point which needs to be assigned to a cluster, and that's what the KNN classifier performs. The input to the classifier is the data clusters and new data point, whereas the output of the algorithm is the cluster class that the data point is assigned to. This can be thought of as training the classifier by allowing it to learn the clusters beforehand and then performing testing. Another input or initialization step in the algorithm the is the value for 'k' which is the number of neighbors that the new data point would be checked against for classification. An important thing to note here is that 'k' must be an odd number because if it is even and the algorithm may detect equal possibility of assignment to all the clusters, due to which the algorithm would fail to classify the data point to any cluster. The classification uses the property of comparing Euclidian distances of the new data point to its 'k' nearest neighbors and whichever neighbor has the least distance is utilized to assign its cluster to the new data point i.e. the point is assigned to that cluster. Altman was the pioneer to introduce the idea of nearest neighbor classifier kernel for nonparametric regression in 1992 [53]. This algorithm is extremely powerful for detecting outliers in large datasets as shown in [54], [55], [56].

Algorithm:

1. Input the clustered data with each dataset having its labelled class
2. Initialize the 'k' value for number of nearest neighbors
3. Initialize a for loop to iterate through all the data points in all clusters

a. Calculate the Euclidian distance of the test data point with all the training data points

b. Sort these in an ascending order of distance values calculated, while keeping track of the cluster too

c. Extract the 1st k values and check the cluster class of these values

d. Assign the new data point to the most frequent class obtained

One of the shortcomings of this algorithm is that if one of our data clusters tends to be skewed then it would create a bias by increasing the density of data points in that particular region and classify a new data point into its own cluster thereby increasing the skewness [57]. Hence an alternative approach was to use a weighted classification approach. In this approach each cluster class value for the 'k' nearest clusters, is multiplied by a weight that is inversely proportional to the distance between that cluster's data point and test data point. A score is then calculated for the final frequency value of that cluster and whichever cluster has the highest score is then assigned to the test data [58].

A simple example of KNN is shown in Figure 2.27.

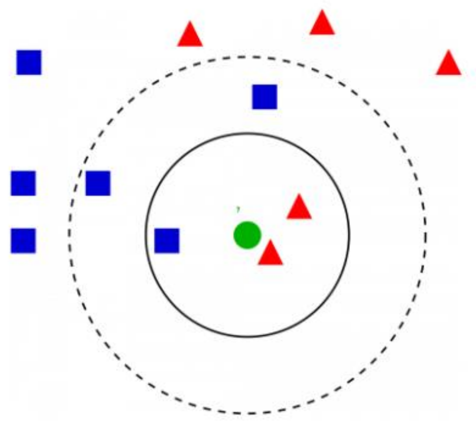


Figure 2.27: Classification example for data point(green) using 3 neighbors and 5 neighbors

It can be seen in the above example that the green dot would be classified in the red cluster if we choose ‘k’ = 3, and if ‘k’ = 5 then the green dot gets classified in the blue cluster. This shows that the choice of ‘k’ is an important parameter which must be chosen carefully. This falls into the category of hyper-parameter tuning, a very complex area of data science and machine learning which is based more on heuristic approach.

In general, higher the ‘k’ value lesser is the effect of noisy data in the system as it leads to more averaging leading to a more deterministic classification [59]. If the clustered data has outliers or noisy data, it can greatly affect the accuracy of the KNN algorithm. In general, we plot curves for training error and testing error to see what is an optimal value of ‘k’. One of the blogs [60] shows how to find the optimal ‘k’ value. The error curves from the blog are as shown in Figure 2.28 and Figure 2.29.

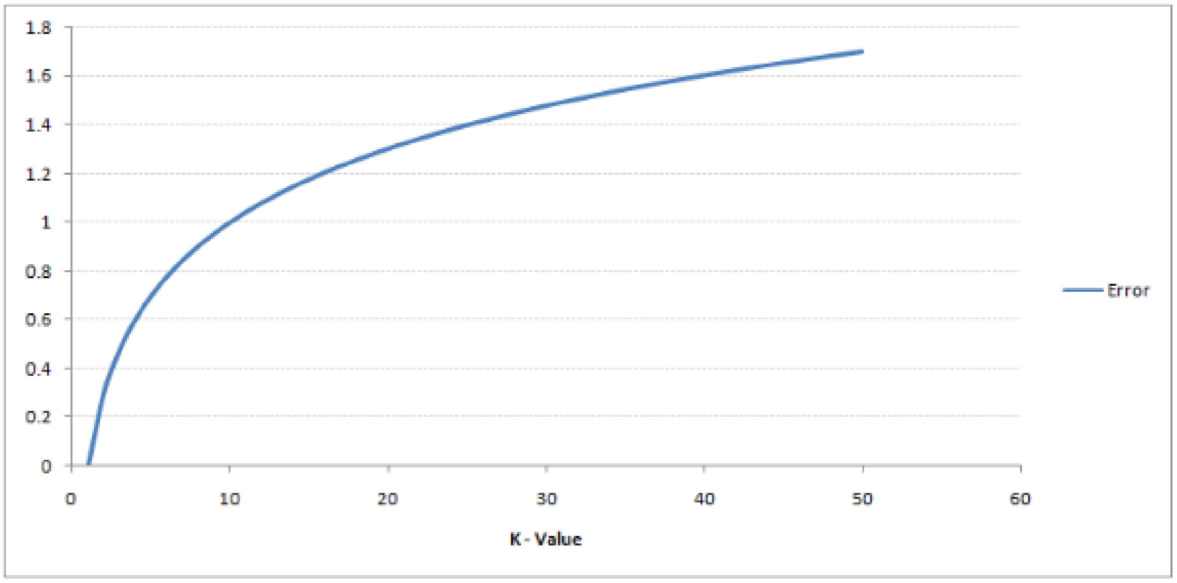


Figure 2.28: Training error for KNN algorithm vs ‘k’ value

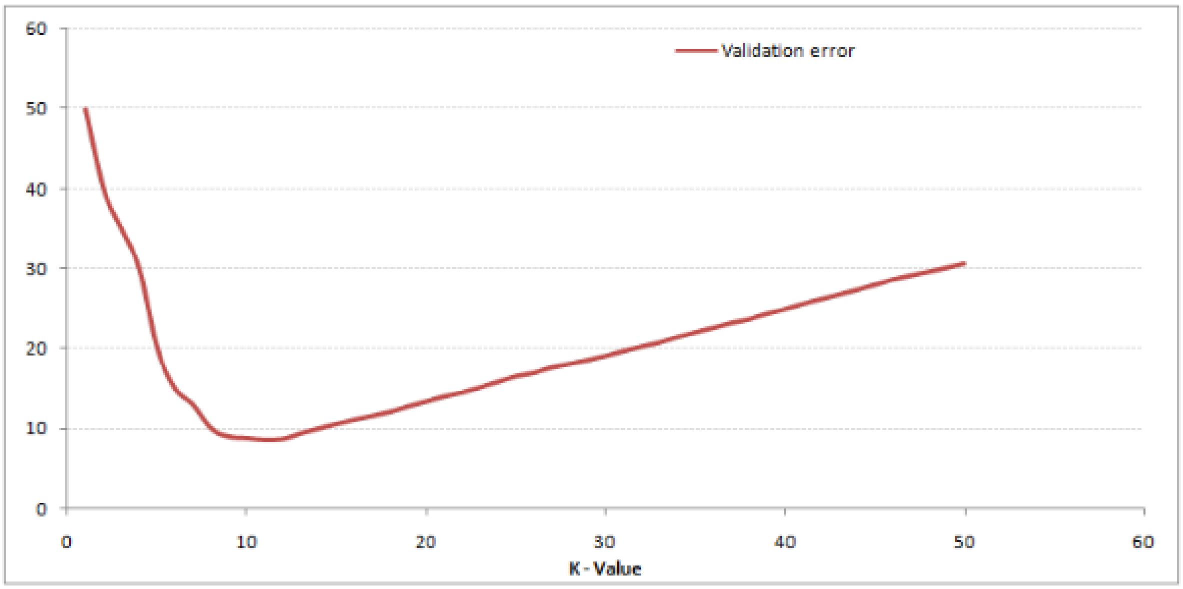


Figure 2.29: Validation error for KNN algorithm vs ‘k’ value

It can be seen as we increase ‘k’, the training error keeps increasing suggesting that we are under-fitting our classifier because as we increase ‘k’ our data points have reduced in comparison to our model features, whereas our validation error reduces to a minimum and then increases again meaning that initially we over-fit our classifier model and then go on to perform under-fitting. Thus, it is important to evaluate our classifier with a number of ‘k’ values which will give different results based on the data or features we have as inputs to the classifier.

There are different feature scaling and feature selection techniques that can be used to improve the classifier’s performance. We will however not be going into more details, as they are most widely used for computer vision applications.

In conclusion, KNN is a feature similarity-based classification algorithm which can work with classifying non-linear data too. It is not a model based predictive algorithm rather just distance based. It makes no assumptions about the data in general and is highly accurate. Although it is computationally expensive and has a high memory requirement to storing of all training data, yet it is a simple algorithm to understand and can be used for both classification and regression. It is sensitive to noise in the data hence one may have to perform feature scaling and dimensionality reduction of the data to achieve better classifier performance. [61]

2.4.2 Neural Networks

Artificial Neural Networks is a framework that models the working of a human brain. In essence a neural network learns to identify features based on the input data, independently. It consists of nodes called neurons in a layered fashion. Each neuron can communicate to the next one and through a weighted signal transmission either activating the cell or keeping it in a deactivated state. Once the input signal to the network progresses completely through the framework, the weights for every neuron need to be updated based on how closely the network output matches the expected output. The updating of weights is based on how the error from the last layer of the network is back propagated to the 1st layer. The network has to be trained with a set of data whose inputs and outputs are known and the network learns the weights for every neuron in the system. Once you have trained your neural network the next step is to test it with some new data and check how closely the network can predict your output. This is a high-level explanation of the neural network algorithm. Such a neural network is also called a MLP (multi-layer perceptron) and is represented as shown in Figure 2.30 [62]. Simplest model of a neural network is shown in Figure 2.31, in which a neuron is modelled as a logistic unit.

Logistic regression is the basic building block of neural networks. The output of this unit is a linearly weighted sum of inputs which are passed through an activation function, the sigmoid function in this case. The output lies between 0 and 1. We then compute the cost function for this unit using Equation 3, where m is the total number of training input datasets. The cost function is computed using the log-loss function and this cost function must be minimized to calculate the weights from every input neuron to the output neuron. Gradient descent method is the most popular cost function optimizer [63].

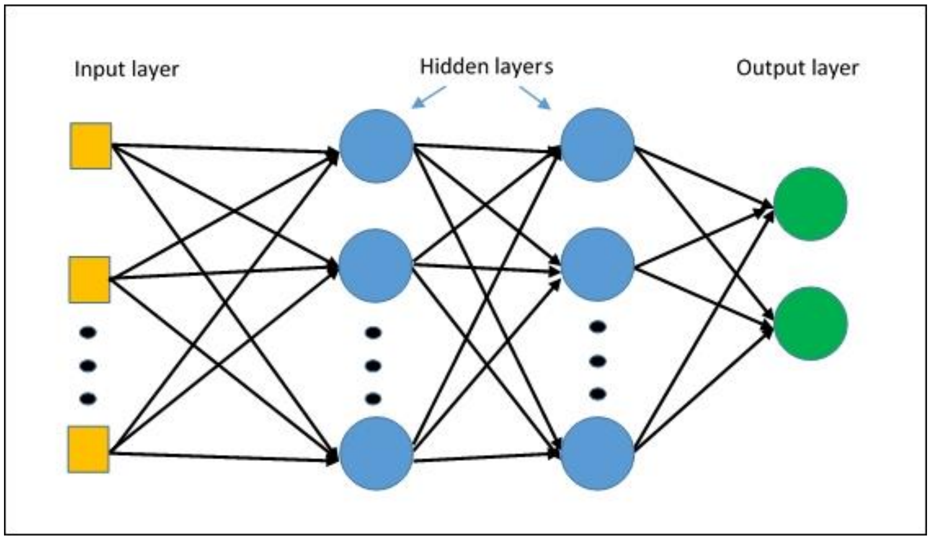


Figure 2.30: Multi-layer perceptron network with an input layer, 2 hidden layers and an output layer

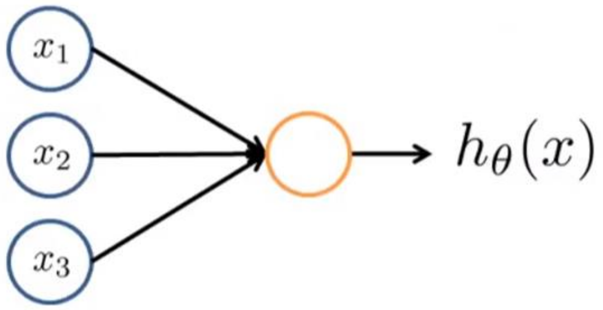


Figure 2.31: Simplest example of neural network with 3 input neurons and 1 output neuron whose value is between 0 and 1

$$z(x) = \theta^T x \tag{1}$$

$$h_\theta(x) = \frac{1}{1 + e^{-z}} \tag{2}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[y\big(\log\big(h_\theta(x)\big)\big) - (1 - y)\log(1 - h_\theta(x)] \tag{3}$$

Figure 2.32 shows a general representation of a neural network architecture with 3 input nodes, 1 hidden layer having 3 neurons and 1 output layer with a single neuron.

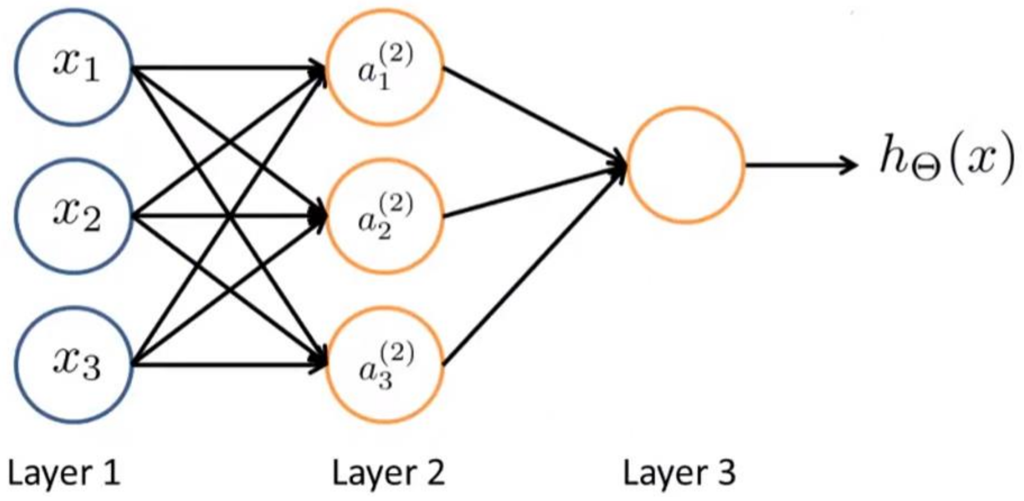


Figure 2.32: Single class neural network with 1 hidden layer

An important point to note is that although the network can learn non-linear functions, building a complex network may lead to overfitting the data thereby having a higher variance in the model. At the same time if we use less hidden layers with a smaller number of nodes in every layer it may lead to under-fitting the data thereby resulting in a biased model. In both the cases the performance of our network will be affected and prediction accuracy would decrease. In order to achieve good results from our network we must tune these parameters keeping in mind the dataset we work with and thereby play with the number of hidden layers, number of nodes in every hidden layer, characteristics of the activation function being used and lastly the cost function optimizer being used.

As shown in Equations 1 and 2, the neural network uses the nonlinear activation function on the weighted sum of inputs to perform a step called forward propagation. Now in order to learn the weights of the network for our given data we need to perform a step called the back propagation.

Once we have our predicted output we compute the error with the expected output for the output layer, L, as shown in Equation 4. This error is then propagated back into the subsequent hidden layers using the Equation 5. These errors are related to the cost function by the Equation 6.

$$\delta^{(L)} = a^{(L)} - y^{(L)} \tag{4}$$

$$\delta^{(2)} = \left(\theta^{(2)}\right)^{T} \delta^{(3)} (a^{(2)}(1 - a^{(2)})) \tag{5}$$

$$\frac{\partial L(\theta)}{\partial \theta_{ij}^{l}} = a_j^{(l)} \delta_i^{(l+1)} \tag{6}$$

In the above equations, we have $a_i^{(j)}$is the activation of unit i in layer j, $\theta^{(j)}$is the matrix of weights that govern the mapping from layer j to j+1. Here 'a' is computed by the sigmoid activation function for unit 'i' in layer 'j'. The matrix $\theta^{(j)}$has a dimension of $s_{j+1} \times (s_j + 1)$, where s_j is number of units in layer j and $s_{j+1}$is number of units in layer j+1 [64].

The weights can now be learned using the gradient descent approach. We will be performing multiclass classification rather than a binary classification as shown in Figure 2.32, as we have multiple output nodes in our system for classification. A piece of research [18] also points that it is advantageous to use one classifier per class than a multi-class classifier for better computational efficiency and interpretability which can be traced back to [65]. However, we choose to employ a multi-class classifier since computational power is not a limitation for the scope of thesis.

Chapter 3

3.1 Process

In this chapter we will be revisiting how an intrusion is performed in the CAN bus and explain the utility of this thesis for detecting it. Next, we will be describing the equipment used to perform intrusion and validate our method. Wiring diagram will be shown for this process and we will show the process of data collection using Picoscope and TDC7201-EVM. Lastly, we will revisit the thesis goal in terms of process description for intrusion detection.

3.2 What is meant by intrusion in CAN bus? (revisited)

Let's say we have 2 nodes on a CAN bus, Node A and Node B. Node A has an identifier of 01 and node B has an identifier of 10 as shown in Figure 3.1.

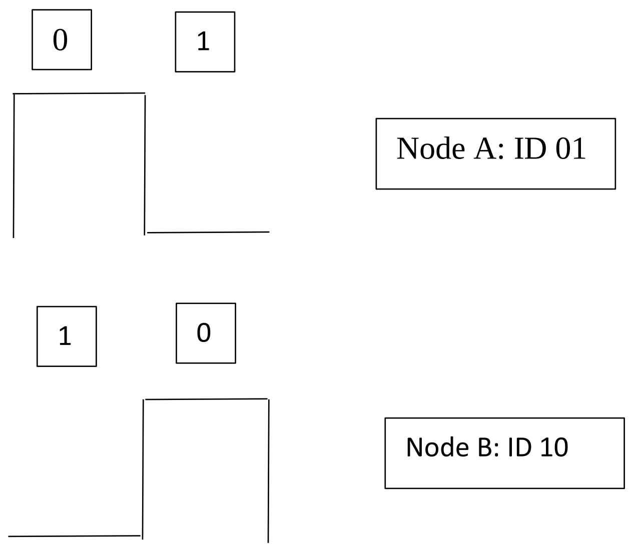


Figure 3.1: Node identifiers before intrusion

If a potential hacker intrudes node A to control node B, it essentially means that node A will be having an identifier similar to node B but the data that is carried by the message packet may be different than what should be carried and so by that means a hacker may spam the node with a set of messages he or she desires to send over the bus. This is shown in the Figure 3.2.

Now in a typical attack [66] the hacker may gain access to a disabled/inactive node and use it to spoof the bus while disabling the intruded node (sometimes called a masquerade attack) or sending messages at a different frequency than the original node (called fabrication attack). This means that node B may be an inactive node and while the broadcast rate of Node A could be 10Hz, under intrusion it might be broadcasting at 20Hz.

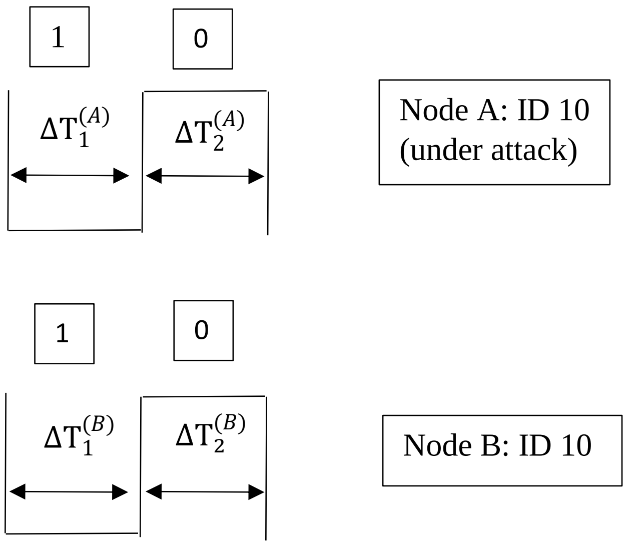


Figure 3.2: Node identifiers after intrusion

Our thesis provides a method to detect this intrusion by pointing out that the bit timing of the identifier being sent by node A ($\Delta T_1^{(A)}$ or $\Delta T_2^{(A)}$ or $\Delta T_1^{(A)} + \Delta T_2^{(A)}$) is different as compared to node B (($\Delta T_1^{(B)}$ or $\Delta T_2^{(B)}$ or $\Delta T_1^{(B)} + \Delta T_2^{(B)}$) due to manufacturing inconsistencies of the clocks of the nodes producing the bits. This means that if we can measure the bit timings of a node's identifier, we can fingerprint a node with this timing value characteristic, and when a node is under attack (Node A in our case) sending the identifier of a different node (Node B in our case), which the bus doesn’t recognize to be the original node itself (Node A) and rather the node whose identifier is being transmitted (Node B), will have an identifier bit timing of its own (Node A) and we aim to detect it in this work using machine learning technique of k-nearest neighbors algorithm. This essentially means if we plot these timing values in the form of a histogram we can expect something as shown in Figure 3.3, which can then be classified using KNN algorithm and thereafter detect intrusion based on classification of new incoming bit timing value.

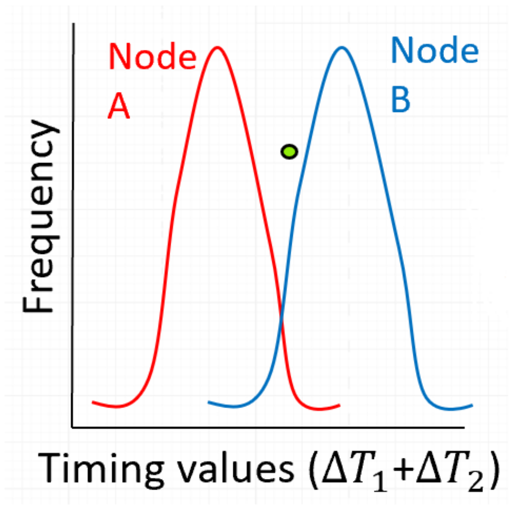


Figure 3.3: Expected histogram for bit timing values from 2 nodes A and B sending CAN messages with same identifier

As mentioned above, the goal of this thesis is to identify CAN bus nodes based on their identifier bit timings. The identifier of a node is characteristic to every node and hence is a good metric to be assessed for bit timing and fingerprint the node based on that characteristic or feature. This also allows for our metric to be consistent across all the nodes irrespective of the data they send, since in the process of intrusion, it is the identifier that must be duplicated and not the data.

3.3 Equipment

The CAN bus setup consists of 3 Maxon EPOS2 50/5 position controllers or nodes, which are used to control 3 motors using the CANopen protocol. The CAN cable length from node 1 to node 2 was nearly 37.5 cm and from node 2 to node 3 was nearly 41 cm. The actual picture of the test equipment is shown in Figure 3.4.

Each node has a 11-bit identifier of which we can configure the last 7 bits which actually serve as the identifier for characterization of the nodes. We only collect data from one node at a time, since we are interested in configuring the same identifier value to all the nodes and collect the bit timing values associated with the same identifier for all nodes, thereby giving us consistency for detecting intrusion for a given identifier value. An important thing to note is that the actual value we set or use for the identifier is irrelevant as long as we test the system for the same identifier through all nodes.

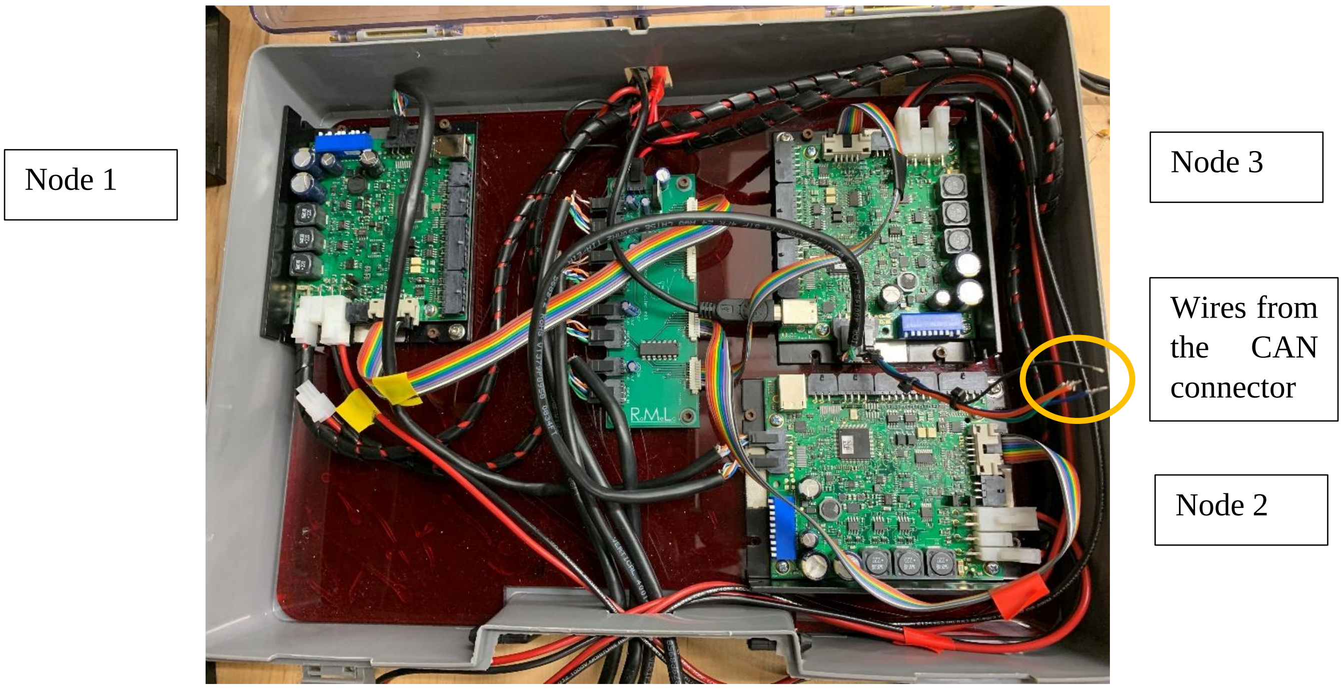


Figure 3.4: Actual test setup

This however means that if there are ‘n’ nodes on the bus then we would have to perform our validation method for each node identifier i.e. ‘n’ times. This is because although the identifier is a measure of the clock characteristic unique to every node, we have to train our KNN algorithm to work for one identifier value at a time. If we perform one test with the same identifier for all nodes we can successfully detect an intrusion for that particular identifier value only. In other words, if we were to deploy our solution for intrusion detection on a real automotive CAN bus, we would have to add this piece of hardware consisting of a TDC booster pack on MSP430 to every node on the bus, with each node also capable of running a KNN algorithm in its background on an FPGA and trained to work with that particular node’s identifier data.

Figure 3.5 shows a simple wiring diagram for the CAN bus setup we have.

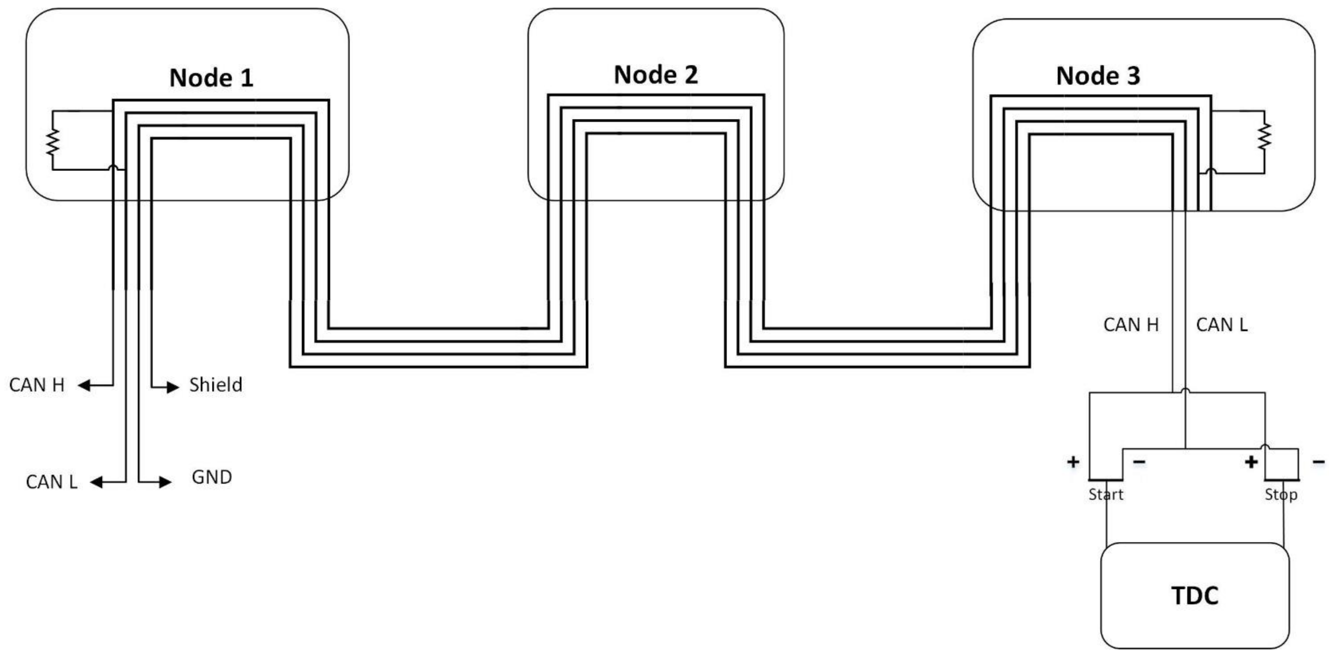


Figure 3.5: CAN bus setup wiring diagram showing 3 nodes and TDC connected to acquire the differential data from the system for each node active at a given time

We initially used a Picoscope to get an intuition of the data coming from the CAN bus and verify the correctness of it based on the protocol, results of which are shown in the next section Figure

3.11. A Picoscope 5442A from Picotech was utilized to do so. A CAN connector was created to tap into the CAN H, CAN L and Ground lines on the CAN bus. The P6060 probes were connected to CAN connector wires for tapping into the system directly. Note that the Picoscope behaves as a voltmeter with very high impedance as compared to the bus impedance of 120 ohms and hence does not affect the working of the bus in any way. None of the internal wiring of the system was tampered with during data collection either from the Picoscope or the TDC.

The CAN connector is a Molex 4pin connector as shown in Figure 3.6.

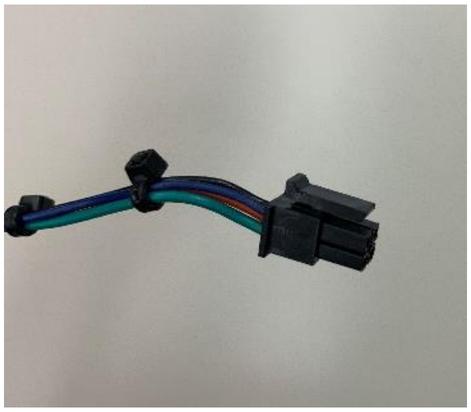


Figure 3.6: Molex 4-pin CAN connector for tapping into the bus

A more robust wiring diagram pertaining to our data collection process is as shown in Figure 3.7.

3.4 Wiring Diagram and Data Acquisition

It can be seen in the wiring diagram, Figure 3.7, that to the extreme left there are 3 Maxon motors which can be controlled using the Motor Controllers shown in the middle of the diagram through CAN bus communication protocol. The motors provide feedback using encoders. Each motor controller has 2 sets of wires going into the motor, one is the power wire and the other is the encoder wires. All the motor controllers are powered from BK Precision power supply at 12V, which in turn power the motors. At no load it was observed that nearly 0.45 A of current was drawn at 12V from the power supply.

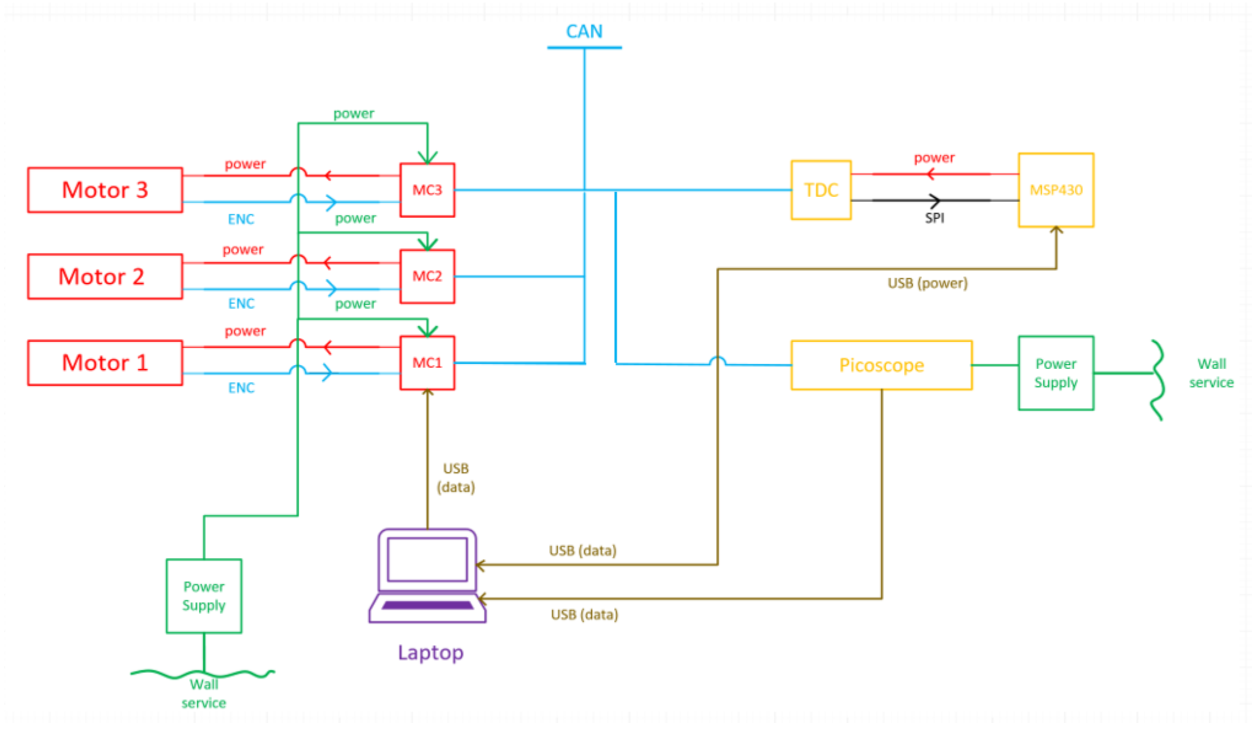


Figure 3.7: Wiring diagram for full setup

We tap the CAN bus using the Molex pin connector as mentioned before and connect the Picoscope and TDC to the CAN H and CAN L wires thereby using the differential potential, for reasons explained later in the next chapter dealing with design of experiments. Next, we know the TDC serves as a booster pack for the MSP430 which is powered using a laptop. The laptop is sometimes charged using AC wall power along with th Picoscope being powered from AC wall power supply. The MSP and Picoscope communicate with the laptop over USB communication protocol. Both the TDC and motor controllers are configured using API from their respective vendors. [67] [68]

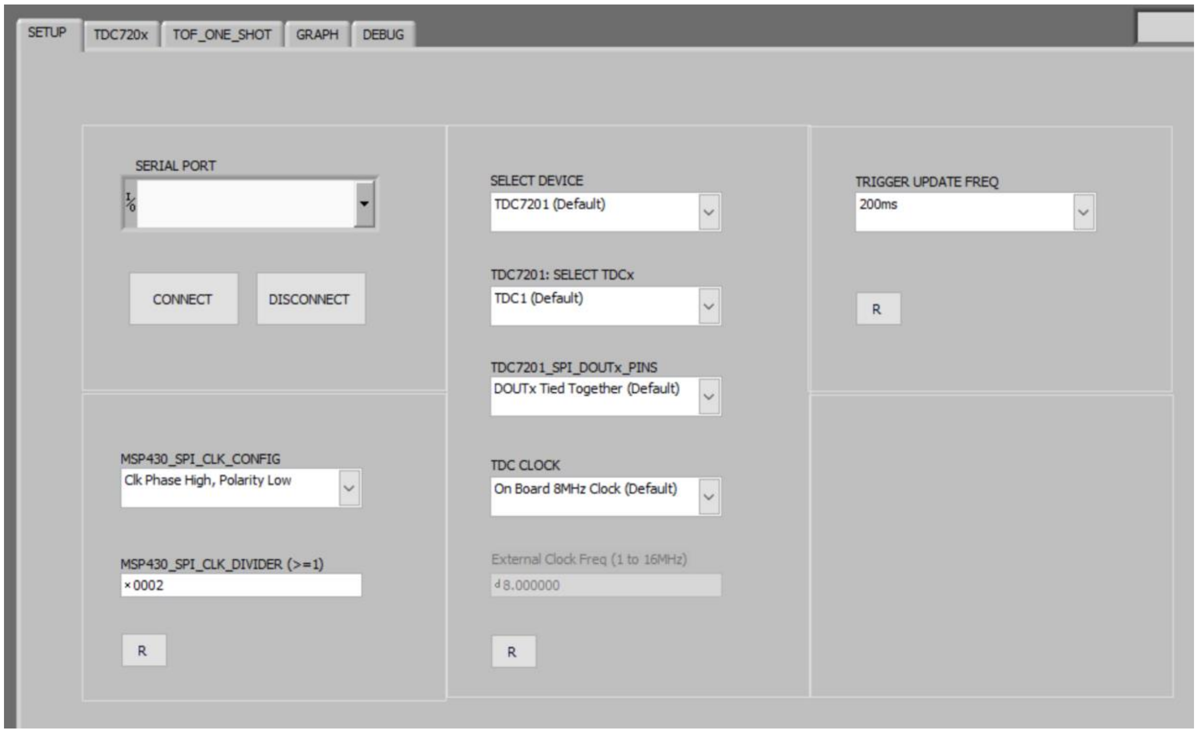
The reason for connecting both Picoscope and TDC simultaneously was done in order to understand what the data is coming in from the CAN bus and if the TDC time values agreed to the expected identifier bit timing values. The Picoscope gives us a digital representation of the bit timing interval approximately to the order of nanoseconds, as the data collection from the Picoscope was done at 125MHz with a 15-bit resolution and signal setting to be between ±5V.

TDC setup parameters:

Shown below are some of the features from the GUI for configuring the TDC in Figure 3.8 and Figure 3.9.

It can be seen that we need to connect the TDC to the MSP and in turn the MSP to the USB port

for serial communication with the GUI from Figure 3.7. Figure 3.8 shows that the SPI clock is set to a value of 4MHz, since the system clock is running at 8MHz and the SPI clock has a predivider of 2. This means that the SPI communication for retrieving data from the TDC takes place at a frequency of 4MHz, meaning that the time needed to read 1 bit of data value is 0.25 microseconds and so if we need to read 100 bits of data value from TDC registers it would take us 25 microseconds. An important thing to note here is that the number of bits to be read from the TDC depends on how many stop pulses do we need to read for, how many averaging cycles are we waiting for and 2 calibration clock periods.



SERIAL PORT

MSP430_SPI_CLK_CONFIG
Clk Phase High, Polarity Low

MSP430_SPI_CLK_DIVIDER (>=1)
x0002

SELECT DEVICE
TDC7201 (Default)

TDC7201: SELECT TDCx
TDC1 (Default)

TDC7201_SPI_DOUTx_PINS
DOUTx Tied Together (Default)

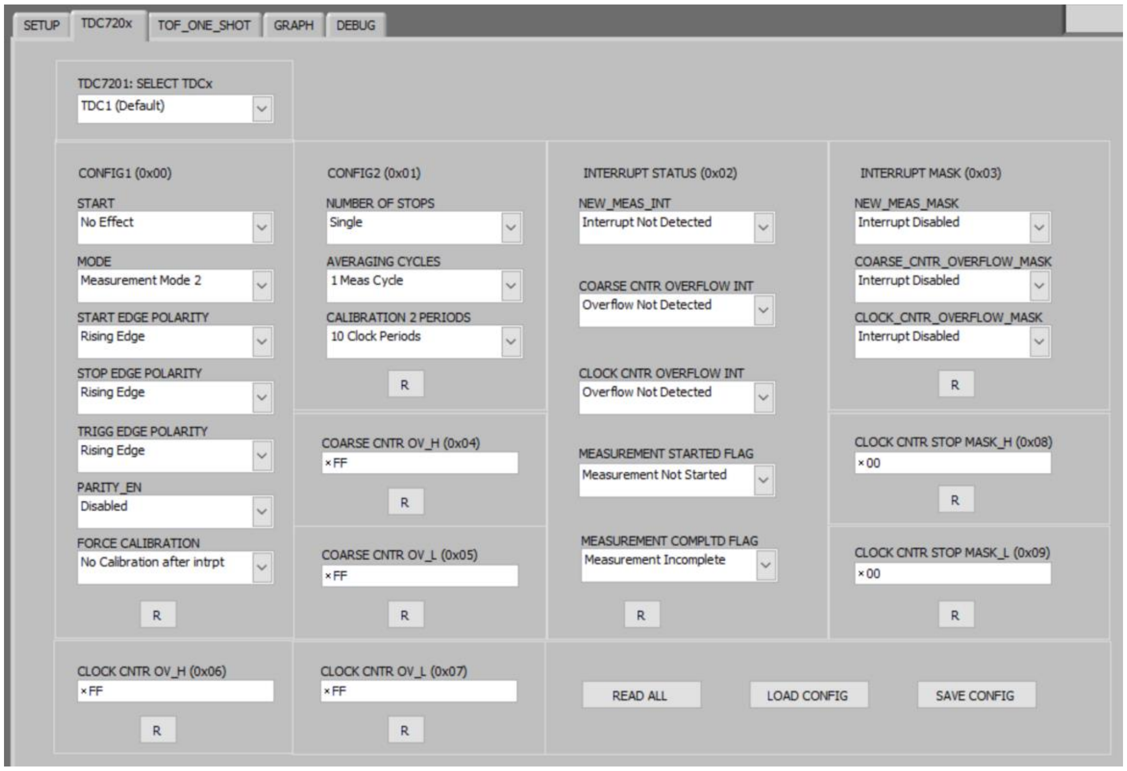
TDC CLOCK
On Board 8MHz Clock (Default)

External Clock Freq (1 to 16MHz)
d 8.000000

TRIGGER UPDATE FREQ
200ms

Figure 3.8: Initial setup of TDC using GUI

The trigger frequency is set to a default of 200ms or 5Hz meaning that the TDC will make a new measurement every 200ms. Note that this is way lower than what our application demands since we want to measure the bit timings for every bit in the identifier, a drawback we realized just recently during the final phase of this thesis but however does not impact our analysis in any major way. Figure 3.9 shows us the configuration settings we can do for the TDC.



TDC7201: SELECT TDCx
TDC1 (Default)

CONFIG1 (0x00)
START: No Effect
MODE: Measurement Mode 2
START EDGE POLARITY: Rising Edge
STOP EDGE POLARITY: Rising Edge
TRIGG EDGE POLARITY: Rising Edge
PARITY_EN: Disabled
FORCE CALIBRATION: No Calibration after intrpt

CONFIG2 (0x01)
NUMBER OF STOPS: Single
AVERAGING CYCLES: 1 Meas Cycle
CALIBRATION 2 PERIODS: 10 Clock Periods

COARSE CNTR OV_H (0x04): ×FF
COARSE CNTR OV_L (0x05): ×FF

INTERRUPT STATUS (0x02)
NEW_MEAS_INT: Interrupt Not Detected
COARSE CNTR OVERFLOW INT: Overflow Not Detected
CLOCK CNTR OVERFLOW INT: Overflow Not Detected
MEASUREMENT STARTED FLAG: Measurement Not Started
MEASUREMENT COMPLTD FLAG: Measurement Incomplete

INTERRUPT MASK (0x03)
NEW_MEAS_MASK: Interrupt Disabled
COARSE_CNTR_OVERFLOW_MASK: Interrupt Disabled
CLOCK_CNTR_OVERFLOW_MASK: Interrupt Disabled

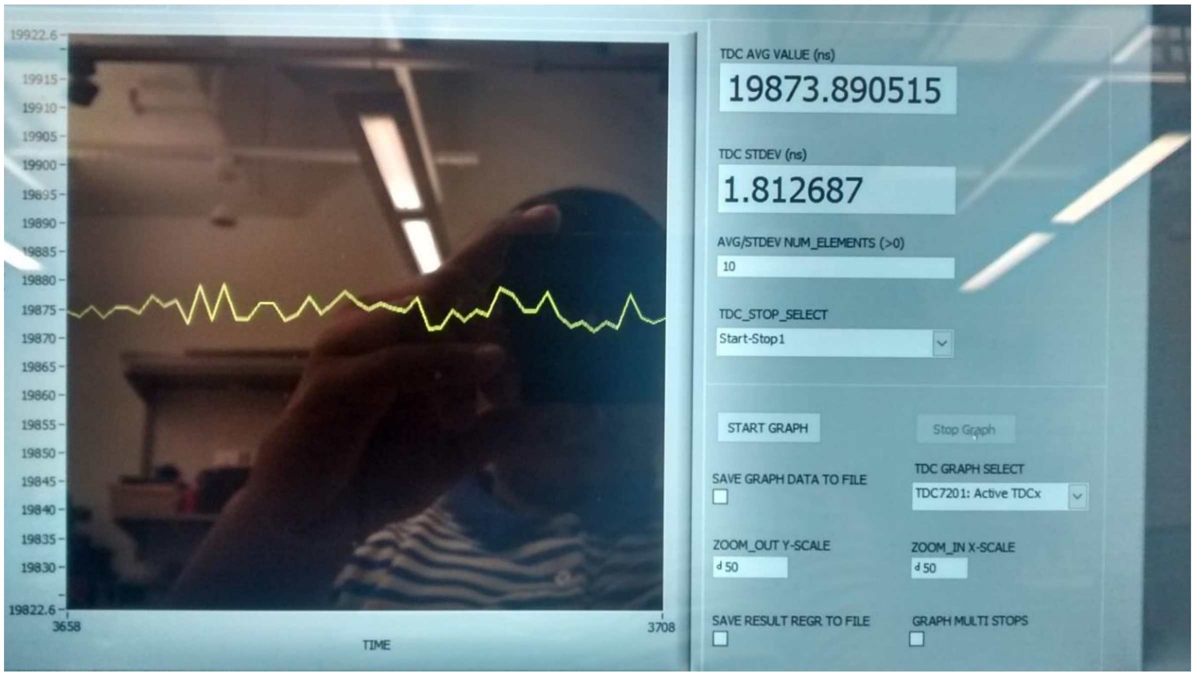
CLOCK CNTR STOP MASK_H (0x08): ×00
CLOCK CNTR STOP MASK_L (0x09): ×00

CLOCK CNTR OV_H (0x06): ×FF
CLOCK CNTR OV_L (0x07): ×FF

Figure 3.9: Configurations for edge polarity and number of stops for TDC

We can select the TDC we are working with either TDC1 or TDC2. We have had to work with both of them at one point due to removal of one of the 50-ohm resistors for TDC2, which is explained later in the thesis in Section 4.6. We must set the start bit to start a new measurement for the TDC to be active and take measurements every 200ms. The start edge polarity is chosen to be falling edge and stop edge polarity is chosen to be rising edge, although TI doesn't recommend doing that for propagation delay reasons in the ring oscillator. The reason for choosing these polarities is explained later in the thesis in Section 3.5, specifically because it aids us to have consistent measurement of 1-bit timing value. The trigger edge polarity is chosen to be the default of rising edge as it does not affect our measurement in any way. The number of stops is chosen to be single as we are interested in measuring just 1-bit timing values and not multiple bit timing values. The averaging cycle is kept to its default of 1 because since the bits coming into the CAN bus can have more than 1 successive 0'1 or 1's and hence if our identifier has consecutive 0's and 1's then averaging the timing value will completely defeat the purpose of measuring identifier bit timing. Calibration is set to 10 clock periods and this does not affect our timing measurement as it only makes our measurement more accurate by having a better calibration. Lastly, we just enable all the interrupt masks while leaving the other values to default.

Figure 3.10 shows the graph window, displaying the incoming time measurement values in a graphical form. The figure shows a measurement value of 19.87 microseconds which was our simulation experiment which will be explained in the next chapter in Section 4.3. But an important thing to note here is that the resolution of the TDC is well within the picosecond range.


TDC AVG VALUE (ns) 19873.890515

TDC STDEV (ns) 1.812687

AVG/STDEV NUM_ELEMENTS (>0) 10

TDC_STOP_SELECT Start-Stop1

TDC GRAPH SELECT TDC7201: Active TDCx

ZOOM_OUT Y-SCALE 50

ZOOM_IN X-SCALE 50

Figure 3.10: TDC measuring 19873 ns with a resolution of picoseconds

Motor Controller setup parameters:

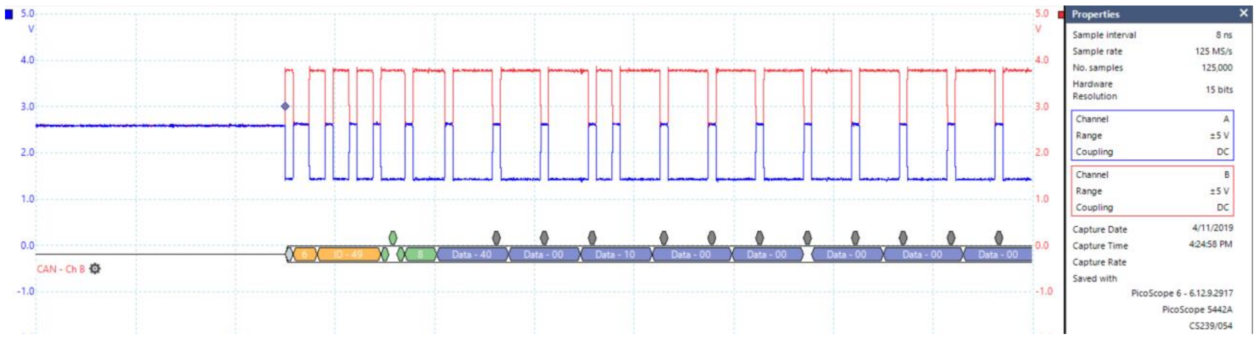
In order to configure the motor controllers, we must install the API EPOS Studio and follow the quick start guide by EPOS position controllers. We will highlight the important parameters in a tabular format that were configured for the correct startup sequence.

Interface	USB
Port	USB Φ
Baud rate	1Mbps
Timeout	500ms
Auxiliary Regulation	None
Motor Type	Maxon DC motor
Main sensor type	Incremental Encoder 1 with index (3ch)
Encoder Position	System with gear, sensor on motor
Max Application Speed	4650.5 rpm
Nominal Current	2500mA
Max Output current limit	5000mA (2*nominal current)
Thermal Constant	4s
Encoder resolution	500 pulse/turn
Position Resolution	2000 qc/turn (4*position resolution)
Max following error	2000qc
CAN identifier setting	DIP switches 1-7
CAN termination	DIP switch 9 in 'ON' position for node 1 and 3

Table 3-1: EPOS position controller configuration parameters

Some of the parameters used above are default from the startup guide, however the parameters pertaining to the motor are different because we have used the GM9000 series DC servo motor by Pittman [69],[70]. To setup the system we will set 3 different identifier values on the nodes and plug-in the CAN connector to node 3 and collect data from just the node having the identifier of interest.

Once we setup the EPOS position controllers using the above parameters and follow the startup guide we can now work on collecting CAN bus data using Picoscope and visualize it. As per procedure just have one node active at a time on the bus and collect data for that node from the Picoscope. A sample signal from the CAN bus is shown in Figure 3.11.



Sample interval	8 ns
Sample rate	125 MS/s
No. samples	125,000
Hardware Resolution	15 bits
Channel	A
Range	±5 V
Coupling	DC
Channel	B
Range	±5 V
Coupling	DC
Capture Date	4/11/2019
Capture Time	4:24:58 PM
Capture Rate	
Saved with	PicoScope 6 - 6.12.9.2917
	PicoScope 5442A
	CS239/054

Figure 3.11: Example CAN signal from EPOS controller using Picoscope confirming to the CANopen protocol

The red signal is the CAN H signal and the blue signal is the CAN L signal both referenced to CAN ground. We can see that the signals are biased at 2.5 V and agree with the CAN protocol definition of signal voltages as mentioned in Section 2.2.3, Figure 2.4. It is seen that the start bit is a dominant value of 0 followed by the function code for the identifier, as per CANopen protocol, of 4 bits in length, followed by the actual identifier value that we set, which is 0x49 in this case. The DIP switch setting for this identifier can be found by referring [71].

3.5 Thesis goal revisited: (Process description)

Based on the goal of this thesis where we are trying to characterize each node on the basis of their bit timing values for a node identifier, we want to utilize the TDC in getting time of flight values. From the working of the TDC we know that we need a start and stop pulse to measure the time interval between arrival of those pulses (Figure 2.21). In our case of the CAN bus we can just utilize one pulse for both our start and stop pulses and TI also allows for this assuming you have a delay of at least 12ns between the 2 pulses [72]. This means that we must now decide on the polarity of the edges for triggering or measuring our timing values. On the basis of the observed data we thought it was best to use the falling edge for the start pulse and then the rising edge for the stop pulse. This way it gives us consistency as most of the time interval measurements would be just 1 bit long not only in the identifier bits but also along the data bits as well giving us an advantage of having more measurements representing a node along with identifier bit timings. This also means that we measure alternate bit timing values and not successive, however any bit timing for the same node is still a representation of the clock in every sense and so this is not a major concern for now.

In Figure 3.12 the single dashed black line shows falling edge polarity for the start pulse and falling edge polarity for the stop pulse.

As we keep getting timing values from the TDC, we will store them for respective nodes and use the entire data into our KNN algorithm as the training data and testing data. We will then evaluate the accuracy of our algorithm and thereafter collect more testing data to observe the new accuracy as a representation of a fully deployed model. That is the main goal of this thesis and we will be going into more details regarding evaluating the performance of our TDC, KNN algorithm and Neural Networks in our chapter on Experimental Design and Results.

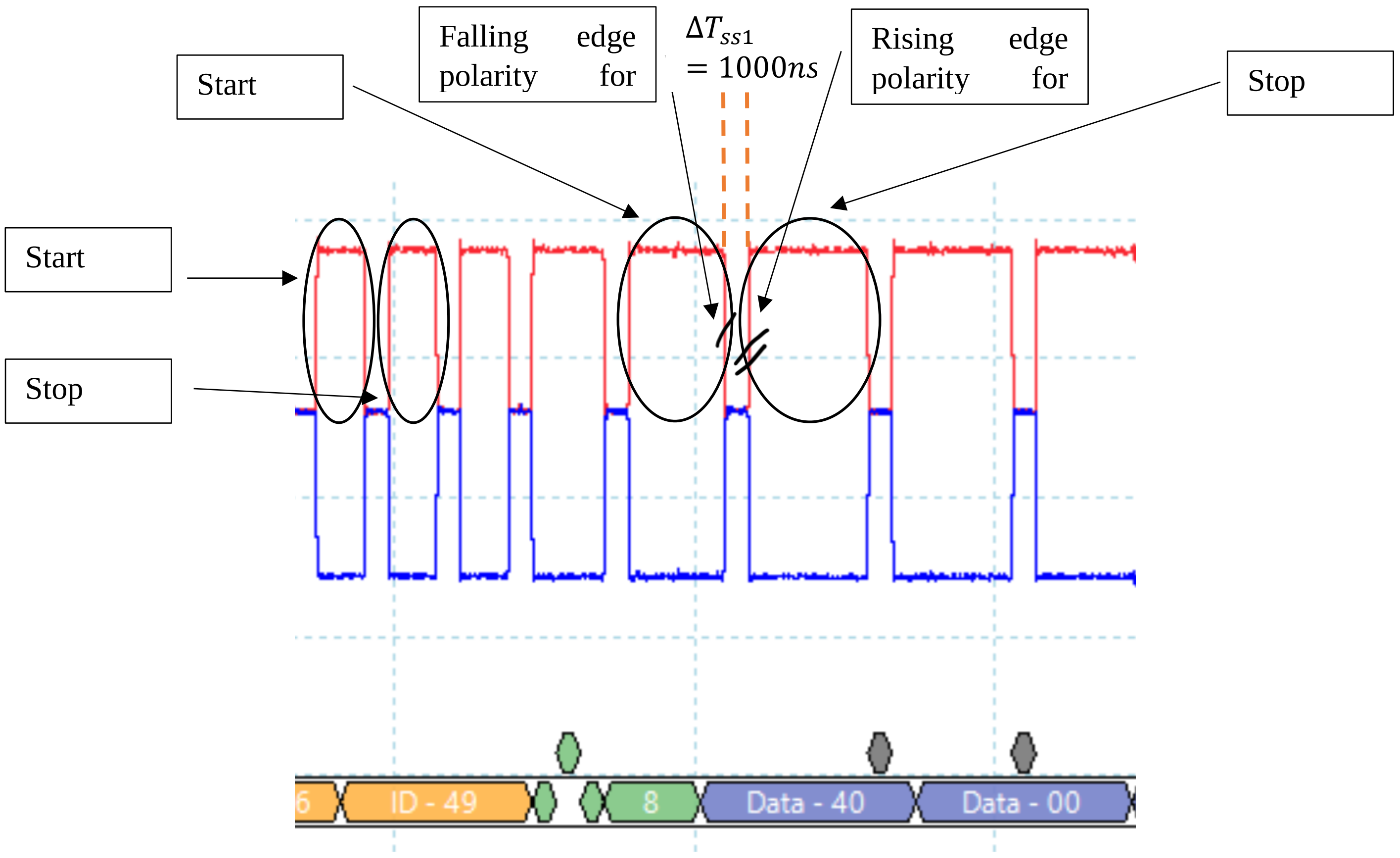


Figure 3.12: Edge polarity depiction from CAN signal for TDC start and stop pulses and expected bit timing

Chapter 4

4.1 Experimental Design

In this chapter we will explaining the design of experiments process where we will be going over evaluating our idea in a simulation experiment using PWM signals from MSP432. We will then look into setting up the CAN bus to match the requirements of the TDC. Data collection and algorithm implementation will be covered next. Lastly, we go over a few modifications done to the TDC for achieving stable and good readings.

4.2 Functionality of TDC to be verified

The first step in our experimentation process was to verify the functioning of the TDC7201-ZAX-EVM. It was essential to understand if the instrument worked as it was supposed to outlined in the manual. In order to check this functionality, a function generator was used to generate 2 PWM waves with a time interval of 19 us at a frequency of 40khz. The TDC was mounted on the MSP430 and firmware for the MSP430 was updated as per instructions in the TDC EVM manual. It was confirmed that the TDC connection to the MSP was reflected in the USB connection of the MSP to the computer in the device manager ports section. Then the connections to the TDC were made for the start and stop signals using USB-to-SMA cable as shown in Figure 4.1 from the function generator output channels. A Tektronix AFG3102 function generator was used for this purpose as shown in Figure 4.2. The Trigger pin (TP9) of the TDC was connected to the trigger input of the function generator allowing the TDC to generate a trigger for the function generator to start the PWM pulses. The expected PWM are as shown in Figure 4.3. The output signals shown are captured using an oscilloscope.

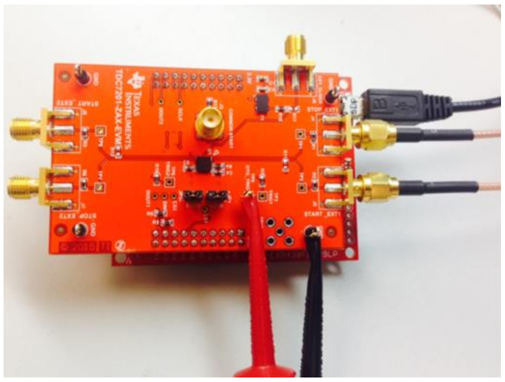


Figure 4.1: Physical connections to the TDC for start, stop and trigger pulses

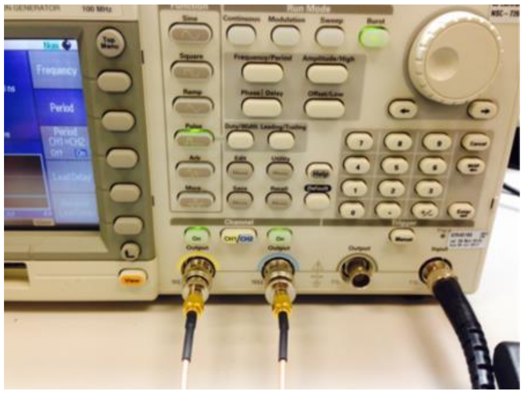


Figure 4.2: Tektronix AFG3102 function generator connections to/from the TDC

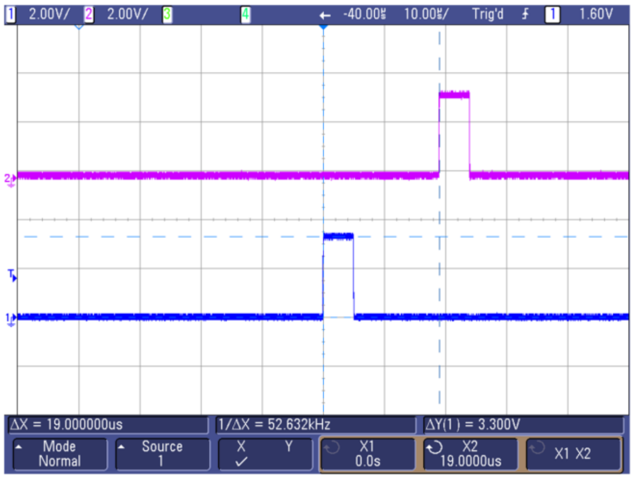
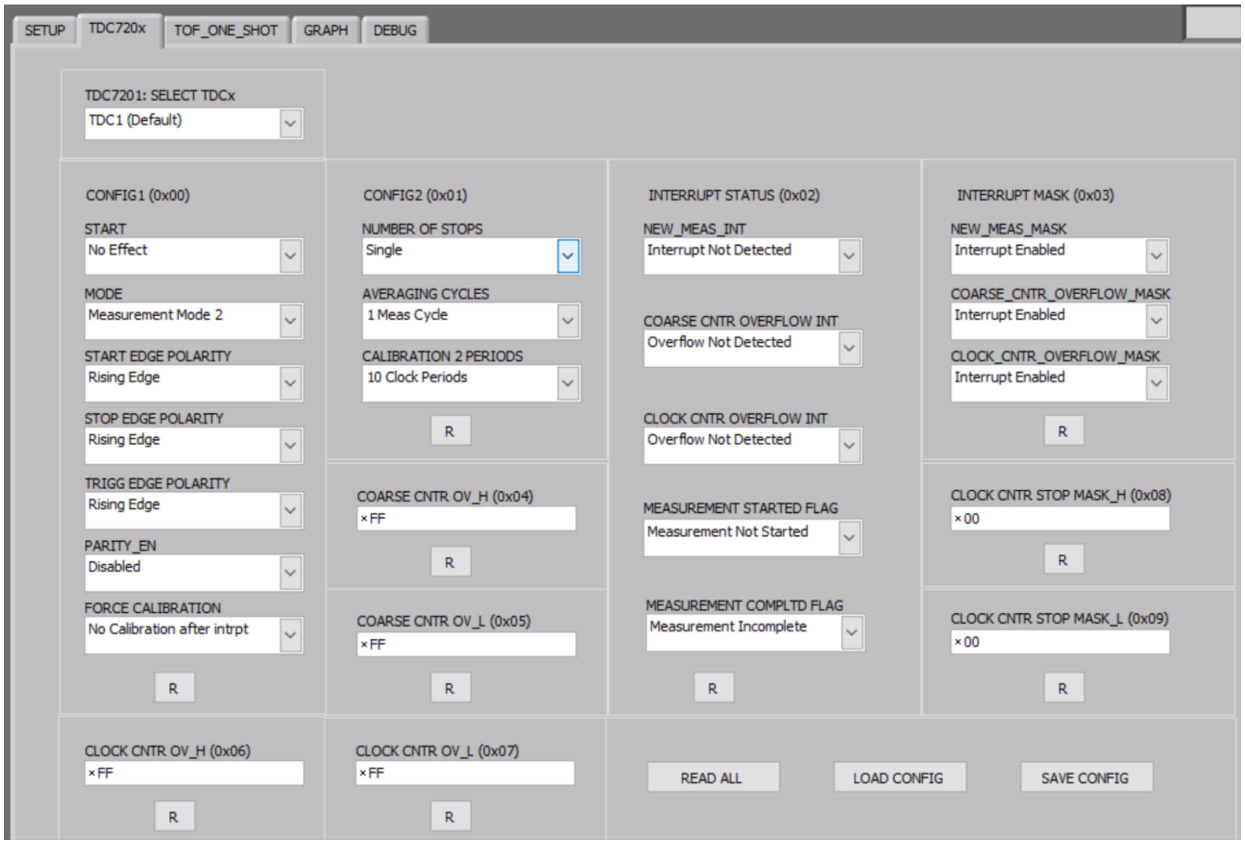


Figure 4.3: PWM signals for start and stop pulse for TDC from signal generator captured using an oscilloscope, having a delay of 19us

The TDC configuration is as shown in Figure 4.4. We follow the standard operating configuration as recommended in the user manual as shown in the Table 4-1.



TDC7201: SELECT TDCx: TDC1 (Default)

CONFIG1 (0x00)
START: No Effect
MODE: Measurement Mode 2
START EDGE POLARITY: Rising Edge
STOP EDGE POLARITY: Rising Edge
TRIGG EDGE POLARITY: Rising Edge
PARITY_EN: Disabled
FORCE CALIBRATION: No Calibration after intrpt

CONFIG2 (0x01)
NUMBER OF STOPS: Single
AVERAGING CYCLES: 1 Meas Cycle
CALIBRATION 2 PERIODS: 10 Clock Periods

COARSE CNTR OV_H (0x04): x FF
COARSE CNTR OV_L (0x05): x FF

INTERRUPT STATUS (0x02)
NEW_MEAS_INT: Interrupt Not Detected
COARSE CNTR OVERFLOW INT: Overflow Not Detected
CLOCK CNTR OVERFLOW INT: Overflow Not Detected
MEASUREMENT STARTED FLAG: Measurement Not Started
MEASUREMENT COMPLTD FLAG: Measurement Incomplete

INTERRUPT MASK (0x03)
NEW_MEAS_MASK: Interrupt Enabled
COARSE_CNTR_OVERFLOW_MASK: Interrupt Enabled
CLOCK_CNTR_OVERFLOW_MASK: Interrupt Enabled

CLOCK CNTR STOP MASK_H (0x08): x 00
CLOCK CNTR STOP MASK_L (0x09): x 00

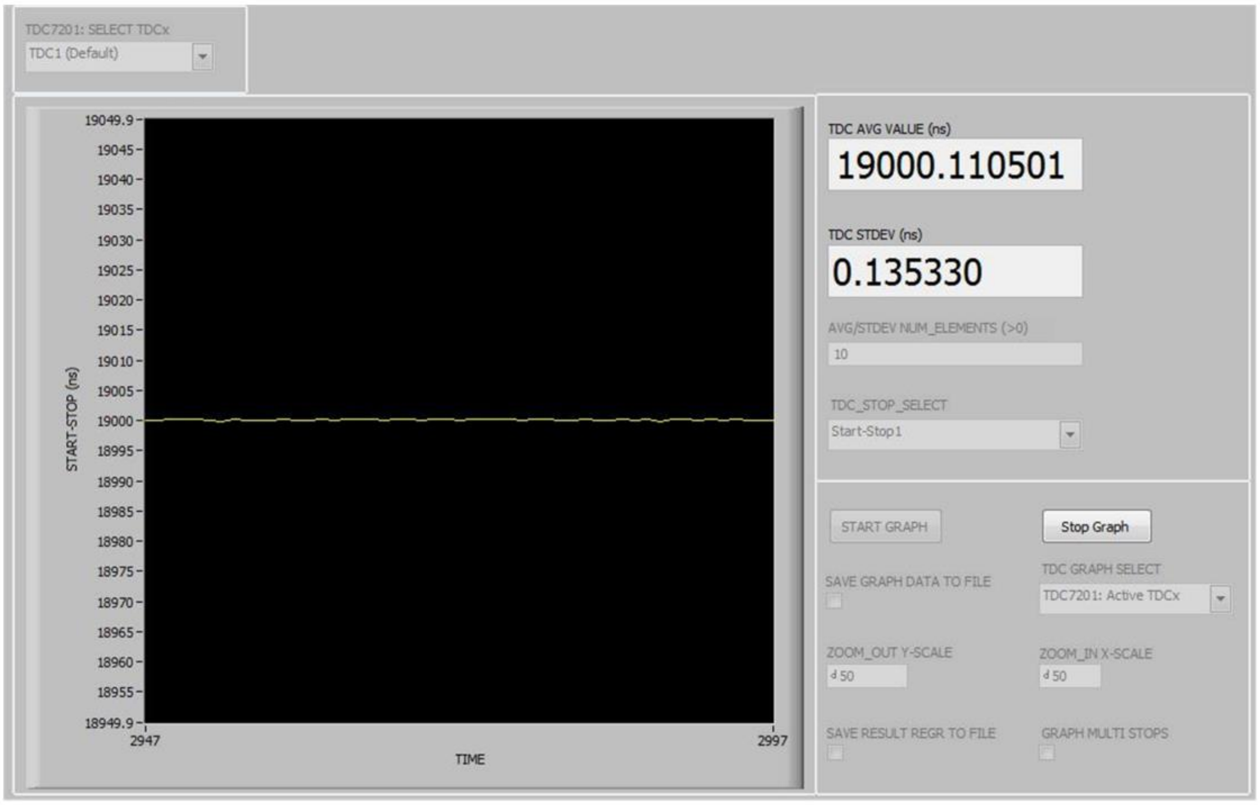
CLOCK CNTR OV_H (0x06): x FF
CLOCK CNTR OV_L (0x07): x FF

Figure 4.4: TDC GUI with setup configuration

Operating Mode	Measurement Mode 2
Start Edge polarity	Rising Edge
Stop Edge polarity	Rising Edge
Trigger Edge polarity	Rising Edge
Number of Stops	1
Averaging Cycle	1
Calibration 2 periods	10
Interrupts	All interrupts enabled

Table 4-1: Important TDC configuration parameters

The TDC time of flight results from the GUI are shown in Figure 4.5.



TDC AVG VALUE (ns): 19000.110501

TDC STDEV (ns): 0.135330

AVG/STDEV NUM_ELEMENTS (>0): 10

TDC_STOP_SELECT: Start-Stop1

Figure 4.5: GUI graph for time of flight values from function generator

The histogram for above time of flight values obtained is shown in Figure 4.6 using MATLAB.

Next, we need to design an experiment to check if our TDC can be used for CAN bus applications. Thus, we need to simulate PWM signals on a microcontroller which can be as close to a CAN bus controller identifier signal as possible, and see if we can indeed find any difference in the timing values to characterize or fingerprint the microcontrollers.

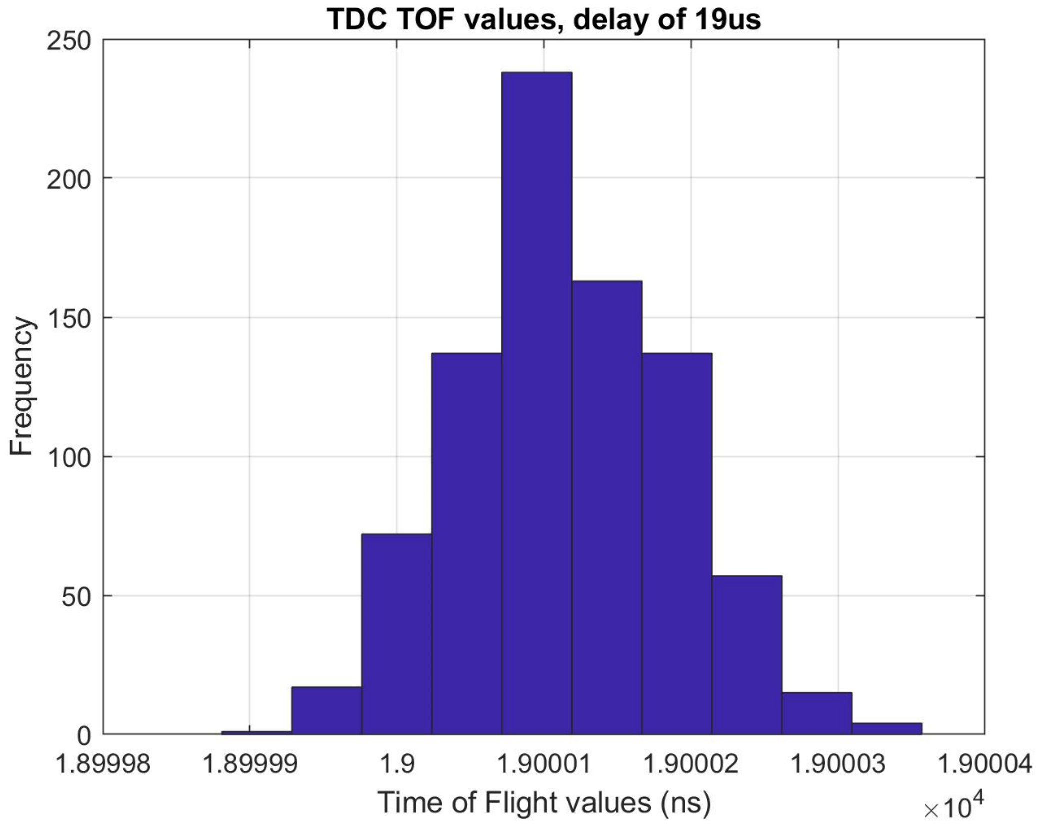


Figure 4.6: Histogram obtained for time interval between 2 PWM signals from signal generator using TDC, delay between 2 pulses is 19us

4.3 Simulating CAN identifier signal using microcontroller and checking for differences in relative timing values for fingerprinting nodes

CAN bus identifier signals consist of logical 1s and 0s transmitted up to a frequency of 1MHz. The simplest CAN bus identifier signal can be 0x01, or 0b0000001. Since the identifier is directly linked to the microcontroller clock and we are only interested in measuring the time difference between any 2 pulses or in this case time between 2 logical 0’s of an identifier, we will simulate 2 PWMs from a microcontroller and find the time intervals between the respective rising edges. This allows us to directly check the accuracy of the clock timings as the pulses are generated referencing from the microcontroller clock directly.

Now the most accurate representation of a clock timing using PWM signal is to generate these signals using clock timers on the microcontroller. In our case we will be using MSP432 as our CAN bus controller which generates PWM signals using its internal TimerA, at a frequency of 1kHz. Both the start and the stop pulse were referenced using the same TimerA for consistency of results and avoiding any timing differences that may rise from using 2 different timers running asynchronously thereby adding unwanted time delays between pulse measurements. Table 4-2 shows the important PWM characteristics used for our simulation study.

Duty cycle of start and stop PWM pulse	10% and 80% respectively
Frequency of pulses	1 kHz
Time delay between the rising edges of the pulses	~20 us

Table 4-2: Important signal characteristics to meet the requirements of TDC functioning

Figure 4.7 shows the transition from a CAN bus identifier signal to our simulated signal as a direct representation of clocks.

Based on the above parameters the PWM signals generated on the MSP432 are as shown in Figure 4.8 captured using an oscilloscope. The TDC GUI graph for time interval being measured is shown in Figure 4.9. It can be clearly seen that since the resolution of the oscilloscope is limited it says that the time interval being measured is 19.891 us, whereas our TDC which can measure to an order of picoseconds tells us that the time interval being measured is 19.899 us, a difference of nearly 8 ns which can be very critical to characterize microcontrollers based on their clock timing inaccuracies.

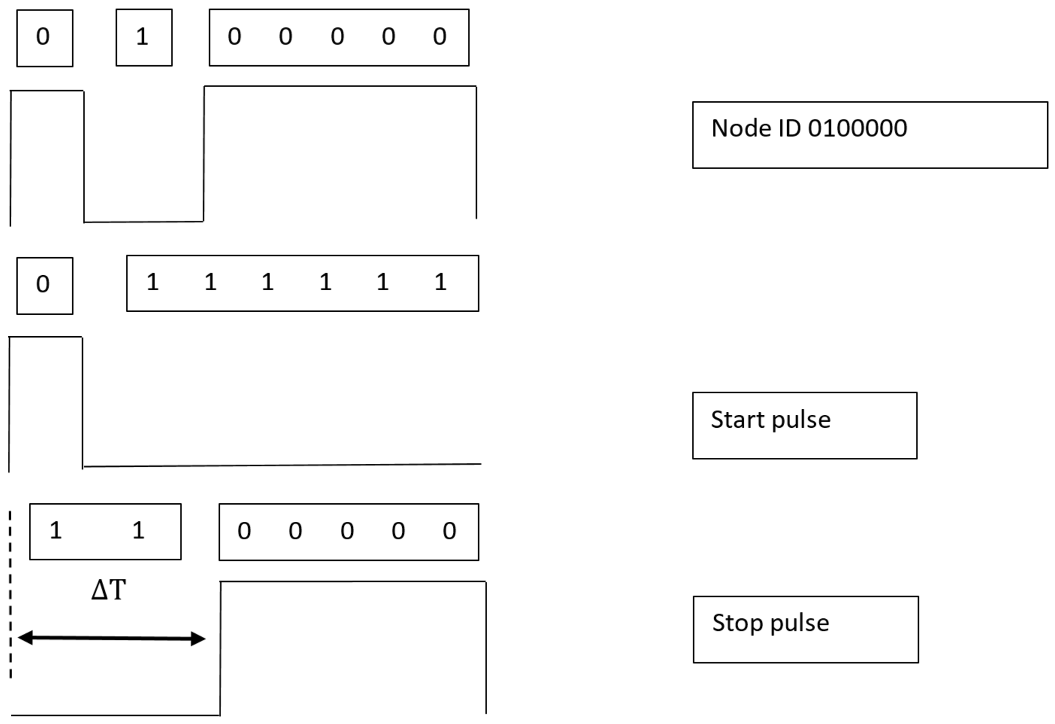


Figure 4.7: CAN bus identifier signal split into start and stop pulses

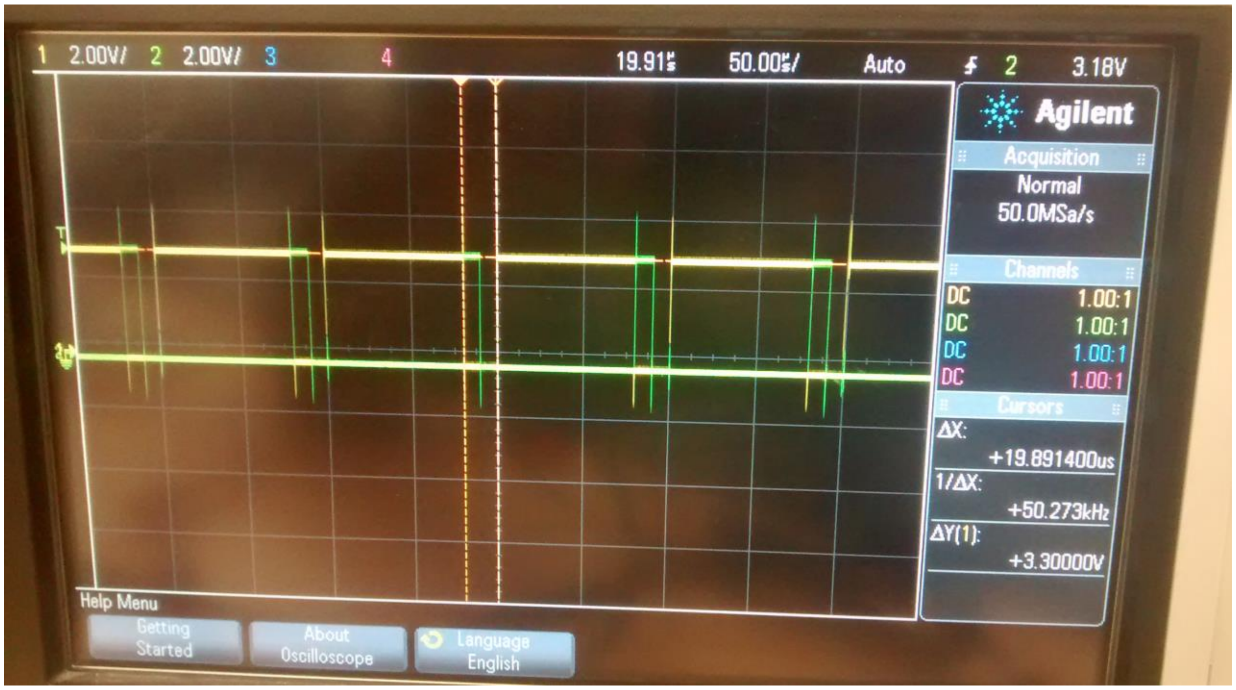
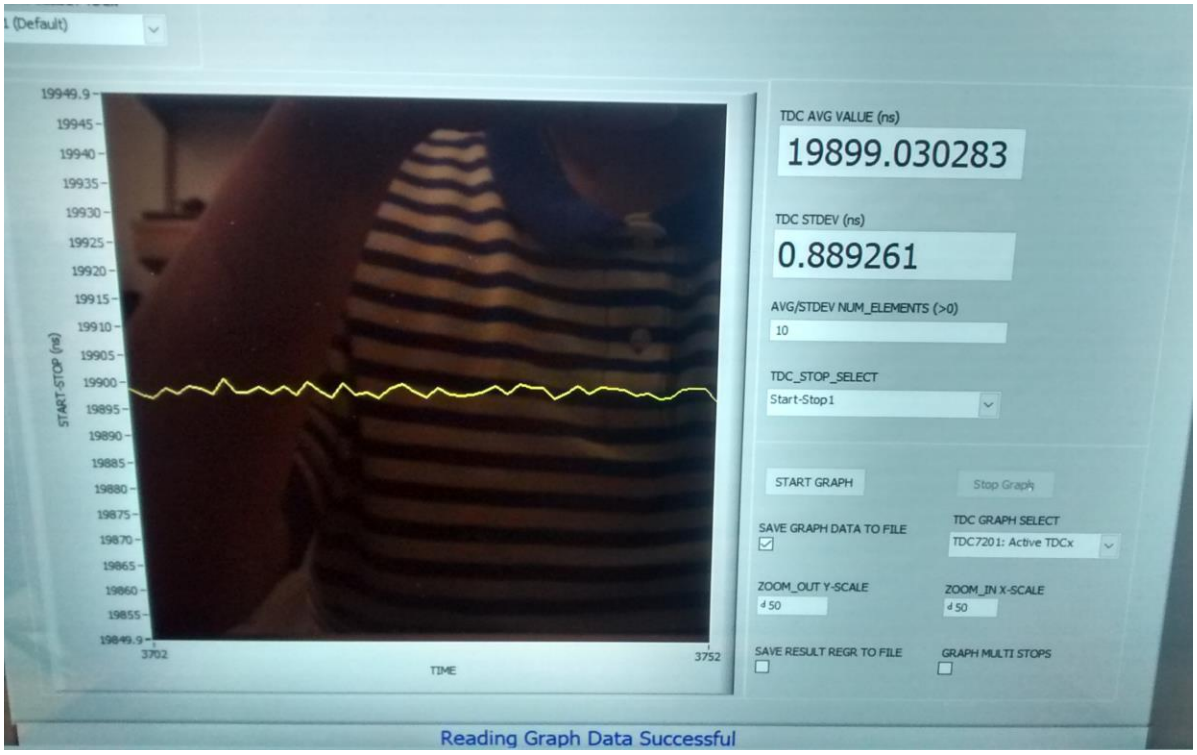


Figure 4.8: PWM signals from TimerA of MSP432 captured using an oscilloscope



TDC AVG VALUE (ns) 19899.030283

TDC STDEV (ns) 0.889261

AVG/STDEV NUM_ELEMENTS (>0) 10

TDC_STOP_SELECT Start-Stop1

SAVE GRAPH DATA TO FILE ☑

TDC GRAPH SELECT TDC7201: Active TDCx

ZOOM_OUT Y-SCALE 50

ZOOM_IN X-SCALE 50

SAVE RESULT REGR TO FILE ☐

GRAPH MULTI STOPS ☐

Reading Graph Data Successful

Figure 4.9: TDC GUI showing time of flight values for PWMs generated

We do the same experiment of measuring the time difference between these PWM signals from 6 different MSP432s and below is the graph of the timing differences for all of them based on the TDC reading, plotted as a histogram in Figure 4.10.

Now that we can see a clear demarcation between the timing intervals for all the nodes, as we expected from Figure 3.3, we can apply KNN technique to classify a new time of flight value obtained by the TDC into one of the 6 different node categories. If there is an intrusion and say node 1747 is spoofed to send the identifier of node 1834, and the time of flight value for the identifier bit noted from the TDC is classified by KNN algorithm to be as node 1747, then we would be clearly able to identify that node 1747 is hacked by node 1834. The results of our KNN classifier will be discussed in the results section of this thesis.

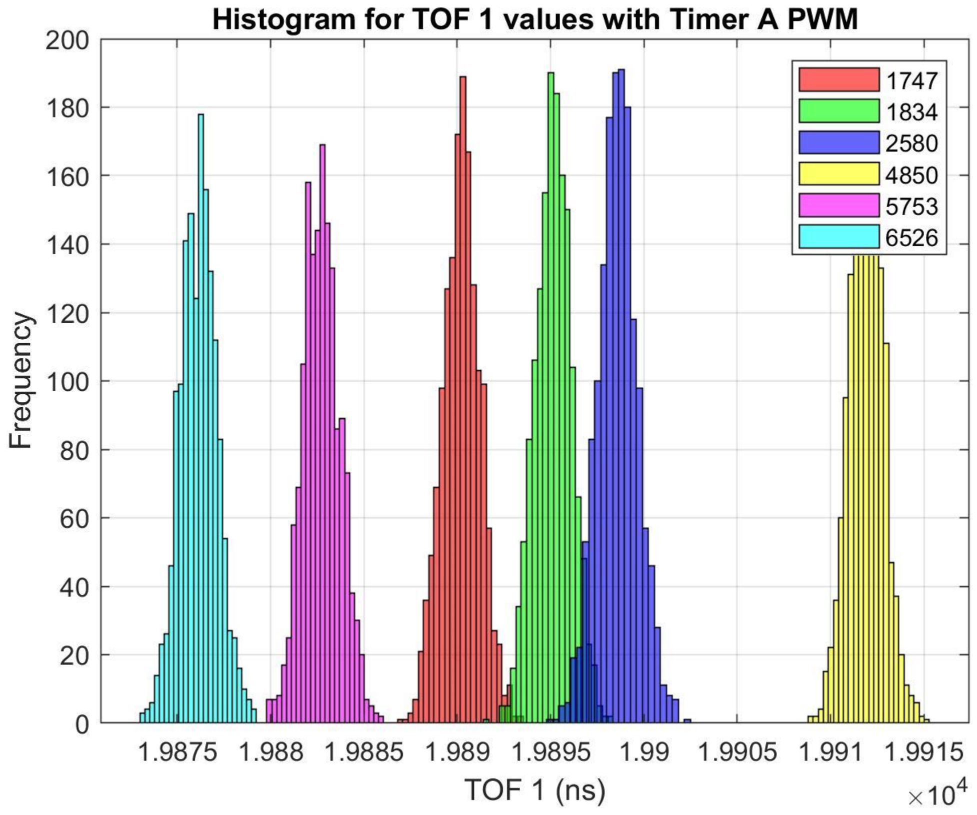


Figure 4.10: Histogram showing time intervals for PWMs observed from 6 different MSP432s using the TDC

4.4 Collecting actual CAN bus identifier bit timing data with the TDC EVM

Now once we have seen that the clocks of microcontrollers differ in their timing characteristics based on our above experiment, we can now proceed to apply the same concept in an actual CAN bus setup which was described in our hardware section, Section 3.3. The aim here is to program is the same identifier for all the 3 nodes in the system and capture the time interval between successive bits of the identifier using the TDC. Since we want to program the same identifier on all the nodes, it is quite clear that we can have only one node operating in the system at a time. The identifier can be programmed externally using dip switches as shown in Figure 4.11. Figure 4.12 shows the actual CAN bus signal from node 1 with identifier 0d73 (decimal value of 73).

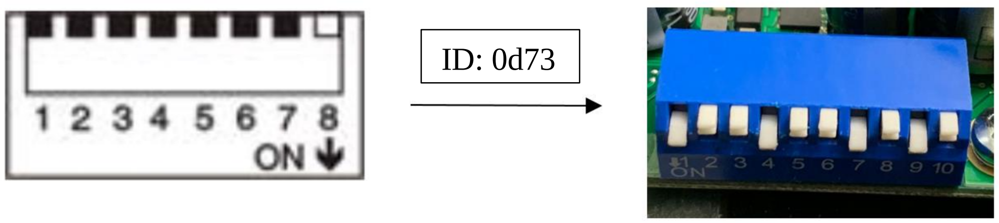


Figure 4.11: DIP switch configuration to program the identifier with ID 0d73

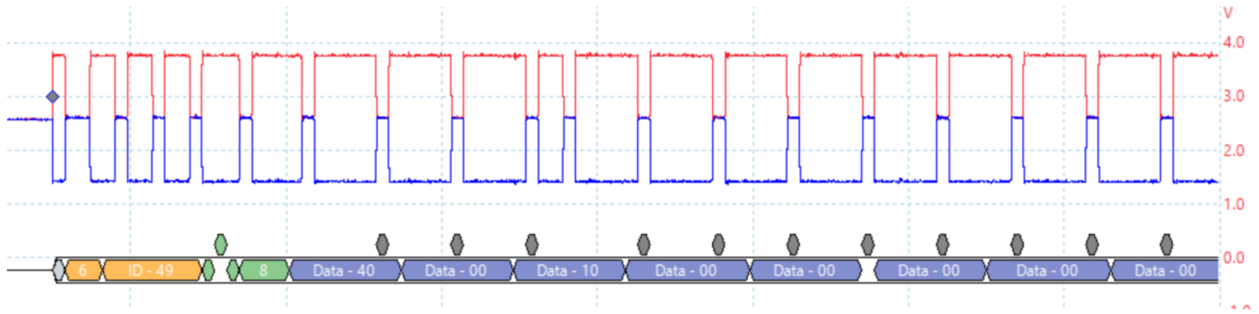


Figure 4.12: CAN bus signal from Node 1 with ID 0d73 (0x49) captured in Picoscope

Note that the ID in the above CAN signal is 49. It may seem to disagree with our ID of 73. However, it is important to note that the Picoscope does serial-decoding of the CAN bus signal and displays the identifier in a hexadecimal format. Indeed, the identifier of 0x49 is equivalent to 0d73.

The TDC starts measuring time intervals from the falling edge of the start pulse to the rising edge of the stop pulse, based on our set configuration. The criteria for the TDC to detect a rising edge is a voltage transition from a logical low to a logical high, where the logical low voltage is between 0-1.1V and logical high is between 2.2-3.3V, when powered by 3.3V. As it can be seen from Figure 4.12 above that neither our CAN H (red) nor our CAN L (blue) meet this criterion for the TDC to denote a rising edge using either of the signals. It was decided to use the differential voltage between the CAN H and CAN L signals as the inputs to the TDC start and stop pulses. This differential voltage signal is as shown in the Figure 4.13 without the TDC connected to the CAN bus for measuring time of flight values.

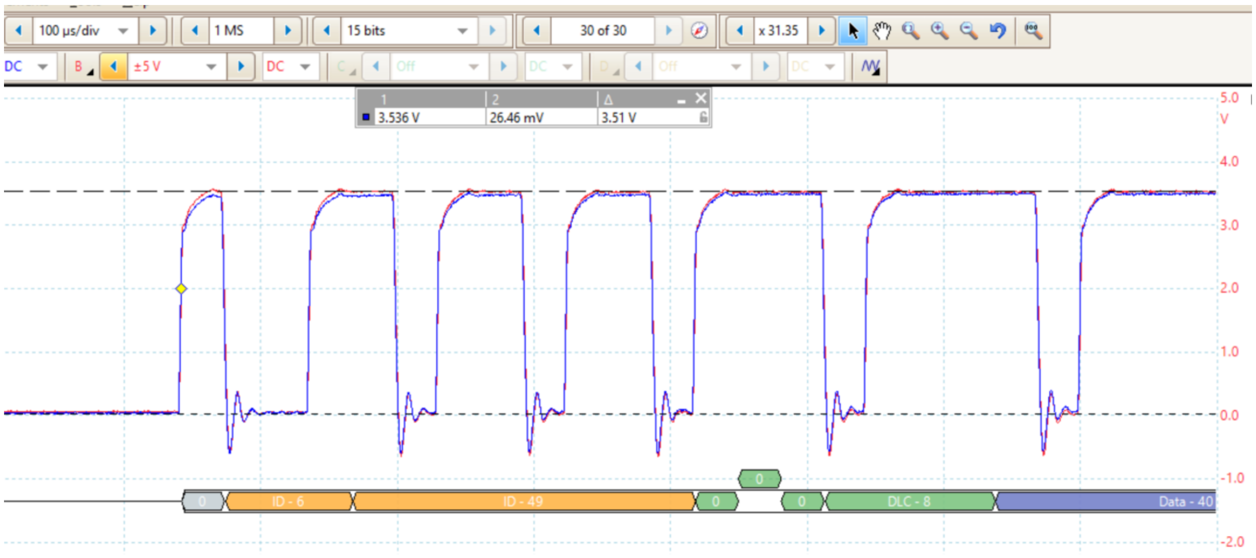


Figure 4.13: CAN differential signal between CAN H and CAN L measured using Picoscope without the TDC connected to the CAN bus setup for acquiring timing data

Now this differential signal is very suitable for the input to the TDC for the start and stop pulses. However, since we do not have a separate start and stop pulse we are now limited to using the same signal for both start and stop pulse. Once we connect the TDC to the CAN bus the differential signal as obtained using the Picoscope is shown in Figure 4.14. This shows that the TDC is drawing current from the system, because the voltage differential is getting reduced to almost half of what was originally observed without the TDC being connected.

In order to overcome this problem, it was decided to reduce the voltage input to power up the TDC from 3.3V to 2V using a voltage divider. This also proportionately reduces the voltage levels for the logical low to be 0-0.7V instead of 0-1.1V and logical high to be 1.4V-2.1V instead of 2.2-3.3V. Thus, we can see a proper transition from logical low to logical high for a rising edge to be recorded by the TDC for starting the measurement of time intervals for the start and stop pulses.

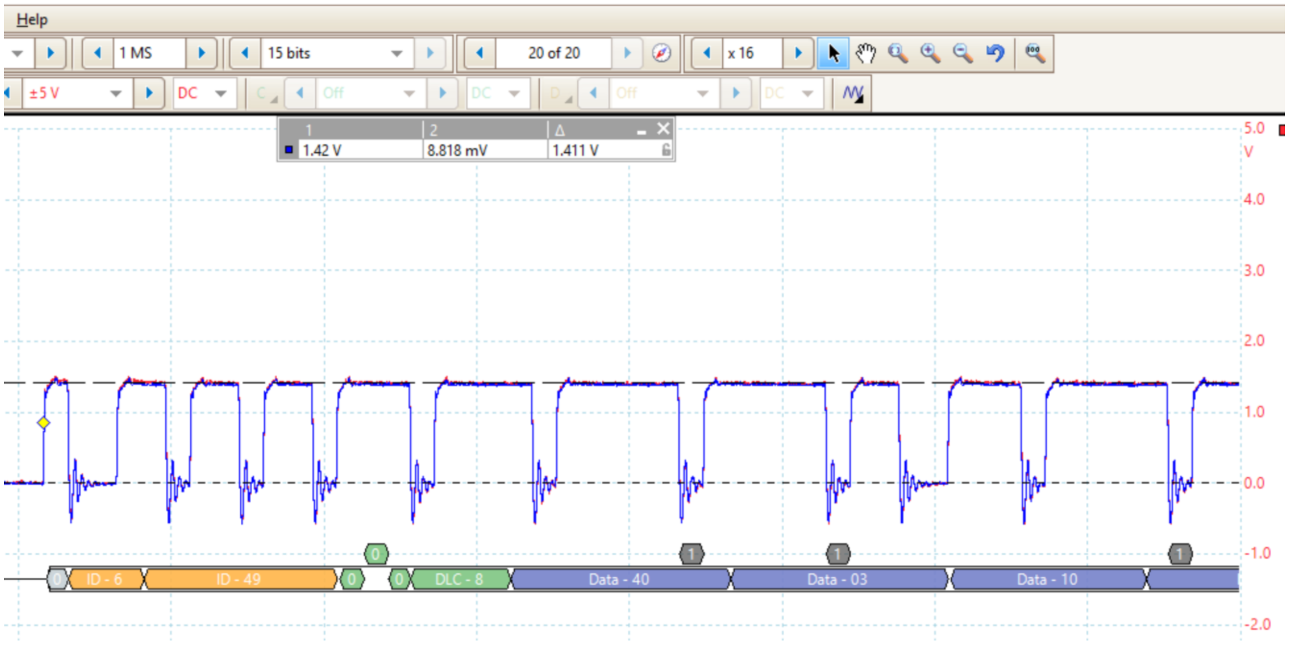


Figure 4.14: CAN differential signal with the TDC connected to the bus captured using Picoscope

Now in order to estimate the timing values that must be observed, we again take a look at our configuration settings for the TDC. The trigger signal is generated every 200ms which is surely more than an entire CAN packet timing. We know that several CAN packets will pass by before every new measurement from the TDC. We also know that since we have the same CAN signal going into the start and stop signals for the TDC with start edge polarity to be falling and stop edge polarity to be a rising edge, we can estimate a time interval of either 2000ns or 1000 ns depending on the trigger signal generation. Say if the trigger signal is generated before the start of the CAN packet, we will see a time value of 2000 ns because 2 bits pass between the falling edge and rising edge of the start and stop signals respectively and if the trigger signal is after the start bit we can expect a time value of 1000ns as shown in Figure 4.15. These values are based on the number of bits that pass by between the falling edge of the start signal and rising edge of the stop signal. Since the CAN bus operates at a speed of 1Mbits/s, every bit is transferred in 1us or 1000 ns.

Figure 4.15 shows the zoomed in view of the CAN signal and the circles show the start and stop pulses that would be observed by the TDC. It also shows the bit breakdown and hence the timing values that can be expected from the TDC for the appropriate case.

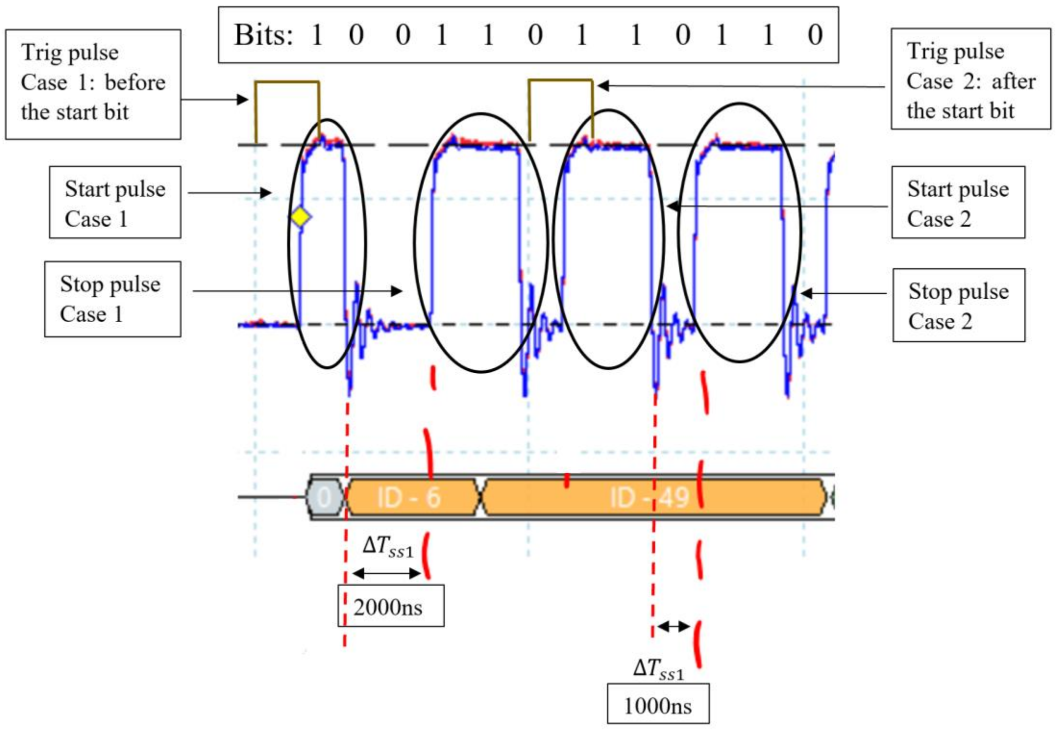


Figure 4.15: Zoomed in view of the identifier signal with bits laid out and expected start and stop pulses by the TDC

With a better understanding of our expected results we then go ahead and take the readings for each node identifier bit timing values using the TDC. The data is then plotted in MATLAB and the histograms are shown in Figure 4.16.

It can be seen that the time intervals are significantly unique to fingerprint each node and, their value (in the range of 900 ns) does agree to our expected value of around 1000 ns. Most of the readings turned out to be about 1000ns and the ones which were about 2000ns were excluded for the sake of reducing processing. This also means that the trigger signal was in fact sometimes before the start bit of the CAN packet and sometimes after the start bit, but again was not a major point of concern. KNN technique is applied to this dataset and the classifier is trained. We also obtain a new set of test data and check for the model accuracy. The test data is shown in Figure 4.17.

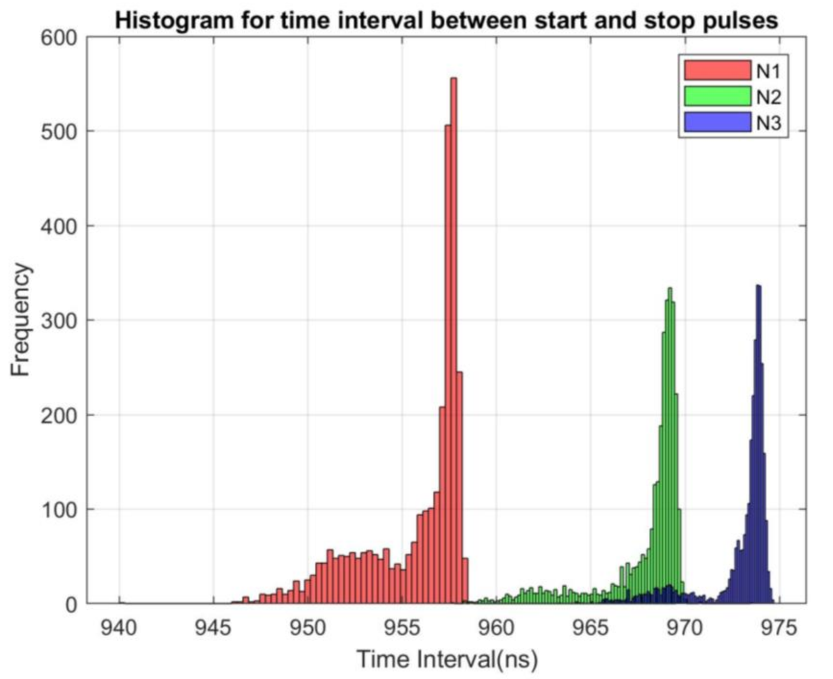


Figure 4.16: Time interval between start-stop1 plotted as histogram (for training dataset)

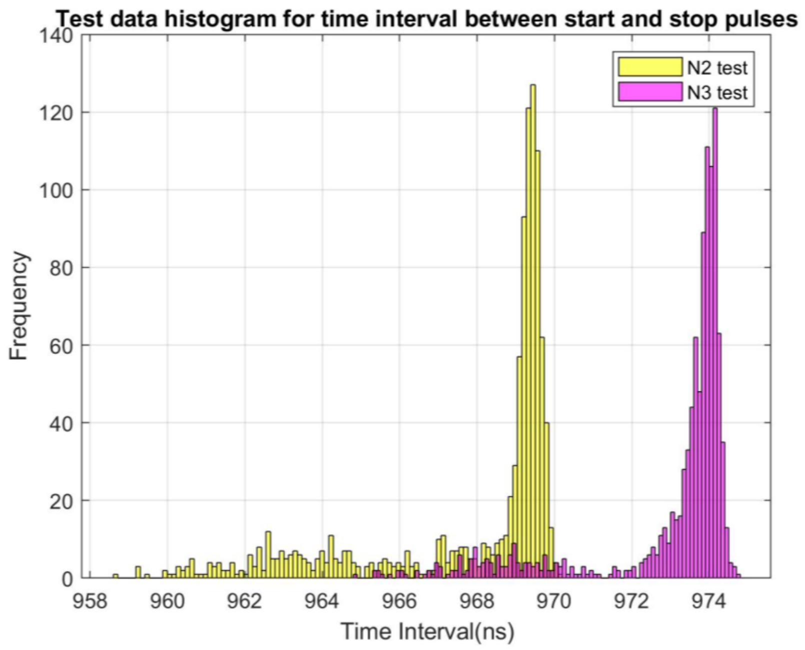
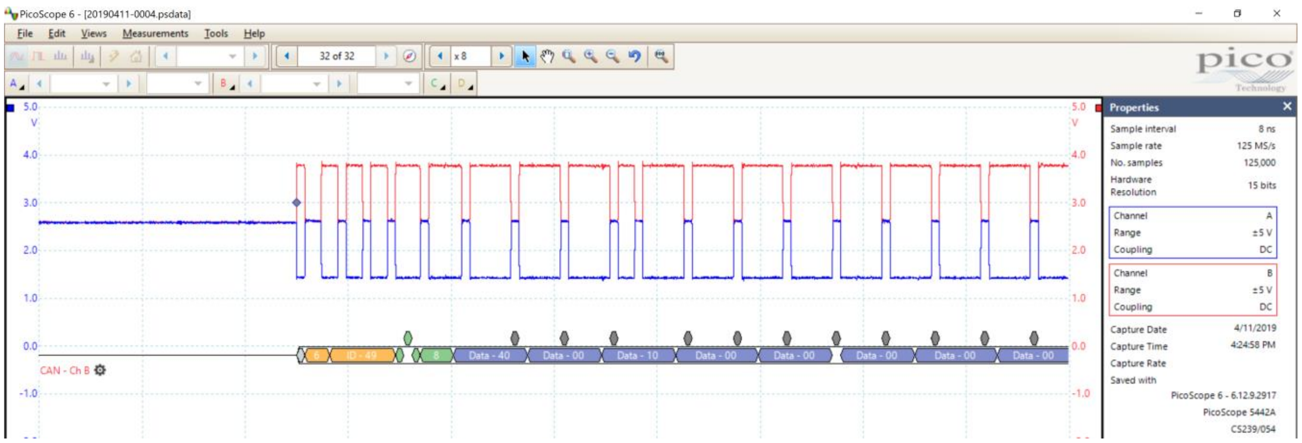


Figure 4.17: Time interval between start-stop1 plotted as histogram (for test dataset)

4.5 Neural Network trained on the Identifier data from Picoscope

Since KNN algorithm is not a model-based approach rather a feature-based technique, we also resorted to a model-based approach in order to try detection i.e. to implement a neural network on the CAN bus identifier signal directly and see if we could indeed detect intrusions with an alternative approach. Hence the next experiment was to just collect the CAN bus signal using a Picoscope, extract the identifier data into MATLAB and train a neural network classifier.

Figure 4.18 shows the CAN bus signal obtained from the Picoscope. The Picoscope collected data at a sample rate of 125 MS/s which is significantly higher that the CAN bus baud rate of 1Mbits/s. As it can be seen from the figure that we collect a total of 125,000 data samples for the entire session out of which we now have to extract the identifier data samples. Each identifier is 7 bit long and we have data for 1000ms, which means each bit has roughly 1000 data points representing it. This means once we can get the index value for the start of the identifier bit we can extract 1024 samples from that index to get the identifier data in MATLAB.



Sample interval	8 ns
Sample rate	125 MS/s
No. samples	125,000
Hardware Resolution	15 bits
Channel	A
Range	±5 V
Coupling	DC
Channel	B
Range	±5 V
Coupling	DC
Capture Date	4/11/2019
Capture Time	4:24:58 PM
Capture Rate	
Saved with	PicoScope 6 - 6.12.9.2917 PicoScope 5442A CS239/054

Figure 4.18: CAN signal for identifier ID73 using Picoscope showing different Picoscope parameters for data acquisition

The resolution of the Picoscope is 15 bits for a voltage range of ± 5V which means the LSB that can be measured is around 0.3 mV. The total number of training cases used was roughly 100 for each node. The extracted identifier data from MATLAB is shown in Figure 4.19.

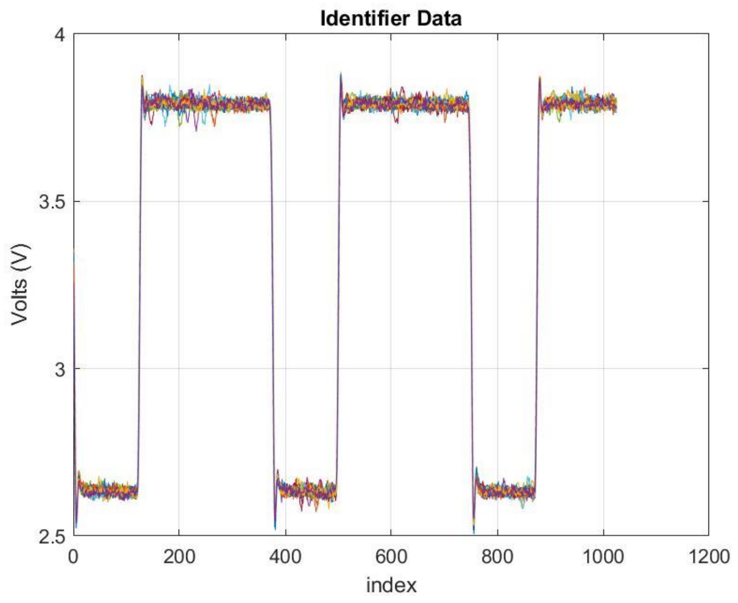


Figure 4.19: Identifier data extracted from the CAN signal using MATLAB showing 32 datasets plotted over each other

The above figure has the identifier data for Node 1 with ID 73, 32 datasets plotted over one another in MATLAB. As mentioned before we collect about 100 training cases for 3 nodes and supply them into the neural network for training it. Our neural network has 1025 inputs because a single dataset of the identifier consists of 1025 data points, 1 hidden layer with variable number of nodes inside the layer such as 510, 250, 25, and 3 output layers since we are trying to achieve multi-class classification with the neural network. An interesting thing to note is that we are using every single data point of an identifier i.e. 1025 values as features to the neural network. This is because we are assuming that every data point is conditionally independent of the other and there is no correlation amongst each other. Based on our result we can check if our assumption actually holds or not.

We also go ahead apply the same neural network structure to just the noise data at the high end of the identifier signal just to exploit the fact that if the noise data is actually unique to every node, then the neural network should pick up the unique features of the noise and be able to classify the nodes correctly. The results from the trained neural network such as the accuracy of the network will be later presented in the results section of this thesis, Section 5.5. Interpretation of these results will also be explained. We use the conjugate gradient descent method for optimizing the cost function of our neural network and finding the weights for every layer of the network. An important point to remember is that neural networks have several parameters that must be tuned to achieve higher accuracies and such parameters are called hyper-parameters, such as number of input features, number of hidden layers, activation function, cost optimizer functions for determining layer weights, etc. and it is sufficiently time consuming to tune your network to attain high accuracies.

4.6 Modifications to the TDC EVM

From Figure 4.13 and Figure 4.14 in Section 4.4, it was clear that the TDC was drawing current from the CAN bus due to the reduction of the voltage level. It is a case of impedance mismatching which is causing the TDC to draw current instead of having a resistance much higher than the CAN bus terminating resistance of 120 ohms. At first, we sought to match the impedance by adding external resistance of 70 ohms between the TDC start (+) and the TDC start (-) to create the balance of impedance, however, the results were not significantly changing.

Looking more carefully at the TDC EVM schematic in Figure 2.26 from Section 2.3.8 (zoomed in view shown in Figure 4.20), it was observed that there was a 50-ohm resistor connected between the TDC input (TP1) and ground. This was the primary cause of the current draw, since now an additional 50-ohm was introduced in parallel to the terminating 120-ohm of the CAN bus which led to a reduced termination voltage and therefore the impedance mismatch. This 50-ohm resistor was unsoldered from both the start and stop terminals on the TDC EVM board and then the new signal was checked using the Picoscope.

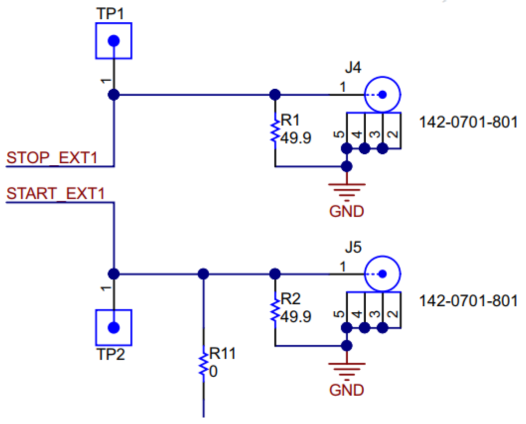


Figure 4.20: TDC EVM schematic showing 50-ohm resistor between the TDC input pin and ground

The new CAN bus differential signal after removing the 50-ohm resistance is as shown in Figure 4.21. It can be seen that when the TDC is connected to the bus, the differential voltage no longer reduces and is the same as before, without the TDC being connected. This can be compared to Figure 4.13. The signal although is shown for a different identifier just for comparison purposes and that the identifier doesn’t matter for checking the voltage level for the differential.

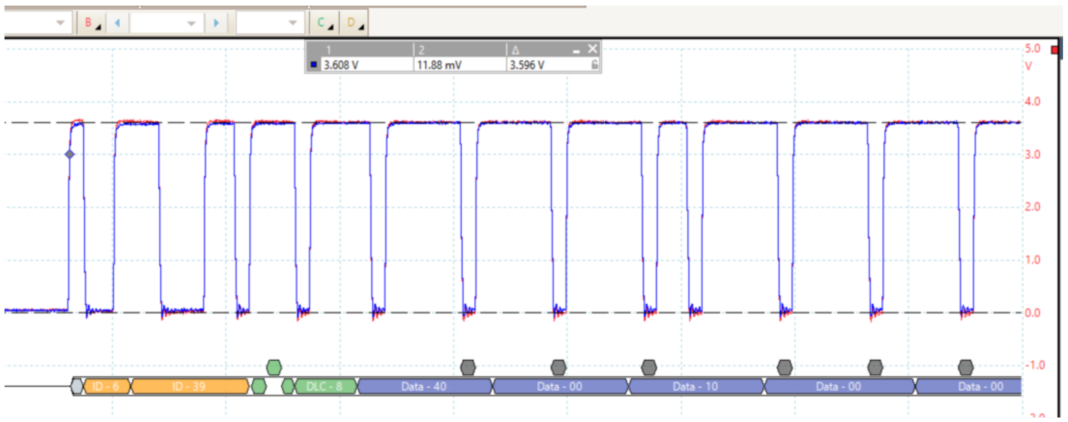


Figure 4.21: CAN bus signal with TDC connected (after removing 50-ohm resistor) as measured from Picoscope

Next, we now collect new data with the TDC as we did in Section 4.4. This time we change the TDC settings a bit to configure the start pulse edge polarity as rising edge and stop pulse polarity as rising edge. Since the ID 73 is used as shown in Figure 4.22, we expect the TDC to read around 3000 ns for start-stop 1 timing value. This time we also collect results up to 3 start-stop values, with the expected timing values being 6000 ns and 9000 ns respectively. The MATLAB plots for the histograms of the collected data are as shown in Figure 4.23 and Figure 4.24.

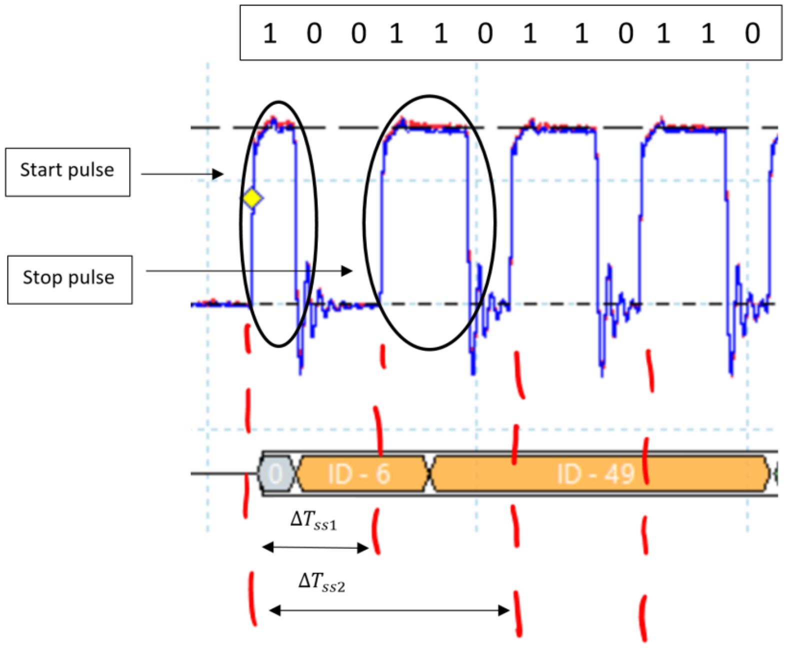


Figure 4.22:Identifier signal with bits laid out and expected start and stop pulses by the TDC with timing values for multiple start-stops

It can be seen from Figure 4.23 and Figure 4.24, that we do get the timing values for identifier bit timings as expected, one being about 2999.5 ns and the other being 5999.5 ns. However, now it is seen that these timing values are not nearly as distinct or separable as they were in Figure 4.16 for the 3 nodes under test. This really gets us to a point where our removal of the 50 ohm resistor just for getting the correct differential voltage is under scrutiny.

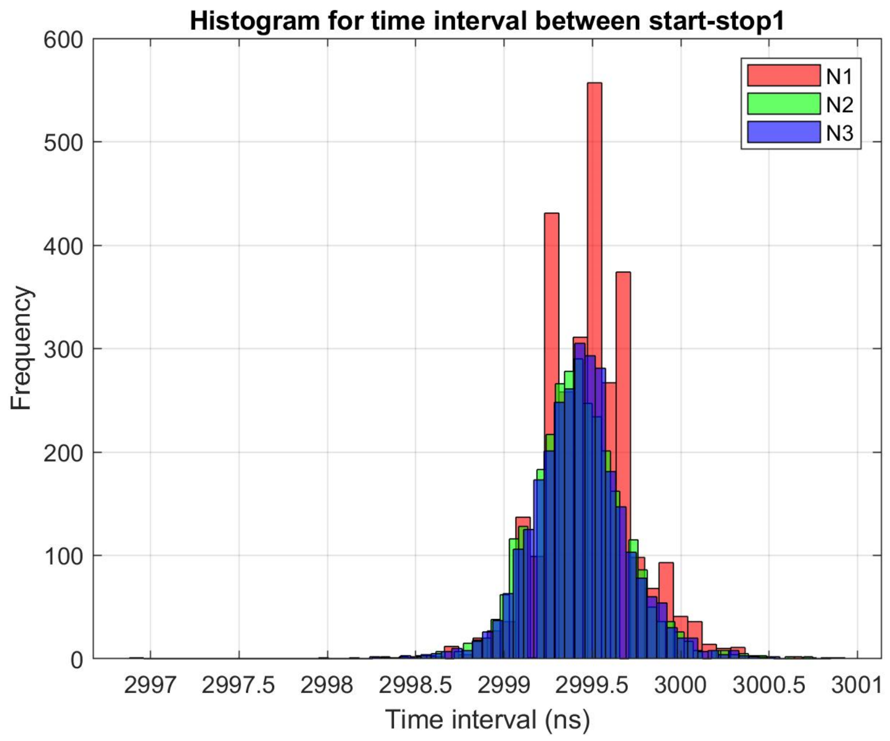


Figure 4.23: Histogram for time interval between start-stop1

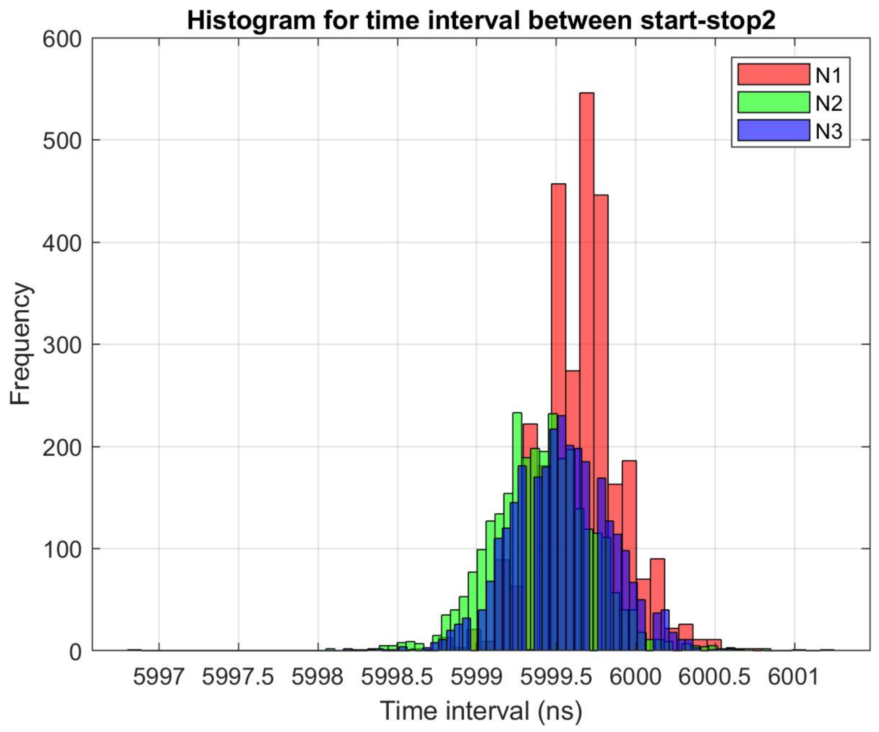


Figure 4.24: Histogram for time interval between start-stop2

Chapter 5

5.1 Results and Discussions

In this section we will go through the results from chapter 4 and provide interpretations to them. The results will be described sequentially in the same manner as for chapter 4.

5.2 Verification of TDC functionality

As shown in Figure 4.6, the histogram obtained looks very much like a normal distribution. It however looks to be left skewed. We decided to fit a normal distribution around the histogram using the histfit function in MATLAB and obtained the mean, standard deviation and skewness of the original data. These parameters are shown in Table 5-1 and the fitted normal distribution is shown in Figure 5.1.

Mean (ns)	19000.121
Standard deviation (ns)	0.0708
Skewness	-0.0603

Table 5-1: table showing mean, standard deviation and skewness of the data for TDC functionality verification experiment in Section 4.2

It can be seen from the table that the TDC very precisely and accurately measures the time interval mean which was theoretically 19000 ns, with an accuracy of 121 ps. The standard deviation is 70ps, which is roughly double the value from Figure 2.15. There is also some skewness observed from the table above. From [73], it is seen that skewness in the range of ±0.5 is acceptable and that the distribution can be regarded to be almost perfectly symmetric. It is important to check for skewness because if the data comes out to be highly skewed, then our KNN algorithm would need to do some form of weighting on the data to avoid false classification as described in Section 2.4.1

of theory. We can now safely assume that TDC does not introduce any skewness into the system, works as it is supposed to and can now proceed to our simulation results.

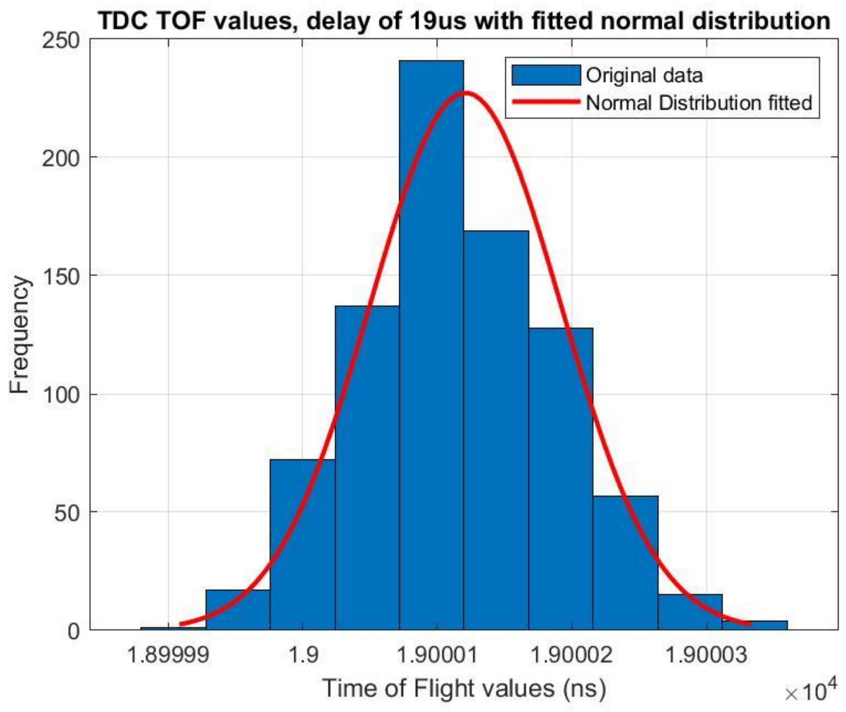


Figure 5.1: Histogram fitted with normal distribution for TDC functionality verification experimental data obtained in Section 4.2

5.3 CAN identifier signal simulated on MSP432 using TimerA module

Figure 4.10 shows the different histograms obtained from the simulation experiment study. They have distinct means and very little overlap and hence serve as good training data for our KNN algorithm. Total number of data points for each node is 1500. We pass the entire dataset for each of the nodes as input to the KNN classifier model in MATLAB along with each class label. Once the model is trained, we will input new test data to assess the performance of our classifier. For testing purposes, we use ‘k’ = 5 as number of neighbors to be compared against for the test data. Table 5-2 shows the re-substitution loss and cross validation loss for our trained KNN model.

Re-substitution loss	1.3 %
K-fold cross validation loss	1.65 %

Table 5-2: loss results for trained KNN model on data from Section 4.3

The re-substitution loss is obtained when we use the entire dataset for training as the test dataset for our trained classifier model. The k-fold cross validation loss is obtained by creating a new cross validation model from our trained model [74], [75]. By default, MATLAB uses a 10-fold cross validation on the training data and each data point is randomly assigned into one of the 10 groups created. Then the cross validation model is trained on all the remaining 9 groups except one for which it is tested on, and the loss is computed for each model, and the average loss is returned [76]. If the re-substitution loss and k-fold loss are not much different from each other then it means our model has fit the data accurately and will perform as per expectations given that the test data is not widely different from the training dataset.

We will show the performance of our classifier for 3 test cases and the accuracies for each case is reported in Table 5-3. The histograms for these test data overlaid with the training data are shown in Figure 5.2, Figure 5.3 and Figure 5.4 respectively with the appropriate labels for each of the test data vectors.

Test case class label	Accuracy (%)
1834	95.23
6526	100
2580	81.53

Table 5-3: Test case and corresponding accuracy of KNN trained on data from Section 4.3

We can see that the accuracy for class label 2580 is significantly less which can be attributed to the lesser overlap of the test data to the training data as shown in Figure 5.3. This could be a result of temperature variation during data collection which may have shifted the mean of the data altogether. A similar trend can be observed for the test case label 6526 and 1834, for which the mean too has shifted slightly on the lower bound of the original mean. However, it can be seen that the shift is not significant and more testing may need to be conducted to verify this assumption.

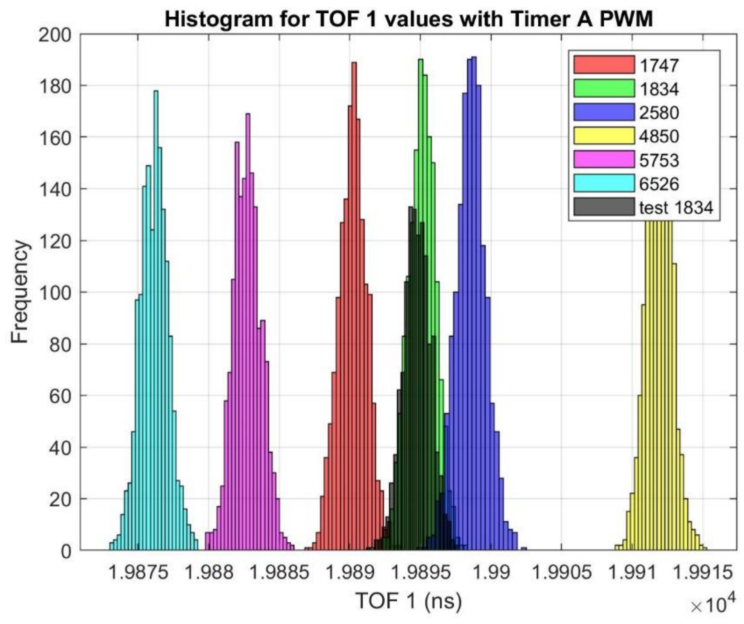


Figure 5.2: Test data for label 1834 overlaid with training data histogram

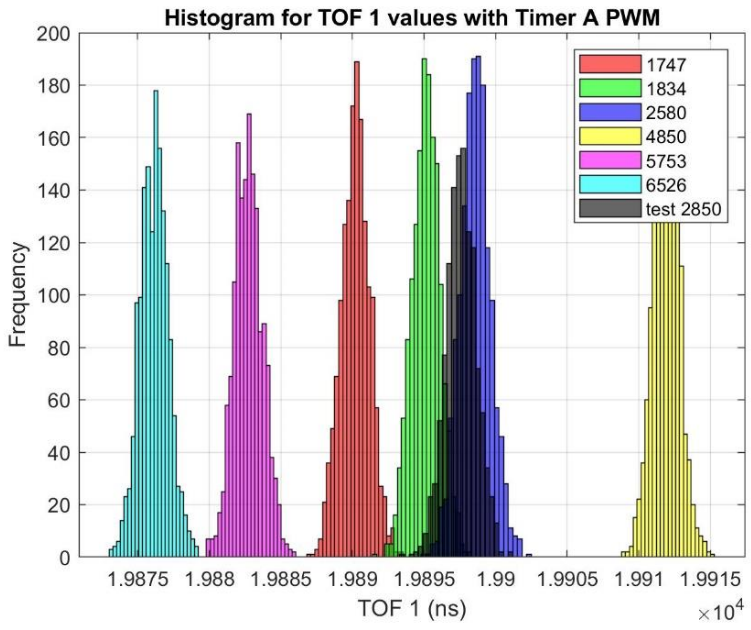


Figure 5.3: Test data for label 2580 overlaid with training data histogram

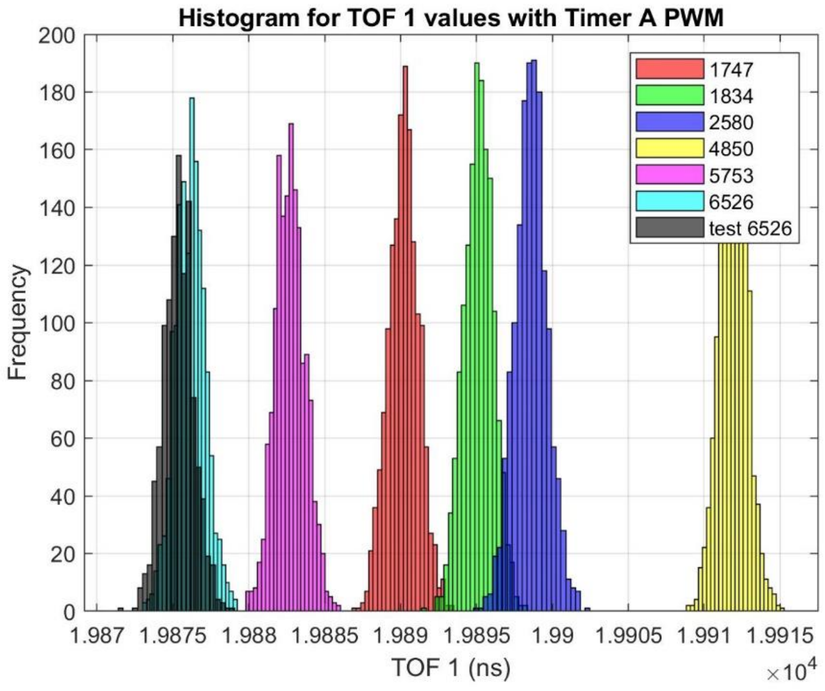


Figure 5.4: Test data for label 6526 overlaid with training data histogram

With these accuracies, it can be safely concluded that our KNN classifier can predict the right class for test data and our concept of detecting intrusion using timing characteristics holds well. Next step is to apply this KNN algorithm to the actual data from the CAN bus setup.

5.4 KNN applied to CAN bus identifier bit timing data

In chapter 5 we have shown both the training and the test data that have been collected using the TDC. Now it was simply a matter of putting these into the KNN classifier, creating the model, obtaining the loss values and checking the accuracy of the classifier. The loss values and accuracies are shown in Table 5-4. An interesting thing to note is the bimodal nature of the data that is obtained as seen in Figure 4.16 and Figure 4.17. Since the frequencies of these lower modes are roughly 10% of the dominant mode, these lower mode values can be neglected and discerned as noise which is not truly representing the clocking characteristic of the CAN bus node. The training data uses a total of 3000 data samples for each node whereas the test data has 1000 samples for the 2 nodes under test. The reason for node 1 having a significantly higher mode frequency is not completely known at this point and hence the test was performed for node 2 and node 3 data, while still using node 1 data as an input for the KNN classifier.

Re-substitution loss	3.9 %
K-fold loss	4.6 %
Accuracy for test case label node 2	96.15 %
Accuracy for test case label node 3	88.54 %

Table 5-4: Loss and Accuracy of KNN classifier for actual CAN bus identifier data

The combined histogram showing both training and test data is shown in Figure 5.5.

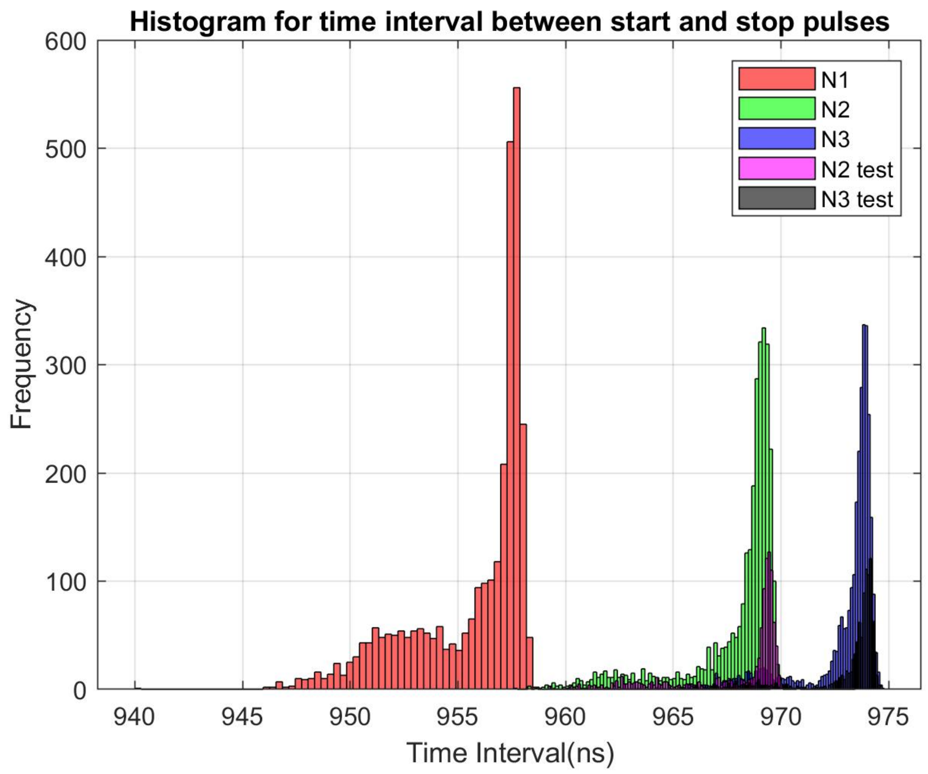


Figure 5.5: Combined histogram depicting training and test data for actual CAN bus setup

Again, we can see from Table 5-4 that our loss values agree with each other which means that our KNN classifier will be misclassifying data about 4% of the time. This is also in tandem with the accuracy for node 2 test data, however, the accuracy for node 3 test data is significantly less. Two possible reasons for this are that since the lower mode in the data for both train and test cases has been retained, i.e. lower mode for node 3 is overlapping with the dominant mode of node 2, implies that KNN algorithm is losing accuracy in that region. Secondly, it can be seen from both the test case data that the means have been slightly shifted to the higher bound of the training data, which could be either due to fewer data samples in the testing data with respect to the training data or again possibly due to the temperature drift. Based on our figures Figure 5.2 through Figure 5.4 we can see that the temperature drift may lead to shifting the mean to the lower bound of the training data than to the higher bound and so temperature drift is less likely to have the above effect. This again concludes that we can indeed detect intrusions using identifier bit timing data.

5.5 Neural Network Results for model trained on raw identifier data

As mentioned in chapter 5.4 we created a neural network with 1025 inputs, 1 hidden layer with variable number of nodes and 3 output neurons denoting the node that the data is classified into, and we run the cost optimization algorithm for determining the weights up to 100 times. The results are only computed for training set accuracy, which uses the training data as the test data to compute accuracy of the neural network classifier. We present our results below in Table 5-5.

Sr no	Data samples	Input size	Hidden layer size	Output size	Cost func start	Cost func at n-1 iter	Cost func end	Accuracy	Iteration executed	Probable Reason
	Actual identifier data of 1025 data points and 96 samples for each of the 3 nodes									
1	96 * 3	1025	510	3	-	-	5.754469	70.833333	38	Probably just right, saddle
2	96 * 3	1025	510	3	-	-	6.273919	43.402778	6	
3	96 * 3	1025	510	3	6.634212	5.713412	5.712029	65.625000	77	overfitting
	Just change in the size of hidden layer									

4	96 * 3	1025	250	3	4.313190	3.731188	3.631884	33.333333	20	Overfitting
5	96 * 3	1025	250	3	4.088960	3.796877	3.796591	46.875000	39	
6	96 * 3	1025	250	3	4.095681	3.702221	3.702083	47.222222	57	
	Again, reducing size of hidden layer									
7	96 * 3	1025	25	3	2.916976	2.916976	2.916976	33.333333	1	Under-fitting, Not learning
8	96 * 3	1025	25	3	2.958226	2.958226	2.958226	33.333333	1	
9	96 * 3	1025	25	3	2.883506		2.181049	33.333333	2	
	Actual identifier data of 1025 data points and 48 samples for each of the 3 nodes									
10	48 * 3	1025	510	3	11.07627	5.903668	5.884937	50.000000	100	overfitting
11	48 * 3	1025	510	3	10.76431	6.491211	6.490278	61.805556	89	
12	48 * 3	1025	510	3						
	Just noise on CAN High signal 1st 100 samples with 96 datasets for each node									
13	96 * 3	100	25	3	1.946236	1.942040	1.941733	34.027778	7	No features to learn
14	96 * 3	100	25	3	2.618851	1.944847	1.938569	33.333333	3	Or Getting stuck
15	96 * 3	100	25	3	1.984095	1.938283	1.937886	33.333333	4	
	Just noise on CAN High signal 1st 100 samples with 96 datasets for each node									
16	96 * 3	100	50	3	2.001839	1.954617	1.954467	33.333333	8	
17	96 * 3	100	50	3	2.086574	1.954691	1.954428	33.333333	6	
18	96 * 3	100	50	3	2.010597	1.954621	1.954319	33.333333	10	
	Down-sampled data by 10									
19	96 * 3	103	10	3	2.771048		1.933809	33.333333	2	Again, no features to learn
20	96 * 3	103	10	3	1.947754		1.947754	33.333333	1	Or just getting stuck
21	96 * 3	103	10	3	2.934435	1.929988	1.929968	33.333333	8	Saddle point

	Down-sampled by 10 but different layer size									
22	96 * 3	103	25	3	1.951826	1.938755	1.937029	33.333333	3	
23	96 * 3	103	25	3	1.946157		1.946157	33.333333	2	
24	96 * 3	103	25	3	2.514251	1.937692	1.937425	33.333333	6	

Table 5-5: Neural Network performance statistics

We will briefly describe the results of few of the test cases just to develop a broad understanding of our neural network performance and the hyper-parameters that may need to be tuned for better accuracy. In test case 1, we see that the accuracy, although is 71%, the number of iterations executed for the cost function optimization is just 38, far from 100, which means that the cost function has converged and there is no further possibility of reducing the cost function. However, the accuracy is not high enough in the range of 80-90% which means that there is a possibility of the network getting stuck at a saddle point based on the initial weights' initialization. A saddle point is nothing but a local optimum which the gradient descent method may get stuck in depending on the initialization weights in the beginning. Now this also means that if indeed a saddle point is present and we do have a local minimum instead of a global optimum then our assumption of every data point being independent and uncorrelated may not be true after all and there may be some correlation between certain data points.

To examine the above issue, we conducted tests 13-18, which consisted of just extracting the noise signal on the higher voltage side of the CAN H signal and checking the accuracy of the newly trained neural network. It was observed that the cost function always converged to nearly 1.95 and in less than 10 iterations meaning that the cost function can't be reduced any further than that no matter the number of nodes in the hidden layer. This means that the noise is random and uncorrelated at least in its raw form and an accuracy of almost 33% indicates that the network is making a random guess of the label class because the least accuracy achievable is 33%. It also leads to the conclusion that there are no unique features in the noise data than can be learned by the network to predict the correct class label.

The cost function clearly decreases in value with the decrease in the number of nodes in the hidden layer, as is evident from all the test case results. In test 2 we see that the converged cost is higher than in the test 1 and the accuracy is lower, which is what is expected. In test 3 we see that the cost function is lower than test 1 but accuracy is also lower meaning the model is overfitting the data and hence the lower accuracy with the decreased cost. One of the reasons for the number of iterations not increasing after a certain number for all the test cases is because, once the cost function has converged there is no other search direction to explore.

Now from tests 4-9 we see that our accuracies are no more than 50% which is clearly a case of under-fitting because we are trying to use 250 or lesser nodes in the hidden layer, meaning we are trying to reduce the feature set from 1025 to 250 or less. Under-fitting occurs when our model learns lesser number of features than optimum and hence very low accuracies. Tests 19-24 were done with down-sampled data by a factor of 10 because down-sampling would reduce our input vector size from 1025 to 100 while preserving the shape and features of the signal. However, the accuracy still remains in the range of 33% which means that the network is just randomly predicting the class without actually learning the input features.

Thus, all the above tests show that our neural network may need more tuning if there is indeed a possibility to detect intrusions by using the raw identifier data directly as input to the neural net. Some of the modifications that can be done, to try and achieve a better accuracy, are suggested as follows.

Firstly, the sigmoid function, shown in Figure 5.6 for neuron activation introduces 2 pitfalls. The

1st pitfall is that the rate of update of the gradient in order to optimize the cost function is slow. This is because the sigmoid function can only have values between 0 and 1. So when a change of value is calculated for updating the weights, the change will also be between 0 to 1. This leads to a fairly slow gradient update and hence slower optimization of cost [77]. The 2nd shortcoming posed by the sigmoid function is vanishing gradient issue, which is basically the cause to the 1st problem, i.e. the gradient becomes smaller and smaller as we move from one layer to the next and hence there may come a point where the gradient is almost 0, leading vanishing of the gradient and the infinitesimally slow update of the weights [78]. A proposed solution to this is the rectified linear unit activation function which overcomes the issues posed by sigmoid function [79]. The rectified linear unit activation function is shown in Figure 5.7 [77].

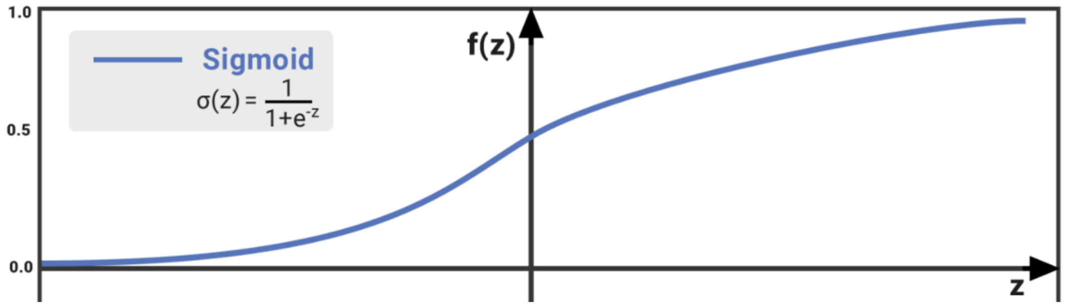


Figure 5.6: Sigmoid function for neuron activation in neural network

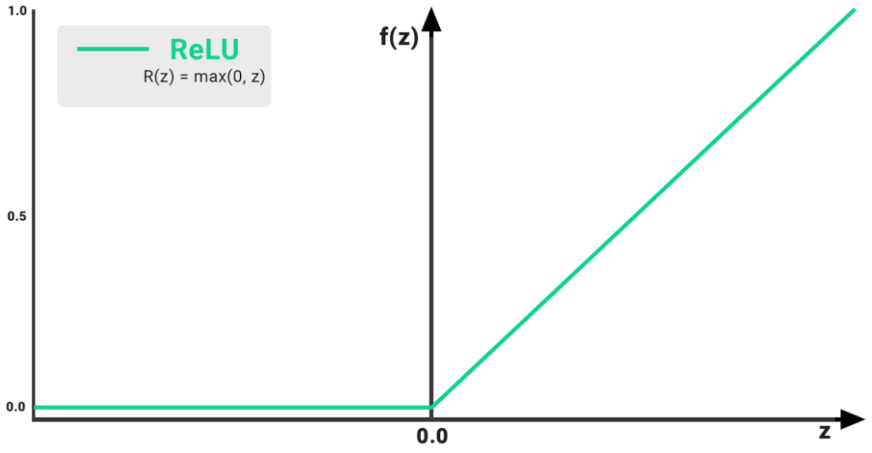


Figure 5.7: Rectified Linear Unit (ReLu) activation function for neurons

Another parameter that can be changed for better convergence of the cost is the optimization function i.e. gradient descent method. The disadvantage of gradient descent method is that it has a slower update rate and can get stuck at a saddle point if any exist. One possible way to avoid the slow update is to use better optimization functions for the cost and hence weight updates. Some of the optimization methods that can be used are Adagrad, Adadelta and Momentum methods which have ways of updating the learning rate based on gradients [80]. Figure 5.8 shows the performance of some of these methods in presence of a saddle point.

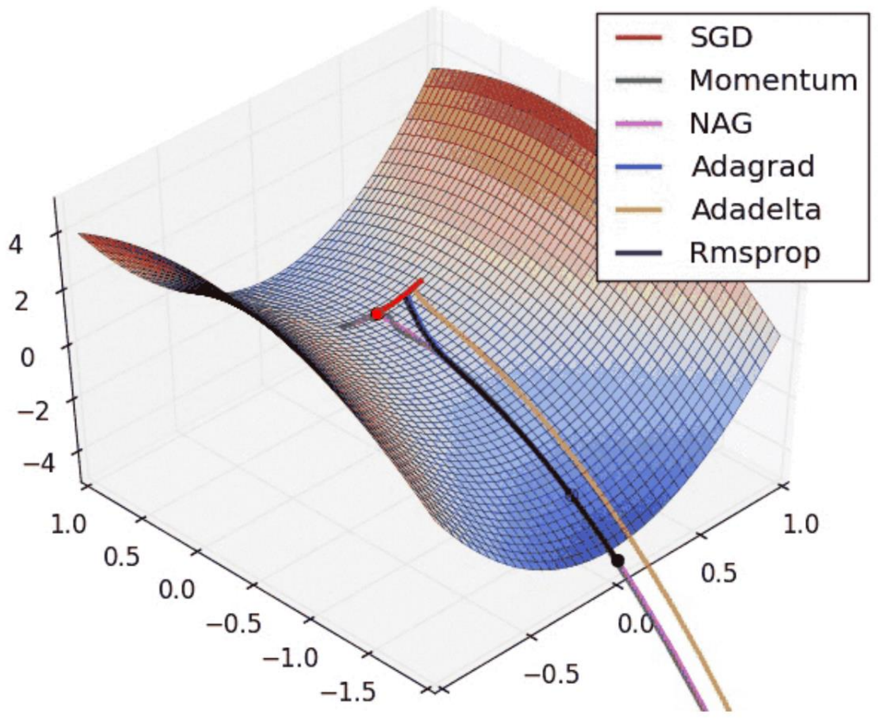


Figure 5.8: Performance of different optimization functions in presence of saddle points

One last thing that can be tried, is to increase the number of hidden layers and also play with the number of nodes in every layer and tune the neural network for achieving better accuracies. Neural networks are highly complex classifiers that must be tuned from problem to problem to achieve desired results.

5.6 Actual CAN bus identifier bit timings

As seen from Figure 4.23 and Figure 4.24, the time interval between the start bit rising edge and 2 consecutive identifier bit rising edges are extremely precise for all 3 nodes under test. Their means are all close to 2999.5 ns and 5999.5 ns respectively. Since the histograms for all 3 nodes have a very high overlap it is almost impossible for any machine learning technique to classify a test data point into the correct cluster with a high accuracy. In conclusion this means that the clocks for the CAN bus setup under consideration could be of very high-quality manufacturing, although we may need to conduct more experiments to verify the validity of this data. Rigorous testing may need to be performed to ascertain the validity of the collected data since the removal of a resistor in no way means that a change in the timing values should be observed. This however does not mean that our original hypothesis of detecting intrusions using timing characteristics is completely invalid. It just means that if the data obtained is truly valid, then detecting intrusions in a CAN bus system using timing characteristics with high precision crystal oscillators can prove to be quite challenging. We will however propose some future experiments that may aid to make a more educated guess of applying this method for CAN bus systems in the next section.

Chapter 6

6.1 Conclusions and Future Work

This chapter will briefly discuss the process to detect intrusions based on timing measurement, revisit the thesis goal for CAN bus intrusion detection, describes the success and limitations of this work and propose future work experimental suggestions.

6.2 Intrusion detection based on timing measurement

The main goal of this thesis was to fingerprint microcontrollers based on their timing characteristics which would be an indirect measure of the clock timing and thereby detect intrusion when applying this technique for CAN bus controllers. Any 2 microcontrollers running at the same clock frequency would still have a small difference in their time period due to manufacturing inaccuracies, which would be reflected in the signals generated using a microcontroller not having exactly same timing characteristic. In order to exploit this fact and fingerprint a microcontroller using its timing attribute, an experiment was conducted using 6 different MSP432s sending 2 PWM signals using the same timer module, same frequency, but different starting time and duty cycle as shown in Figure 4.8. A timing measurement unit called time-to-digital convertor (TDC) was used to measure the time difference between the rising edges of the 2 PWMs. These PWMs are an indirect measure of the microcontroller clock frequency. These timing values are then plotted in the form of histograms for all 6 microcontrollers in Figure 4.10. It is clear that these microcontrollers have a distinct timing values for the same pulses and hence can be easily distinguishable. The k-nearest neighbor machine learning technique was then applied to classify an incoming timing data value into one of the 6 clusters with a minimum of 80% accuracy Table 5-3. Once this proof of concept was verified and turned out to be very successful, it was time to apply this idea to an actual CAN bus setup and see the results.

6.3 Fingerprinting applied to CAN bus setup

An EPOS2 CAN controller was used for applying our fingerprinting technique and detecting intrusion. There were 3 CAN controllers under study, each running at 1Mbits/s frequency. Since in the CAN protocol, the uniqueness of a CAN controller is its identifier value or the CAN ID, which is the 11-bit identifier after the start bit in every CAN packet (Figure 2.5), it was decided that the fingerprinting technique would be applied to the unique identifier bit timing value for every node (Figure 4.15). This essentially meant that the time-to-digital convertor would measure the timing value between the rising edge of the start bit in the CAN packet and rising edge of the successive identifier bits up to a maximum of 5 timing values, each individually less than 8ms which is the maximum range of time interval measurement of the TDC.

There were 2 major experiments conducted to go about this. In the 1st experiment we calculated the bit timing interval values for just 1 bit and observed that we were indeed able to fingerprint the CAN bus nodes correctly using the KNN classifier. Since we wanted to correct for the incorrect differential voltage (Figure 4.14), we performed another experiment with some modifications to our TDC, in order to expel our concerns regarding the impedance mismatch problem created by the TDC, and thereby collected new data. This time our timing values matched our expected values of 3000ns and 6000ns for 2 consecutive stop pulses. However, as we saw from Figure 4.23 and Figure 4.24, we cannot distinguish between the timing values from all the 3 nodes under test and hence our data after removal of 50-ohm resistor is subject to further testing and validation.

In conclusion we can say that it is possible to classify microcontrollers based on their timing characteristics and can also be applied to CAN bus nodes, however we may have to conduct more testing to ascertain the validity of compatibility of the TDC with CAN bus system. We may need perform more experiments to assert the validity of the data collected along with correct functionality of the CAN hardware and TDC hardware.

6.4 Future Work

One of the first things that we can do is to eliminate the use of the differential signal, rather than just use the CAN H signal with a 0 DC offset and amplified signal to match the TDC requirements of 2-3.3V. The DC offset can just be removed by using a blocking capacitor and the signal amplification can be done using an op-amp. We would definitely need a very high-speed op-amp because we are concerned about doing time measurements and we don’t want a high delay in the time signals due to the op-amp. Figure 6.1 shows an example circuit and conditioned CAN H signal.

The blue signal in the figure is the original CAN H signal and the green signal is conditioned after the removal of DC offset and amplified. The op-amp used is AD-8091 with a bandwidth of 110 MHz and unity gain. This is important because when we are trying to implement a gain of 3 to our CAN H signal, our bandwidth would reduce to 37 Mhz. Hence the square wave of CAN H will ultimately be distorted because a square wave is composed of all frequencies and by using an op-amp like the one we used, we are essentially limiting ourselves to recreate the square wave with frequencies only up to 37 Mhz. The LTspice simulation in Figure 6.1 does not reflect that because the op-amp model used is for an ideal op-amp. One may use a non-ideal op-amp model to see the distorted signal and gauge an effect of distortion on the timing measurement too.

Another experiment that can be performed is to eliminate the bias from both CAN H and CAN L signals, amplify them to the expected voltage range for the TDC and use one signal as the start pulse and another as the stop pulse. Figure 6.2 shows the 2 signals in the correct form to be used for this experiment. Blue signal is CAN L and green signal is CAN H.

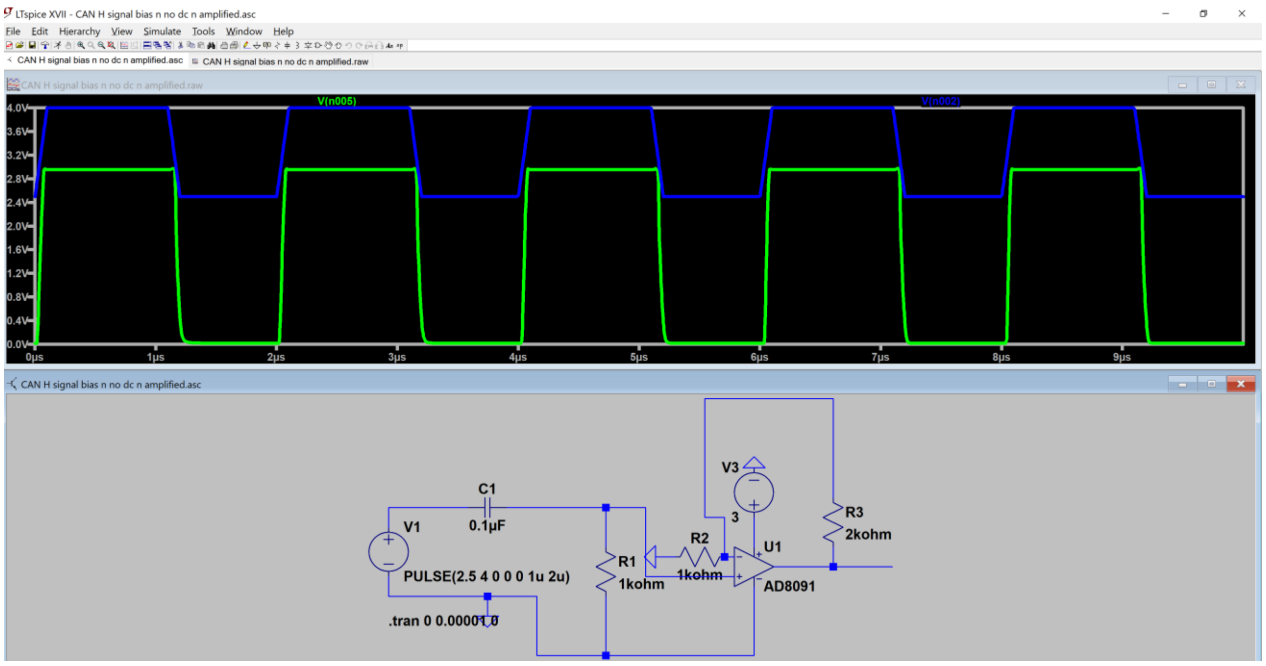

LTspice XVII - CAN H signal bias n no dc n amplified.asc
File Edit Hierarchy View Simulate Tools Window Help
CAN H signal bias n no dc n amplified.asc
CAN H signal bias n no dc n amplified.raw
V(n005)
V(n002)
4.0V
3.6V
3.2V
2.8V
2.4V
2.0V
1.6V
1.2V
0.8V
0.4V
0.0V
0µs
1µs
2µs
3µs
4µs
5µs
6µs
7µs
8µs
9µs
C1
0.1µF
V1
PULSE(2.5 4 0 0 0 1u 2u)
.tran 0 0.00001 0
R1
1kohm
R2
1kohm
V3
3
U1
AD8091
R3
2kohm

Figure 6.1: Conditioned CAN H signal to be used with TDC

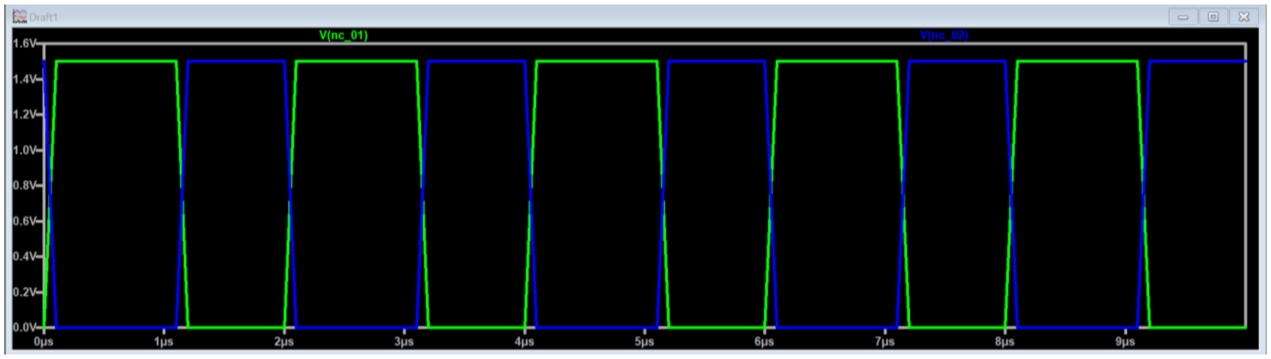


Figure 6.2: Conditioned CAN H and CAN L signals to be used as the start and stop pulse for the TDC

Another important thing to do is find a time interval measuring instrument which has a much higher update rate for measuring timing than just 200ms. This is because given our CAN bus runs at a speed of 1Mbits/s and if we have about 100bits in one CAN frame then we would need an update rate for measurement of every consecutive identifier bit timing of about 100us. FPGAs can be used which can run at 500 MHz or time measurement units which have a high event rate of 200MHz. In addition, it is important to see the effect of temperature on the clock timing and frequency, since we did see a shift in the timing mean between the training and test data in our simulation study with the microcontroller time measurement in Figure 5.2 through Figure 5.4. Some form of temperature correction may be needed to have a more robust classification of incoming timing data.

Changing the length of CAN cables in the current setup is also an important parameter that must be checked with the timing measurement. The effect of cable length, if significant, also means that the position of the TDC along the bus would be of concern and we may need to come up with a model to estimate the effect of TDC’s position in regard to the measured timing. It is also important to see how the bus speed affects the timing measurement because lower the bus speed higher is the time interval being measured, lower is the clock frequency and perhaps better clock performance at lower speeds.

To that extent we may need to explore different machine learning techniques for more robust classification of highly correlated data. Different frequency domain intrusion detection techniques such as statistical noise modelling and SVM classification can be looked into to supplement this work’s claim and have a robust identification based on time and frequency analysis of CAN bus data. Lastly it is essential to look into the practical implementation challenges that are posed by real time intrusion detection and computational complexity of machine learning technique implementations.

6.5 Final Remarks

This thesis presents a robust method to characterize microcontrollers based on their clock timing behaviors. More robust data acquisition instruments along with well-tuned machine learning algorithms will lead to an improved and strong classification of CAN bus nodes using our idea.

References

[1] A. Nuñez, “An Introduction to the CAN Bus: How to Programmatically Control a Car,” *Voyage*, 04-Jun-2017. [Online]. Available: https://news.voyage.auto/an-introduction-to-the-can-bus-how-to-programmatically-control-a-car-f1b18be4f377. [Accessed: 22-May-2019].

[2] “Car Hacker’s Handbook.” [Online]. Available: http://opengarages.org/handbook/. [Accessed: 22-May-2019].

[3] A. Greenberg, “A $60 Gadget That Makes Car Hacking Far Easier,” *Wired*, 25-Mar-2015.

[4] K. Smith, “a complete guide to hacking your vehicle bus on the cheap & easy – part 1 (hardware interface) | theksmith.” [Online]. Available: https://theksmith.com/software/hack-vehicle-bus-cheap-easy-part-1/. [Accessed: 22-May-2019].

[5] V. Joen, “Hack Your Vehicle CAN-BUS With Arduino and Seeed CAN-BUS Shield,” *Instructables*. [Online]. Available: https://www.instructables.com/id/Hack-your-vehicle-CAN-BUS-with-Arduino-and-Seeed-C/. [Accessed: 22-May-2019].

[6] J. Markoff, “Researchers Hack Into Cars’ Electronics,” *The New York Times*, 09-Mar-2011.

[7] S. Checkoway *et al.*, “Comprehensive Experimental Analyses of Automotive Attack Surfaces,” p. 16.

[8] C. Miller and C. Valasek, “A Survey of Remote Automotive Attack Surfaces,” p. 94.

[9] “CAN Message Injection - Paper,” *Duo Security Community*, 15-Aug-2016. [Online]. Available: https://community.duo.com/t/can-message-injection-paper/269. [Accessed: 22-May-2019].

[10] E. J. Markey, “All Info - S.1806 - 114th Congress (2015-2016): SPY Car Act of 2015,” 21-Jul-2015. [Online]. Available: https://www.congress.gov/bill/114th-congress/senate-bill/1806/all-info. [Accessed: 22-May-2019].

[11] "Intrusion detection system," *Wikipedia*. 18-May-2019.

[12] C. Douligeris and D. N. Serpanos, *Network Security: Current Status and Future Directions*. John Wiley & Sons, 2007.

[13] R. A. Sadek, M. S. Soliman, and H. S. Elsayed, "Effective Anomaly Intrusion Detection System based on Neural Network with Indicator Variable and Rough set Reduction," vol. 10, no. 6, p. 7, 2013.

[14] Chris. Sanders and Jason. Smith, *Applied network security monitoring: collection, detection, and analysis*, 1 online resource (497 pages) vols. Waltham, MA: Syngress, 2013.

[15] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures," *Reliab. Eng. Syst. Saf.*, vol. 96, no. 1, pp. 11–25, Jan. 2011.

[16] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, 2015, pp. 45–49.

[17] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *2016 International Conference on Information Networking (ICOIN)*, 2016, pp. 63–68.

[18] M. W. Spicer, "Intrusion Detection System for Electronic Communication Buses: A New Approach," Thesis, Virginia Tech, 2018.

[19] D. K. Vasistha, "Detecting Anomalies in Controller Area Network for Automobiles," Thesis, 2017.

[20] S. Lokman, A. T. B. Othman, and M. Abu-Bakar, "Optimised Structure of Convolutional Neural Networks for Controller Area Network Classification," in *2018 14th International*

Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2018, pp. 475–481.

[21] "CAN bus," *Wikipedia*. 06-May-2019.

[22] "CAN in Automation (CiA): History of the CAN technology." [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-history/. [Accessed: 10-May-2019].

[23] "Mercedes-Benz S-Class W 140." [Online]. Available: https://www.mercedes-benz.com/content/com/en/mercedes-benz/classic/history/mercedes-benz-s-class-w-140. [Accessed: 22-May-2019].

[24] "can-newsletter.org - Applications." [Online]. Available: https://can-newsletter.org/engineering/applications/160322_25th-anniversary-mercedes-w140-first-car-with-can/. [Accessed: 22-May-2019].

[25] "CAN protocol license," *Bosch Semiconductors*. [Online]. Available: http://www.bosch-semiconductors.com/ip-modules/can-ip-modules/can-protocol/. [Accessed: 22-May-2019].

[26] N. Carter, "Fixed Wireless Broadband Blog | Accel Networks: What is Layer 2, and Why Should You Care?" [Online]. Available: https://web.archive.org/web/20100218075030/http:/www.accelnetworks.com/blog/2009/09/what-is-layer-2-and-why-should-you-care.html. [Accessed: 22-May-2019].

[27] B. Fair, "The Physical Layer," *InterWorks*, 30-Jul-2011. .

[28] "CAN Physical Layer Standards: High-Speed vs. Low-Speed/Fault-Tolerant CAN - National Instruments." [Online]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019LzHSAU&l=en-US. [Accessed: 22-May-2019].

[29] “Wayback Machine,” 11-Dec-2015. [Online]. Available: https://web.archive.org/web/20151211125301/http:/www.boschsemiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can_fd_spec.pdf. [Accessed: 22-May-2019].

[30] Young EC, *The Penguin Dictionary of Electronics*, 1988, ISBN 0-14-051187-3 11-Apr-2019.

[31] E. C.-C. Magnetics, “Common Mode Signals vs. Differential Mode Signals | Wurth Electronics Midcom.” [Online]. Available: https://www.we-online.com/web/en/passive_components_custom_magnetics/blog_pbcm/blog_detail_electronics_in_action_44030.php. [Accessed: 29-Dec-2016].

[32] U. Mike1024, *This is a 9 pin d-sub connector commonly called a DB-9, or more properly called a DE-9. This is the male end of a cable; this end would typically plug into non-PC hardware in a PC-to-other-hardware arrangement.* 2005.

[33] “can-connectors-1.gif (600×575).” [Online]. Available: https://www.kvaser.com/wp-content/uploads/2014/01/can-connectors-1.gif. [Accessed: 22-May-2019].

[34] “A Brief Introduction to Controller Area Network.” [Online]. Available: https://copperhilltech.com/a-brief-introduction-to-controller-area-network/. [Accessed: 22-May-2019].

[35] S. Corrigan, “Introduction to the Controller Area Network (CAN),” May-2016. [Online]. Available: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf. [Accessed: 22-May-2019].

[36] B. Kinicki, “Bit and Byte Stuffing,” p. 13. 24-Mar-2019.

[37] mbedlabs Technosolutions, “CAN Bus,” 03:18:24 UTC.

[38] M. D. Natale, “Understanding and using the Controller Area Network,” p. 47, Oct. 2008.

[39] “PCAN-View: PEAK-System.” [Online]. Available: https://www.peak-system.com/PCAN-View.242.0.html?&L=1. [Accessed: 22-May-2019].

[40] “Time-to-digital converter,” *Wikipedia*. 06-May-2019.

[41] T. Instruments, “TDC7201 Time-to-Digital Converter for Time-of-Flight Applications in LIDAR, Range Finders, and ADAS.pdf.” [Online]. Available: http://www.ti.com/lit/ds/snas686/snas686.pdf. [Accessed: 23-May-2019].

[42] G. Roberts and M. Ali-Bakhshian, “A Brief Introduction to Time-to-Digital and Digital-to-Time Converters,” *Circuits Syst. II Express Briefs IEEE Trans. On*, vol. 57, pp. 153–157, Apr. 2010.

[43] U. eckenheimer, *English: circuit diagram of a tapped delay line*. 2012.

[44] “Optical Supplemental Navigation Device.” [Online]. Available: http://www.eecs.ucf.edu/seniordesign/fa2016sp2017/g24/doc.html. [Accessed: 23-May-2019].

[45] D. M. Santos, S. F. Dow, and M. E. Levi, “A CMOS delay locked loop and sub-nanosecond time-to-digital converter chip,” in *1995 IEEE Nuclear Science Symposium and Medical Imaging Conference Record*, 1995, vol. 1, pp. 289–291 vol.1.

[46] A. H. Chan and G. W. Roberts, “A jitter characterization system using a component-invariant Vernier delay line,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 12, no. 1, pp. 79–95, Jan. 2004.

[47] P. Chen, Shen-Luan Liu, and Jingshown Wu, “A CMOS pulse-shrinking delay element for time interval measurement,” *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, vol. 47, no. 9, pp. 954–958, Sep. 2000.

[48] M. K. Mandal and B. C. Sarkar, “Ring oscillators: Characteristics and applications,” *APPL PHYS*, vol. 48, p. 10, 2010.

[49] J. Kalisz, “Review of methods for time interval measurements with picosecond resolution,” *Metrol. Metrol.*, vol. 41, pp. 17–32, Feb. 2004.

[50] J. Kalisz, M. Pawlowski, and R. Pelka, “Error analysis and design of the Nutt time-interval digitizer with picoecond resolution,” *J. Phys. [E]*, vol. 20, p. 1330, Nov. 2000.

[51] J. Kalisz, R. Pelka, and A. Poniecki, “Precision time counter for laser ranging to satellites,” *Rev. Sci. Instrum.*, vol. 65, pp. 736–741, Apr. 1994.

[52] “TDC7201: TDC7201 question - Sensors forum - Sensors - TI E2E Community.” [Online]. Available: https://e2e.ti.com/support/sensors/f/1023/t/584424?TDC7201-TDC7201-question. [Accessed: 23-May-2019].

[53] N. S. Altman, “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression,” *Am. Stat.*, vol. 46, no. 3, pp. 175–185, 1992.

[54] E. M. Knorr, R. T. Ng, and V. Tucakov, *Distance-Based Outliers: Algorithms and Applications*. 2000.

[55] S. Ramaswamy, R. Rastogi, and K. Shim, “Efficient Algorithms for Mining Outliers from Large Data Sets,” in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2000, pp. 427–438.

[56] F. Angiulli and C. Pizzuti, “Fast Outlier Detection in High Dimensional Spaces,” in *Principles of Data Mining and Knowledge Discovery*, 2002, pp. 15–27.

[57] D. Coomans and D. L. Massart, “Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-Nearest neighbour classification by using alternative voting rules,” *Anal. Chim. Acta*, vol. 136, pp. 15–27, Jan. 1982.

[58] T. G. Siong, “classification - How Does Weighted KNN Work?,” *Data Science Stack Exchange*. [Online]. Available: https://datascience.stackexchange.com/questions/42376/how-does-weighted-knn-work. [Accessed: 23-May-2019].

[59] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, “Miscellaneous Clustering Methods,” in *Cluster Analysis*, 5th ed., John Wiley & Sons, Ltd, 2011, pp. 215–255.

[60] T. Srivastava, “Introduction to KNN, K-Nearest Neighbors : Simplified,” *Analytics Vidhya*, 25-Mar-2018. .

[61] A. Bronshtein, “A Quick Introduction to K-Nearest Neighbors Algorithm,” *Noteworthy - The Journal Blog*, 11-Apr-2017. [Online]. Available: https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7. [Accessed: 23-May-2019].

[62] G. Zaccone, “Multi Layer Perceptron - Getting Started with TensorFlow [Book].” [Online]. Available: https://www.oreilly.com/library/view/getting-started-with/9781786468574/ch04s04.html. [Accessed: 23-May-2019].

[63] J. R. Shewchuk, “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain,” Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[64] A. Ng, “Backpropagation Algorithm,” *Coursera*. [Online]. Available: https://www.coursera.org/learn/machine-learning/home/welcome. [Accessed: 23-May-2019].

[65] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[66] K.-T. Cho and K. G. Shin, “Fingerprinting Electronic Control Units for Vehicle Intrusion Detection,” presented at the 25th {USENIX} Security Symposium ({USENIX} Security 16), 2016, pp. 911–927.

[67] “TDC7201-ZAX-EVM User’s Guide (Rev. A).pdf.” [Online]. Available: http://www.ti.com/lit/ug/snau198a/snau198a.pdf. [Accessed: 23-May-2019].

[68] "EPOS2 50/5 Getting Started," p. 36, 2016.

[69] "HEDS-9000/9100: Two Channel Optical Incremental Encoder Modules Data Sheet," p. 10.

[70] "Servo Components - Pittman Motors & More," *Servo Components Website*. [Online]. Available: http://www.servocomponents.com. [Accessed: 23-May-2019].

[71] "EPOS2 50/5 Hardware Reference," p. 50, 2014.

[72] H. Landman, "TDC7201-ZAX-EVM: Is there a datasheet for the board? - Sensors forum - Sensors - TI E2E Community." [Online]. Available: https://e2e.ti.com/support/sensors/f/1023/t/629138. [Accessed: 23-May-2019].

[73] "Normality Testing - Skewness and Kurtosis - Documentation." [Online]. Available: https://help.gooddata.com/doc/en/reporting-and-dashboards/maql-analytical-query-language/maql-expression-reference/aggregation-functions/statistical-functions/predictive-statistical-use-cases/normality-testing-skewness-and-kurtosis. [Accessed: 23-May-2019].

[74] "Cross-validated k-nearest neighbor classifier - MATLAB crossval," *Mathworks*. [Online]. Available: https://www.mathworks.com/help/stats/classificationknn.crossval.html. [Accessed: 23-May-2019].

[75] "Cross-validated classification model - MATLAB," *Mathworks*. [Online]. Available: https://www.mathworks.com/help/stats/classreg.learning.partition.classificationpartitionedmodel-class.html. [Accessed: 23-May-2019].

[76] "Cross-validation loss of partitioned regression model - MATLAB," *Mathworks*. [Online]. Available: https://www.mathworks.com/help/stats/regressionpartitionedmodel.kfoldloss.html. [Accessed: 23-May-2019].

[77] H. Mahmood, “Activation Functions in Neural Networks,” *Towards Data Science*, 31-Dec-2018. [Online]. Available: https://towardsdatascience.com/activation-functions-in-neural-networks-83ff7f46a6bd. [Accessed: 23-May-2019].

[78] C.-F. Wang, “The Vanishing Gradient Problem,” *Towards Data Science*, 08-Jan-2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. [Accessed: 06-May-2019].

[79] D. Maker, “machine learning - What are the advantages of ReLU over sigmoid function in deep neural networks?,” *Cross Validated*. [Online]. Available: https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks. [Accessed: 23-May-2019].

[80] A. S. Walia, “Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent,” *Towards Data Science*, 10-Jun-2017. [Online]. Available: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f. [Accessed: 23-May-2019].

Appendix A

Here we present some results on hypothesis testing for our simulation experiment described in Section 4.3. The goal of this testing to check if we can conduct some hypothesis test to verify if classification is achievable with just a few samples of test data, using statistical methods as opposed to machine learning techniques like k-nearest neighbors and neural networks which may be computationally expensive. We use MATLAB to perform these tests. First, we perform the Z-test in order to see if it is possible to classify the mean of few samples of test data into the correct population distribution amongst the 6 microcontrollers. After, testing for the number of samples required it was seen that with 100 samples of test data it was possible to achieve correct classification. The hypothesis and results are as shown below.

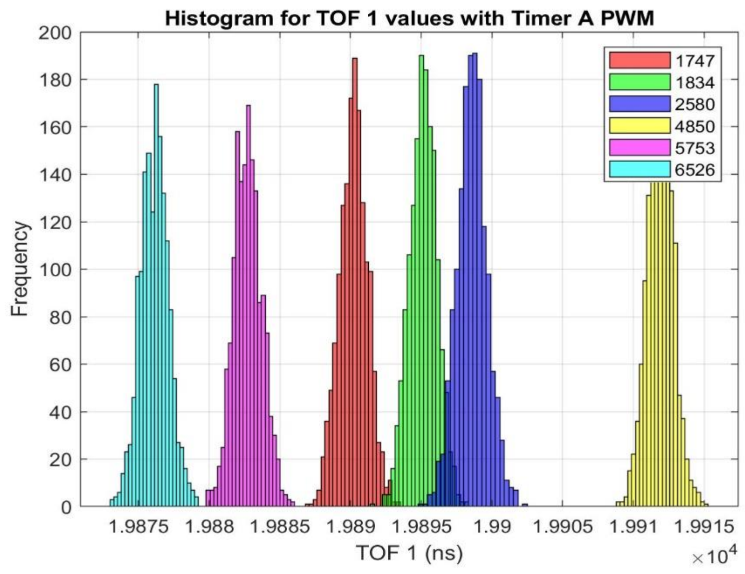


Figure A.1: Histogram for Time of Flight values from simulation experiment

Time (ns)	6526	5753	1747	1834	2580	4850
Mean	19876.15	19882.72	19890.22	19895.14	19895.96	19911.91
Std dev	0.990	0.995	0.998	0.967	1.060	0.936

Table A.1: Mean and standard deviation values for histograms from simulation experiment

Hypothesis:

For any test case, our hypothesis are as follows:

H_o: sample mean is representative of the population mean

H_α: sample mean is not representative of the population mean

The function ztest used for testing this hypothesis has inputs (in MATLAB) as follows:

[h(i),pval(i),ci(:,i)] = ztest(mean_test, mean_all_microcontrollers (i), std_dev_all_microcontrollers (i)), where 'i' is meant for a 'for' loop

The mean vector is as follows:

Mean_all_microcontrollers = [m_1747, m_1834, m_2580, m_4850, m_5753, m_6526]

A 'h' value of 1 means we reject the null hypothesis and a 'h' value of 0 means we do not reject the null hypothesis.

Result 1: Test case 1834 (100 samples)

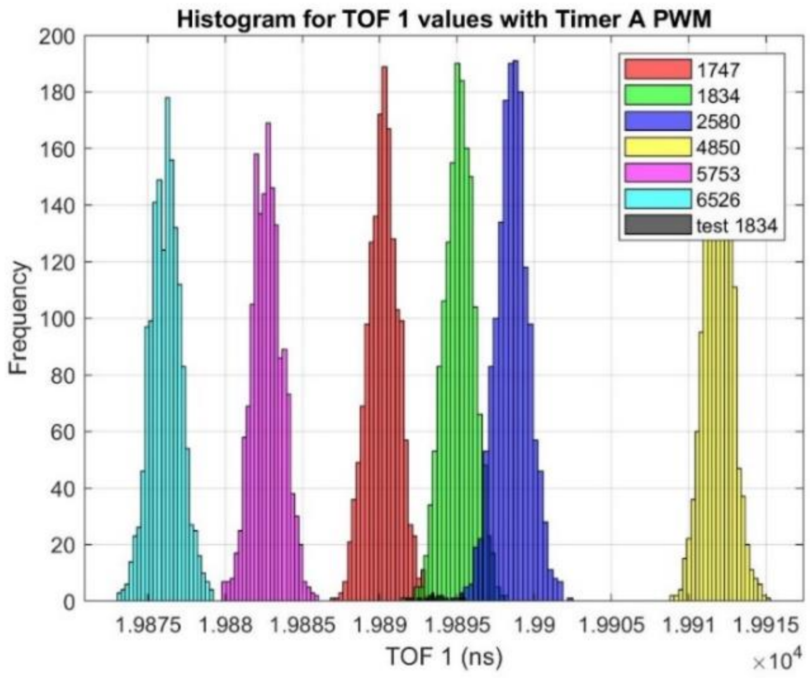


Figure A.2: Test case 1834 histogram for time of flight values overlaid on simulation histograms

Mean	19894.18
Std dev	0.926

Table A.2: Mean and standard deviation for test case 1834 data

The following results are obtained using the ztest function in MATLAB.

h	1 0 1 1 1 1

Table A.3: H values for test case 1834 under hypothesis test

It can be seen from the above results that we do not reject the null for 1834 microcontroller meaning that sample mean for test case of 1834 could be a representative of the population mean for 1834 microcontroller.

The pvalue is as follows:

Pval	0.319

Table A.4: p-value for test case 1834 under hypothesis test

A p-value of <0.05 means we need to reject the null hypothesis.

The confidence interval is as follows:

CI	19892.282
	19896.073

Table A.5: Confidence Interval for test case 1834 under hypothesis test

It can be seen that the confidence interval covers the actual mean of the population i.e. 19895.14 ns.

We can see that doing the above hypothesis testing we can classify the incoming test data with just 100 test sample data.

Result 2: Test case 6526

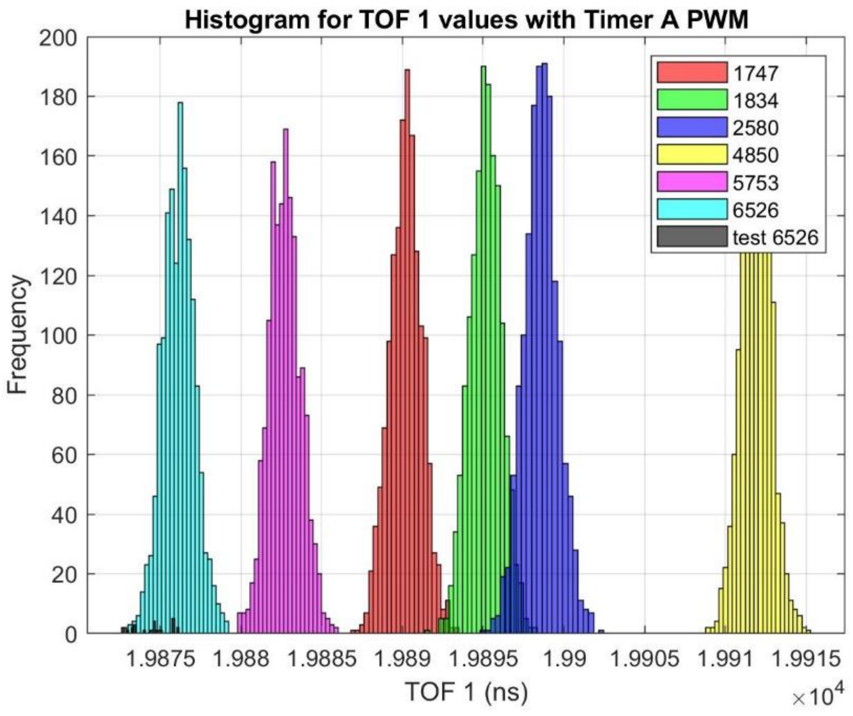


Figure A.3: Test case 6526 histogram for time of flight values overlaid on simulation histograms

Mean	19874.622
Std dev	0.984

Table A.6: Mean and standard deviation for test case 6526 data

H	1 1 1 1 1 0
P-val	0.1225
CI	19872.68
	19876.56

Table A.7: H values, P-values and Confidence Interval for test case 6526 under hypothesis test

Again, p-value is not < 0.05 and confidence interval covers the population mean of 19876.15 ns.

Result 3: Test case 2580

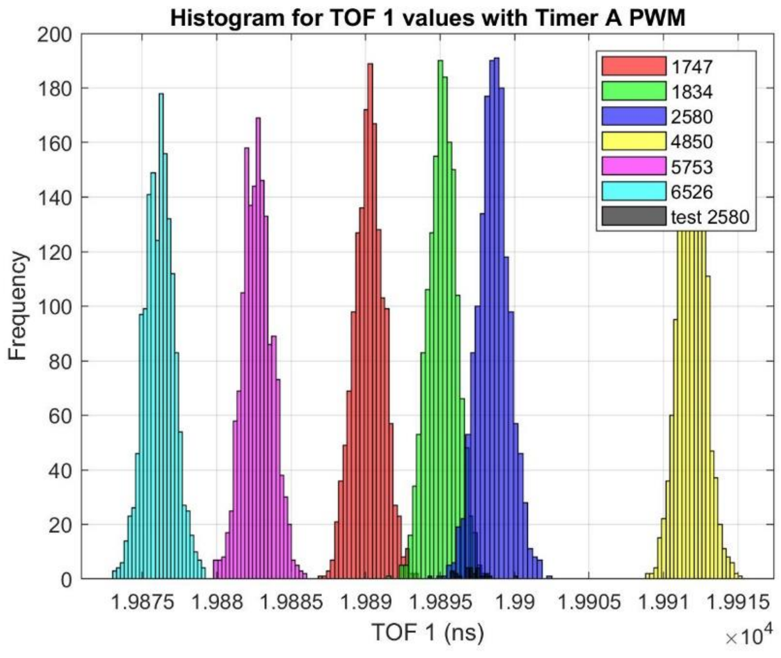


Figure A.4: Test case 2580 histogram for time of flight values overlaid on simulation histograms

Mean	19897.39
Std dev	1.017

Table A.8: Mean and standard deviation for test case 2580 data

H	1 1 0 1 1 1
P-val	0.2587
CI	19895.32
	19899.48

Table A.9: H values, P-values and Confidence Interval for test case 2580 under hypothesis test

Again, p-value is not < 0.05 and confidence interval covers the population mean of 19895.96 ns.

Lastly, we will perform the chi-squared variance test for comparing the sample variance to the population variance. However, given that all the test cases have a variance in the same range as all the microcontrollers, we do not obtain any classification advantage.

The function used is vartest in MATLAB to perform this test and the results are as shown.

```
h =
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

The reason for equivalent variance is due to the fact that we might be actually measuring the jitter in the TDC combined with the uncertainty of all the microcontrollers. Since we use the same instrument to measure the signals across all microcontrollers of similar type, the error amount to be the same throughout and hence the same variance.