

THESIS

SECURE REMOTE SENSOR SIMULATOR FOR HEAVY VEHICLE ELECTRONIC  
CONTROL UNITS

Submitted by

Ram Rohit Gannavarapu

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2022

Master's Committee:

Advisor: Edwin Chong

Jeremy Daily

Sudeep Pasricha

Copyright by Ram Rohit Gannavarapu 2022

All Rights Reserved

## ABSTRACT

### SECURE REMOTE SENSOR SIMULATOR FOR HEAVY VEHICLE ELECTRONIC CONTROL UNITS

Heavy Vehicle Event Data Recorders (HVEDRs) have the capability to record crash-related data and are valuable tools for traffic crash investigators. The data extracted from HVEDRs contain information to help reconstruct the driver's behaviors and determine the events leading to a crash. Data extraction is commonly performed using diagnostic tools when the electronic control unit (ECU) with the HVEDR is available on the vehicle's network. In the cases where the electrical system of the vehicle is compromised, the ECU is often removed and connected to a harness for power and communications. These harnesses are not designed to preserve fault codes or diagnostic trouble codes which can result in overwriting data related to a particular crash event.

This thesis describes the open-source hardware and software design of a remotely accessible sensor simulator used to create a fault-free environment for a bench download of an HVEDR. The sensor simulator device reduces the chance of any alteration of the original fault code data inside the HVEDRs by emulating the presence of actuators and sensors to the ECU. It does this using analog voltage outputs, pulse-width modulated signals, digital potentiometers, and CAN messages. The settings for these are adjustable remotely through a web-based interface.

A contribution of the thesis focuses on a process to increase the security posture of the embedded IoT devices wherein it utilizes a hardware security module to offload cryptography operations. The hardware security module was also used for secure key storage and implement Elliptic Curve Digital Signature Algorithm (ECDSA) to sign and verify messages for integrity, which is a key process in Transport layer security (TLS). The device also securely connects to a cloud infrastructure using TLS, enabling investigators to operate these devices remotely using a web-based

graphical user interface. Secure remote access enables further research and investigation of heavy vehicle electronic systems.

## ACKNOWLEDGEMENTS

Throughout the writing of this thesis, I have received a great deal of assistance. I would like to first thank my mentor, Dr. Jeremy Daily, who has guided me through this project with his valuable knowledge and dedication. I would also like to thank Dr. Edwin Chong, Dr. Sudeep Pasricha for being on my thesis committee. Thanks to all my research colleagues who have helped me throughout the project. More importantly, I would like to thank my parents and my sister for their constant support throughout. I would like to thank Colorado State University for providing the facilities and resources to conduct the research.

## DEDICATION

*I would like to dedicate this thesis to my late grandfather Addanki Marthanda Manikya Sharma.*

## TABLE OF CONTENTS

ABSTRACT . . . . .	ii
ACKNOWLEDGEMENTS . . . . .	iv
DEDICATION . . . . .	v
LIST OF TABLES . . . . .	ix
LIST OF FIGURES . . . . .	x
Chapter 1      Introduction . . . . .	1
1.1          Background . . . . .	1
1.2          Related Research . . . . .	2
1.3          Objective . . . . .	3
1.4          Approach . . . . .	4
1.5          Background on Heavy Vehicle Systems . . . . .	6
1.5.1      Standard sensors found in heavy vehicles . . . . .	6
1.5.2      Creating a fault-free environment . . . . .	8
1.5.3      Data extraction from ECMs . . . . .	9
1.6          Contribution . . . . .	10
1.7          Organization of Thesis . . . . .	11
Chapter 2      System Design . . . . .	12
2.1          Requirements . . . . .	12
2.1.1      Fault free environment . . . . .	12
2.1.2      Remote connection . . . . .	13
2.1.3      Compatibility . . . . .	13
2.1.4      Cybersecurity requirements . . . . .	13
2.1.5      Provisioning . . . . .	13
2.1.6      Real-time feedback . . . . .	13
2.1.7      Open Source . . . . .	13
2.2          Functions . . . . .	14
2.2.1      Simulating Resistive sensors . . . . .	14
2.2.2      Generation of Analog DC Signals . . . . .	14
2.2.3      PWM signal generation . . . . .	15
2.2.4      Monitoring voltage on the output . . . . .	16
2.2.5      CAN Read/Write messages . . . . .	16
2.2.6      User Interface . . . . .	16
2.3          Functional Allocation . . . . .	17
2.3.1      Generating PWM signals . . . . .	17
2.3.2      Controlling digital potentiometers . . . . .	18
2.3.3      Communication over CAN bus . . . . .	18
2.3.4      Monitor Voltage . . . . .	18
2.3.5      Application Programming Interface . . . . .	19
2.3.6      User Interface . . . . .	19

2.4	Architecture . . . . .	19
2.5	Conclusion . . . . .	21
Chapter 3	Hardware Design . . . . .	22
3.1	Introduction . . . . .	22
3.2	Requirements . . . . .	22
3.3	Hardware Building Blocks . . . . .	23
3.4	Detailed Schematics . . . . .	26
3.4.1	Voltage Regulation and power protection . . . . .	29
3.4.2	Development Board . . . . .	30
3.4.3	Digital Potentiometers . . . . .	31
3.4.4	Pulse Width Modulated Signals . . . . .	35
3.4.5	Ethernet Module . . . . .	36
3.4.6	CAN Transceivers . . . . .	37
3.4.7	Crypto Trust Platform . . . . .	38
3.4.8	External Connections . . . . .	39
3.5	Enclosure and Printed Circuit Board . . . . .	40
3.6	Bill of Materials . . . . .	44
3.7	Functional Unit Tests . . . . .	45
3.7.1	Digital Potentiometer test . . . . .	45
3.7.2	PWM test . . . . .	47
3.7.3	Voltage monitoring test . . . . .	49
3.7.4	CAN Test . . . . .	52
3.7.5	ATECC608B test . . . . .	56
3.7.6	Ethernet test . . . . .	57
3.8	Hardware Design and Testing Summary . . . . .	60
Chapter 4	Software Design . . . . .	61
4.1	Introduction . . . . .	61
4.2	Requirements . . . . .	61
4.3	API Design . . . . .	61
4.3.1	PWM process flow . . . . .	65
4.3.2	Potentiometers process flow . . . . .	67
4.3.3	CAN Message generation process flow . . . . .	69
4.4	Graphical User Interface (GUI) . . . . .	71
4.4.1	Digital Potentiometers . . . . .	71
4.4.2	Pulse width Modulation . . . . .	72
4.4.3	CAN message viewer . . . . .	73
4.4.4	CAN Message Generator . . . . .	74
4.5	Functional Unit Tests . . . . .	76
4.5.1	GET method tests . . . . .	76
4.5.2	POST method tests . . . . .	82
4.6	Conclusion . . . . .	84
Chapter 5	Securing Cloud and External Communication . . . . .	85



5.1	Introduction . . . . .	85
5.2	Transport Layer Security 1.2 (TLS) . . . . .	85
5.2.1	ECDSA sign and verify test . . . . .	87
5.3	Implementation . . . . .	95
5.4	Cloud Communications Summary . . . . .	98
Chapter 6	Conclusion . . . . .	99
6.1	Limitations and Future work . . . . .	99
Bibliography	. . . . .	101

## LIST OF TABLES

3.1	Mini-SSS3 bill of materials(BOM) . . . . .	44
3.2	PAC1934 voltage measurement results . . . . .	51
3.3	Hardware test case summary . . . . .	60

## LIST OF FIGURES

1.1	V diagram for product development at the system level . . . . .	5
2.1	Two wire sensor configuration where terminal A is not connected. . . . .	14
2.2	Three wire sensor configuration . . . . .	15
2.3	Mini-SSS3 function allocation . . . . .	17
2.4	Mini-SSS3 architecture overview . . . . .	20
3.1	Different layers of hardware components . . . . .	24
3.2	High-level printed circuit board block diagram . . . . .	26
3.3	Mini-SSS3 schematic page-1 . . . . .	27
3.4	Mini-SSS3 schematic page-2 . . . . .	28
3.5	Power Regulation . . . . .	29
3.6	Power monitoring IC . . . . .	30
3.7	Primary Microcontroller . . . . .	31
3.8	MCP41HV51 Terminal connections . . . . .	32
3.9	Digital Potentiometers . . . . .	32
3.10	MCP41HV51 TCON register . . . . .	33
3.11	MCP41HV51 block diagram . . . . .	34
3.12	PWM Signal Generator Schematic . . . . .	35
3.13	operational amplifier configuration . . . . .	36
3.14	Mini-SSS3 Schematic for Wiznet850i Ethernet module . . . . .	37
3.15	Mini-SSS3 schematic for CAN Transceivers . . . . .	38
3.16	ATECC608A cryptographic co-processor . . . . .	38
3.17	Pin definitions for the external connections . . . . .	40
3.18	Mini-SSS3 Printed Circuit Board . . . . .	41
3.19	Assembled Mini-SSS3 circuit card assembly . . . . .	41
3.20	Mini-SSS3 enclosure Ethernet side . . . . .	42
3.21	Mini-SSS3 enclosure Molex side . . . . .	43
3.22	Mini-SSS3 digital potentiometer test . . . . .	47
3.23	Mini-SSS3 PWM signal generation test observations from the Saleae Logic analyzer. . . . .	48
3.24	Mini-SSS3 PWM signal frequency test . . . . .	49
3.25	Mini-SSS3 voltage monitor test . . . . .	52
3.26	Mini-SSS3 CAN message viewer test . . . . .	54
3.27	Mini-SSS3 CAN message generation test . . . . .	56
3.28	ATECC608 get serial number test . . . . .	57
3.29	Mini-SSS3 Ethernet test . . . . .	59
4.1	Mini-SSS3 HTTP API flow diagram . . . . .	63
4.2	PWM process flow diagram . . . . .	65
4.3	Potentiometers process flow diagram . . . . .	67
4.4	CAN generation flow diagram . . . . .	69
4.5	Digital Potentiometer GUI . . . . .	72

4.6	PWM GUI . . . . .	73
4.7	CAN viewer GUI . . . . .	74
4.8	CAN message generator GUI . . . . .	75
4.9	POST request serial monitor output . . . . .	83
5.1	TLS Handshake . . . . .	86
5.4	AWS Certificate Registration . . . . .	93
5.5	Certificate signing request test . . . . .	95
5.6	AWS user login screen . . . . .	96
5.7	AWS user signup . . . . .	97
5.8	Access control with AWS Cognito user pool . . . . .	98

# Chapter 1

## Introduction

### 1.1 Background

A statistical projection of traffic fatalities shows that an estimated 2,811,185 people were involved in motor vehicle accidents in the United States in 2018 [1]. Different agencies, such as law enforcement, attorneys, and accident investigators, need to find the actual cause of those accidents. Determining critical information that happened just a few seconds before and during the accident is crucial. For passenger vehicles, Event Data Recorders (EDRs) can aid investigations of the causes of crashes [2]. For heavy vehicles, Heavy Vehicle Event Data Recorder (HVEDR) functionality is integrated into one or more Engine Control Modules (ECMs) [3].

Even though ECMs were initially designed to improve fuel efficiency and achieve optimal engine performance while meeting emission regulations, they also serve to store critical information such as vehicle speed, driving logs, diagnostic fault codes, brake status, and throttle position [4], [5]. Modern ECMs record event data related to diagnostic faults, hard or quick stops, and the vehicle's last or most recent stop [5]. This information is stored in a manner that allows reports to be generated and may be of value in an accident investigation and the data must be preserved in a forensically sound manner [6].

The early designs of the smart sensor simulator [7] were concentrated on understanding the different types of sensors and actuators that are needed to be simulated to create a fault free environment which would reduce the likelihood of data alterations during a bench download from a HVEDR. Córcega's work focused on validating this process on seven different ECM's and creating a fault free environment for those modules. Later a new version the smart sensor simulator 2 (SSS2) was designed as a commercial solution by Dr. Daily at Synercon Technologies. The SSS2 had some hardware improvements with a smaller form factor compared to the SSS. SSS2 also had a windows based graphical user interface application which communicated over USB. Those were

later open sourced and were made available at [8]. Both the SSS and SSS2 only communicated through USB and had an overhead of installing additional software to interface with them. In this thesis we also focus on creating a web based graphical user interface making it compatible with any operating system and removing the overhead of installing additional software/drivers. With the addition of a web interface and remote access, cybersecurity controls will need to be designed into simulator solutions going forward.

Internet of Things (IoT) devices are deployed in a variety of domains, including public and private networks. Cybersecurity is becoming critical to avoid the threat of leakage of sensitive information. Unfortunately, many low-end IoT devices become vulnerable due to the lack of implementation of security measures. A hacker who gains access to one device can replicate the same attack on the others. Because embedded device (IoT) firmware architects prioritize functionality over security, exploiting SPI flash and other device components is becoming common [9] [10]. That could be due to various factors requiring high processing power for performing cryptographic operations, which require high memory requirements and optimizing costs [11]. Once we have access to the hardware, it is possible to dump data from a serial flash or a JTAG debugger and gain access to stored secrets or code [9] [10]. Once someone gets access to this sensitive information, the whole network could be compromised. This thesis aims to utilize hardware security modules(HSMs), also commonly known as Trusted Platform Modules, to implement security measures on microcontrollers with comparatively low memory and processing power.

## **1.2 Related Research**

The preferred method for obtaining HVEDR records is to download the information directly from the vehicle via the in-cab Deutsche connector, using an RP1210 compliant Vehicle Diagnostic Adapter (VDA) and Original Equipment Manufacturer (OEM) software [3]. However, this cannot be accomplished in some instances due to the extent and nature of the damage to the vehicle, particularly if the electrical or communications networks have been compromised. Suppose this information cannot be downloaded directly from the truck. In that case, there are three alternative

methods for obtaining the data: using a surrogate vehicle, performing a benchtop download without a simulator harness, or performing a benchtop download with a simulator harness [5].

In the surrogate vehicle method, the ECM(s) are removed from the vehicle involved in the accident and placed into an undamaged vehicle of the same make and model. Finding a suitable surrogate can be difficult and is often only feasible for large fleet operators. Additionally, there is the opportunity cost of having the surrogate vehicle out of service while the ECMs are swapped, and the data is retrieved [12]. An alternate method is to perform a benchtop download, in which the event data is retrieved from the ECM(s) while out of the vehicle. When an ECM is powered on while disconnected from the vehicle, the absence of sensor inputs can create new fault codes; these new faults may overwrite potentially valuable information [13], [14]. A simulator harness can be connected to the ECM using actual and emulated truck components. However, these are typically limited to a specific truck configuration and are expensive [12]. Some simulator harness solutions, such as [15], can only simulate passive sensors, although many ECMs require active signals to be fault free [14].

A similar subsystem often called the "Truck-in-a-Box" (TIB), allows investigators to emulate sensors and actuators often seen in different engine configurations to create a fault-free environment [3]. The TIB renders all signals to the ECM by using the actual sensors or utilizing equivalent electrical components. The setup in [13] is used as a teaching tool for technical schools. It can also be used to simulate sensor malfunctions, typically seen in crash events, and to study the ECM's response without an actual truck.

### **1.3 Objective**

The initial motivation for the remote sensor simulator builds upon the work done by Corgega [7] where he initially designed a smart sensor simulator, a hardware tool for bench top downloads that is able to simulate both passive and active signals in order to create a fault-free environment. Later another version called the smart sensor simulator 2 (SSS2) was designed by Synercon technologies [16] making it a commercial product.

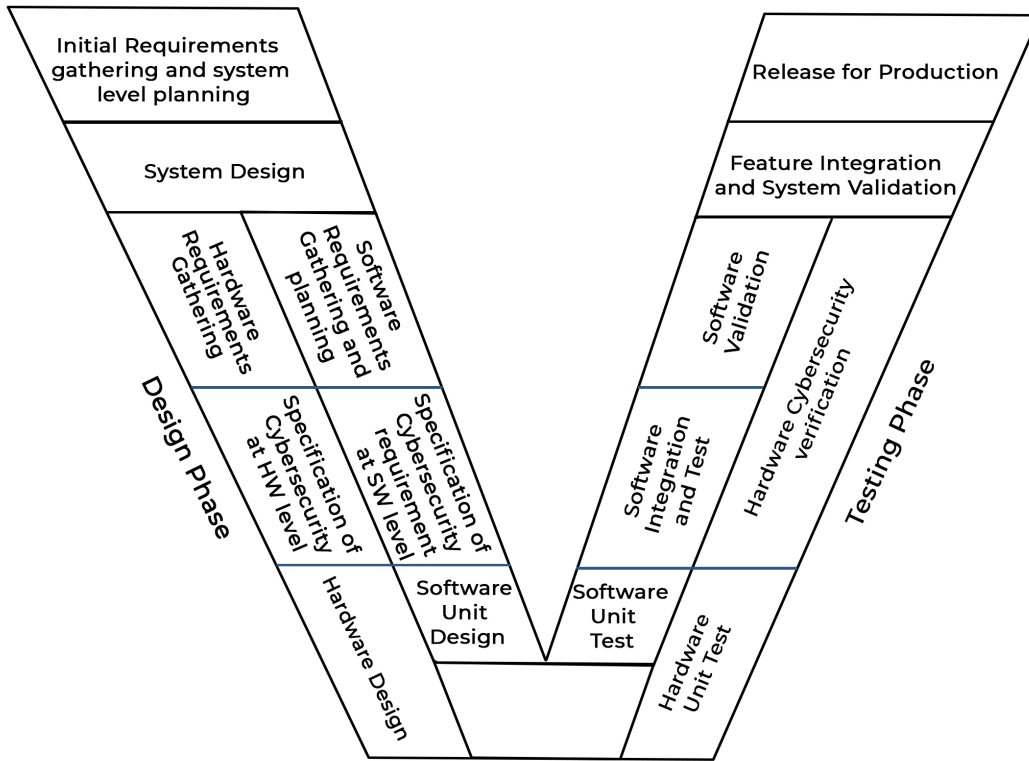
The remote sensor simulator or called as Mini Smart Sensor Simulator 3 (Mini-SS3) is designed to be general purpose and able to work with various ECMs - across different manufacturers but most importantly give the investigators an option to operate these devices remotely. Various passive and active signals are simulated via a combination of digital potentiometers and Pulse Width Modulated (PWM) outputs. Additionally, the Mini-SS3 supports common vehicle communications standards, such as Controller Area Network (CAN). The hardware is paired with software that enables users to simulate different sensors and actuators through a Graphical User Interface (GUI). Both the hardware and software solutions are open source and available at [17]. The key objective of the Mini-SS3 was to give the device a way to be controlled remotely in a secure manner.

## 1.4 Approach

Figure 1.1 shows a V-diagram followed during the product development of the Mini smart sensor simulator. Cryptography was given importance at every design, development, test, and validation stage following the ISO/SAE21434 standard [18]. During the initial phase of system design, we did system-level analysis to understand the initial requirements for the overall system and this analysis was then used to do a system level planning to include new features. Then, based on the resulting requirements the functionalities were further allocated to hardware and software design or both respectively. Further, Cybersecurity requirements were then drafted to make sure the communication between the IoT device (Mini-SS3) and cloud infrastructure secure. Finally, at the last the stage of the design process all the functionalities were translated into hardware and software components. This concludes the design phase of the Mini-SS3 system.

In the testing phase, initially each of the individual component were tested for validating their functionalities. Then, the whole system was integrated and tested again to make sure the functionalities were delivering the results as per the requirements. Finally, after the feature integration and system validation the Mini-SS3 was released for production.





**Figure 1.1:** V diagram for product development at the system level [19]

## 1.5 Background on Heavy Vehicle Systems

Electronic Control Units (ECUs) are designed for many tasks, including:

- Improve fuel efficiency.
- Meet emission regulations.
- Aid with troubleshooting and diagnostics.
- Protect the system/asset they monitor (i.e., Engine, Transmission, among others).

There are numerous sensors and switches throughout the truck that allow these ECUs to make decisions based on electrical inputs. The truck's battery and the ignition switch power these ECUs. After then, the ECU(s) will utilize this voltage as a reference for the 5V, 8V, and 12V sources used to power the various sensors and switches triggered by these units. J1708/J1587 [20] [21], or J1939 [27] are common protocols used by ECUs to connect with other modules. For example, the vehicle's diagnostic port provides access to at least one of these communication protocols

### 1.5.1 Standard sensors found in heavy vehicles

The ECM monitors engine sensors. Sensor outputs monitored by the ECM take many forms, including resistance, analog, Pulse Width Modulated (PWM), or a communication standard such as CAN, J1708, etc. The ECM is the most common ECU across all OEMs. This module contains all the timing calibrations needed to drive fuel injectors and maximize fuel efficiency [28]. It also monitors all the engine sensors and switches to protect this asset. There are at least two harnesses connected to an ECM. These harnesses are often referred to as vehicle side and engine side. Among the sensors connected to the engine side harness, there are:

- Engine Oil Pressure Sensor
- Intake Manifold Pressure Sensor
- Intake Manifold Temperature Sensor

- Barometric Pressure Sensor
- Engine Coolant Temperature Sensor
- Fuel Temperature Sensor
- Engine Oil Temperature Sensor
- Inlet Air Temperature Sensor
- Inlet Air Temperature Sensor
- Crankcase Pressure Sensor
- Ambient Air Temperature Sensor

These are commonly thermistors (temperature) or variable capacitance (pressure) sensors. Some sensors have built-in signal conditioning and provide a voltage output. Therefore, the signal expected by the ECM is within a voltage range, usually from 0.25V and 4.75V.

The engine harness is also connected to variable reluctance sensors that help the ECM determine what position the engine is in at any given time during its cycle. The ECM uses two sensors located near the crankshaft and camshaft. These sensors often include:

- Engine Position Sensor 1
- Engine Position Sensor 2
- Crankshaft position sensor
- Camshaft position sensor
- Timing reference sensor
- Synchronous reference sensor

The engine uses the input from these sensors to make decisions and drive different actuators such as:

- EGR Valve Motor
- Variable Geometry Turbocharger Actuator
- Engine compression Brake Solenoids
- Injectors
- Engine Cooling Fan Solenoid
- Stop Engine Lamp
- Check Engine Lamp

The vehicle side harness enables the ECM to monitor sensors and switches available on the truck's dashboard. This harness also houses all the communications and power terminals needed for the ECM to function. Through these communication lines, the ECM can obtain data, such as vehicle speed, from other modules. One example of a sensor connected to the vehicle side harness is the Pedal Position Sensor (PPS). This sensor can differ on the make and model of the vehicle and engine. For example, Caterpillar uses a PPS that produces a single channel PWM signal to the ECM. The PPS used by Detroit Diesel engines (from DDEC IV to DDEC X) utilizes a variable resistance sensor (potentiometer) which provides an analog voltage signal, typically from 0.25V to 5V, which the ECM reads to sense the position of the pedal.

Some actuators are controlled over CAN rather than an analog or PWM signal depending on the engine configuration. However, neither engine CAN nor any standard CAN network (SAE J1939) are used to actuate these sensors. Instead, modules such as the After-treatment Control Module (ACM) or MCM have dedicated CAN lines to control these controllers. In some cases, The device must emulate CAN frames in order to avoid fault codes.

## **1.5.2 Creating a fault-free environment**

Fault codes are generated when any sensors on the vehicle side or the engine side harness are missing. Missing sensors causes new diagnostic records to be written, and these new files

can potentially overwrite any existing records that may have collision-related data [13]. Existing literature showed that active fault codes on the Mercedes-Benz ECMs had overwritten existing diagnostic records on subsequent power cycles [13]. Hence its important to create a fault-free environment before the data is extracted from the HVEDR to reduce the chance of overwriting data.

### **1.5.3 Data extraction from ECMs**

Under normal circumstances, communication with the ECM is done over the diagnostic port, which is present inside the driver's cabin. This connector is often the standardized 9-pin connector described by the SAE J1939-13 standard [16]. This 9-pin connector is also called a Deutsch connector. The OEM software and an RP1210 compliant device that serves as a translator are necessary to establish communication between the computer and the truck.

When communications with the ECM(s) cannot be established through the diagnostic port, the ECM can be removed and a Direct to Module (DTM) or bench download is performed. This process requires the investigator to remove the engine and vehicle side harness and retrieve the ECM from the truck. Using an OEM Reprogramming Harness and an external source of power, the download can be performed. However, this method will certainly introduce new diagnostic fault codes that will alter the original image of the ECM and could potentially overwrite crash-related records. A bench download can also be done without an OEM reprogramming harness.

Instead of using a reprogramming harness to perform a bench download, sockets crimped to wires are used to connect to the ECM. Opposite to the crimped end, banana plugs are used to connect to a breakout box. A power source and RP1210 compliant device connect to this breakout box to power the system and carry out the download. Similar to the bench download using a reprogramming harness, this download will also introduce diagnostic fault codes. Moreover, the sockets are not mechanically fastened, which risks losing electrical continuity with the ECM's terminals.

In cases when a bench download is required, a fault-free environment must be created to guarantee that all data records available within the ECM are preserved. Based on the practices for ECM downloads shown in the previous section, the only way to achieve a fault-free environment is to simulate the different sensors and actuators available to the ECM through both the engine and vehicle side harnesses. A miniature Smart Sensor Simulator 3 (Mini-SS3) was designed to be able to perform this task.

The Mini-SS3 integrates the adaptability of the TIB and the ability to repeat an identical bench download of the fault-free cables described in previous sections. Even though the Mini-SS3 allows the investigator to emulate a great variety of sensors, actuators, switches, and modules, the fault-free environment must be constructed before the actual forensic download is performed. The crash-related events stored in the ECM can be overwritten seconds after a fault code is set. Therefore, investigators must follow a proper method to create a fault-free environment must be followed to minimize opportunities for data tampering. Also, due to the Mini-SS3's smaller size, multiple units may be necessary to simulate all the sensors necessary for an actual engine control module.

## **1.6 Contribution**

The key contributions of this thesis involve:

1. Documentation of the hardware and software design of the Mini Smart Sensor Simulator (Mini-SS3).
2. creating a web based graphical user interface for the Mini-SS3 making it a standalone device without the need of installing additional drivers or software.
3. Implementing a voltage feedback loop and providing users with a real time feedback of the current state of the device.
4. Implementing cybersecurity protocols on memory-constrained micro-controllers with the purpose of increasing security posture.

5. Connecting the sensor simulator to cloud infrastructure allowing users to control the sensor simulator remotely.

## 1.7 Organization of Thesis

The thesis is divided into five chapters:

- Chapter 1 provides a basic introduction to the project, literature review of related researches, objective and motivation of the project, background on sensors on a heavy vehicle, an approach to achieve the objective, and the contribution.
- Chapter 2 provides the hardware design, which lists the project requirements, system block diagram, detailed component schematics showing how the electrical components are connected, housing and printed circuit board (PCB) layout displaying the placement of those components on the device, bill of materials (BOM) listing all required parts, and results from hardware functional tests.
- Chapter 3 reviews the software design, which consists of the process overview indicating the interactions between all the system components. The chapter also talks about the embedded firmware of the device, the interface of the cloud services for each of the operation modes.
- Chapter 4 talks about securely connecting the sensor simulator to third-party cloud services such as Amazon Web Services utilizing the ATECC608 cryptographic co-processor.
- Chapter 5 concludes the thesis with a restatement of the abstract, contribution and lists some future works for project improvement.

# Chapter 2

## System Design

### 2.1 Requirements

To create a fault-free environment, the device should simulate a wide range of sensors and actuators. The solution should also be highly generalized and universal, such that a wide variety of vehicle makes and models are supported. Allowing most of the configuration to happen via software makes the solution more user-friendly and limits room for user error - a configuration representing a specific vehicle make, and model can be saved and reloaded as needed. While some requirements have not been vetted against industry standards, they have worked for laboratory uses. A few major requirements are summarized below.

#### 2.1.1 Fault free environment

All sensors, actuators, switches, and other modules required for a heavy vehicle control unit to behave as if they were connected to an actual sensor must be emulated by the system to ensure a fault-free environment. Most sensors that are present in any commercial truck can be classified into the following categories:

- Two-wire sensors
- Three-wire sensors
- Pulse Width Modulation
- Actuators or Solenoids to +12V
- Actuators or Solenoids to Ground
- Switches



### **2.1.2 Remote connection**

The next primary requirement is that users can securely connect to cloud infrastructure and control these devices remotely. The system should have an interface to connect to the internet.

### **2.1.3 Compatibility**

The system shall be backward compatible with the pinouts from the earlier versions of the smart sensor simulator. This allows re-use of the Molex connectors built for various ECM's.

### **2.1.4 Cybersecurity requirements**

The system shall have a way to securely store private keys to maintain confidentiality. The system shall also have integrity of the keys stored on these devices and while also ensuring the uniqueness of those keys.

### **2.1.5 Provisioning**

The system shall have a provisioning process that shall involve configuring and locking the hardware security modules. This process also involves registering certificates from the device with the cloud provider.

### **2.1.6 Real-time feedback**

The system shall have a way to provide real-time feedback to the user about changes taking place on the device. The settings on the device must be accurately reflected in a user interface.

### **2.1.7 Open Source**

An open-source solution that allows third parties to investigate and validate the underlying source code lends credibility and transparency to the process and bolsters the forensic soundness of the results.

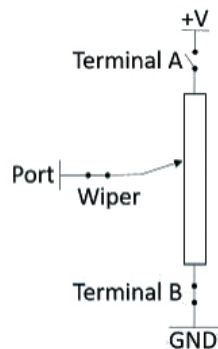
## 2.2 Functions

Based on the system requirements a few key functionalities for the Mini-SSS3 are described in the below sections.

### 2.2.1 Simulating Resistive sensors

#### Two-wire sensors

Two-wire sensors are loop-powered and do not require a separate supply voltage. The sensor may be read via a Wheatstone bridge in the ECU for resistance-based sensors, such as thermistors. Potentiometers can be used to create arbitrary resistance values to emulate these by disconnecting terminal A and connecting terminal B to the ground, as shown in Figure 2.1. By modifying the wiper position we can change the resistance value. This can be used to let the ECU detect current flow which is expected for some actuators or to emulate switches to ground, like idle validation.



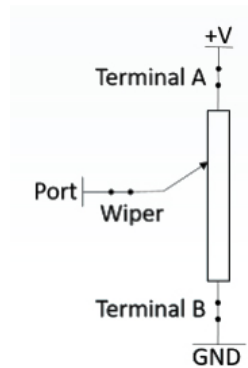
**Figure 2.1:** Two wire sensor configuration where terminal A is not connected.

### 2.2.2 Generation of Analog DC Signals

A potentiometer is a variable resistor with three terminals, a knob or slider, which can be moved or rotated to vary the resistance between the middle terminal and either of the remaining terminals. Digital potentiometers have similar functionality, but instead of a mechanical slider or knob, the resistance can be controlled using digital signals.

### Three-wire sensors

Three wire sensors usually have one additional terminal connected to supply voltage compared to a two wire sensor. Modifying the wiper position creates a voltage divider which provides analog voltages on the wiper connector. By modifying the wiper position The voltage divider can produce a desired analog voltage. A schematic of a three-wire sensor emulator can be found in Figure 2.2.



**Figure 2.2:** Three wire sensor configuration

### 2.2.3 PWM signal generation

An anti-lock braking system (ABS) is a common feature in most vehicles. This prevents the wheels from locking up during a hard brake event. Wheel speed sensors play an essential role in providing information to the ABS and other systems such as traction control and stability control. The wheel speed sensors on most vehicles are magnetic and generate an alternating current signal or a variable DC signal, that changes its frequency and amplitude based on the wheel speed. The teeth of the tone ring rotate and changes the magnetic field around the sensor, which induces a current in the sensor. This results in a harmonic wave pattern that changes frequency with wheel speed. Generating periodic waves in the digital world is only possible through pulse-width modulated signals. To be able to simulate rotary sensors the system needs to have a functionality to generate PWM signals. The system shall also provide options to adjust PWM parameters such as the duty cycle and frequency.

## **2.2.4 Monitoring voltage on the output**

Based on the Requirement 2.1.6 the system shall have a periodic function that updates the voltages of the output pins. This information can be helpful for the user to have real-time feedback on the changes they perform.

## **2.2.5 CAN Read/Write messages**

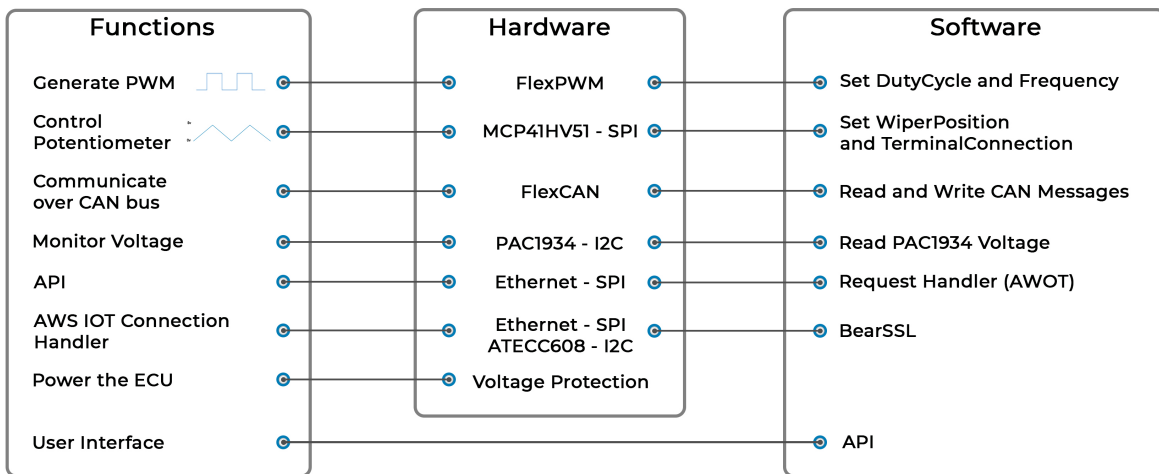
Modern vehicles have more than one controller module. An engine controller is solely responsible for looking after the engine's functionality, A brake controller looking after all the brake applications, and a body controller for controlling all the electronic components inside the cabin such as the switch states, air conditioning, etc. All these different modules communicate over the CAN. The system shall be able to send out periodic CAN messages simulating the presence of an electronic control module.

## **2.2.6 User Interface**

The users should have a way to interact and control various peripherals of the Mini SSS3 device. The user needs to be able to see the status of the different settings on the device and be able to read the output values on the different pins of the device. The user interface should also be accessible remotely.

## 2.3 Functional Allocation

After carefully analyzing the requirements and functions of the system, Teensy 4.0 development board was deemed a right fit for this application. Teensy 4.0 is an ARM Cortex-M7 based development board that works with the Arduino development environment. Some features that make the Teensy 4.0 an effective solution for the Mini-SS3 include three inbuilt CAN channels with one channel supporting CAN FD, seven serial, three SPI and three I2C interfaces. It also has 1984Kb Flash, 1024Kb RAM, 1Kb EEPROM (emulated). The Teensy 4.0 processor operates at 600 MHz, which is fast enough to support most requirements.



**Figure 2.3:** Mini-SS3 function allocation

The following sections describe how each functions discussed in Section 2.2 are allocated over hardware and software.

### 2.3.1 Generating PWM signals

PWM signals can be generated by toggling digital signals on and off. This could be achieved through software methods but a major disadvantage is that any interrupts will affect the timing, which can cause jitter in the signal. Many micro-controllers utilize hardware timers and registers to generate PWM signals which make them more timing critical and not have to depend on the

software execution. By manipulating the chip's timer registers directly, This provides more control than toggling digital signals through software.

The use of hardware based PWM signals allows the system to generate signals with high frequencies. The software allows us to then configure the hardware timers and registers for the desired frequency and duty cycle.

### **2.3.2 Controlling digital potentiometers**

Microchip's MCP41HV51 digital potentiometer were chosen for resistance based outputs that can tolerate voltages up to +18V. It has an 8-bit resolution for the wiper position giving us 256 distinct resistance values. The device also provides us with an option to connect and disconnect the other two terminals. This provides us the ability to emulate both 3-wire and 2-wire sensors as required in 2.2.1. The functionality of generating the DC signals has been offloaded to an external IC but configuration of the IC is controlled by the main microprocessor over SPI.

### **2.3.3 Communication over CAN bus**

The next major functionality for the device is to send and receive CAN messages from the CAN bus. The Teensy 4.0 has 3 CAN controllers, with one channel supporting CAN FD. An external CAN transceiver chip is added to complete the electrical interface between Teensy 4.0 and the external CAN bus. The Teensy 4.0's Flexible Controller Area Network (FLEXCAN) module is a communication controller that follows the CAN 2.0B protocol specification and implements the CAN protocol on hardware. The FLEXCAN module supports both standard and extended message frames. The FLEXCAN module supports 64 message buffers. The interface to the FLEXCAN modules is through memory mapped input/output.

### **2.3.4 Monitor Voltage**

To provide users with real-time feedback about the changes on the output terminals, Microchip's PAC1934 was incorporated in the design. The PAC1934 is a four-channel power/energy monitor that includes a current sensor amplifier and bus voltage monitors that feed high-resolution

ADCs. The embedded controller can retrieve bus voltage, sense shunt resistor voltage, and accumulated proportional power from registers over I2C. To reduce offset and gain errors, the PAC1934 uses real-time calibration.

### **2.3.5 Application Programming Interface**

An API allows other systems/users to interact with Mini-SS33 programmatically. HTTP API is a protocol that describes how a client can access information from the server. It works as a request-response protocol between a client and server. It specifies the types of calls or requests that can be made, how they should be made, the data formats that should be utilized, and the protocols that should be followed, among other things. An HTTP API deemed necessary for the Mini-SS33 design as it would help develop a user interface (UI) on top of it.

### **2.3.6 User Interface**

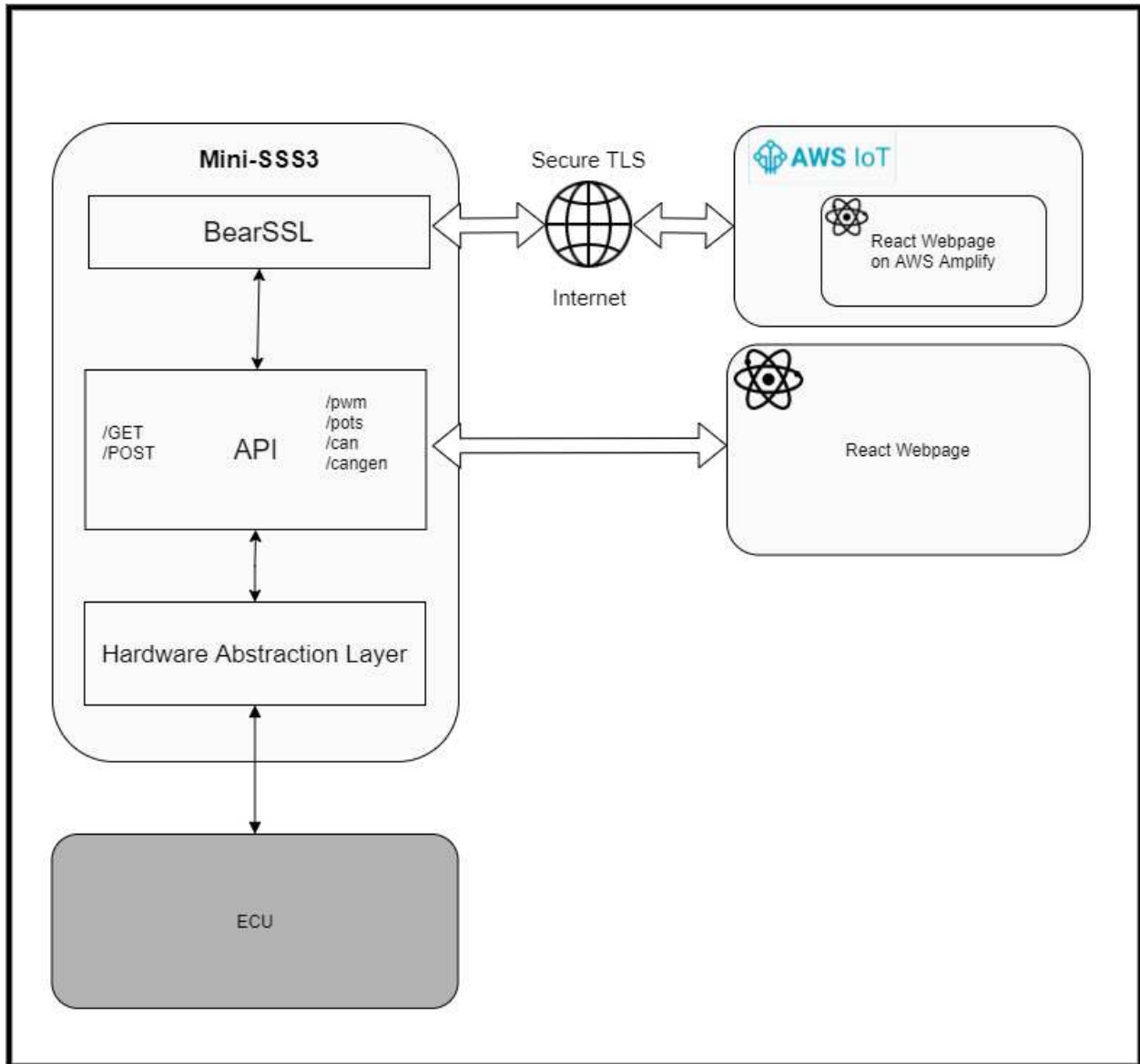
The graphical user interfaces builds upon the API to fetch and update state of the different peripherals on the Mini-SS33. The user interface is further discussed in Section 4.4. This is allocated to software only as it is portable and relies on the hardware for networking layers as a foundation.

## **2.4 Architecture**

The Mini-SS33 provides two modes of operations, one where the user can locally connect to the device and use the web-based front end served on the device to control all the peripherals of the Mini-SS33. In the second mode, the device can be operated remotely through the internet using AWS IoT. The second mode utilizes the Microchip's ATECC608 hardware security module, which involves an initial provisioning process. The provisioning process involves the generation of a certificate signing request (CSR) and registering it with AWS IoT. We assume that the following things are uncompromised to ensure the security and privacy of this model are intact.

1. The program used for the initial provisioning process

2. The Internet connection over TLS
3. The third-party cloud party (Amazon Web Services in this case)
4. The ATECC608 hardware device.



**Figure 2.4:** Mini-SS3 architecture overview

Figure 2.4 gives an overview of the system level architecture. The Hardware Abstraction layer controls the hardware peripherals like generating PWM signals, controlling potentiometers, read-



ing voltage values and, reading and writing CAN messages. The API layers acts like an interface for accessing the functionalities of the Mini-SS3. The react webpage interact with API to allow the users to control the device and, also enable the users to understand the current state of the device. BearSSL is used for implementing TLS which is essential for establishing secure communication with third party cloud servers to provide users with secure remote access.

## **2.5 Conclusion**

In this section the overall system requirements were gathered and the main functions of the Mini-SS3 were formulated. These functions were then allocated to hardware, software or both respectively. The following section dives deep into the hardware design of the Mini-SS3.

# Chapter 3

## Hardware Design

### 3.1 Introduction

In this section, the hardware design of Mini-SS3 will be explained in detail. Based on the functional allocations from the previous chapter, the hardware requirements of Mini-SS3 were gathered. Further, depending on the requirements, the hardware design of Mini-SS3 was formulated. A deep dive into the requirements, hardware components used, and their schematics will be discussed in detail in the upcoming sections.

### 3.2 Requirements

The major hardware requirements based on the system requirements are the following:

1. The device should utilize hardware timer peripherals to offload the generation of PWM signals from the main microprocessor and give the users an option to change both the duty cycle and the frequency.
2. Generate variable DC signals, emulating a three-wire sensor with support for varying voltage ranges from 0 to 5 volts.
3. The device should be able to emulate passive resistive sensors with resistance from 100 ohms to 200,000 ohms.
4. The device should read CAN messages at bus speeds of 125k, 250k, 500k, 666k, and 1M, giving the user an option to monitor messages on the CAN bus.
5. The sensor simulator should be able to generate an extended frame CAN message mimicking a control module.

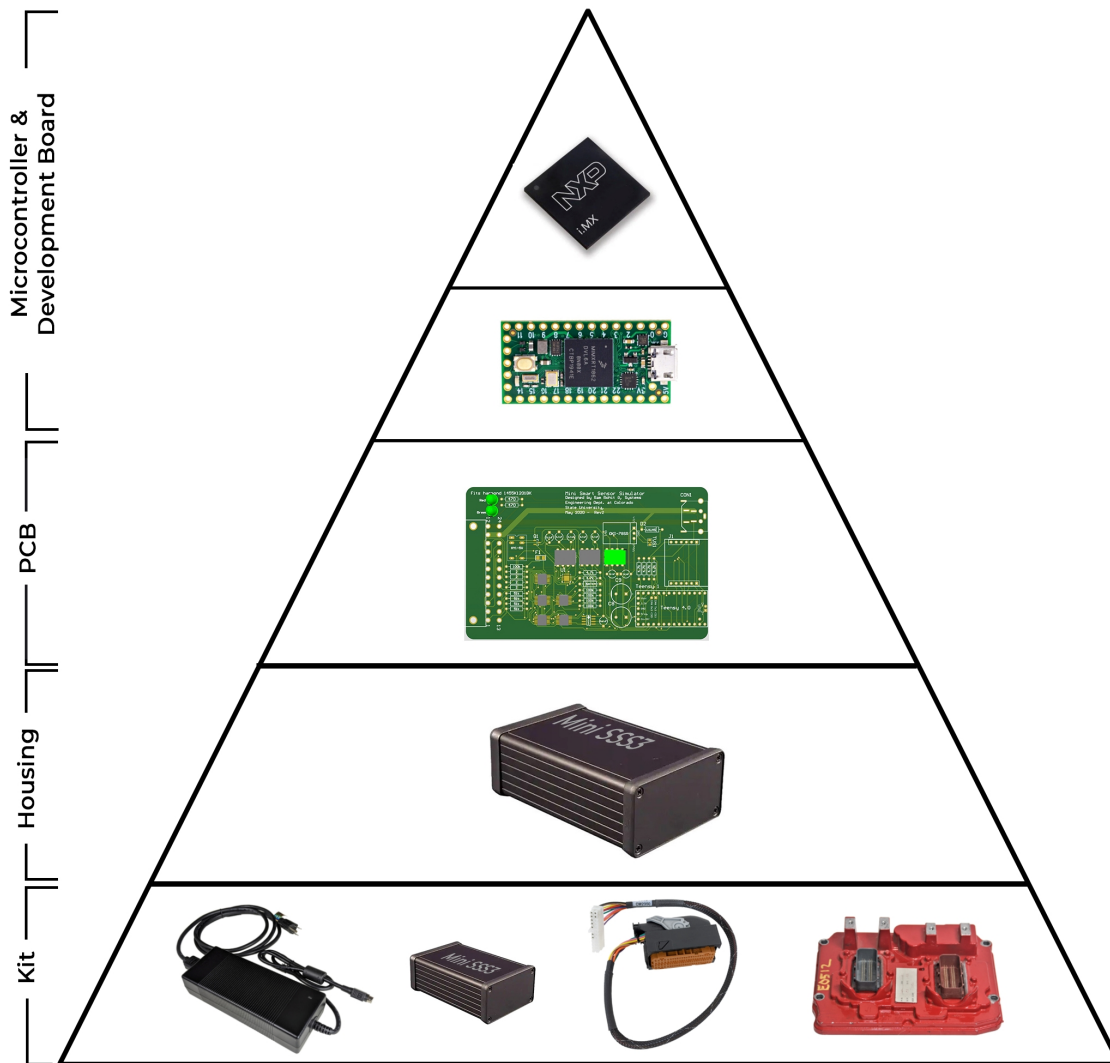
6. The device should be able to provide +12V power, ground, and switched ignition control to the ECM.
7. The device should have a way to store total configurations in non-volatile memory so it can accommodate different settings for various ECMs.
8. The device shall have voltage and current monitors that can provide real-time feedback on the output pins of the device.

### **3.3 Hardware Building Blocks**

In the previous section, the hardware requirements were discussed in detail. Based on these requirements, the hardware design of Mini-SS3 is explained in a top-down approach. Figure 3.1 shows the different layers of the hardware design of the Mini-SS3.

The top layer is the main processor for the Mini-SS3. Based on the requirements, it was decided to choose a microcontroller that had sufficient peripherals. The NXP iMXRT1062, an Arm® Cortex®-M7 processor, was deemed a better option due to its support for multiple communication protocols, including three CAN Bus (1 with CAN FD), three SPI, three I2C ports, and one USB port. The processor also has 40 digital input/output pins, of which 31 pins support PWM output.

Therefore, due to the above-mentioned features offered by the NXP iMXRT1062, the Teensy 4.0 development board was selected. The development board has all the hardware components necessary to program and run the iMXRT1062 chip such as a Micro USB port for programming and serial communication. A voltage regulator to reduce the USB's 5 volts to 3.3 volts was used. The board has 2 Mb of SPI flash memory intended for storing the user code. The board also has a bootloader chip that is responsible for uploading bootloader code to the main MCU's RAM and start executing the user code. A 24 Mhz crystal is present on Teensy 4.0 which is used by the system and other peripherals. A phase-locked loop (PLL) raises the system clock speed from 24 MHz to the desired system clock speed. Another 32 kHz crystal is used by the real-time clock (RTC).



**Figure 3.1:** Different layers of hardware components

In the next stage, the Teensy 4.0 was integrated into a printed circuit board (PCB) along with other integrated circuits such as

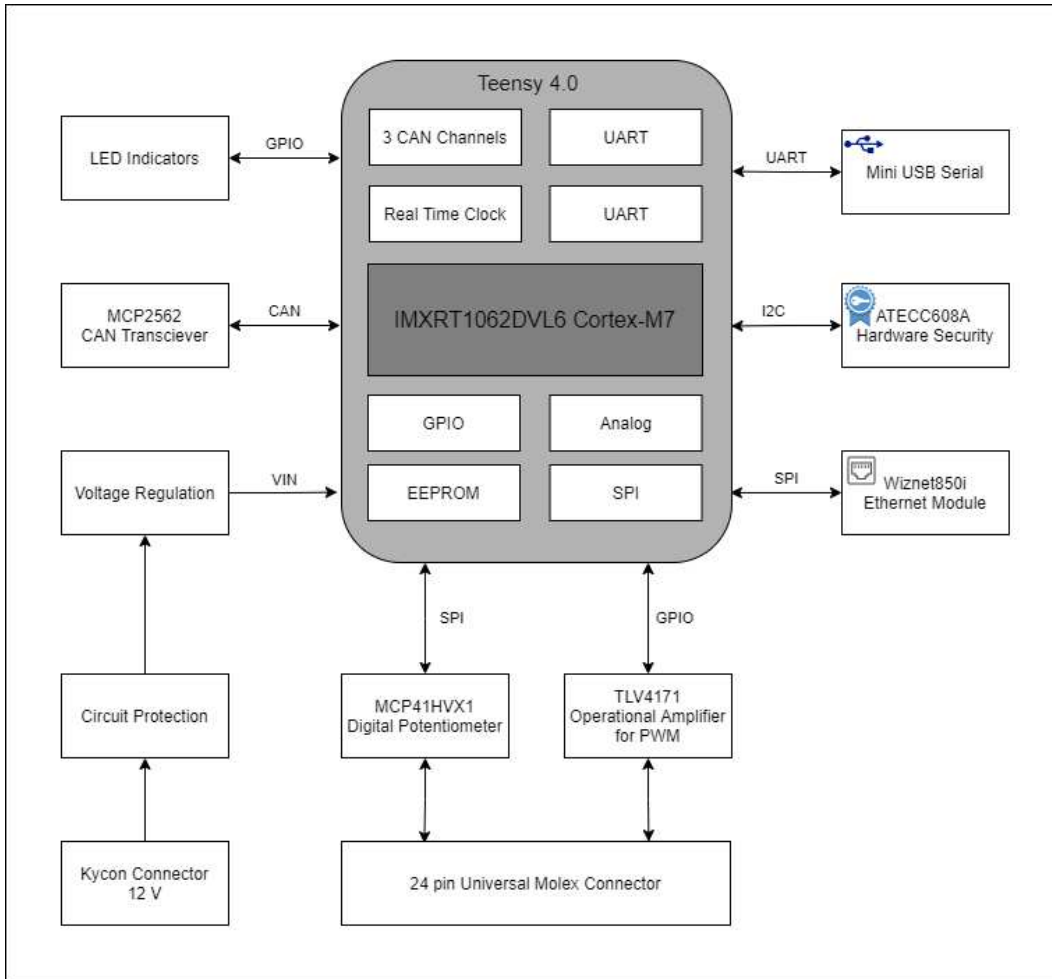
1. A voltage regulator and power protector to power and protect the circuit from any high voltage spikes from the raw input.
2. Microchip's MCP41HV51, a digital potentiometer [22] used for analog signal generation.

3. Microchip's PAC1934, [23] a voltage and power management IC was used for voltage and current monitoring.
4. Texas Instruments TLV4171, a general-purpose operational amplifier was used to amplify PWM signals generated by the Teensy 4.0 from 3.3 to 5 volts.
5. Microchip's MCP2562, a CAN transceiver is used to interface between the CAN controller on the Teensy 4.0 and the physical two-wire CAN bus [24].
6. The Wiznet WIZ850io, an Ethernet module to provide Ethernet capabilities to the Teensy 4.0 [25].
7. Microchip ATECC608B crypto authentication device was used for secure key storage and to perform cryptographic operations [26].

The Mini-SSS3 PCB design is further illustrated through the block diagram in Figure 3.2. The individual components are discussed in detail in the following sections.

Next, in stage 3 an aluminum housing was designed to protect the internal circuitry of the PCB from any kind of damage. The housing exposes external connectors such as the Kycon female power connector to power the device, the ethernet jack to connect an ethernet cable, and the 24 pin Molex connector to connect to the ECM harness.

Finally, a kit with other necessary accessories, like a power supply module and a harness, was included to connect the device to an electronic control module.



**Figure 3.2:** High-level printed circuit board block diagram

### 3.4 Detailed Schematics

Altium software was used to design the schematics and the printed circuit board of the Mini-SSS3. The schematics for the Teensy 4.0 development board are publicly available at [27]. The individual components from the architecture were then incorporated into the schematics. These components are also carefully distinguished on the schematics with the help of bounding boxes based on their functionality. The schematic PDFs are made available on Github [28].

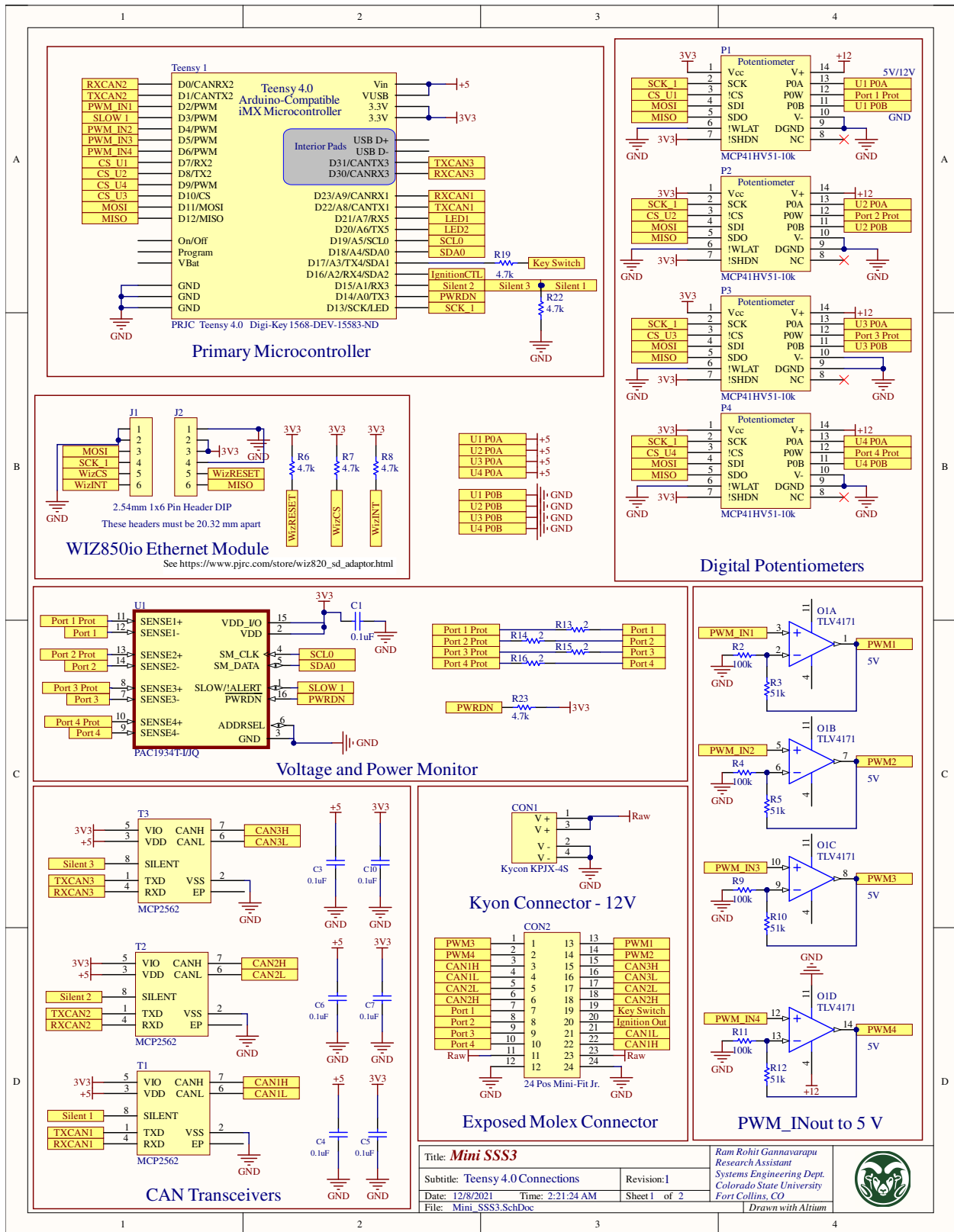


Figure 3.3: Mini-SSS3 schematic page-1

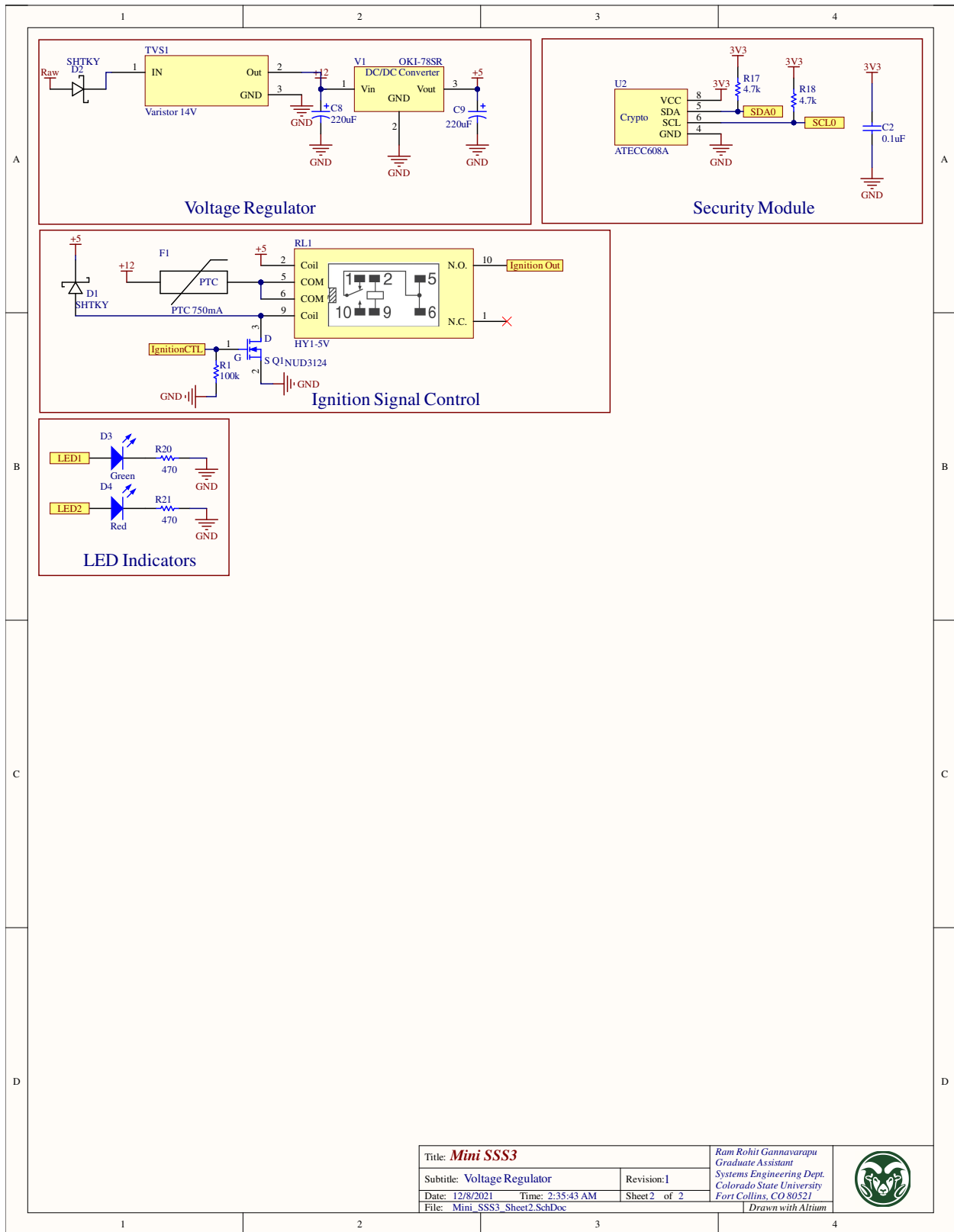


Figure 3.4: Mini-SSS3 schematic page-2

Title: <b>Mini SSS3</b>		Ram Rohit Gannavarapu Graduate Assistant	
Subtitle: Voltage Regulator		Systems Engineering Dept. Colorado State University	
Date: 12/8/2021	Time: 2:35:43 AM	Revision: 1	Fort Collins, CO 80521
File: Mini_SSS3_Sheet2.SchDoc		Sheet2 of 2	
			Drawn with Altium





### 3.4.1 Voltage Regulation and power protection

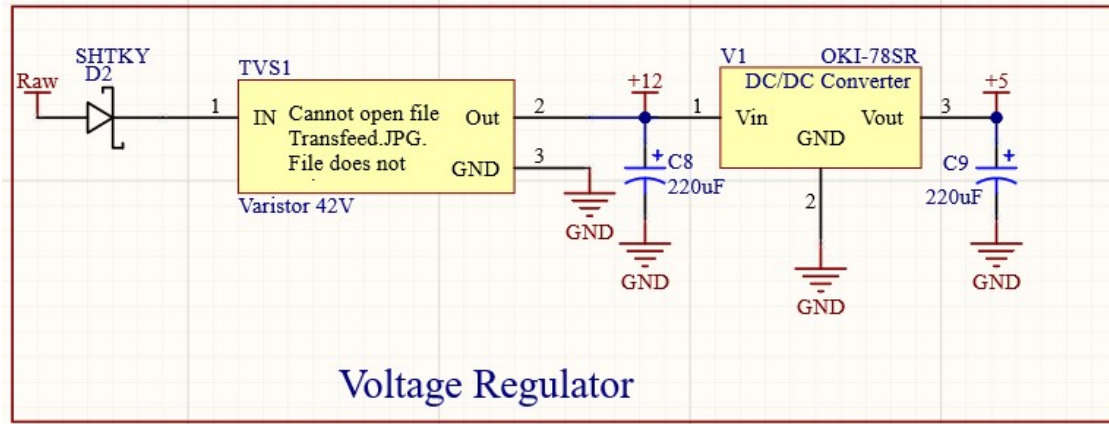


Figure 3.5: 12 volts to 5 volts step down circuit

Figure 3.5 shows the schematics for voltage regulation and power protection on the smart sensor simulator 3. A transient voltage suppression (TVS) device was used to protect the device from any high voltage spikes coming in from the raw input. A Schottky diode was used to protect the device from any reverse polarity. The Schottky diode allows current flow only in a single direction while only dropping about 0.3 volts (compared to 0.7 V for a traditional PN junction diode). The TVS diode is bi-directional and helps protect sensitive electronic equipment from voltage transients induced by transient voltage events.

Most microcontrollers and other peripherals operate on much lower voltages, usually 5 or 3.3 volts. DC to DC converter is used to achieve such voltages. In this case, we have used an OKI-78SR voltage regulator, which steps down the voltage to 5V, which is used to power the Teensy 4.0 development board. Even though the iMX processor runs on 3.3V, the 5V feed works for the Teensy 4.0 because it has a low dropout voltage regulator built into the development board to supply power for the microprocessor.

Figure 3.6 shows the schematic of PAC1934 connections. Precision voltage measurement is provided by PAC1934's four-channel, bidirectional, high-side current sensing capabilities [23]. The four sense channels are connected in parallel with the digital potentiometer output and the

24-pin Molex connector along with a 2-ohm sense resistor which is recommended in the datasheet [23]. The PAC1934 communicates with the microprocessor over I2C and hence pins SM\_CLK and SM\_DATA pins on the PAC1934 are connected to SCL0(19) and SDA0 (18) pins of the Teensy 4.0. The PAC1934 also has a low power mode during which the IC is set to sleep mode and can be controlled by the PWRDN pin. The pin is pulled high with help of a pull-up resistor making the device powered on by default. The SLOW/ALERT pin is used to decide the sampling rate on the PAC1934. if the SLOW/ALERT pin is pulled high the sampling rate is set to 8 samplers per/second and when the pin is pulled low the sampling rate is set to 1024 samples per second.

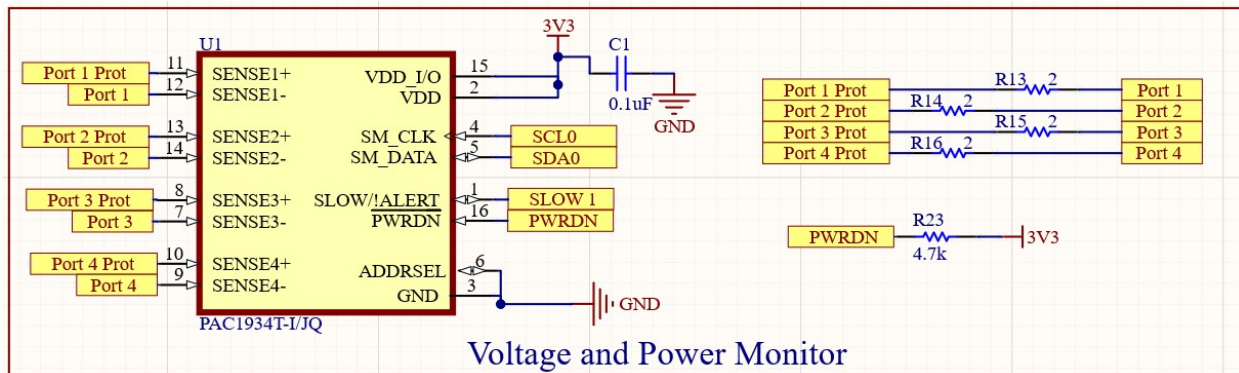


Figure 3.6: PAC1934 power and voltage monitoring device.

### 3.4.2 Development Board

The Mini-SSS3 was designed with the Teensy 4.0 development board as its primary processing unit. Teensy 4.0 has 40 input/output signal pins, out of which 24 are accessible as through-hole headers with 0.100" spacing. The Mini-SSS3 uses female headers to connect the Teensy 4.0 to the PCB. When pin headers are attached, this makes it easy for swapping out development boards during the prototype phase. The Teensy 4.0 houses an NXP iMXRT1062 chip which is an ARM Cortex-M7 processor. The development board also offers power on/off management, a real-time clock for date and time, dynamic clock scaling support, and can be overclocked above 600 MHz. Furthermore, the Teensy 4.0 board consists of 1024K RAM, 2048K flash, 3 CAN buses, 3 SPI ports, and 3 I2C ports. A complete device specification can be found at [29].

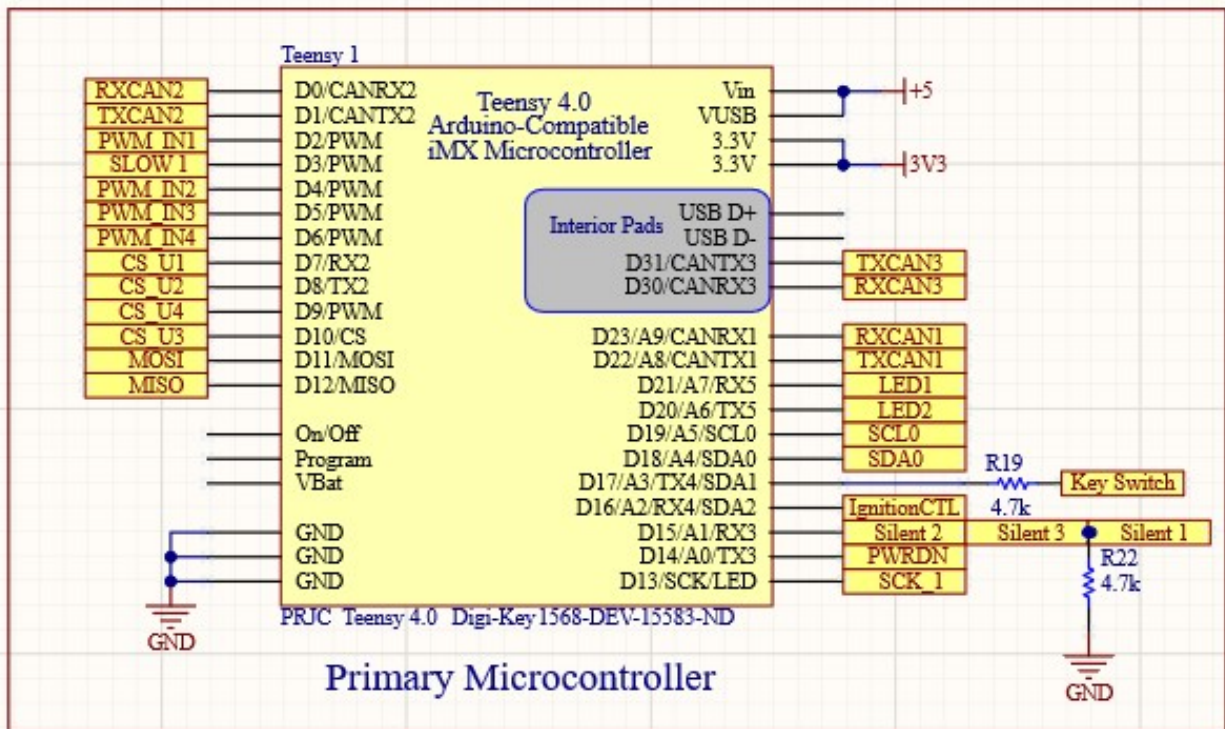
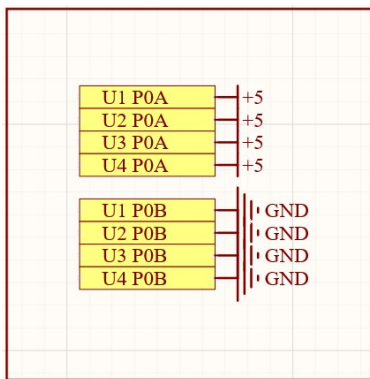


Figure 3.7: Primary Microcontroller evaluation board Teensy 4.0

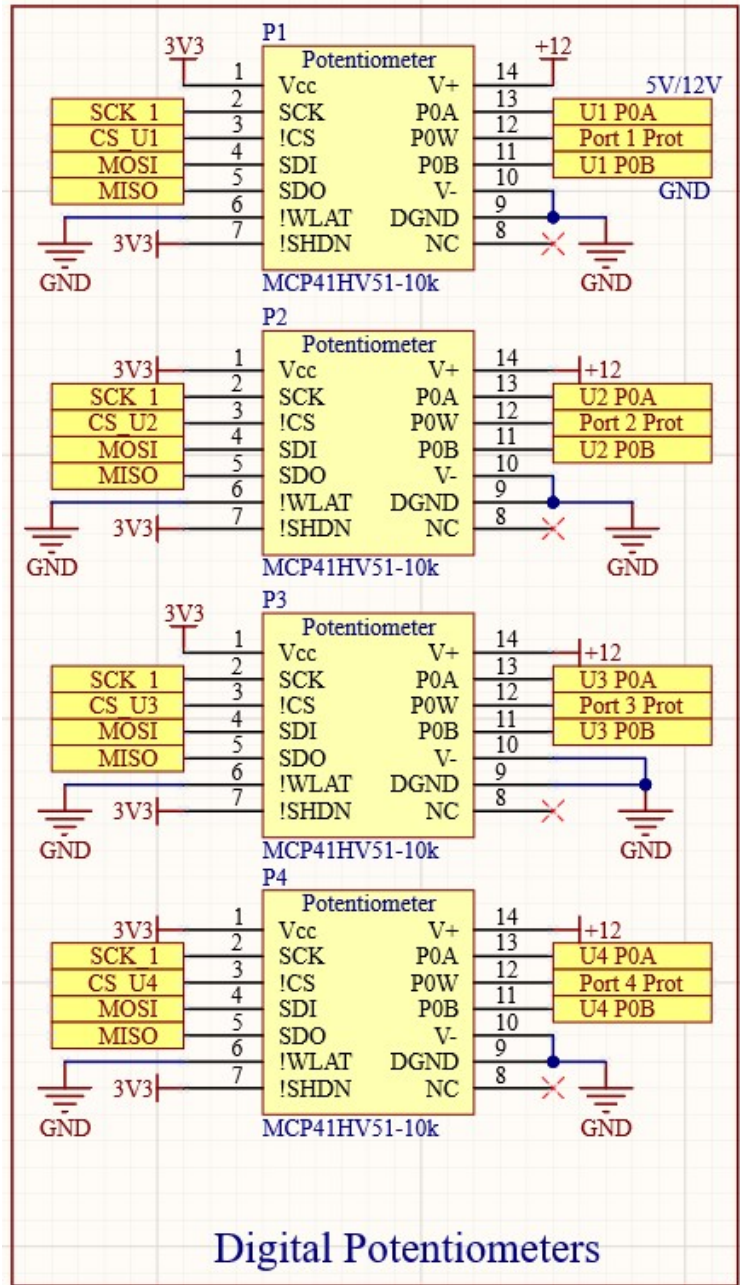
### 3.4.3 Digital Potentiometers

Microchip's MCP41HV51 chip supports one potentiometer in each integrated circuit chip and four of them are used in the Mini-SSS3 design. The MCP41HV51 communicates over the SPI interface, so for each device to connect over the SPI, four pins are required, which are the Clock (SCK), Master In Slave Out (MISO), Master Out Slave In (MOSI), and the Chip Select (CS) pins. Although the Teensy 4.0 has 3 SPI channels, all the devices were connected to the SPI0 interface with MOSI on pin 11, MISO on pin 12, SCK on pin 13, and the CS pins at 7,8,9,10 respectively. In SPI communication, the master can decide which slave to talk to, by pulling the chip select (CS) pin low, allowing for multiple slaves on the same SPI bus. The drawback with SPI is that each slave requires a separate chip select pin.

The V+ and V- pins on the MCP41HV51 are the analog power rails used to power the resistor network terminal pins. Analog switch resistances increase when the analog supply voltage decreases, impacting the limits on the input voltage range. The V+ is connected to 12 volts and



**Figure 3.8:** MCP41HV51 Terminal connections



**Figure 3.9:** Microchip MCP41HV51 Digital Potentiometers schematic

the V- is connected to ground. Pin P0A of the MCPHV51 is connected to 5 volts and pin P0B is connected to ground as shown in Figure 3.8.

The MCP41HV51 has a Terminal Control (TCON) Register, which connects/disconnects terminals A, B, and Wiper individually from the resistor network based on the bit value in the TCON register. This allows the system to emulate both two-wire and three-wire sensor configurations as described in Section 2.2.1.

The contents of the TCON register are shown in Figure 3.10. Bits 7–4 are reserved, and Terminal A is connected to the resistor network if bit 2 (R0A) is set to 1 and disconnected if set to 0. Similarly, Terminal B is based on R0B (bit 0), and Wiper is based on R0W (bit 1).

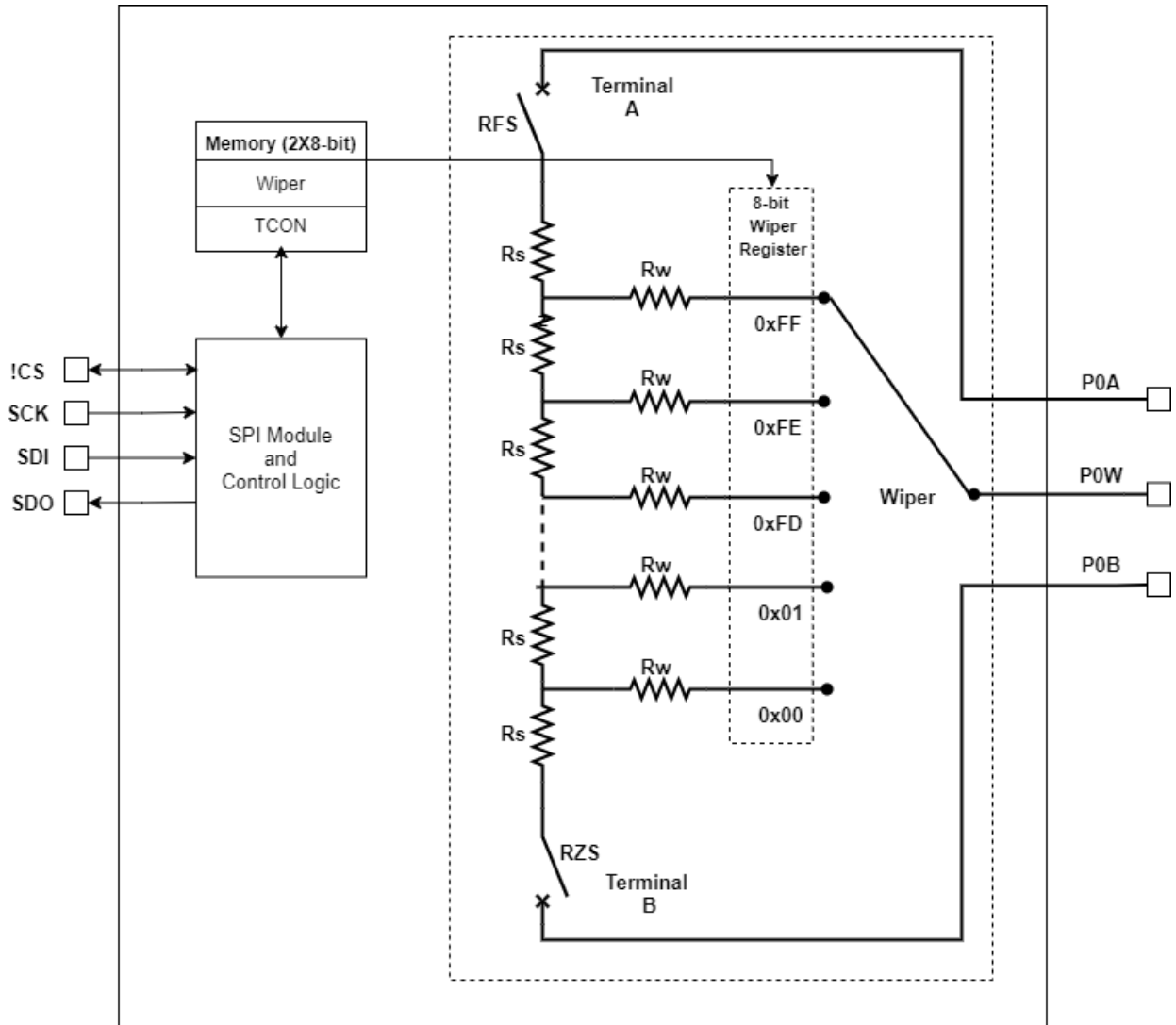
R-1	R-1	R-1	R-1	R/W-1	R/W-1	R/W-1	R/W-1
D7	D6	D5	D4	R0HW	R0A	R0W	R0B
bit 7							bit 0

**Figure 3.10:** MCP41HV51 TCON register

MCP41HV51 also has a register for wiper position at address 0x00. The device has two variants with 7 and 8-bit registers for the wiper resistor network. Each resistor network allows zero-scale to full-scale connections. Figure 3.11 shows a block diagram for the resistive network of MCP41HV51. The resistor network has three external connections: Terminal A, Terminal B, and the wiper (or Terminal W). The RAB resistor ladder is composed of the series of equal value Step resistors ( $R_S$ ) and the Full-Scale ( $R_{FS}$ ) and Zero-Scale ( $R_{ZS}$ ) resistances.

$$R_{AB} = R_{ZS} + n * R_S + R_{FS}$$

The  $R_{FS}$  and  $R_{ZS}$  resistances are artifacts of the RAB resistor network implementation. In the ideal model, the  $R_{FS}$  and  $R_{ZS}$  resistances would be 0. More information about  $R_{FS}$  and  $R_{ZS}$  is available in the data sheet [22].



**Figure 3.11:** MCP41HV51 block diagram

### 3.4.4 Pulse Width Modulated Signals

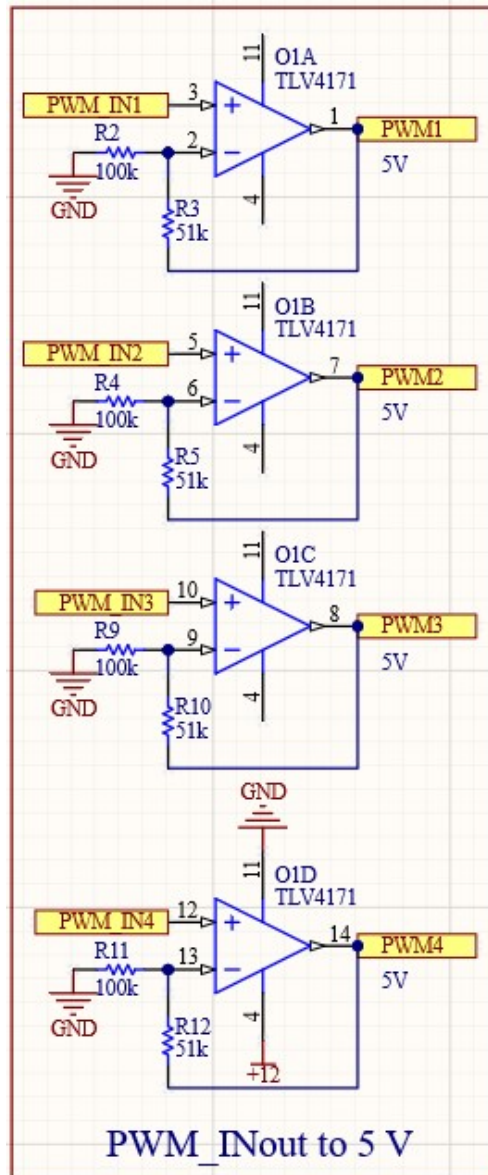


Figure 3.12: PWM Signal Generator Schematic

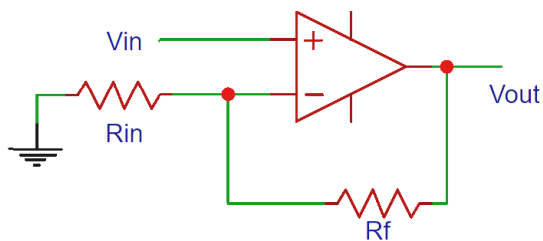
Pulse Width modulated signals are generated by toggling the digital pins at high frequency. Most of the Teensy 4.0 pins can generate PWM signals. The pins on the Teensy generate an output voltage of 3.3 volts. To step up the voltage to 5 volts, a non-inverting operation amplifier as shown in Figure 3.13 is used.

$$Gain = 1 + \frac{R_f}{R_{in}}$$

$$Gain = 1 + \frac{51k}{100k} = 1.51$$

$$Gain = \frac{V_{out}}{V_{in}}$$

$$V_{out} = Gain * V_{in} = 1.51 * 3.3 = 4.983V \approx 5V$$



$$Gain = V_{out}/V_{in} = 1 + R_f/R_{in}$$

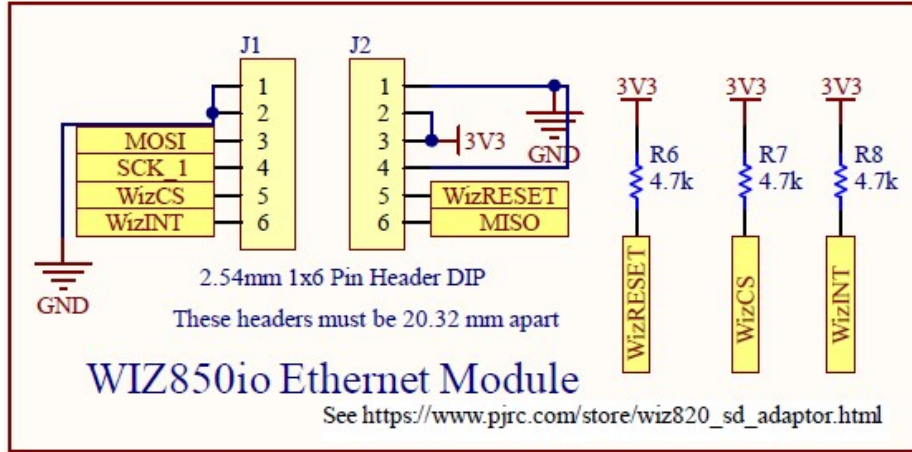
**Figure 3.13:** Non Inverting operational amplifier configuration

On the iMXRT1062 microcontroller, PWM signals are created by hardware timers. Internally, set of pins are tied to a timer and, when frequency of one of the pin is changed, the frequency of all the other pins tied to the same timer also change. This means each PWM output may have unique duty cycles, their frequencies may be shared, depending on which timer is providing the trigger for the PWM signal.

### 3.4.5 Ethernet Module

The main requirement for the Mini-SSS3 is to be able to control it remotely over the internet. A lot of thought went into deciding the network interface between Ethernet and WiFi. Ethernet





**Figure 3.14:** Mini-SSS3 Schematic for Wiznet850i Ethernet module

supports larger bandwidths and also gives us an option to route CAN data over Ethernet for future works. The WIZ850io network adapter was selected for the Mini-SSS3. The device has a hard-wired TCP/IP and a PHY layer embedded in it, making it a plug-in system for developing internet enabled systems rapidly. The device communicates over the Serial Peripheral Interface (SPI) over the MOSI, MISO, SCK and CS pins of the SPI0 interface.

### 3.4.6 CAN Transceivers

The Microchip’s MCP2562 [24] CAN transceiver were used for all three CAN channels on the Teensy 4.0, as shown in the schematics in Figure 3.15. The chip supports CAN FD with speeds up to 8Mbps. Besides the normal functions of a CAN transceiver, it also provides a silent mode which gives the Mini-SSS3 the ability to enable/disable CAN channel transmission. The MCP2562 meets the ISO-11898-1:2015 specifications [30]. On the Teensy 4.0, pins 22 and 23 were used for the CAN1 controller, pins 0 and 1 were used for the CAN2 controller and pins 30 and 31 were used for CAN3 controller.

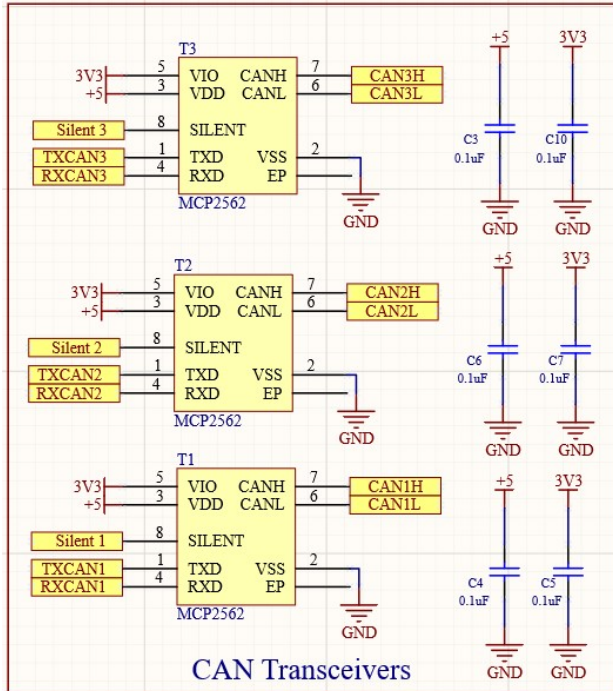


Figure 3.15: Mini-SSS3 schematic for CAN Transceivers

### 3.4.7 Crypto Trust Platform

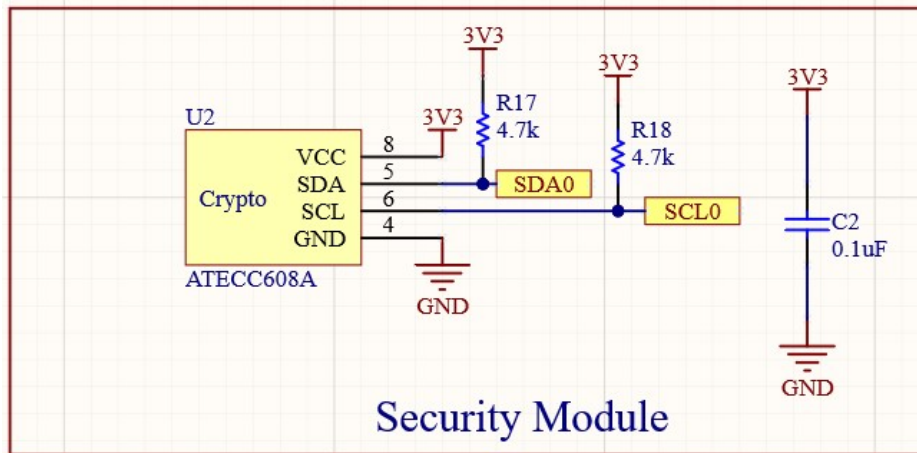


Figure 3.16: ATECC608A cryptographic co-processor Schematic

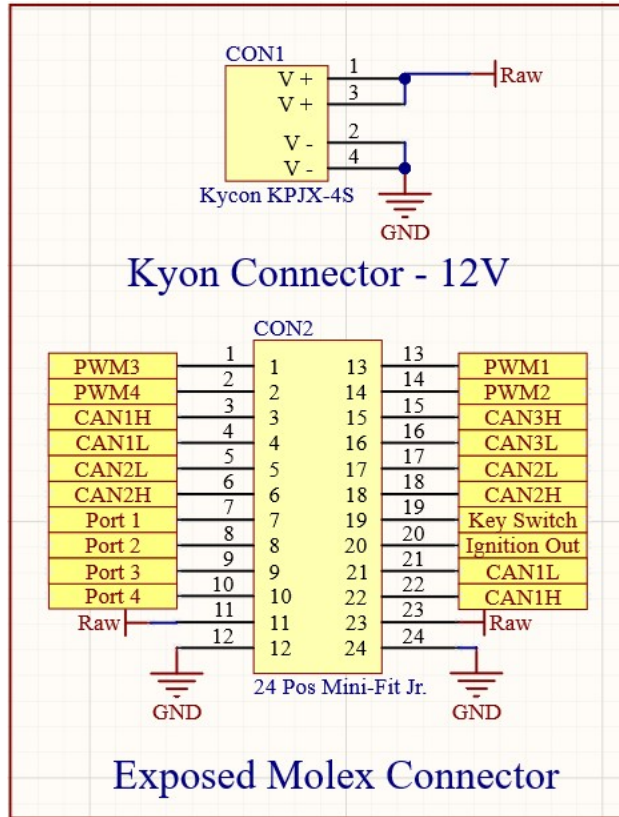
To add hardware security to the Mini-SSS3 a cryptographic co-processing unit was deemed necessary. Microchip ATECC608B was chosen for this project because of its hardware-based

cryptographic key storage and cryptographic countermeasures that eliminate back doors tied to firmware flaws. The device integrates the Elliptic Curve Diffie Hellman (ECDH) security protocol, enabling secure communication on an unsecure channel. The device is agnostic of any microprocessor (MPU) or microcontroller (MCU) and communicates using the I2C protocol over the SCL and SDA pins. The device can store up to 16 keys, and once the device is locked in the EEPROM, only the internal hardware functions have access to the private keys. This feature ensures that the secrets are maintained confidentially and reduces the risk of exposing and compromising systems. The ATECC608A also supports the following cryptographic operations:

1. SHA-256 and HMAC hash.
2. AES-128 encryption and decryption
3. Store compressed X.509 certificates
4. 256-bit ECC following NIST standard with Elliptic-Curve Digital Signature Algorithm (ECDSA) following FIPS186-3.

### **3.4.8 External Connections**

A 24-pin Molex Mini-Fit Jr. male right angle pin header was used to interface the connection to the cabling for the connection to the arbitrary external vehicle electronic control unit. The pin mapping were kept uniform with the earlier version of the smart sensor simulator allowing for re-usability of harness connectors [31]. A Kycon power connector was used for the power and ground connections for the Mini-SSS3 [32]. A Meanwell GST220A12-R7B desktop power supply provided up to 15 amps of 12V power through a matching Kycon connector.



**Figure 3.17:** Pin definitions for the external connections

### 3.5 Enclosure and Printed Circuit Board

With the engineering of the system completed, the last step was shifting to manufacturing and assembly. The PCB Gerber files and NC Drill File were then exported from Altium designer software and sent out to Oshpark LLC for PCB manufacturing. Based on the earlier design of the Smart Sensor Simulator, an Aluminium extruded enclosure made by Hammond (PN: 1455k120BK) was determined the best fit for this system [33]. The printed circuit board for the Mini-SS3 was designed using Altium Designer software, and the resulting 3-D rendering is shown in Figure 3.19. The PCB was built with two layers with a dimension of 75mm L by 120mm H. Since the Mini-SS3 was a prototype board for the SSS3, there may be room available for more improvements and additions.

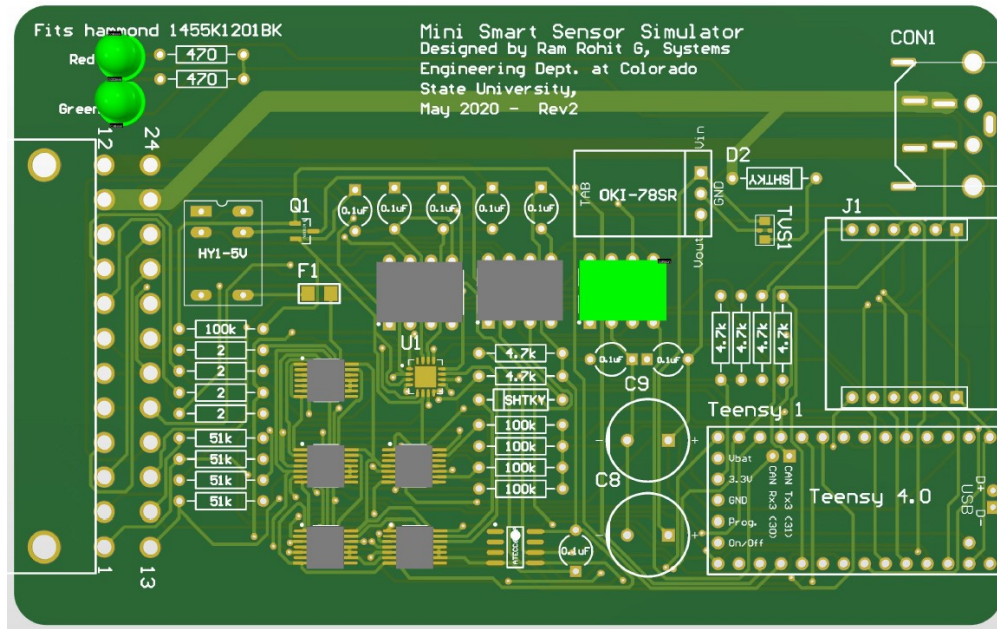


Figure 3.18: Mini-SSS3 Printed Circuit Board

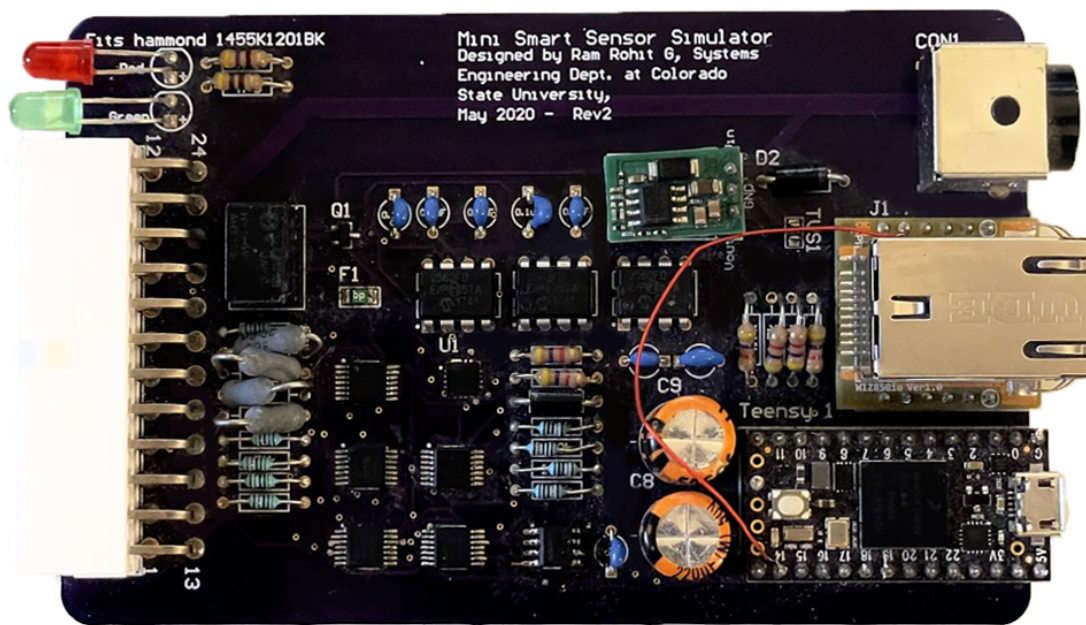


Figure 3.19: Assembled Mini-SSS3 circuit card assembly

As the Mini-SS3 development board was a prototype, most of the components were through hole and all of these components were soldered manually by using soldering iron. Further, a few components such as MCP41HV51 Digital Potentiometer, PAC1932 voltage and power monitoring IC, and ATECC608B hardware security module were surface mounted onto the Mini-SS3 using a hot air soldering station.



**Figure 3.20:** Mini-SS3 enclosure Ethernet side



**Figure 3.21:** Mini-SSS3 enclosure Molex side

## 3.6 Bill of Materials

The complete Bill of Materials is shown in Table 3.1 for the printed circuit board.

**Table 3.1:** Mini-SSS3 bill of materials(BOM)

Designator	Comment	Description	Quantity	Manufacturer
C1, C2, C3, C4, C5, C6, C7, C10	0.1uF	0.1µF -20%, +80% 25V Ceramic Capacitor Y5V (F) 0603 (1608 Metric), 0.1µF ±10% 50V Ceramic Capacitor X7R Radial	8	AVX Corporation, TDK Corporation
C8, C9	220uF	Polarized Capacitor (Radial)	2	AVX Corporation, TDK Corporation
CON1	Kycon KPJX-45	DC Power Connectors 4P JACK SKT SHIELDED SNAP AND LOCK	1	Kycon
CON2	24 Pos Mini-Fit Jr.	24 Positions Header Shrouded Connector 0.165in (4.20mm) Through Hole Right Angle Gold	1	Molex
D1, D2	SHTKY	Diode Schottky 40V 2A Surface Mount DO-214AC (SMA), Diode Schottky 40V 2A Surface Mount DO-214AC (SMA) CDBA240LL-HF	2	Comchip Technology
D3	Green	Green 569nm LED Indication - Discrete 2.1V Radial	1	Lite-On Inc.
D4	Red	Red 623nm LED Indication - Discrete 2V Radial	1	Lite-On Inc.
F1	PTC 750mA	PTC RESTTBLE 0.75A 16V CHIP 1206	1	Bel Fuse Inc
J1, J2	2.54mm 1x6 Pin Header DIP	2.54mm 1x6 Pin Header DIP	1	WIZnet
O1	TLV4171	Quad Low-Power JFET-Input General-Purpose Operational Amplifier, 7 to 36 V, 0 to 70 degC, 14-pin SOP (PW14), Green (RoHS & no Sb/Br)	1	Texas Instruments
P1, P2, P3, P4	MCP41HV51-10k	7/8-Bit Single, +36V (±18V) Digital POT with SPI Serial Interface and Volatile Memory	4	Microchip Technology
Q1	NUD3124	IC INDCT LOAD DRVR AUTO SOT23	1	ON Semiconductor
R1, R2, R4, R9, R11	100k	100 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Automotive AEC-Q200 Thick Film	5	Panasonic Electronic Components
R3, R5, R10, R12	51k	RES SMD 51K OHM 1% 1/10W 0603	4	Panasonic Electronic Components
R6, R7, R8, R17, R18, R19, R22, R23	4.7k	4.7 kOhms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Automotive AEC-Q200 Thick Film, 4.7 kOhms ±5% 0.25W, 1/4W Through Hole Resistor Axial Flame Retardant Coating, Safety Carbon Film	8	Panasonic Electronic Components
R13, R14, R15, R16	2	2 Ohms ±1% 0.125W, 1/8W Chip Resistor 0805 (2012 Metric) Automotive AEC-Q200 Thick Film	4	Stackpole Electronics Inc.
R20, R21	470	330 Ohms ±1% 0.1W, 1/10W Chip Resistor 0603 (1608 Metric) Automotive AEC-Q200, High Voltage Thick Film	2	Stackpole Electronics Inc
RL1	HY1-5V	Telecom Relay SPDT (1 Form C) Through Hole	1	Panasonic Electric Works
T1, T2, T3	MCP2562	Microchip CAN FD Transceiver with Silent Mode	3	Microchip
Teensy 1	Teensy 4.0	32 Bit Arduino-Compatible iMX Microcontroller	1	PRJC
TVS1	Varistor 14V	VARISTOR CAP FEEDTHRU 14V 0805	1	AVX
U1	PAC1934T-I/JQ	Current Monitor Regulator High-Side 16-UQFN _4x4_	1	Microchip Technology
U2	ATECC608A	Authentication Chip IC Networking and Communications 8-SOIC	1	Microchip Technology
V1	OKI-78SR	Linear Regulator Replacement DC DC Converter 1 Output 5V 1.5A 7V - 36V Input	1	Murata Power Solutions



## 3.7 Functional Unit Tests

This section covers a series of unit tests to ensure the proper functionality of the Mini-SSS3. Since some of the hardware is driven by software based commands, code snippets for each of the test is shown below. The complete code for all the test cases is made available on Github at [34].

### 3.7.1 Digital Potentiometer test

Simulating analog signals is a crucial feature of the Mini-SSS3. This test covers all the functionalities of the MCP41HV51 digital potentiometer. A triangle signal was generated to test if the device covers all the desired voltage ranges. The Terminal A is hard wired to a 5 volt supply on the Mini-SSS3 and hence for this test the values range between 0-5 volts. To test the terminal disconnection feature terminal A and Terminal B were disconnected one after the other to check their behavior. The complete test code is available at [35].

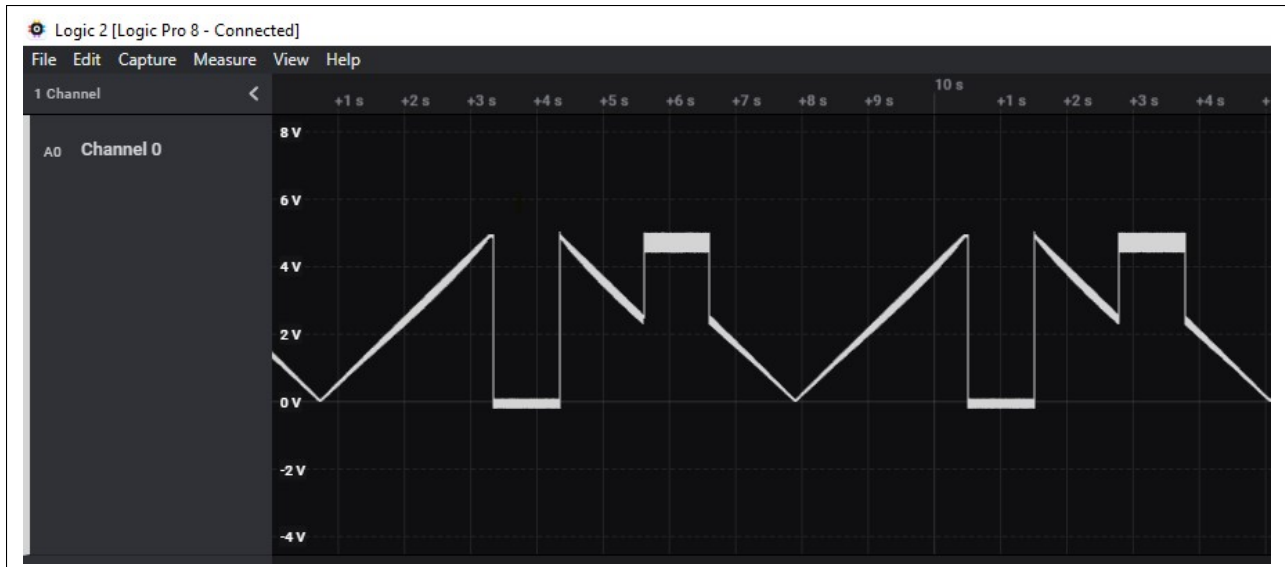
```
#include <SPI.h>
byte address = 0x00;
void setup() {SPI.begin();}
void loop()
{
  int i = 0;
  for (i = 0; i <= 255; i++)
  {
    digitalPotWrite(i, 7);
    delay(10);
  }
  delay(50);
  // Wiper and Terminal B are only connected when TCON value is 3
  MCP41HV_SetTerminals(3, 7);
  delay(1000);
  // All Terminals are connected when TCON value is 7
```

```

MCP41HV_SetTerminals(7, 7);
for (i = 255; i >= 0; i--)
{
    if (i == 128) {
        // Wiper and Terminal A are only connected when TCON value is 6
        MCP41HV_SetTerminals(6, 7);
        delay(1000);
        MCP41HV_SetTerminals(7, 7);
    }
    digitalPotWrite(i, 7);
    delay(10);
}
}

```

Figure 3.22 shows the output from the logic analyzer. Channel 0 of the logic analyzer was connected to Potentiometer 1 whose chip select is connected to pin 7 on the Teensy 4.0. During the first half cycle the wiper position was constantly increased from 0 to 255 and the signal also constantly increases from 0 to 5 volts which can be observed in Figure 3.22. After reaching the maximum wiper position i.e. the output reaching 5 volts, Terminal A was disconnected and it can be observed from Figure 3.22 the output is pulled low. When only Terminal A is disconnected the circuit no longer behaves like a voltage divider and now directly connects to ground and a same behaviour is observed on the signal. After one second the terminal A was connected back to 5 volts and it can be observed that the signal jumps to 5 volts. During the second half cycle the wiper position was constantly decreased from 255 to 0 and since terminal A was connected back the circuit behave like a voltage divider and the same behaviour is observed on the output. Midway through the second half terminal B was disconnected and we can observe the output is pulled to 5 volts. When Terminal B is disconnected the output is directly connected to 5 volts and the same behaviour is evident from this test.



**Figure 3.22:** Mini-SSS3 digital potentiometer test

### 3.7.2 PWM test

AnalogWrite() function can be used to generate PWM signals on the Arduino. The function configures the hardware timers and registers to produce the desired duty cycle and frequency of the PWM signal. A digital logic analyzer was connected to the output ports 1, 2, 13, 14 on the Mini-SSS3 to observe the PWM outputs. The following code snippet was used to test the capability of Mini-SSS3 to generate the PWM signals with varying frequencies and duty cycles.

```

const uint8_t numPWMs = 4;
const int8_t PWMPins[numPWMs] = {2, 4, 5, 6};
uint16_t pwmValue[numPWMs] = {500, 1000, 2048, 4096};
uint16_t pwmFrequency[numPWMs] = {245, 245, 200, 200};

void setup() {
    uint8_t i;
    for (i = 0; i < numPWMs; i++)
        pinMode(PWMPins[i], OUTPUT);
    // analogWrite value 0 to 4095, or 4096 for high

```

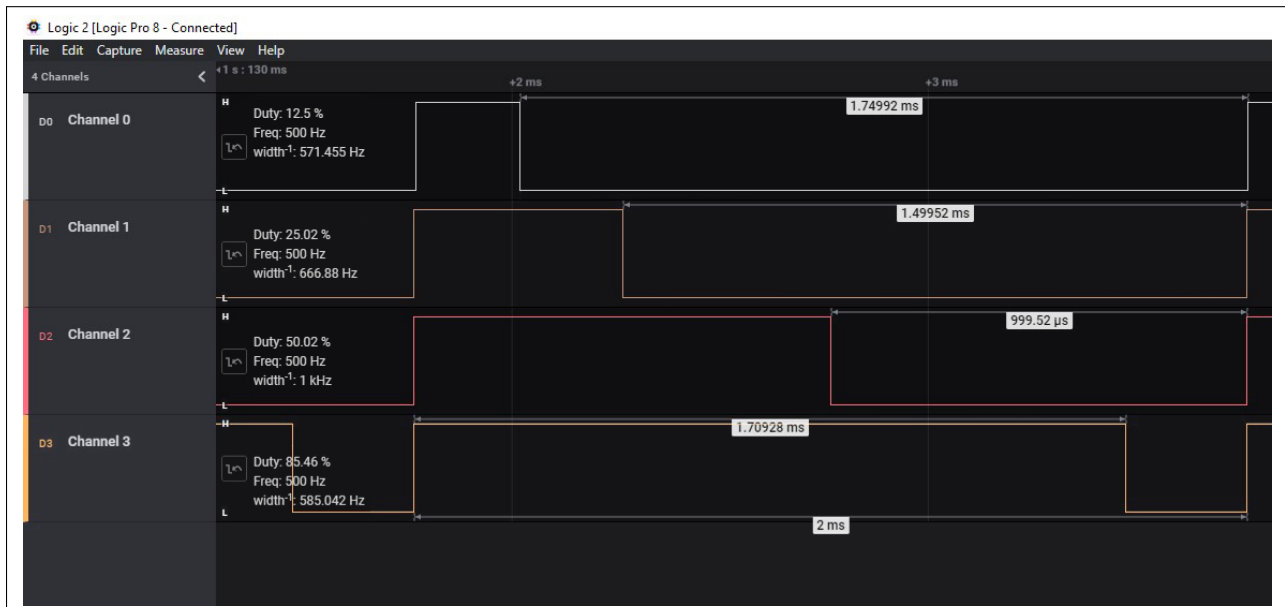
```

analogWriteResolution(12);
analogWrite(PWMPins[0], 512);
analogWrite(PWMPins[1], 1024);
analogWrite(PWMPins[2], 2048);
analogWrite(PWMPins[3], 3500);

analogWriteFrequency(PWMPins[0], 500);
analogWriteFrequency(PWMPins[1], 500);
analogWriteFrequency(PWMPins[2], 500);
analogWriteFrequency(PWMPins[3], 500);
}

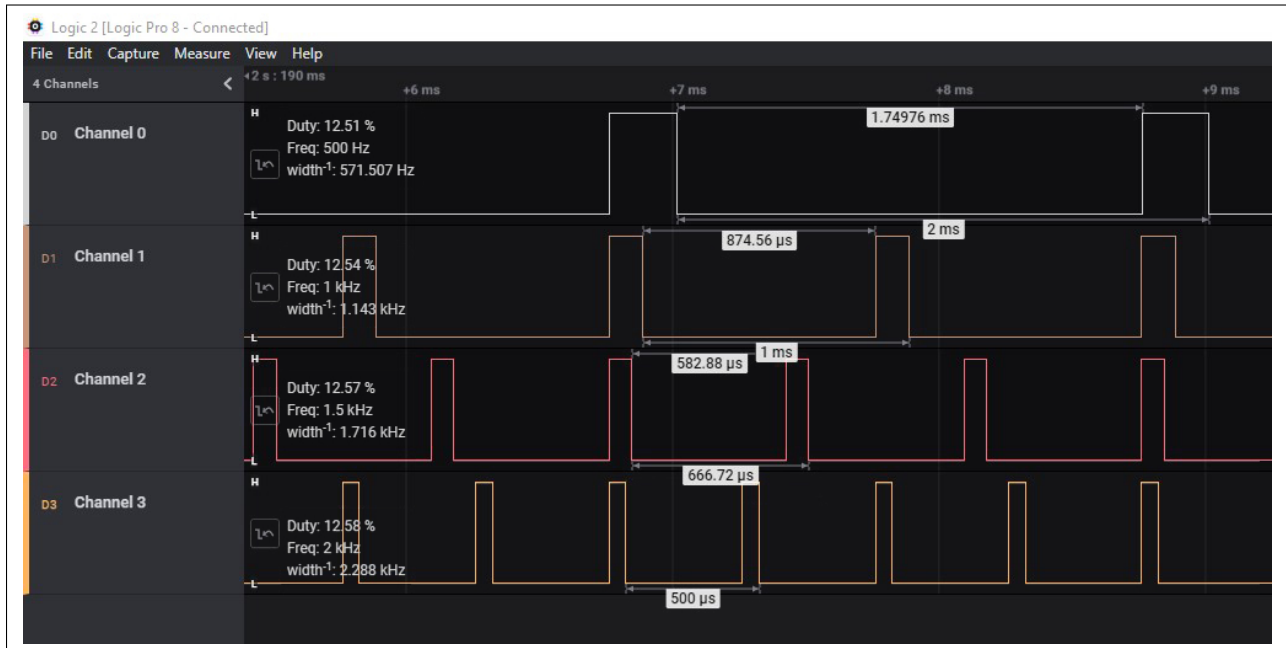
void loop() {
    // put your main code here, to run repeatedly:
}

```



**Figure 3.23:** Mini-SSS3 PWM signal generation test observations from the Saleae Logic analyzer.

Figure 3.23 depicts the multiple test signals that can be produced through different PWM pins present on Mini-SSS3. Channels 0-3 on the digital logic analyzer were connected to PWM pins 0-3 on Mini-SSS3. All the signals produced have varying pulse widths and duty cycles at a fixed frequency. This helps us emulate a wide range of sensors with varying characteristics.



**Figure 3.24:** Mini-SSS3 PWM signal frequency test

Figure 3.20 depicts the multiple test signals produced through different PWM pins present on Mini-SSS3. Channels 0-3 on the digital logic analyzer were connected to PWM pins 0-3 on Mini-SSS3. All the signals produced have varying pulse widths and duty cycles at varying frequencies. This helps us emulate a wide range of sensors with varying characteristics.

### 3.7.3 Voltage monitoring test

Microchip PAC1934's main functionalities on the Mini-SSS3 are to monitor voltages. The PAC1934 gives users a real-time feedback on the desired changes. In the following test, the digital potentiometers are configured on different wiper positions, and the corresponding voltages were measured from the PAC1934 chip. The complete code for the following test case is available at [36]

```

#include <Microchip_PAC193x.h>
#include <Wire.h>
#include <SPI.h>
byte address = 0x00;
int i = 0;
Microchip_PAC193x PAC;
void setup()
{
  Wire.begin();
  PAC.begin();
  SPI.begin();
  Serial.begin(9600);
  while (!Serial)
}

void loop()
{
  digitalPotWrite(10, 7);
  digitalPotWrite(100, 8);
  digitalPotWrite(255, 9);
  digitalPotWrite(200, 10);
  PAC.UpdateVoltage();
  Serial.print("\n\nRead start:");
  Serial.print("\n Voltage1      (mV) = ");
  Serial.print(PAC.Voltage1);
  Serial.print("\n Voltage2      (mV) = ");
  Serial.print(PAC.Voltage2);
  Serial.print("\n Voltage3      (mV) = ");

```

```

Serial.print(PAC.Voltage3);
Serial.print("\n Voltage4      (mV) = ");
Serial.print(PAC.Voltage4);
delay(2000);
}

```

An approximated formula to calculate the output voltage from digital potentiometer is

$$V_{Out} = \frac{Wiper\ value}{2^8} * (V_{P0A-P0B})$$

However, there are other resistances such as  $R_{FS}$  and  $R_{ZS}$  which play role in the calculation of the output voltage but are neglected for approximation purposes.

$$V_1 = \frac{10}{255} * 5 = 196.07mV$$

$$V_2 = \frac{100}{255} * 5 = 1960.78mV$$

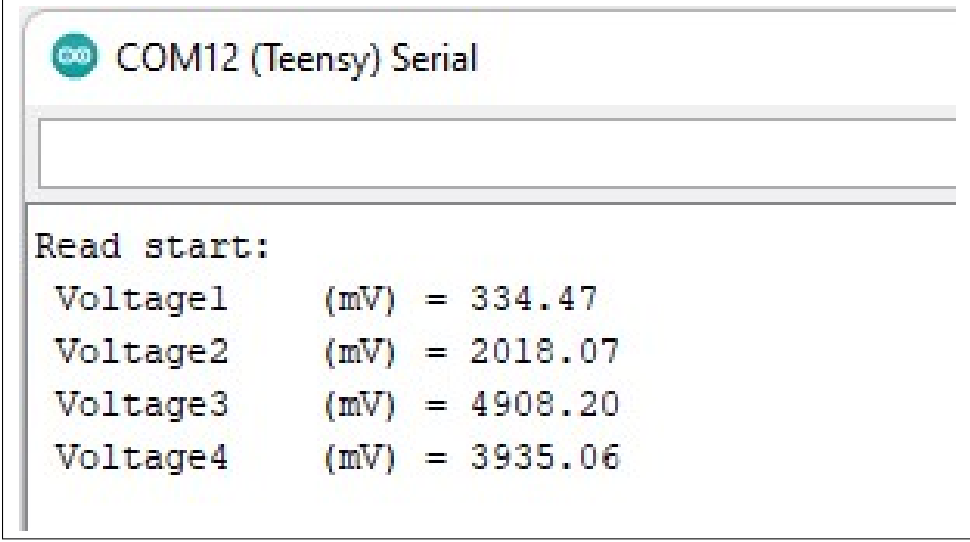
$$V_3 = \frac{255}{255} * 5 = 5000mV$$

$$V_4 = \frac{200}{255} * 5 = 3921.56mV$$

**Table 3.2:** PAC1934 voltage measurement results

	Wiper Setting	PAC1934 Voltage	Multimeter Reading	Difference
Pot-1	10/255	334.36 mV	337 mV	3 mV
Pot-2	100/255	2018.07 mV	2026 mV	8 mV
Pot-3	255/255	4908.20 mV	4906 mV	1 mV
Pot-4	200/255	3935.06 mV	3930 mV	5 mV

Wiper position value on each of the potentiometer were set to 10,100,255,200 respectively. The voltage on the potentiometer outputs on the Mini-SS3 was measured using both a multimeter and PAC9134 and are tabulated in the Table 3.2. The PAC9134 has



```
COM12 (Teensy) Serial

Read start:
Voltage1      (mV) = 334.47
Voltage2      (mV) = 2018.07
Voltage3      (mV) = 4908.20
Voltage4      (mV) = 3935.06
```

**Figure 3.25:** Mini-SS3 voltage monitor test

### 3.7.4 CAN Test

#### Read CAN messages

Teensy 4.0 has 3 CAN channels which enable users to monitor CAN messages on the bus. In the following test script, channels 1 and 2 were tested. The channels were initialized to a 250kbps bit rate. To test the CAN functionality, a brake controller module was connected to the Mini-SS3 over the 24-port Molex connector. The test script reads messages on the bus and prints them on the serial monitor. The complete code for this test case can be found at [37].

```
#include <FlexCAN_T4.h>

FlexCAN_T4<CAN1, RX_SIZE_256, TX_SIZE_16> can1;
FlexCAN_T4<CAN2, RX_SIZE_256, TX_SIZE_16> can2;
```



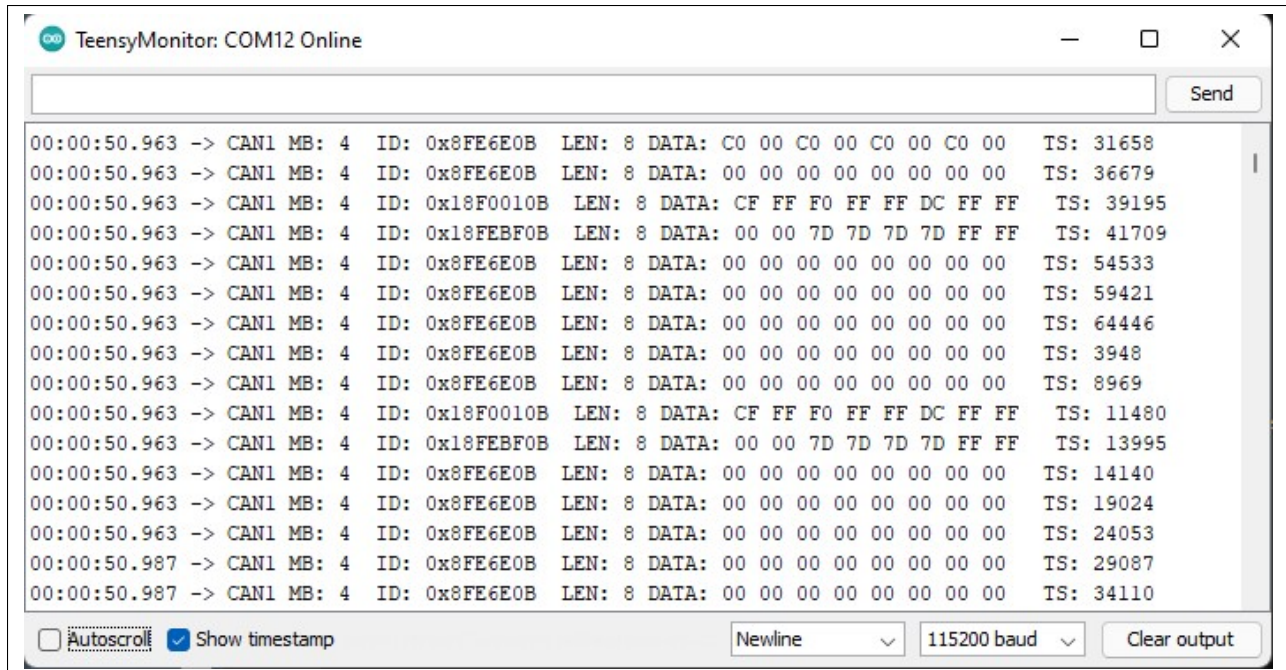
```

CAN_message_t msg;

void setup(void) {
    can1.begin();
    can1.setBaudRate(250000);
    can2.begin();
    can2.setBaudRate(250000);
}

void loop() {
    if ( can1.read(msg) ) {
        digitalWrite(21, HIGH);
        Serial.print("CAN1 ");
        Serial.print("MB: "); Serial.print(msg.mb);
        Serial.print(" ID: 0x"); Serial.print(msg.id, HEX );
        Serial.print(" LEN: "); Serial.print(msg.len);
        Serial.print(" DATA: ");
        for ( uint8_t i = 0; i < 8; i++ ) {
            if (msg.buf[i]<16) Serial.print("0");
            Serial.print(msg.buf[i],HEX); Serial.print(" ");
        }
        Serial.print(" TS: "); Serial.println(msg.timestamp);
        delay(10);
        digitalWrite(21, LOW);
    }
}

```



**Figure 3.26:** Mini-SSS3 CAN message viewer test

We can observe in Figure 3.29 that the messages coming from the brake controller have the J1939 format with a 29-bit message identifier. The brake controller is currently not connected to any sensors and hence we see all 0's in most of the data fields.

### Generate CAN messages

The following code is used to test out the CAN Message generation functionality on the Teensy 4.0. The CAN channel 1 and channel 2 were physically connected to the same CAN bus. To test the CAN Generation functionality, a dummy CAN message was generated on channel 1 and was read on channel 2. The complete code for this test case can be found at [38].

```
#include <FlexCAN_T4.h>

FlexCAN_T4<CAN1, RX_SIZE_256, TX_SIZE_16> can1;
FlexCAN_T4<CAN2, RX_SIZE_256, TX_SIZE_16> can2;

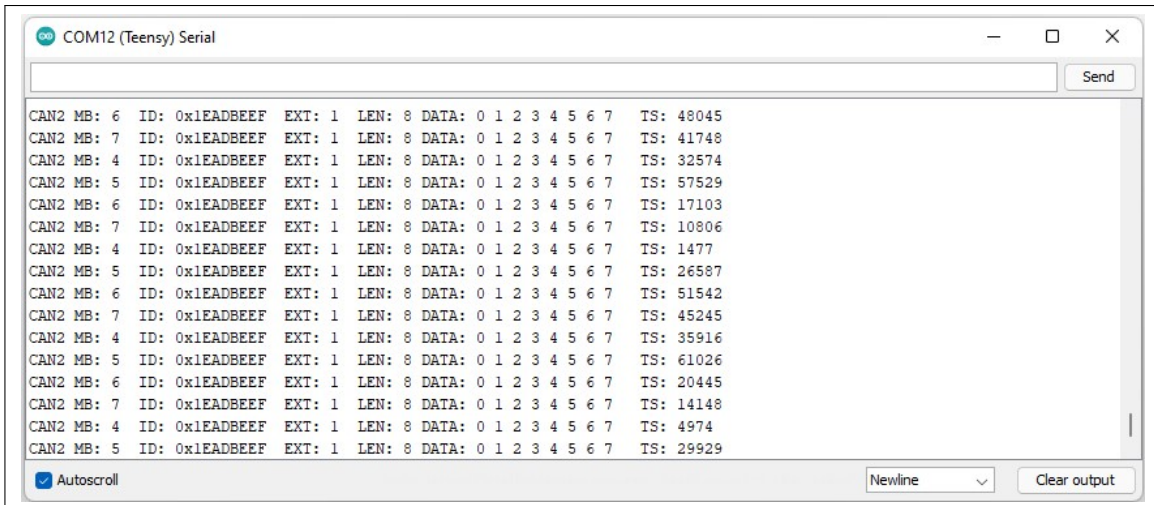
CAN_message_t msg;
CAN_message_t msg1;

void setup(void) {
```

```

can1.begin();
can1.setBaudRate(250000);
can2.begin();
can2.setBaudRate(250000);
}
void loop() {
  msg1.id = 0xDEADBEEF;
  msg1.len = 8;
  for ( uint8_t i = 0; i < 8; i++ ) {
    msg1.buf[i] = i;
  }
  msg1.flags.extended = 1;
  can1.write(msg1);
  if ( can2.read(msg) ) {
    Serial.print("CAN2 ");
    Serial.print("MB: "); Serial.print(msg.mb);
    Serial.print(" ID: 0x"); Serial.print(msg.id, HEX );
    Serial.print(" EXT: "); Serial.print(msg.flags.extended );
    Serial.print(" LEN: "); Serial.print(msg.len);
    Serial.print(" DATA: ");
    for ( uint8_t i = 0; i < 8; i++ ) {
      Serial.print(msg.buf[i]); Serial.print(" ");
    }
    Serial.print(" TS: "); Serial.println(msg.timestamp);
  }
  delay(100);
}

```



**Figure 3.27:** Mini-SSS3 CAN message generation test

We can observe from Figure 3.28 that the messages transmitted on CAN Channel 1 are being read by CAN Channel 2.

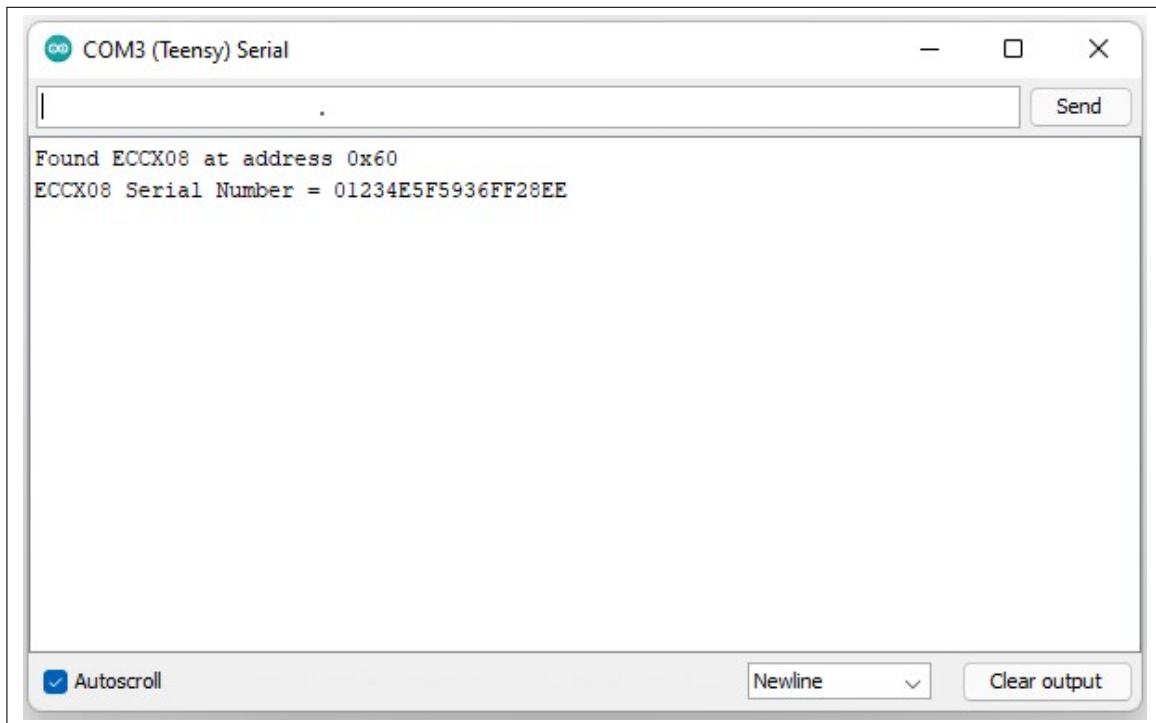
### 3.7.5 ATECC608B test

The following test case checks for proper connectivity between Teensy 4.0 and the ATECC608B over I2C. The main processor tries to retrieve the serial number from the ATECC608B module. Each ATECC608B module has a unique serial number.

```
#include <ArduinoECCX08.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);
  if (!ECCX08.begin(0x60)) {
    Serial.println("No ECCX08 present!");
    while (1);
  }
  else {
    Serial.println("Found ECCX08 at address 0x60");
  }
}
```

```
}  
String serialNumber = ECCX08.serialNumber();  
Serial.print("ECCX08 Serial Number = ");  
Serial.println(serialNumber);  
Serial.println();  
}
```



**Figure 3.28:** ATECC608 get serial number test

### 3.7.6 Ethernet test

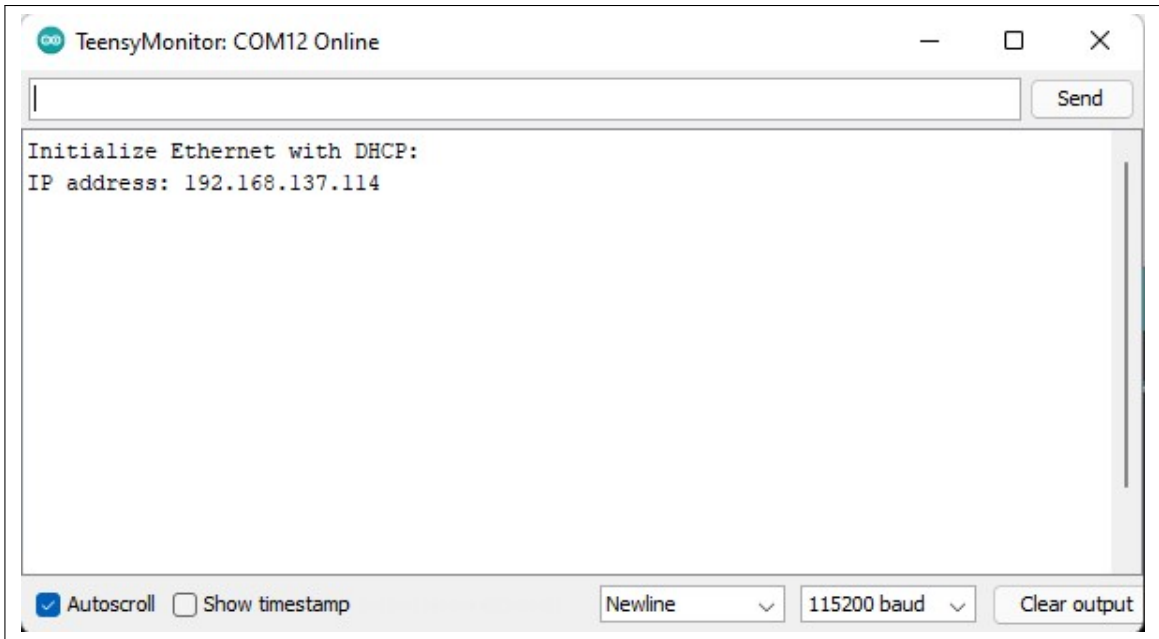
The primary functionality of the remote sensor simulator is to operate it remotely. The following test focuses on the significant functionalities related to the Wiznet850i Ethernet module on the Mini-SSS3. The test initializes the Wiznet850i module and tries to assign an IP address through DHCP.

```

#include <SPI.h>
#include <Ethernet.h>
// Enter a MAC address for your controller below.
byte mac[] = {    0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
void setup() {
    Ethernet.init(14); // 14 is the CS pin
    // start the Ethernet connection:
    Serial.println("Initialize Ethernet with DHCP:");
    if (Ethernet.begin(mac) == 0) {
        Serial.println("Failed to configure Ethernet using DHCP");
        if (Ethernet.hardwareStatus() == EthernetNoHardware) {
            Serial.println("Ethernet shield was not found.  Sorry, can't run
            ↪ without hardware. :(");
        } else if (Ethernet.linkStatus() == LinkOFF) {
            Serial.println("Ethernet cable is not connected.");
        }
        // no point in carrying on, so do nothing forevermore:
        while (true) {
            delay(1);
        }
    }
    // print your local IP address:
    Serial.print("IP address: ");
    Serial.println(Ethernet.localIP());
}

void loop() {
}

```



**Figure 3.29:** Mini-SSS3 Ethernet test

## 3.8 Hardware Design and Testing Summary

**Table 3.3:** Hardware test case summary

Device	Test Description	Status	Notes
PAC1934 voltage and power monitor	Measure voltage measurements from PAC1934	Passed	verifies proper I2C connections on the PAC1934
MCP41HV51 Digital potentiometer	Modify wiper settings and observe change in output voltage, connect/disconnect terminal settings and observe output behavior	Passed	verifies proper SPI connections on the MCP41HV51
Wiznet850io Ethernet module	Initialize Ethernet and obtain IP address	Passed	verifies proper SPI connections on the MCP41HV51
MCP2562 CAN transceiver	Read and Write CAN messages on the CAN bus	Passed	The CAN controller on the Teensy was also tested as part of this test
OLV1471 Op-Amp	Amplify signals from 3.3 to 5 volts for the generated PWM signals	Passed	The PWM generation functionality of the teensy was also tested as part of this test case
ATECC608B HSM	Initialize device on I2C and read device serial number	Passed	
Heat and environmental extremes	Test the device functionality at various environmental situations	Not Tested	The Mini-SSS3 was designed as a prototype board to test out functional feasibility and hence this test was not conducted
shock and vibration	Test the device stability at sudden impacts	Not Tested	same as above
Reverse polarity and over voltage	Test the device voltage protection systems	Not Tested	There were a handful of prototype boards assembled and as this test has a chance of burning out the PCB. The test has been put on hold.
PAC1934 voltage and power monitor	Measure current measurements from PAC1934	Not Tested	The following feature is not implemented in the current design but PAC1934 has the ability to measure current

In this section the overall hardware requirements were gathered and the main hardware functions of the Mini-SSS3 were designed. Further, individual components were tested to verify their functionalities. The following section dives deep into the software design of the Mini-SSS3.



# Chapter 4

## Software Design

### 4.1 Introduction

In this section, the software design of Mini-SS3 is discussed in detail. Based on the functional allocations from the system design chapter, the software requirements of Mini-SS3 were gathered. Further, depending on the requirements, the software design of Mini-SS3 was formulated. A deep dive into the requirements, API design, graphical user interface design, and the functional unit tests are discussed in detail.

### 4.2 Requirements

The primary software requirements for the system are enumerated below:

1. The system should provide an API to control and configure all the hardware peripherals such as the PWM, Digital Potentiometers, CAN message generation.
2. The system should provide a graphical user interface for users to interact with the device and modify various parameters related to the hardware peripherals
3. The system should provide digital feedback allowing users to look at the changes that take place in real-time.

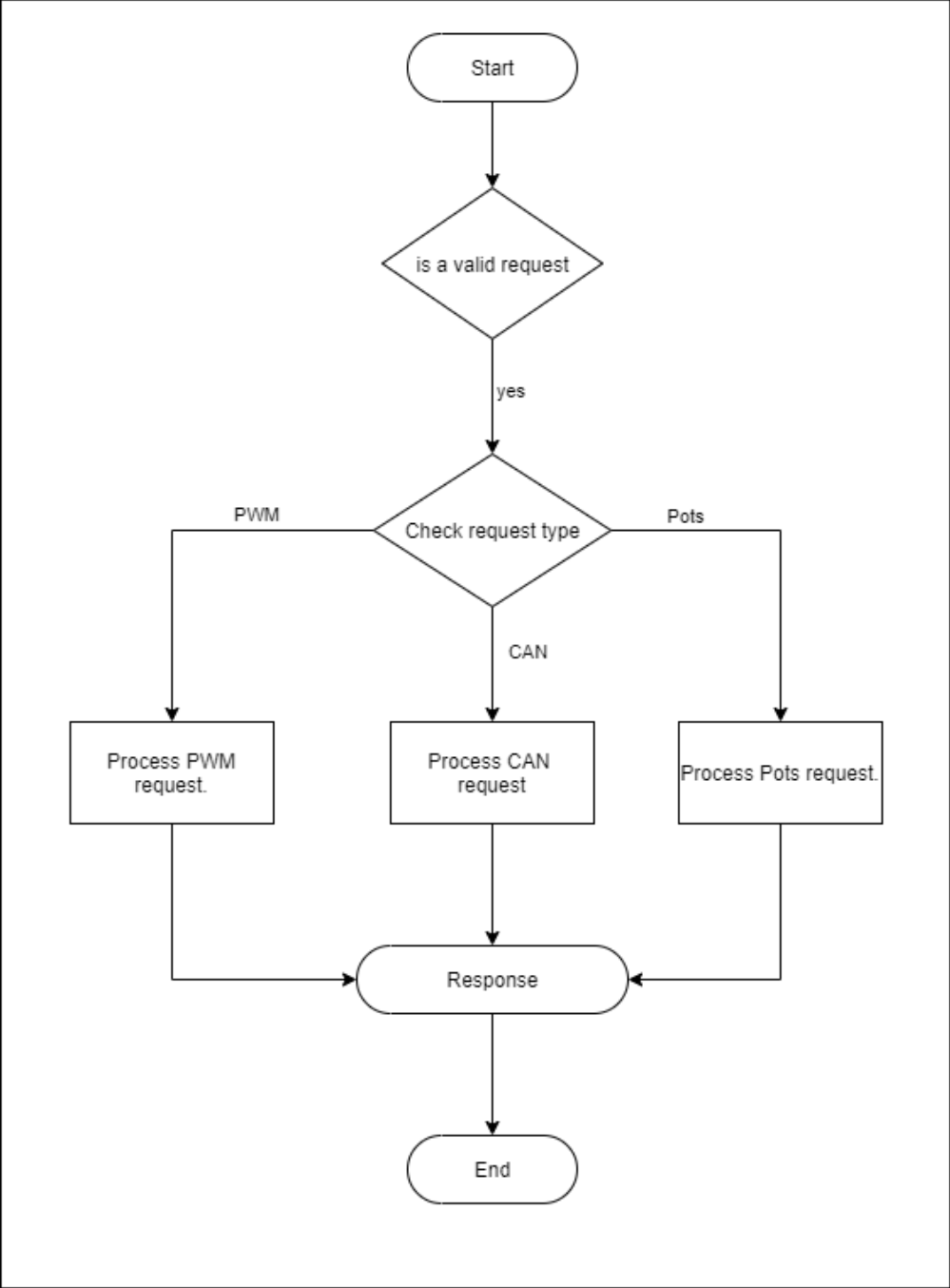
### 4.3 API Design

The earlier version, i.e., the smart sensor simulator, could only be operated through serial commands, which are not always user-friendly. The Mini-SS3, with the option to connect to an Ethernet port, opens up multiple ways to communicate with it. An HTTP API lets users control the different peripherals on the Mini-SS3 programmatically. The HTTP API is a protocol that describes how a client can access information from a server. It works as a request-response protocol

between a client and a server. The HTTP API also allows us to build a web interface to control the different functionalities of the Mini-SSS3. The web interface also eliminates the overhead of installing additional drivers or software, which was required earlier. An Arduino library called aWOT [35] [39] that provides essential web application features on memory-constrained micro-controllers, is used in designing the HTTP API.

Multiple API endpoints were set up for each peripheral of Mini-SSS3 to let clients communicate, get status and also update parameters.

1. <ip-address-of-Mini-SSS3>/pots
2. <ip-address-of-Mini-SSS3>/pwm
3. <ip-address-of-Mini-SSS3>/can
4. <ip-address-of-Mini-SSS3>/cangen
5. <ip-address-of-Mini-SSS3>/voltage

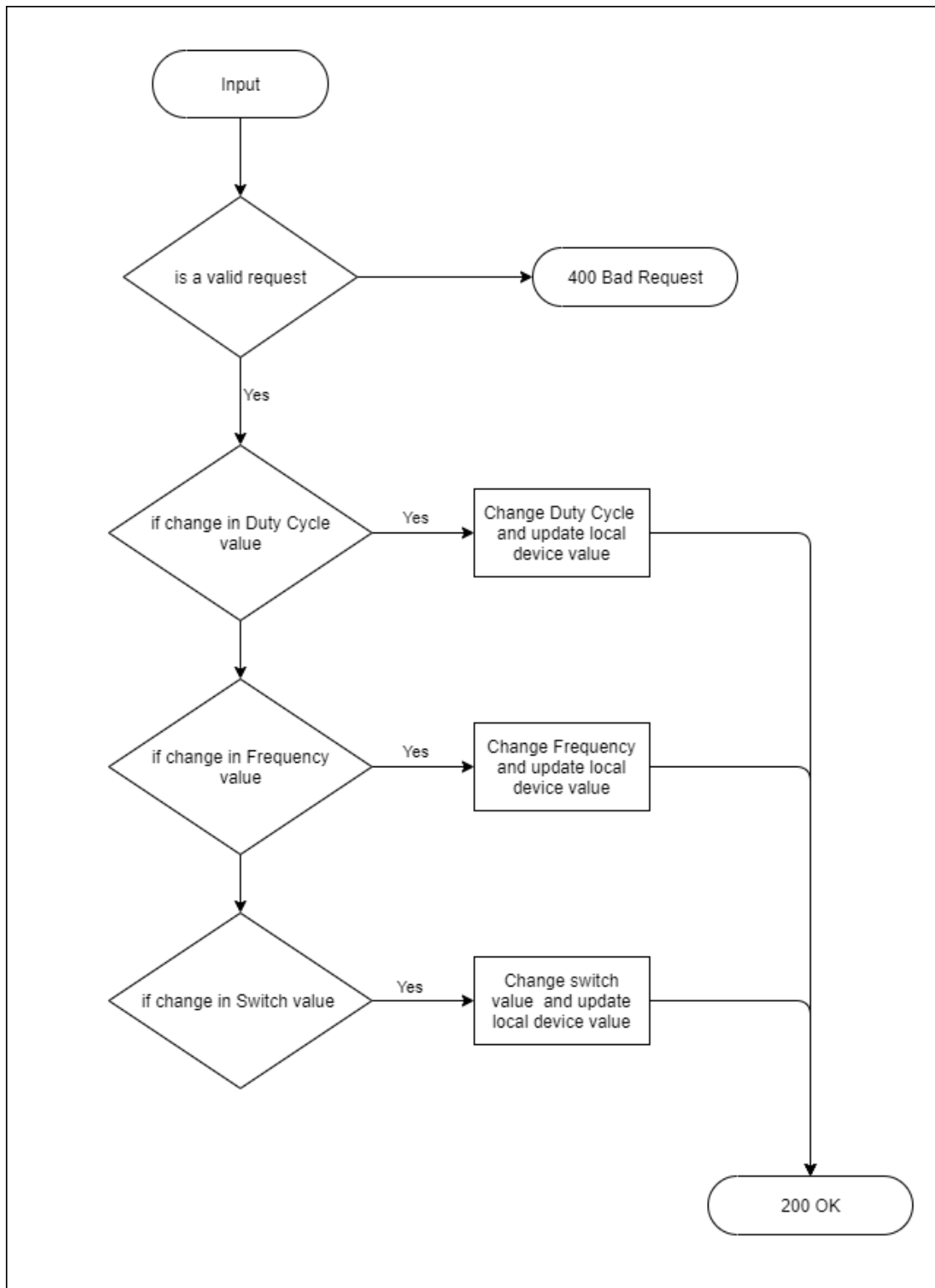


**Figure 4.1:** Mini-SSS3 HTTP API flow diagram

The above flowchart explains how an HTTP request is processed. message received from the client is in the form of a JSON message. The following checks are performed to ensure its a valid request.

- Check for Valid JSON structure
- Check for required key value pairs
- Check if the values are within limits
- Check if the value is different from the current setting on the local device

### 4.3.1 PWM process flow



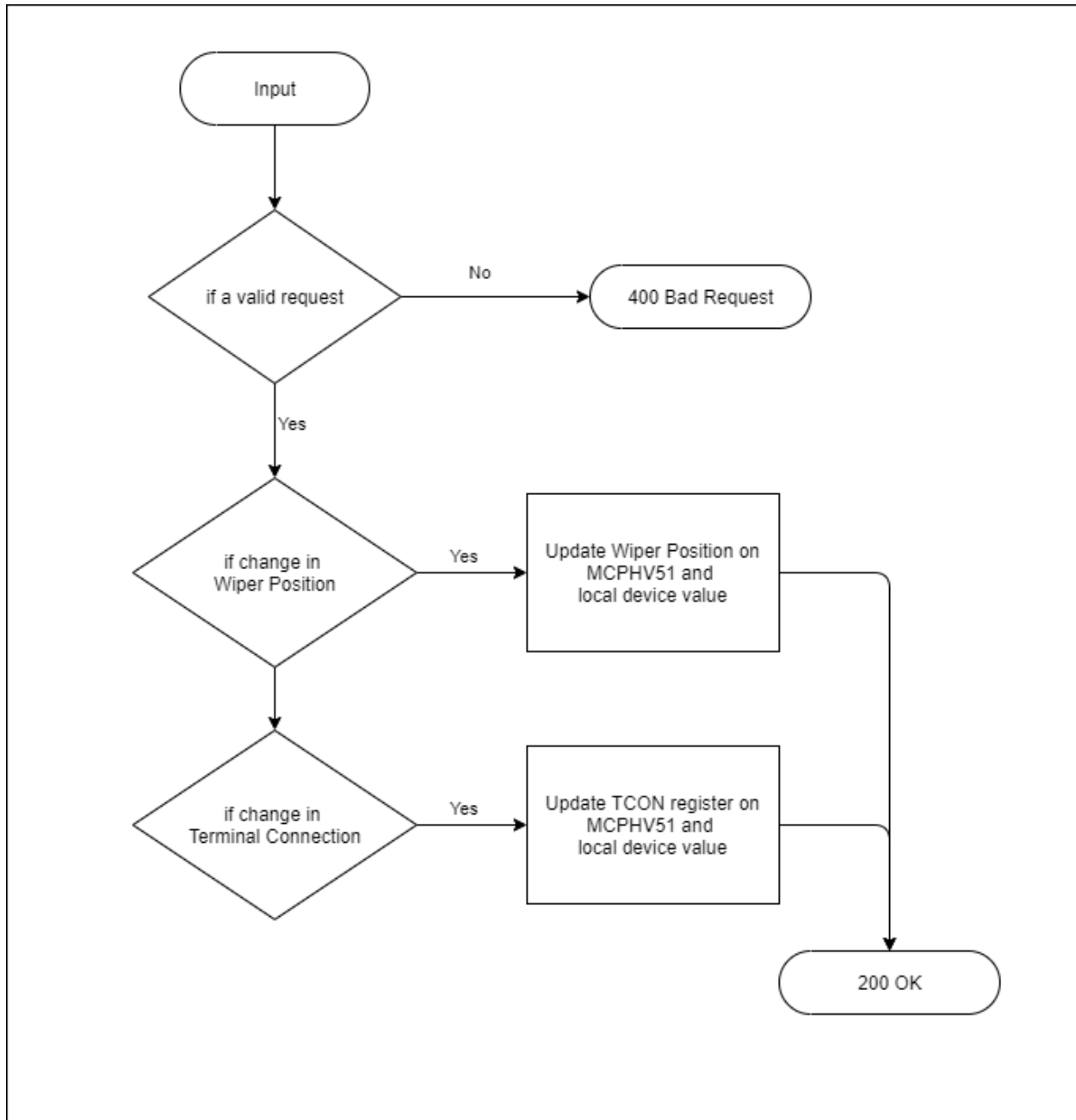
**Figure 4.2:** PWM process flow diagram

If the JSON message is valid, then it is further processed and goes to the next step. Otherwise, it reports a 400 bad request status to the client. The validation process checks for the following conditions:

- Check for valid JSON structure
- Check for required key value pairs
- Check if the values are within limits
- Check if the value is different from the current setting on the device

If the current and received values are different for the duty cycle, the system updates the duty cycle value in the global variable and then modifies it on the hardware by calling the `analogWrite` function for the specific pin. Similarly, if the frequency value is changed the system updates the frequency value in the global variable and modifies the frequency on the hardware by calling the `analogWriteFrequency()` function. The `analogWriteFrequency()` makes it easy to use as the function calculates and sets the prescaler and divisor for the timer related to the specific pin. The switch enables the user to globally enable/disable the PWM signal that is being generated on a particular pin. Once each of the following conditions are met and the values are successfully modified the system responds to the client with a 200 OK status message.

### 4.3.2 Potentiometers process flow



**Figure 4.3:** Potentiometers process flow diagram

Figure 4.3 contains the flow diagram of how an HTTP POST request to control the digital potentiometer is processed. Similar to the PWM process flow the JSON message is first validate and only if its a valid message it proceeds to the next step, Otherwise, it reports a 400 Bad Request

status to the client. The system then updates the current values both on the hardware and software levels for each of the Wiper Position, TCON registers and responds with a 200 OK status message to the client.



### 4.3.3 CAN Message generation process flow

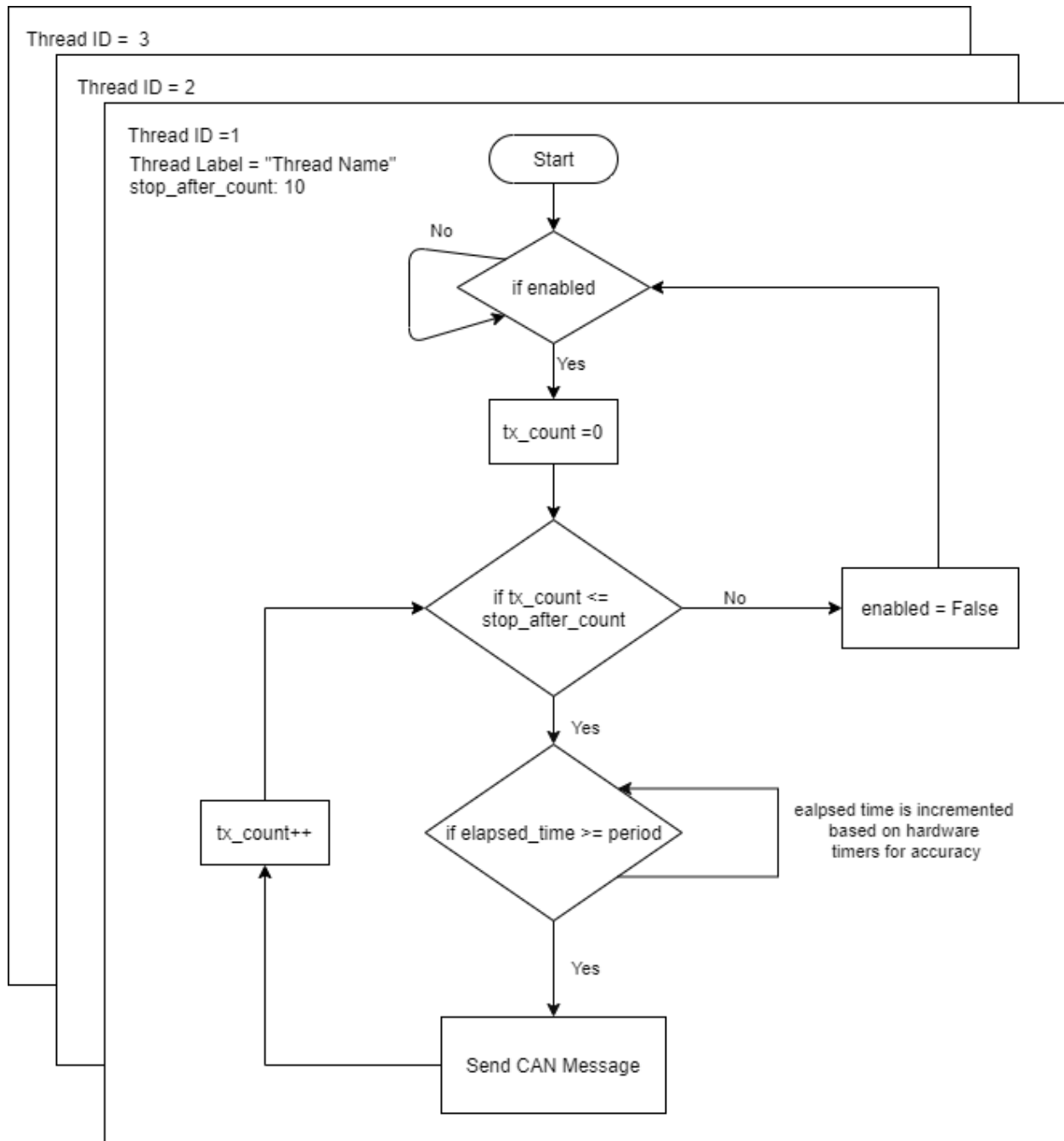


Figure 4.4: CAN generation flow diagram

Figure 4.4 contains the software flow of how the CAN messages are generated on the Mini-SSS3. A thread based logic is implemented to better handle timing in generation of these messages. Each CAN message thread has the following parameters that the user can modify.

1. CAN Message ID - It specifies the CAN message identifier.
2. DLC - It specifies the length of the CAN message field.
3. B0-B7 - it specifies the data fields of the 8 byte CAN message.
4. Stop\_after\_count - The CAN message thread is disabled after the transmission count(tx\_count) has reached the specified stop\_after\_count value.
5. TX Count - It specifies the current status of the number of message transmitted. this is not a user modifiable field.
6. Period - It specifies the time delay between consecutive CAN messages.
7. Channel - It specifies the CAN channel on which the message has to be transmitted.
8. enabled - It specifies if the thread is currently enabled. Thread Label - It specifies the thread name for each CAN message thread.

When initialized all the thread as running in idle. The actual logic of sending the message is only entered when the enabled variable is set to True. After the thread is enabled the tx\_count variable is initialized to zero. The tx\_count is incremented every time a CAN message is sent. A while loop constantly compares the tx\_count reaches stop\_after\_count value the enabled variable is set to false and the thread is in idling mode.

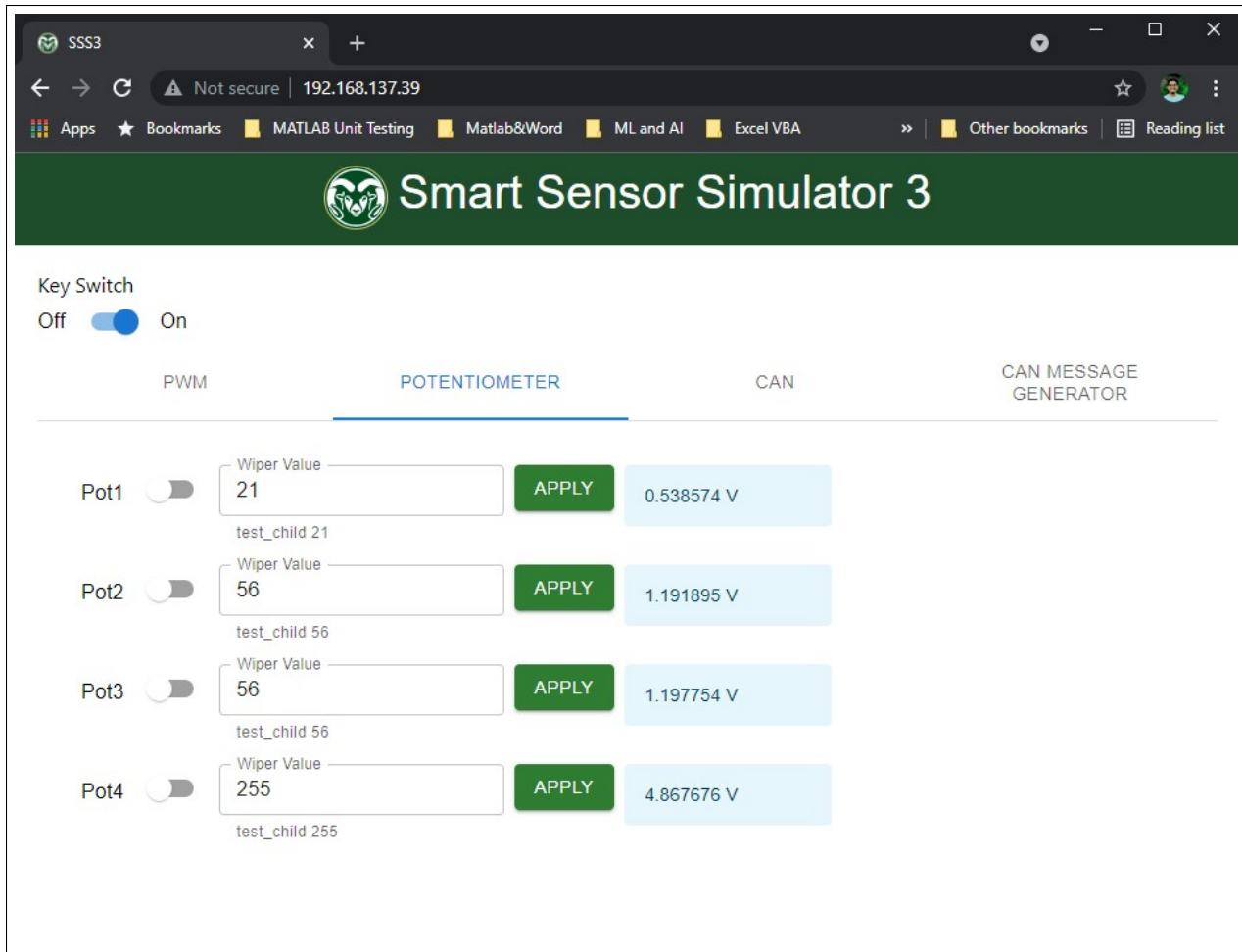
When a HTTP POST request to control the digital potentiometer is processed. Similar to the PWM process flow the JSON message is first validate and only if its a valid message it proceeds to the next step. Otherwise, it reports a 400 Bad Request status to the client. The system then updates the current values both on the hardware and software levels for each of the Wiper Position, TCON registers and responds with a 200 OK status message to the client.

## 4.4 Graphical User Interface (GUI)

A react-based web page was designed to give users an interface to operate the Mini-SS3. The React framework was developed by facebook as a single page application (SPA) to enable dynamic webpages running JavaScript. The aWOT Arduino library [39] also enables us to host react-based web pages from memory-constrained devices. The library is compatible with most Arduino-based microcontrollers. The GUI is split into multiple tabs each one for a separate peripheral of the Mini-SS3. The following sections talk about the graphical user interface for each of the peripherals of the Mini-SS3.

### 4.4.1 Digital Potentiometers

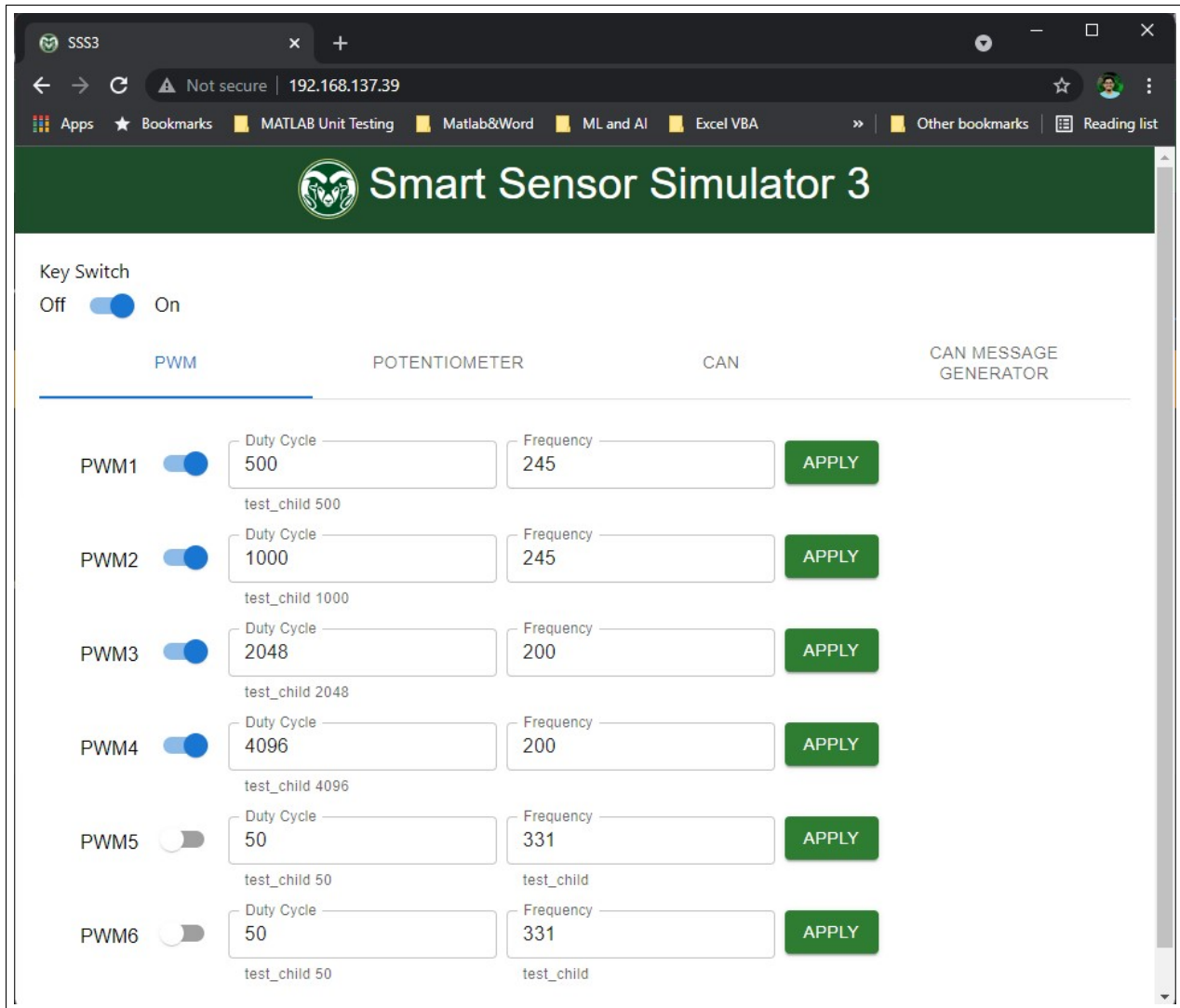
The GUI for the Digital Potentiometers is shown in Figure 4.5. The GUI offers the option to change the wiper value and also control the terminal switches on the MCP41HV51 as shown in Figures 2.1 and 2.2. The readings from the PAC1934 [23] is periodically queried using the HTTP API and relayed on the GUI as shown in the blue box adjacent to the green "Apply" button in Figure 4.5. The data from PAC1934 gives users real-time feedback about the actual voltage on the 24-pin Molex connector.



**Figure 4.5:** Digital Potentiometer GUI

#### 4.4.2 Pulse width Modulation

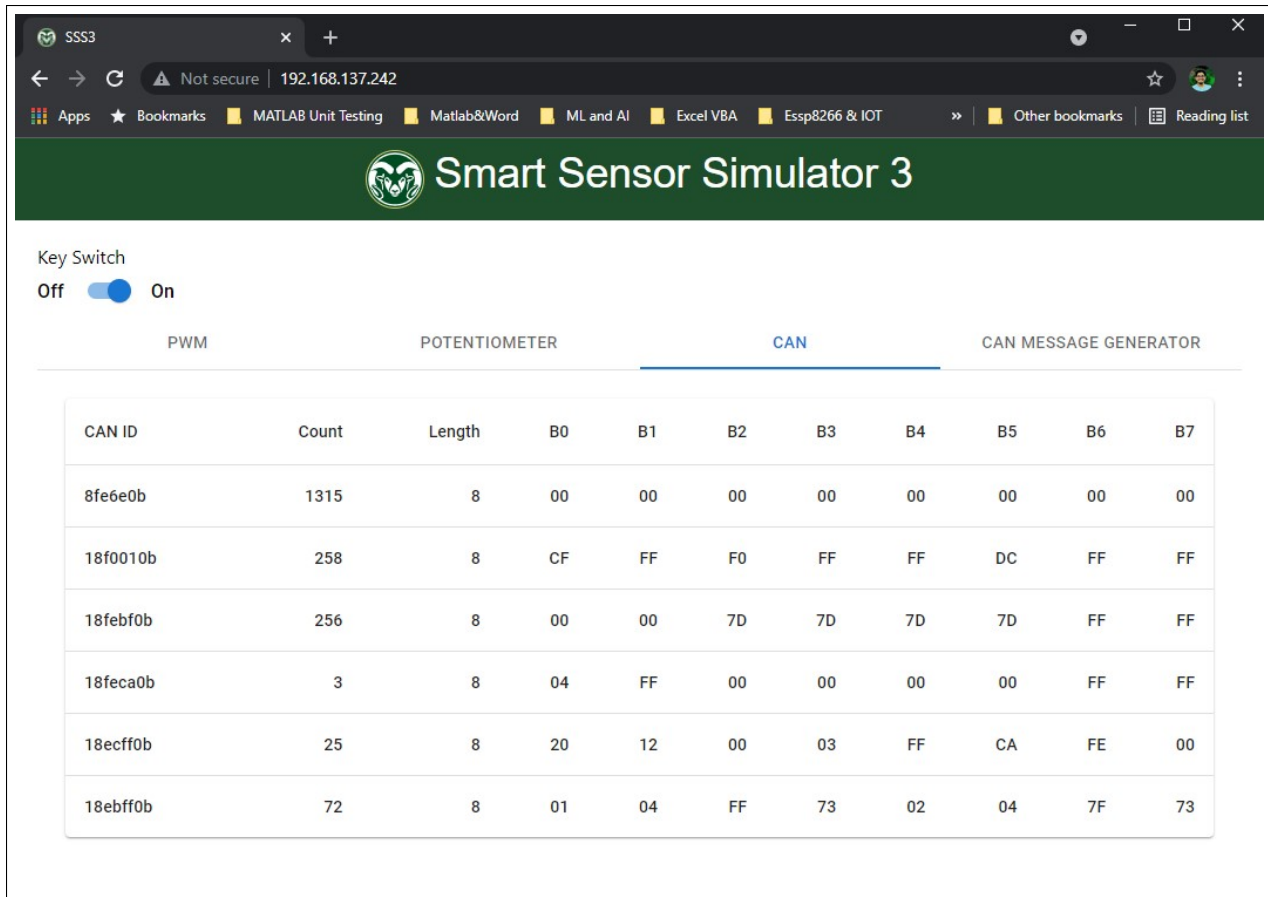
The PWM tab allows users to modify the duty cycle and frequency for all the available pins, as shown in Figure 4.6. Each PWM pin also has a toggle switch that allows users to enable/disable the pin globally. The apply button then sends out an HTTP Post request to pwm endpoint the Mini-SSS3



**Figure 4.6: PWM GUI**

### 4.4.3 CAN message viewer

The CAN viewer tab shows the current running messages on the CAN bus. It gives out a summary of all the unique messages filtered by CAN message identifier. Figure 4.7 shows the GUI for the CAN viewer tab.



**Figure 4.7: CAN viewer GUI**

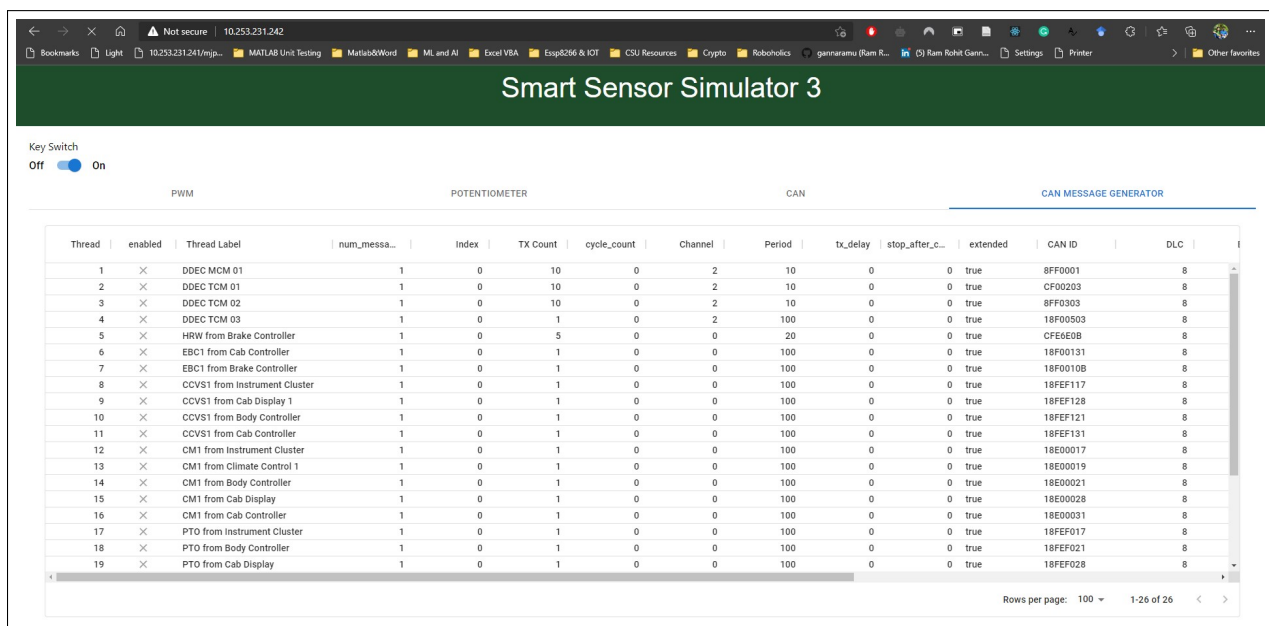
#### 4.4.4 CAN Message Generator

The CAN Message generation tab is shown in Figure 4.8. This tab contains a table with different messages that can be transmitted from the Mini-SSS3. Each row in the table is a CAN message thread that is responsible for generating CAN messages. Each CAN message thread has the following parameters that the user can modify.

1. CAN Message ID
2. DLC - It specifies the length of the CAN message field
3. B0-B7 - it specifies the data fields of the 8 byte CAN message

4. Stop\_after\_count - The CAN message thread is disabled after the transmission count(tx\_count) has reached the specified stop\_after\_count value.
5. TX Count - It specifies the current status of the number of message transmitted. (this is not a user modifiable field)
6. Period - It specifies the time delay between consecutive CAN messages
7. Channel - It specifies the CAN channel on which the message has to be transmitted.
8. enabled - It specifies if the thread is currently enabled Thread Label - It specifies the thread name for each CAN message thread.

Users have the option to edit the CAN message thread parameters as mentioned above.



**Figure 4.8:** CAN message generator GUI

## 4.5 Functional Unit Tests

The HTTP application programming interface is a main way the GUI gets information from the Mini-SSS3. The following tests were performed using the requests library in Python 3.9.

### 4.5.1 GET method tests

#### /pwm

The following python code is used to test the response from the HTTP GET method for the pwm endpoint. We can observe from the output below, that pwm endpoint returns a json document enumerating different PWM parameters such as the duty cycle, frequency and switch values.

```
import requests
url = 'http://192.168.137.119/pwm'
x = requests.get(url)
print(x.text)
```

#### Output

```
{
  "0": {
    "duty": {"value": 500},
    "freq": {"value": 300},
    "sw": {"value": 1}
  },
  "1": {
    "duty": {"value": 1000},
    "freq": {"value": 245},
    "sw": {"value": 1}
  },
  ...
}
```



## /pots

The following python code is used to test the response from the HTTP GET method for the potentiometer endpoint(/pots). We can observe from the output below, that pots endpoint returns a json document enumerating different parameters such as the wiper position and terminal connections on the MCP41HV51 (TCON registers). The output is truncated for readability.

```
import requests
url = 'http://192.168.137.119/pots'
x = requests.get(url)
print(x.text)
```

Output:

```
{
  "0": {
    "wiper": {
      "value": 21
    },
    "TCON": {
      "value": 7,
      "meta": "TBD"
    }
  },
  "1": {
    "wiper": {
      "value": 56
    },
    "TCON": {
      "value": 7,
      "meta": "TBD"
    }
  },
  ...
}
```

## /voltage

```
import requests
url = 'http://192.168.137.119/voltage'
x = requests.get(url)
print(x.text)
```

The following Python code is used to test the response from the HTTP GET method for the voltage endpoint (/volts). This endpoint is constantly queried by the webpage to get real-time voltage information. From the output below, we can observe that the voltage endpoint returns a JSON document enumerating different channels on the PAC1934, and each channel has the voltage and current information provided by the PAC1934. Current flow can be extracted from PAC1934 but is not currently implemented, but the API has a place holder for future versions.

Output:

```
{
  "0": {
    "voltage": 0.537109,
    "current": -1
  },
  "1": {
    "voltage": 1.183105,
    "current": -1
  },
  "2": {
    "voltage": 1.183105,
    "current": -1
  },
  "3": {
    "voltage": 4.86377,
    "current": -1
  }
}
```

## /can

The following Python code is used to test the response from the HTTP GET method for the CAN endpoint (/can). This endpoint is also constantly queried by the webpage to get a summary of the message that are currently running on the CAN bus. From the output below, we can observe that the CAN endpoint returns a JSON document enumerating different CAN IDs and the count of messages with the same message ID and also the last received data packet information. This API endpoint only provides the users with the type of message ID's running on the CAN bus.

```
import requests

url = 'http://192.168.137.119/can'

x = requests.get(url)

print(x.text)
```

## Output:

```
{
  "8fe6e0b": {
    "count": 317,
    "LEN": 8,
    "ID": "8fe6e0b",
    "DATA": [ "00", "00", "00", "00", "00", "00", "00", "00" ]
  },
  "18f0010b": {
    "count": 64,
    "LEN": 8,
    "ID": "18f0010b",
    "DATA": [ "CF", "FF", "F0", "FF", "FF", "DC", "FF", "FF" ]
  },
  "18febf0b": {
    "count": 64,
    "LEN": 8,
    "ID": "18febf0b",
    "DATA": [ "00", "00", "7D", "7D", "7D", "7D", "FF", "FF" ]
  },
  "18feca0b": {
```

```

    "count": 3,
    "LEN": 8,
    "ID": "18feca0b",
    "DATA": ["04", "FF", "00", "00", "00", "00", "FF", "FF"]
  },
  ...
}

```

## /cangen

The following Python code is used to test the response from the HTTP GET method for the CAN message generator endpoint (/cangen). This endpoint is queried by the webpage to get a summary of the CAN threads that are currently running on the Mini-SSS3. From the output below, we can observe that the cangen endpoint returns a JSON document enumerating different message thread ID's along with all the thread parameters. The output is truncated for readability.

```

import requests

url = 'http://192.168.137.119/cangen'
x = requests.get(url)

print(x.text)

```

## Output:

```

{
  "00": {
    "ThreadName": "CI from SSS2",
    "ThreadID": 0,
    "enabled": false,
    "num_messages": 6,
    "message_index": 0,
    "transmit_number": 6,
    "cycle_count": 0,
    "channel": 0,
    "tx_period": 1,
    "tx_delay": 0,
    "stop_after_count": 6,

```

```
"extended": true,
"ID": "18ECFFFA",
"DLC": 8,
"DATA": [
  "20",
  "1d",
  "0",
  "5",
  "ff",
  "eb",
  "fe",
  "0"
]
},
"01": {
  "ThreadName": "CI from SSS2",
  ...
}
}
```

## 4.5.2 POST method tests

### /pwm

The following Python code is used to test the functioning of the HTTP POST method for the PWM endpoint (/pwm). A json payload similar to the one received during a HTTP GET message with the desire frequency and duty cycle values are sent to the Mini-SS3. The Mini-SS3 then processes that request and responds back with the current state of the all PWM channels. Output on the serial monitor of the Mini-SS3 is shown in Figure 4.9.

```
import requests
import json
url = "http://192.168.137.119/pwm"
payload = json.dumps({
    "0": {
        "duty": {"value": 500},
        "freq": {"value": 300},
        "sw": {"value": 1}
    }
})
headers = { 'Content-Type': 'application/json' }
response = requests.request("POST", url, headers=headers, data=payload)
print(response.text)
>>>
{
    "0": {
        "duty": {
            "value": 500
        },
        "freq": {
            "value": 300
```

```
    },
    "sw": {
        "value": 1
    }
},
"1": { ... },
...
}
```

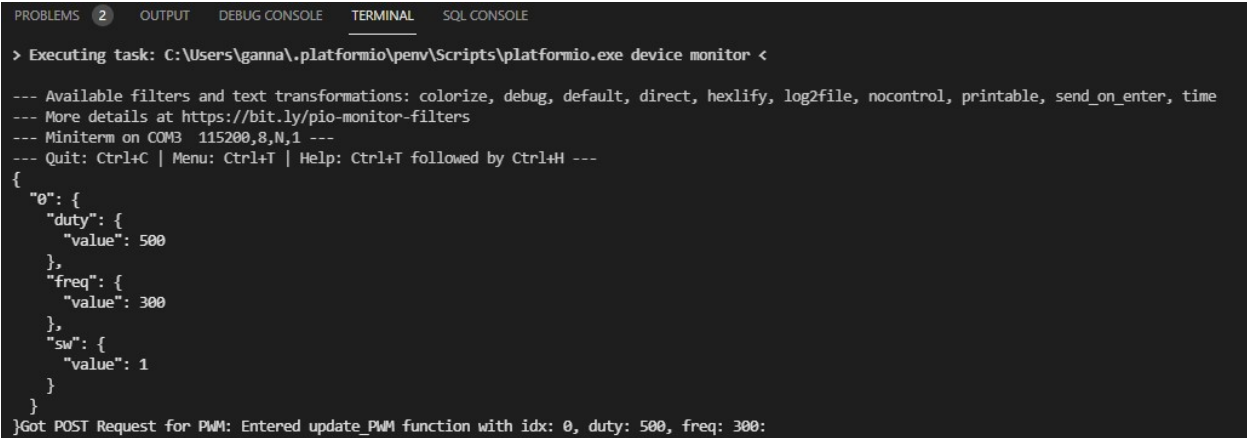


Figure 4.9: POST request serial monitor output

**/pots**

The following Python code is used to test the functioning of the HTTP POST method for the Potentiometer endpoint (/pots). A JSON payload similar to the one received during a HTTP GET message with the desire wiper position and terminal connection values are sent to the Mini-SSS3. The Mini-SSS3 then processes that request and responds back with the current state of the all Potentiometer channels.

```
import requests
import json
url = "http://192.168.137.119/pots"
```

```

payload = json.dumps({
    "0": {
        "wiper": {"value": 22},
        "TCON": {"value": 7},
    }
})
headers = { 'Content-Type': 'application/json' }
response = requests.request("POST", url, headers=headers, data=payload)
print(response.text)
>>>
{
    "0": {
        "wiper": {
            "value": 22
        },
        "TCON": {
            "value": 7
        }
    },
    "1": {...},
    ...
}

```

## 4.6 Conclusion

In this section the overall software requirements were gathered and the main software functions of the Mini-SS3 such as, the API and GUI were designed. Further, these software components were tested to verify their functionalities. The following section discusses in detail about securing the cloud based communication.



# Chapter 5

## Securing Cloud and External Communication

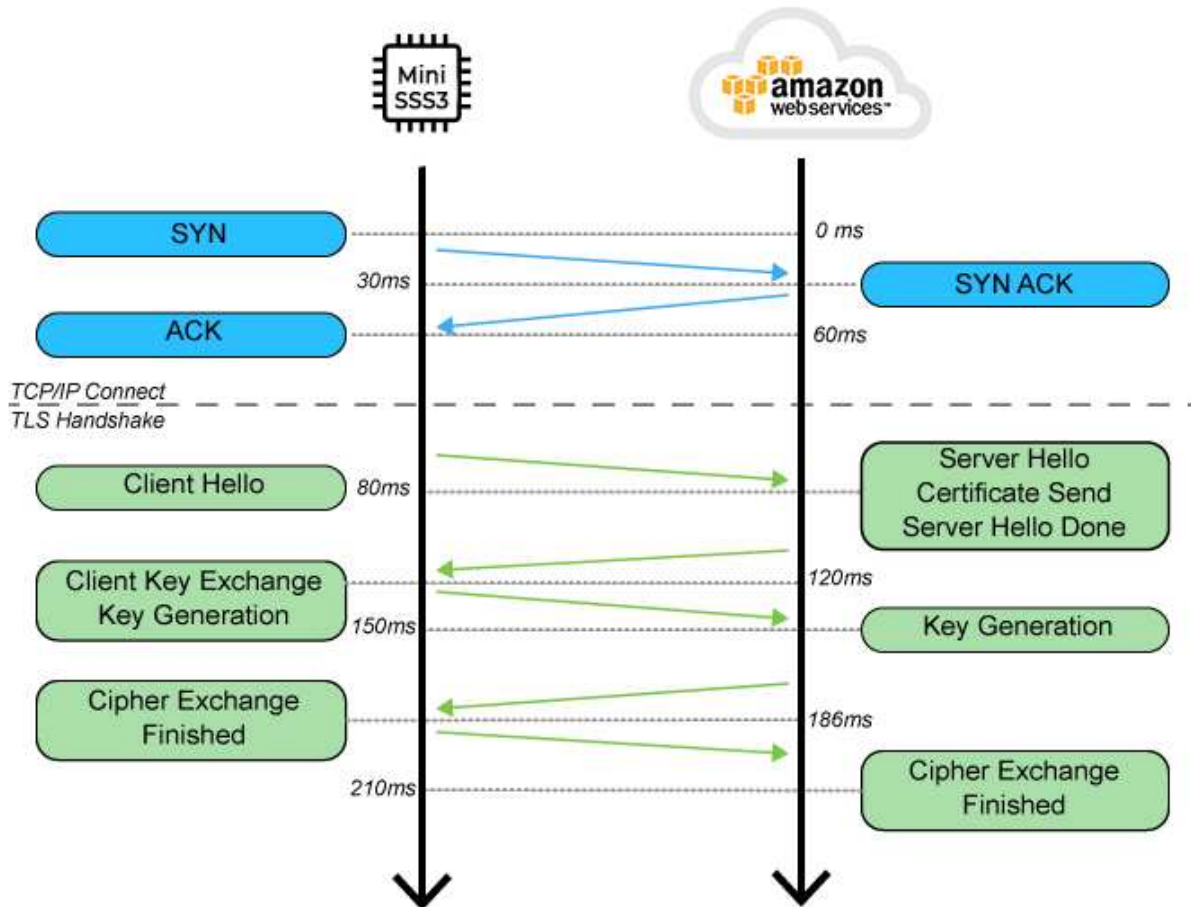
### 5.1 Introduction

Around 21 Billion smart IoT devices are expected to be in the market by the end of 2020, [40] and a lot of these devices are vulnerable to security attacks and tend to be the weakest point in any network. A robust IoT security mechanism allows designers to protect devices from all types of vulnerabilities while deploying the product to production. Implementing cryptography adds overhead on both execution time and energy consumption and is usually costly to implement, as not everyone is skilled to implement them on constrained embedded devices. This section describes the use of Microchip's ATECC608B security module to store private keys and establish a TLS connection to a server. Often, building and maintaining your server infrastructure could be complex and time-consuming. Therefore, third-party cloud providers offer ready-to-go infrastructure at affordable prices. For this thesis, Amazon Web Services (AWS) enables remote connectivity on the Mini-SSS3. AWS IoT Core is a cloud service to enable connected devices to communicate with cloud applications and other devices. Amazon IoT Core is capable of handling billions of devices and trillions of communications reliably and securely, routing them to AWS endpoints and other devices. AWS IoT Core requires devices to use X.509 certificates with TLS for authentication. More information about the certificates and the TLS is discussed in the following sections.

### 5.2 Transport Layer Security 1.2 (TLS)

The TLS encryption protocol was designed to help protect Internet communication from eavesdropping. Starting a communication session using TLS encryption is called a TLS handshake. During a TLS handshake, the client and the server exchange messages to verify each other, decide on an encryption algorithm, acknowledge each other, and agree on the session keys. TLS hand-

shakes are also an integral and foundational part of how HTTPS works. The broad steps involved during a TLS handshake are listed below:



**Figure 5.1:** TLS Handshake

1. Client hello: A hello message is sent by the client with the protocol version, the client random, along with the list of cipher suites.
2. Server hello: Then server replies with the server random, SSL certificate, and the selected cipher suite.
3. Server's digital signature: In this, the private key is used by the server to encrypt the server random, its DH parameter\* and client random. The function of this encrypted data is to work

as the server's digital signature for establishing that the particular server has a private key that matches with the public key from the SSL certificate.

4. Digital signature confirmed: The client then decrypts the server's digital signature with the help of the public key and verify that the server controls the private key.
5. Client DH parameter: The DH parameter is sent by the client to the server.
6. Client and server calculate the premaster secret: The client and server use the DH parameters they exchanged to calculate a matching premaster secret separately, instead of the client generating the premaster secret and then sending it to the server.
7. Session keys created: The client and server calculate session keys from the premaster secret, client random, and server random.
8. Client and Server are now set up to use symmetric encryption.

### **5.2.1 ECDSA sign and verify test**

The following test is to validate the Elliptic Curve Digital Signature Algorithm on the Microchip's ATECC608B chip. To truly validate the functionality, the logic has been tested over two separate Teensy 4.0 devices with different ATECC608B chips. Device A signs a message with the private key stored on the ATECC608B chip and then sends the public key (corresponding to the private key used to sign the message), signature, and the message to the other device. Device B then uses that information to verify the signature with the public key and the message to validate that the message was actually sent from device A.

## Device A code:

```
#include <ArduinoECCX08.h>

byte signature[64];

byte message[32] = {
  0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
  0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13,
  0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D,
  0x1E, 0x1F
};

byte publicKey[64];

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  ECCX08.begin();

  String serialNumber = ECCX08.serialNumber();
  Serial.print("ECCX08 Serial Number = ");
  Serial.println(serialNumber);

  ECCX08.ecSign(0, message, signature);
  ECCX08.generatePublicKey(0, publicKey);
  printMessage();
  printPublicKey();
  printSignature();
}
```

```

COM3 (Teensy) Serial
ECCX08 Serial Number = 01234E5F5936FF28EE
byte message[32] = {
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
};
byte publicKey[64] = {
0x98, 0x59, 0xFE, 0x69, 0x8D, 0x69, 0x82, 0x05, 0xEB, 0x3C, 0xCA, 0x40, 0x09, 0xC5, 0x4D, 0xA5,
0x01, 0xC1, 0xF9, 0x1E, 0xF5, 0x0B, 0xD3, 0x37, 0x33, 0x8F, 0xE5, 0xCE, 0xE7, 0x24, 0x83, 0xC6,
0x82, 0x93, 0x0A, 0x9D, 0x13, 0xAE, 0x27, 0x3B, 0xB9, 0x96, 0x3E, 0xD7, 0x82, 0x0F, 0xD7, 0xE7,
0xFD, 0x40, 0x9E, 0x40, 0x75, 0x6B, 0x15, 0xF2, 0x28, 0x35, 0x5C, 0x5F, 0x4D, 0x84, 0x51, 0x2D
};
byte signature[64] = {
0x8E, 0xB2, 0x5C, 0xE7, 0x7A, 0x1A, 0x2B, 0x87, 0xF7, 0x37, 0x30, 0x6A, 0x2F, 0xA9, 0xE5, 0x70,
0x14, 0x52, 0x7B, 0x0D, 0x36, 0xED, 0xD7, 0xB0, 0x7B, 0xA4, 0x60, 0x7F, 0xF0, 0xB9, 0xEF, 0x30,
0x8B, 0x9D, 0x0C, 0x69, 0x40, 0x10, 0x8D, 0xC3, 0x5B, 0x9E, 0x83, 0x07, 0x2F, 0x55, 0x65, 0xF0,
0x83, 0x82, 0x95, 0xBB, 0x95, 0x03, 0x76, 0x31, 0x24, 0x16, 0xED, 0x98, 0x16, 0xB4, 0x9E, 0xC3
};
Autoscroll
Newline
Clear output

```

**Figure 5.2:** ECDSA signature output

The message, signature, and public key from the output of device A are then passed on to the code of Device B to verify. We can clearly distinguish both the outputs have different ATECC serial numbers.

Device B code:

```

#include <ArduinoECCX08.h>

byte message[32] = {
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A,
0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
};

byte publicKey[64] = {
0x98, 0x59, 0xFE, 0x69, 0x8D, 0x69, 0x82, 0x05, 0xEB, 0x3C, 0xCA,
0x40, 0x09, 0xC5, 0x4D, 0xA5, 0x01, 0xC1, 0xF9, 0x1E, 0xF5, 0x0B,
0xD3, 0x37, 0x33, 0x8F, 0xE5, 0xCE, 0xE7, 0x24, 0x83, 0xC6, 0x82,

```

```

0x93, 0x0A, 0x9D, 0x13, 0xAE, 0x27, 0x3B, 0xB9, 0x96, 0x3E, 0xD7,
0x82, 0x0F, 0xD7, 0xE7, 0xFD, 0x40, 0x9E, 0x40, 0x75, 0x6B, 0x15,
0xF2, 0x28, 0x35, 0x5C, 0x5F, 0x4D, 0x84, 0x51, 0x2D
};

```

```

byte signature[64] = {
0x8E, 0xB2, 0x5C, 0xE7, 0x7A, 0x1A, 0x2B, 0x87, 0xF7, 0x37, 0x30,
0x6A, 0x2F, 0xA9, 0xE5, 0x70, 0x14, 0x52, 0x7B, 0x0D, 0x36, 0xED,
0xD7, 0xB0, 0x7B, 0xA4, 0x60, 0x7F, 0xF0, 0xB9, 0xEF, 0x30, 0x8B,
0x9D, 0x0C, 0x69, 0x40, 0x10, 0x8D, 0xC3, 0x5B, 0x9E, 0x83, 0x07,
0x2F, 0x55, 0x65, 0xF0, 0x83, 0x82, 0x95, 0xBB, 0x95, 0x03, 0x76,
0x31, 0x24, 0x16, 0xED, 0x98, 0x16, 0xB4, 0x9E, 0xC3
};

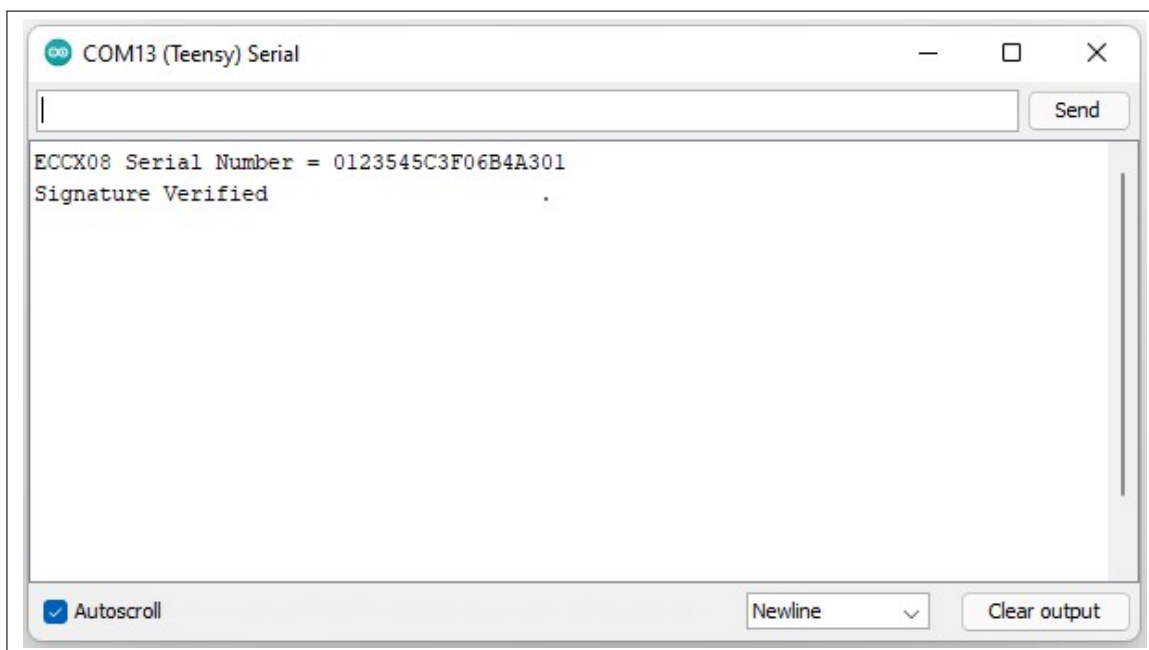
```

```

void setup()
{
  Serial.begin(9600);
  while (!Serial);
  ECCX08.begin(0x35);
  String serialNumber = ECCX08.serialNumber();
  Serial.print("ECCX08 Serial Number = ");
  Serial.println(serialNumber);
  if(ECCX08.ecdsaVerify(message, signature, publicKey))
  {
    Serial.println("Signature Verified");
  }
  else
  {
    Serial.println("Signature Failed");
  }
}

```

```
}  
}  
}
```



**Figure 5.3:** ECDSA verify output

### **X.509 Certificates**

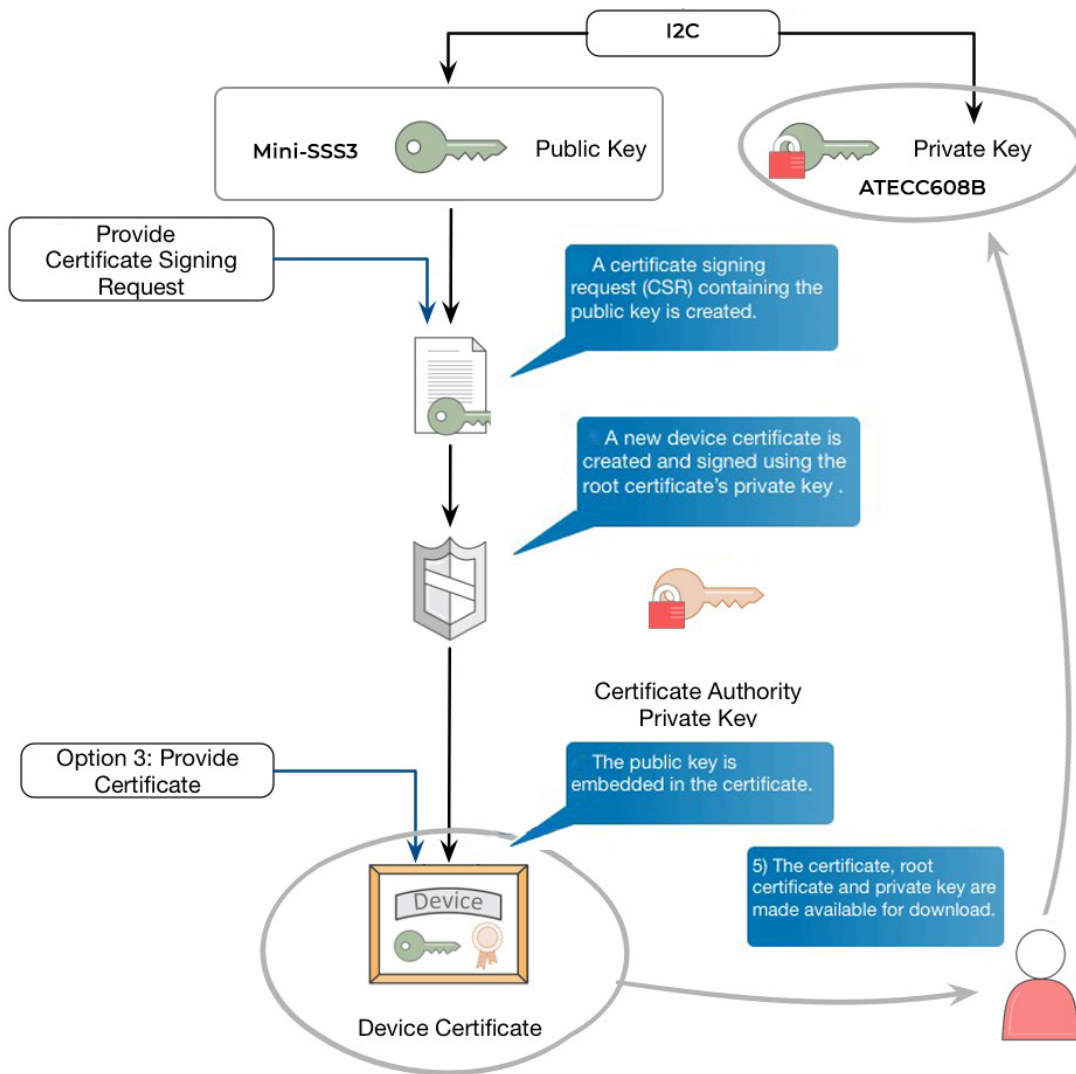
An X.509 certificate document is used to prove ownership of a public key. To generate a new X.509 certificate, the program needs to create a certificate signing request (CSR) and provide it to a certificate authority (CA). The CSR is a digital document that contains the public key and other identifying information. The CA validates the identifying information, and once the identity has been verified, the CA creates a certificate and signs it with a private key. Anyone can now validate the certificate by checking its digital signature with the CA's public key. [41]

AWS IoT provides three different options to create a new certificate. The easiest option is to use one-click generation. Here, AWS will create a public and private key and create a new certificate signed by the AWS IoT CA. The second option is to provide an own CSR. This gives the advantage

of not sharing the private key. The new certificate generated from the CSR is then signed by the AWS IoT CA. The final option is to bring your own certificate signed by your own trusted CA.

A CSR is generated for the public key, which is generated from the private key stored in the ATECC608B chip during the initial provisioning process. This CSR is provided to AWS IOT, which then provides us with a signed X.509 certificate. This process makes sure that the private key has not been compromised even during the provisioning process. The Figure 5.4 shows how a new X.509 certificate is generated from a CSR by AWS IoT.





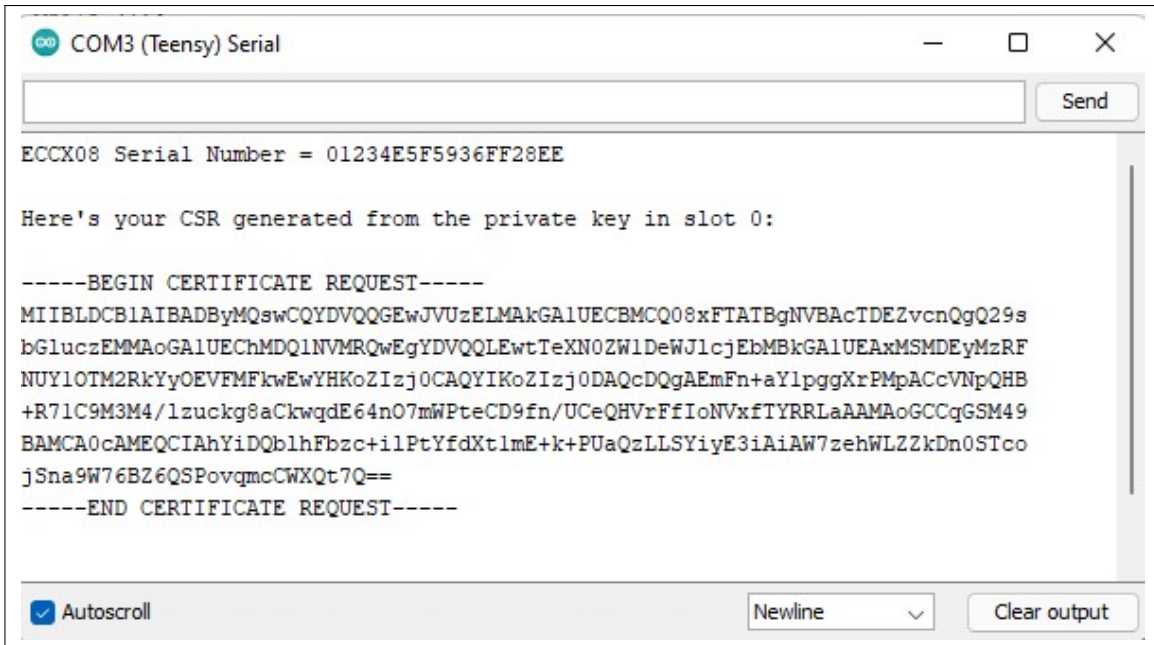
**Figure 5.4: AWS Certificate Registration**

### **ATECC608B CSR generation test**

Generating a certificate signing request (CSR) is a crucial step in the provisioning process. A modified version of the ArduinoECCX08 library [42] was used to interface the ATECC608B chip with Teensy 4.0. The ATECC608 chip communicates with the Teensy over I2C at address 0x60. The following test generates a CSR based on the information about the device (e.g. common name, organization, country), which is then used by the certificate authority (CA) to create a signed X.509

certificate. The certificate also contains the public key and a signature generated from the private key.

```
#include <ArduinoECCX08.h>
#include <utility/ECCX08CSR.h>
void setup() {
  Serial.begin(9600);
  while (!Serial);
  ECCX08.begin(0x60);
  String serialNumber = ECCX08.serialNumber();
  Serial.print("ECCX08 Serial Number = ");
  Serial.println(serialNumber);
  Serial.println();
  ECCX08CSR.begin(0, 0);
  ECCX08CSR.setCountryName("US");
  ECCX08CSR.setStateProvinceName("CO");
  ECCX08CSR.setLocalityName("Fort Collins");
  ECCX08CSR.setOrganizationName("CSU");
  ECCX08CSR.setOrganizationalUnitName("SystemCyber");
  ECCX08CSR.setCommonName(serialNumber.c_str());
  String csr = ECCX08CSR.end();
  Serial.println("Here's your CSR, enjoy!");
  Serial.println(csr);
}
```



**Figure 5.5:** Certificate signing request test

### 5.3 Implementation

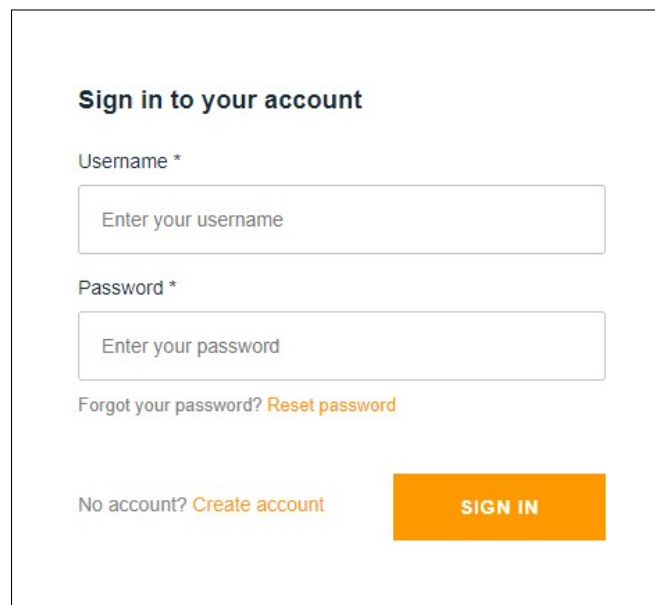
Multiple open-source implementations of SSL/TLS protocol (RFC 5246) [43] are available which help secure the communication over the Internet. BearSSL was one such implementation that was focused on implementing SSL/TLS protocol on embedded devices. ArduinoBearSSL a port of BearSSL for Arduino compatible platforms was used in this project. Initially, during the provisioning process, each device generates a CSR from the private key which is locked in the ATECC608B chip. The CSR is then registered with the AWS IOT and AWS returns a signed X.509 certificate. This certificate is then stored on to the device. The signed X.509 certificate is used to establish TLS communication with AWS.

It is possible for devices to communicate with AWS IoT Core via HTTP, WebSockets, and MQTT. In the current implementation, the Mini-SSS3 communicates with AWS IoT core using the MQTT protocol. MQTT (Message Queuing Telemetry Transport) is an extremely lightweight M2M (machine-to-machine) connection protocol that provides a messaging subscription and pub-

lish transport. To enable access to the device remotely, a similar react webpage as shown in Section 4.4 is implemented on the AWS side.

## User login

When the user first enters the webpage [44] a login screen is presented to the user providing a username and password dialog boxes as show in Figure 5.6. New users can also sign up on this webpage and requires approval by an administrator. An email verification process is implemented to prevent the system from becoming overloaded with invalid email addresses. On completion of account registration and approval from an administrator, the new user is added to the AWS User Pool and is granted access to the web interface to control the Mini-SSS3.



**Sign in to your account**

Username \*

Password \*

Forgot your password? [Reset password](#)

No account? [Create account](#)

**SIGN IN**

**Figure 5.6:** AWS user login screen

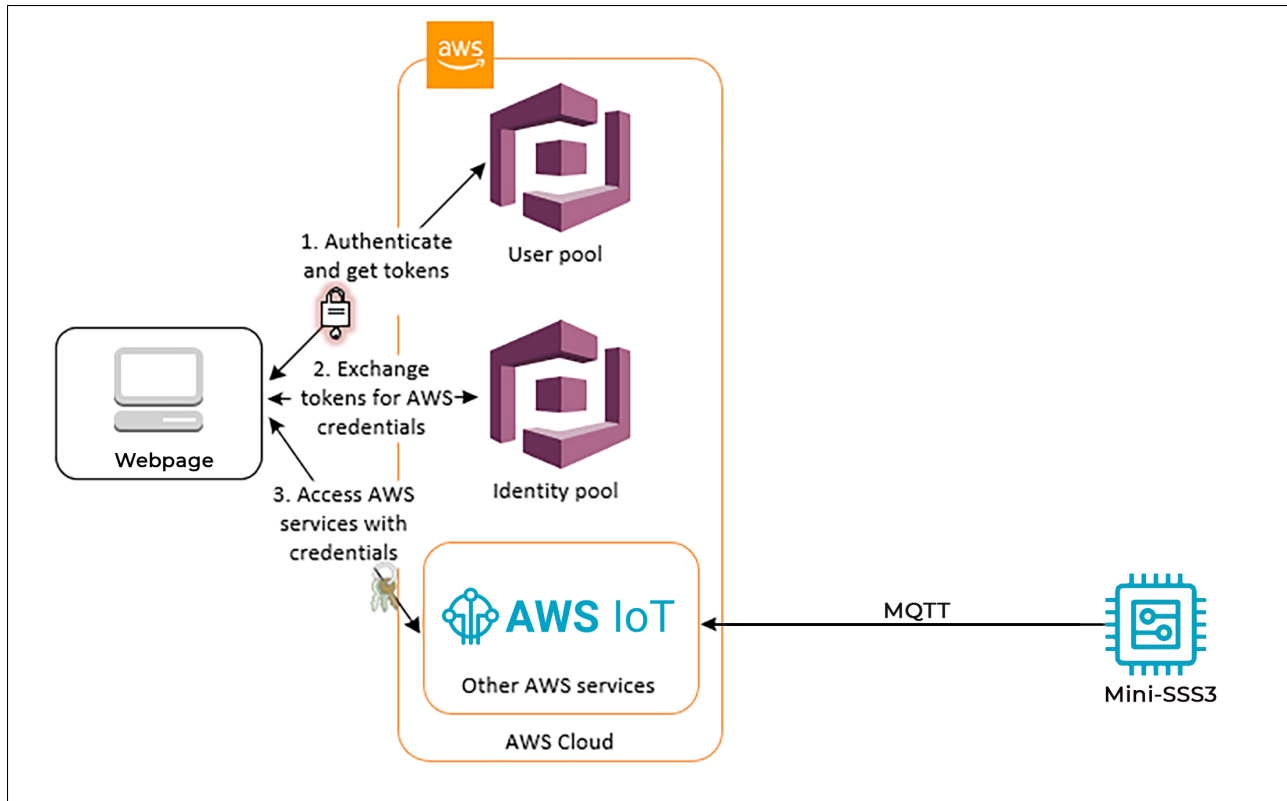
The image shows a web form for creating a new AWS account. The form is titled "Create a new account" and contains the following fields:

- Username \***: A text input field with the placeholder text "Username".
- Password \***: A text input field with the placeholder text "Password".
- Email Address \***: A text input field with the placeholder text "Email".
- Phone Number \***: A field with a dropdown menu showing "+1" and a text input field containing "(555) 555-1212".

At the bottom of the form, there is a link "Have an account? [Sign in](#)" and a prominent orange button labeled "CREATE ACCOUNT".

**Figure 5.7:** AWS user signup

Once the user login's using their username and password, they are authenticated with the AWS user pool, which then returns `accessKeyID`, `session`, and `secretAccessKey` tokens. These tokens are then used to determine which user has access to a particular AWS-IOT (Mini-SSS3) device. The login procedure is implemented in accordance with AWS recommended practices [45]. The webpage to access the Mini-SSS3 is remotely hosted on AWS Amplify, a serverless architecture that enables faster deployments for both front-end and back-end applications. The user interface is similar to that shown in Section4.4. The main difference is how it retrieves information from the device. The Mini-SSS3 and the webpage are subscribed to a common MQTT topic, and they share information on that topic. Any change in parameters on the webpage is communicated to the device, and similarly, the device updates its state to the webpage over the common MQTT topic they are subscribed to.



**Figure 5.8:** Access control with AWS Cognito user pool

## 5.4 Cloud Communications Summary

In this Section, the technicalities about securing the cloud-based communication over TLS utilizing hardware security modules were discussed. Further, buffer length, and a few function prototypes were modified in the Wire library, [46] which is responsible for handling the I2C communication of the Teensy 4.0 for compatibility with the AurdinoECCX08 library for ATECC608 [42], The following Section gives us a brief conclusion of the thesis and the significant contributions and limitations.

# Chapter 6

## Conclusion

The Mini-SS3 was designed following the SAE J3061 guidebook and the following are the major contributions in this thesis:

1. An approach that utilizes hardware security modules to increase the security posture of the embedded IoT devices was presented. This approach can be implemented for various other IoT applications. During the entire process of provisioning, the private keys are never exposed and are only accessible to the functions within the hardware security module.
2. A new hardware sensor simulator Mini-SS3 was designed with improvements over earlier generation devices. As part of the new design, a voltage feedback loop was added to provide users with the actual state of the device. Also, an Ethernet module was added to offer remote operability.
3. A graphical user interface based on the React framework was designed which is served directly from the device's firmware without having to install any additional software or drivers.
4. The Mini-SS3 offers secure remote connection to third-party cloud service providers over TLS utilizing the hardware security module for secure key storage.

### 6.1 Limitations and Future work

The Secure Ethernet-based connection enables the Mini-SS3 to be utilized in advanced research concepts like the software-defined truck, which can accelerate system-level testing for heavy vehicle cybersecurity research. Microchip's ATECC608B-TNGTLS variant is a pre-provisioned variant of the ATECC608B. The device comes pre-configured and pre-provisioned with default thumbprint certificates which can be used to make a connection to AWS IOT, Azure, and Google

cloud. Implementing them in the design can eliminate the steps of provisioning the device which could be a great relief when deploying such devices at a large scale.

The current HTTP API is insecure and vulnerable an HTTPS version needs to be implemented which can make the device standalone and serve secure web pages. A single Mini-SSS3 may not be able to make a fault-free environment for all ECUs and might require multiple devices. Any utilization of the Mini-SSS3 for forensic or investigative work is done at the risk of the user.



# Bibliography

- [1] National Highway Traffic Safety Administration, “Overview of the 2018 Crash Investigation Sampling System,” 2020. [Online]. Available: <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812971>
  
- [2] United States: National Archives and Records Administration: Office of the Federal Register, “EVENT DATA RECORDERS,” in *Transportation. Title 49*. Office of the Federal Register, National Archives and Records Administration, Oct. 2011. [Online]. Available: <https://www.govinfo.gov/app/details/CFR-2011-title49-vol6/CFR-2011-title49-vol6-part563>
  
- [3] J. Daily, A. Kongs, J. Johnson, and J. Corcega, “Extracting Event Data from Memory Chips within a Detroit Diesel DDEC V,” SAE International, Warrendale, PA, SAE Technical Paper 2015-01-1450, Apr. 2015, ISSN: 0148-7191, 2688-3627. [Online]. Available: <https://www.sae.org/publications/technical-papers/content/2015-01-1450/>
  
- [4] S. A. van Nooten and J. R. Hrycay, “The application and reliability of commercial vehicle event data recorders for accident investigation and analysis,” *SAE Transactions*, pp. 1286–1293, 2005.
  
- [5] J. S. Ogden and M. Martonovich, “Forensic Engineering Tools and Analysis of Heavy Vehicle Event Data Recorders (HVEDRs),” *Journal of the National Academy of Forensic Engineers*, vol. 33, no. 2, 2016.
  
- [6] X. Feng, E. S. Dawam, and S. Amin, “A new digital forensics model of smart city automated vehicles,” in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2017, pp. 274–279.
  
- [7] J. L. Córcega, “Design of a forensically neutral electronic environment for heavy vehicle event data recorders,” Master’s thesis, University of Tulsa, 2015.

- [8] Smart Sensor Simulator 2 Github repository. [Online]. Available: <https://github.com/SystemsCyber/SSS2>
- [9] S. Hamdioui, J.-L. Danger, G. Di Natale, F. Smailbegovic, G. van Battum, and M. Tehranipoor, "Hacking and protecting IC hardware," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2014, pp. 1–7, ISSN: 1558-1101.
- [10] "Hardware Hacking 101: Interfacing With SPI." [Online]. Available: <https://www.riverloopsecurity.com/blog/2020/02/hw-101-spi/>
- [11] H. Garg and M. Dave, "Securing IoT devices and securely connecting the dots using REST API and middleware," in *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2019, pp. 1–6.
- [12] J. Daily, J. Johnson, A. Kongs, and J. Corcega, "Wheeled vehicle event data recorder forensic recovery and preservation system," Jan. 9 2018, US Patent 9,865,102.
- [13] D. Plant, T. Austin, and B. Smith, "Data extraction methods and their effects on the retention of event data contained in the electronic control modules of detroit diesel and mercedes-benz engines," *SAE International Journal of Passenger Cars-Mechanical Systems*, vol. 4, no. 2011-01-0808, pp. 636–647, 2011.
- [14] J. Johnson, J. Daily, and A. Kongs, "On the digital forensics of heavy truck electronic control modules," *SAE International Journal of Commercial Vehicles*, vol. 7, no. 2014-01-0495, 2014.
- [15] B. M. Boggess, A. Dunn, D. Morr, T. Martin, A. Cornetto, and F. Bayan, "A New Passive Interface to Simulate On-Vehicle Systems for Direct-to-Module (DTM) Engine Control Module (ECM) Data Recovery," SAE International, Warrendale, PA, SAE Technical Paper 2010-01-1994, Oct. 2010, ISSN: 0148-7191, 2688-3627. [Online]. Available: <https://www.sae.org/publications/technical-papers/content/2010-01-1994/>

- [16] Synercon Technologies, Smart Sensor Simulator 2. [Online]. Available: <https://synercontechnologies.com/sss2/>
- [17] Mini-SS3 Github repository. [Online]. Available: <https://github.com/SystemsCyber/Mini-SS3>
- [18] “ISO/SAE21434: Road Vehicles - Cybersecurity Engineering - SAE International.” [Online]. Available: <https://www.sae.org/standards/content/iso/sae21434/>
- [19] J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems - SAE International. [Online]. Available: [https://www.sae.org/standards/content/j3061\\_201601/](https://www.sae.org/standards/content/j3061_201601/)
- [20] “J1708: Serial Data Communications Between Microcomputer Systems in Heavy-Duty Vehicle Applications - SAE International.” [Online]. Available: [https://www.sae.org/standards/content/j1708\\_201012/](https://www.sae.org/standards/content/j1708_201012/)
- [21] “J1587: Electronic Data Interchange Between Microcomputer Systems in Heavy-Duty Vehicle Applications - SAE International.” [Online]. Available: [https://www.sae.org/standards/content/j1587\\_201301/](https://www.sae.org/standards/content/j1587_201301/)
- [22] Microchip MCP41HV51 digital potentiometer datasheet. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/20005207B.pdf>
- [23] Microchip’s PAC1934 Quad DC Power Monitor Datasheet. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/PAC1931-Family-Data-Sheet-DS20005850E.pdf>
- [24] Microchip MCP2562 High-Speed CAN Transceiver Datasheet. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/20005167C.pdf>
- [25] Wiznet WIZ850io Ethernet module. [Online]. Available: [https://docs.wiznet.io/img/products/w5500/w5500\\_ds\\_v109e.pdf](https://docs.wiznet.io/img/products/w5500/w5500_ds_v109e.pdf)

- [26] Microchip ATECC608B Crypto Trust platform Datasheet. [Online]. Available: <https://www.microchip.com/content/dam/mchp/documents/SCBU/ProductDocuments/DataSheets/ATECC608B-CryptoAuthentication-Device-Summary-Data-Sheet-DS40002239A.pdf>
- [27] PJRC, Teensy 4.0 development board schematics. [Online]. Available: <https://www.pjrc.com/teensy/schematic.html>
- [28] Mini-SS3 Schematics PDF. [Online]. Available: [https://github.com/SystemsCyber/Mini-SS3/blob/main/docs/Mini\\_SSS3.pdf](https://github.com/SystemsCyber/Mini-SS3/blob/main/docs/Mini_SSS3.pdf)
- [29] Teensy 4.0 Development Board Specifications. [Online]. Available: <https://www.pjrc.com/store/teensy40.html>
- [30] International Standards Organization, *11898-2:2016 Road vehicles — Controller area network (CAN) — Part 2: Highspeed medium access unit*. December, 2016. [Online]. Available: <https://www.iso.org/standard/67244.html>
- [31] 24-Pin Molex Connector. [Online]. Available: [https://www.molex.com/pdm\\_docs/sd/039301242\\_sd.pdf](https://www.molex.com/pdm_docs/sd/039301242_sd.pdf)
- [32] Kycon female power connector. [Online]. Available: <http://www.kycon.com/2013Catalogpage/DC%20Power/KPJX.pdf>
- [33] Hammond Aluminium enclosure specification (PN:1455K1201BK). [Online]. Available: <https://www.hammfg.com/part/1455K1201BK>
- [34] Mini-SS3 hardware tests repository. [Online]. Available: <https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/>
- [35] MCP41HV51 Digital potentiometer test. [Online]. Available: [https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test\\_Pots/Test\\_Pots.ino](https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test_Pots/Test_Pots.ino)
- [36] Microchip PAC1934 voltage monitor test. [Online]. Available: [https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test\\_PAC1934/Test\\_PAC1934.ino](https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test_PAC1934/Test_PAC1934.ino)

- [37] Mini-SS3 Read CAN messages test. [Online]. Available: [https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test\\_CAN1/Test\\_CAN1.ino](https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test_CAN1/Test_CAN1.ino)
- [38] Mini-SS3 Generate CAN messages test. [Online]. Available: [https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test\\_CAN1/Test\\_CAN1.ino](https://github.com/SystemsCyber/Mini-SS3/blob/main/Tests/Test_CAN1/Test_CAN1.ino)
- [39] L. Lukkari, *AWOT: Arduino web server library*. May, 2018. [Online]. Available: <https://github.com/lasselukkari/aWOT>
- [40] P. R. Gartner, *Gartner Reveals Top Predictions for IT Organizations and Users in 2017 and Beyond*. October, 2016. [Online]. Available: <http://www.gartner.com/newsroom/id/3482117>
- [41] N. Corbett, *Understanding the AWS IoT Security Model*. May, 2017. [Online]. Available: <https://aws.amazon.com/blogs/iot/understanding-the-aws-iot-security-model>
- [42] ArduinoECCX08, ATECC608B Arduino library. [Online]. Available: <https://github.com/arduino-libraries/ArduinoECCX08.git>
- [43] [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc5246>
- [44] Mini-SS3 webpage hosted on AWS. [Online]. Available: <https://add-aws-front-end.dalv93leipvmw.amplifyapp.com/>
- [45] Amazon Web Services, Access Resources with API Gateway and Lambda with a User Pool. [Online]. Available: <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-scenarios.html>
- [46] Teensy 4.0 Wire library. [Online]. Available: <https://github.com/PaulStoffregen/Wire.git>