ASHESI

ASHESI UNIVERSITY

PROTOTYPING A CAN BUS NODE FOR PREDICTIVE VEHICLE MAINTENANCE.

CAPSTONE PROJECT

B.Sc. Computer Engineering

MAC-NOBLE BRAKO-KUSI

2019

ASHESI UNIVERSITY

PROTOTYPING A CAN BUS NODE FOR PREDICTIVE VEHICLE MAINTENANCE.

CAPSTONE PROJECT

Capstone Project submitted to the Department of Engineering, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Computer Engineering.

Mac-Noble Brako-Kusi

2019

DECLARATION

I hereby declare that this capstone is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……………………………………………………………………………………

Candidate's Name:

……………………………………………………………………………………

Date:

……………………………………………………………………………………

I hereby declare that preparation and presentation of this capstone were supervised in accordance with the guidelines on supervision of capstone laid down by Ashesi University College.

Supervisor's Signature:

……………………………………………………………………………………

Supervisor's Name:

……………………………………………………………………………………

Date:

……………………………………………………………………………………

Acknowledgements

I would like to thank my supervisor, Dr. Robert Sowah, for his immense support and input. I would also like to thank the faculty of engineering for the tools provided, that made the execution of this project possible.

Abstract

The modern-day automobile is no longer just an analog and mechanical entity. Currently, the most basic of vehicular functions have been computerized. The dedicated hardware assigned to these tasks are electronic control units (ECU). Automobiles consist of a number of ECUs networked together to ensure proper functioning of the vehicle. The overall safety of the vehicle relies on real-time communication between the ECUs. Intra-vehicular communication is possible because of the Controller Area Network (CAN). ECUs are responsible for detecting skids, performing anti-lock braking and providing vehicle diagnostic information. Access to CAN bus could prove useful to mechanics, replacing the trial and error method of identifying vehicle faults. Described in this paper is a hardware and software design of a prototype system that provides real-time CAN bus data. Leveraging on the available CAN bus data, the prototype system will provide vehicle performance data over time. This information should aid in the detection of early detection of vehicle irregularities.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1: Introduction.

1.1 Introduction.

In Ghana, there is arguably no appliance more prevalent than the automobile. The estimated national population has grown by 69.3% from 1991 to 2011, while the estimated population of registered vehicles has increased by 828.2% during that period [1]. With the invention of the Electronic Control Unit (ECU), application-specific computers found their way into vehicles. Early implementations of the ECU controlled simple engine function. This resulted in the overall improvement in fuel efficiency and vehicle performance while lowering carbon emissions. The results and performance delivered by ECUs caused its widespread adoption in the automobile industry.

Modern automobiles now more than ever, rely heavily on the integration of application-specific computers for various vehicular functions. The overall safety of vehicles relies on real-time communication between vehicle sensors and control units. From activities such as steering, braking, and engine management, computers have come to play a pivotal role in monitoring and controlling the state of a vehicle.

Prior to the development of the CAN network, sensors and control units required dedicated point-to-point connection to communicate with each other [2]. As consumers demanded more and more from their vehicles, the number of sensors and control units increased. Consequently, this meant more point-to-point connections, which would increase the complexity and cost of wiring. The introduction of a single central network bus system greatly reduced wiring costs and increased overall efficiency in terms of communication speeds between ECUs on the network.

Data is constantly being transmitted over the CAN network. Since the CAN protocol lacks any form of security, direct access to the network via the datalink connector allows one to view CAN packets [3]. In the case of any vehicular malfunction, the corresponding ECU stores a code that is indicative of the fault and in some cases reports this fault by displaying it on the instrument cluster [4] . The availability of vehicle diagnostic fault codes and vehicle performance data allows for fault detection. Equipping vehicle owners and mechanics with the ability to monitor vehicle performance over time provides several benefits. Principal among these, are enabling the early detection of faults thus avoiding prolonged vehicle downtime and ensuring accurate vehicle diagnoses.

1.2 Background

Automobiles are an integral part of the life of the average Ghanaian. Individuals from all walks of life, commute to their various destinations for leisure, business or to conduct transactions. Automobiles constitute the major means by which individuals locomote. The prevalence of automobiles makes the role mechanics play vital. It is therefore of concern how mechanics service faulty vehicles. Accurate vehicle diagnosis does not only shorten the downtime of vehicles, it also reduces the cost vehicle owners incur.

From interviews conducted, information suggests that the majority of mechanics trained through apprenticeship system rely on experience and guess work to diagnose vehicles. Their approach to identifying a probable cause of a vehicular malfunction is based on speculation and past experience. These mechanics subscribe to the trial-and error approach where, for a given fault attempts are made to investigate all possible causes of the fault. A method of elimination is then used to narrow down toward the root cause of the actual fault.

This approach is ineffective since it wastes the time of both the vehicle owner and mechanic. In some cases, vehicle owners incur unnecessary costs as a result of vehicle repairs. This guess-and-check approach stems from the training majority of mechanics in the system receive. Under this system, an individual shadow a superior and is trained to perform specialized vehicle repairs. Mechanics trained under this system are limited in their capacity to carry out maintenance and repair works on other vehicle types they have not been exposed to. As such making them less equipped to service more modern vehicles.

For these informally trained mechanics to stay relevant, there is the need to adapt to changing trends. Vehicles are becoming increasingly complex as more and more vehicle functions are computerized. Consequently, mechanics being churned out into the system are ill-equipped through the apprenticeship training system. Skills acquired under the apprenticeship system apply to a small segment of vehicles that are gradually fading. Transferring this knowledge to other vehicles poses a lot of problems as vehicles from different manufactures are typically incompatible. The intense competition in the automotive industry encourage the use propriety technology and processes. Therefore, only specific service stations that are affiliates of specific auto manufacturers are trained to properly service and repair modern vehicles. The services such companies offer come at a significant cost. Therefore, the majority of the vehicles in circulation are serviced by mechanics who do not use the proper tools and in some cases lack the requisite skill to offer quality services. These concerns present an opportunity where technology can be used to supplement the skills of mechanics to effectively execute their duties. On the other hand, equipping vehicle owners with such tools that can report the performance of their vehicles enables proactive and predictive measures to be taken to avoid unscheduled vehicle failure. This provides owners with the added benefit of possibly identifying and preempting unplanned vehicle failure.

1.3 Objectives

The goal of this project is to develop a system that monitor vehicle parameters, making the information accessible to the user. This is done by:

- Prototyping an embedded system that is able to monitor real-time vehicle parameters.

- Incorporating IoT functionality for remote vehicle data access via a web platform.
- Setting up a web server and a database for logging vehicle parameters.
- Setting up a web interface from which users can view vehicle parameters being monitored.

1.4 Expected Outcome

The project seeks to develop a printed-circuit board housing the required components to interface with the CAN network of a vehicle via the datalink connector. The IoT-enabled hardware monitors CAN traffic and transfers the data to a database. A web application fetches the data from the database and makes it available to a user in real-time. The data is presented in a graphical format to facilitate easy analyses of vehicle performance.

1.5 Delimitations

- The sole focus of the project is developing a proof of concept system as close to a complete end-user product as possible.
- The focus of product development is centered on functionality and trades of security where necessary.
- The only communication protocol handled is the CAN protocol.
- OBD communication is limited monitoring specific vehicle parameters.

Chapter 2: Literature Review

2.1 Introduction

The CAN bus network is a centralized multi-master bus developed by a German company Robert BOSCH GmbH in 1986 [5]. The popularity of the CAN network surged when Federal and state agency namely; National Highway Traffic Safety Administration (NHTSA) in collaboration with the California Air Resources Board (CARB) required standard procedure to monitor vehicle’s emission control systems. As a result, manufacturers implemented the On-Board Diagnostics protocol (OBD) [6]. OBD refers to a vehicle’s ability to diagnose itself in the event of vehicular malfunction. The second generation of OBD (OBD-II) is currently in use in modern vehicles. Since the OBD is a reference point for accessing data from various vehicle sensors and control units, it was necessary to device a means of linking all these independent components. The need for a centralized network that interconnected the various sensors and their corresponding control units created the Controller Area Network (CAN).

The CAN network facilitates the exchange of smaller packets relative to traditional networks lines such as Ethernet or USB [7]. In the CAN network, messages are broadcasted to every control unit on the network. The relevance of a message is subject to the recipient ECU. An example of a data packet that could be broadcasted over the CAN network include commands such as “initiate front vehicle wipers” or "roll down windows." Such messages would be irrelevant to an ECU monitoring engine temperature [8]. There are other networks that are responsible for channeling other types of data between network nodes in a vehicle. Such networks are the Local Interconnect Network (LIN), designed to complement the CAN network. Media Oriented System Transport (MOST), designed for multimedia devices and FlexRay, designed for time-sensitive communication [9]. Of interest to this project is the CAN bus network.

Since 2008, all production vehicles are required to use CAN network as the OBD-II communication protocol [10]. These application-specific computers communicate using specific codes. There are a set of standardized codes used for OBD. Any given code consists of two segments. The first segment of the code is the mode of operation followed by a parameter identifier (PID). For each mode, there are a given set of PIDs that request specific information. For example, using mode '01' and a PID of '0D', the CAN network knows that the current vehicle speed is being requested [10]. As a result, the corresponding ECU replies with vehicle speed in hexadecimal format which would have to be converted to its decimal equivalent.

There are 10 diagnostic modes described by the OBD-II standard SAE J1979 [11]. Of interest to this project is mode 1. A PID request with mode of 1, requests for current vehicle data. In order to extract real-time vehicle data, all vehicle requests made have modes set to 1. The table below shows the standard OBD-II PIDs. In the table below, the expected length of any given response is provided, in addition to formulas for converting the response into intelligible data.

PID (Hex)	Data Bytes	Description	Min Value	Max Value	Units	Formula
0x01	2	Engine RPM	0	16383.75	rpm	(256A+B)/4
0x0C	1	Speed	0	255	Km/h	A
0x0D	1	Temperature	-40	215	^{o}C	A - 40

Table 2.1 Standard OBD-II PIDs with conversion formula

Source: Adapted from [12]

There are two types of CAN packets namely, standard and extended CAN packets. Extended packets are similar to standard packets except that extended packets have a lengthier field for storing arbitration ID. Messages broadcasted over the CAN bus are also referred to as frames. There are four types of frames that can be monitored on the CAN bus; Data, Error, Overload and Remote frames [13]. Each CAN bus frame consists of seven (7) principal constituents:

1. Start of Frame bit (SOF): The SOF bit prompts the bus of an incoming data frame.
2. Arbitration ID: The arbitration ID is a section of the packet that identifies the device attempting to communicate over the bus. The arbitration ID is used to decide which node gets priority to broadcast over the bus.
3. Identifier extension (IDE): This bit distinguishes between a standard and extended arbitration ID. It is 0 for a standard data frame and 1 for an extended data frame.
4. Data Length Code (DLC): This determines the size of the data to be transferred.
5. Data: This is the actual data to be transmitted with a maximum size of 8 bytes.
6. CRC Field: This fiend contains the error-checking code to determine whether the incoming message received contains no errors. The CAN protocol uses 15-bit CRC polynomial for error-checking and detection.
7. End of Frame (EOF): This indicates the end of a CAN packet.

SOF	11-bit ID	RTR	IDE	R0	DLC	0-8 Bytes	CRC	ACK	EOF	IFS

Table 1.2: Standard 11-bit data CAN frame.

2.2 Related Works

Salunkhe, Pravin P Kamble, Rohit Jadhavin in the paper *Design and Implementation of CAN bus Protocol for Monitoring Vehicle Parameters* implement a CAN bus network consisting of three nodes [14]. The implementation measures vehicle speed and monitors the brake status of a vehicle. The values of these parameters are accessed remotely via a web interface. The design consists of two Raspberry Pi 2 and a PiCAN module. In the set-up, each Raspberry Pi 2 acts as single board computer. The Raspberry Pi 2 is a microcontroller that is the recipient of CAN packets for processing. The PiCAN module acts as the interface between the physical CAN bus and the Raspberry Pi 2. The PiCAN module consists of the MCP2551 and the MCP2515. The MCP2551 is a transceiver that intercepts and interprets voltage signals on the CAN bus. The MCP2515 is a CAN controller with three subsystems that help it perform three key roles. The first is the CAN module which handles transmission and receptions of packets. It also contains the control logic module for the control, configuration, and operations of the CAN controller. Lastly, the SPI protocol block of the CAN controller handles serial communication between the CAN controller and the microcontroller via the transmitter (Tx) and receiver (Rx) GPIO pins.

In the setup, the authors construct a CAN network using the Raspberry Pi as the nodes. A Hall sensor is used to measure vehicle speed. The Hall Effect sensor is placed near a test wheel and measures the magnitude of the magnetic force as the wheel rotates. The speed of the wheel is computed and is used to approximate the velocity of the vehicle. A simple switch is used to imitate brake status. Toggling the switch indicates whether the brake is depressed or at rest. A Raspberry Pi is set up as a server. It receives data from the other two Raspberry Pi nodes broadcasting vehicle speed and brake status. The data is uploaded to a webpage hosted on the Raspberry Pi. From the webpage, the brake status and velocity can be monitored.

HaiPing Sun, Hong Zeng, JiaLi Guo in their paper, *Bus Data Acquisition and Remote Monitoring System Based on CAN Bus and GPRS* investigate the feasibility of CAN bus data acquisition and remote monitoring over a GPRS connection [15]. The design focuses on monitoring real-time CAN bus data and remotely communicating the information to a database via a GSM module. The authors propose the design of an embedded system that interfaces with a vehicle over the CAN network. The proposed design allows the embedded hardware to communicate with an external database. The design features a remote monitoring station that allows commands to be sent to the embedded hardware. The hardware implementation uses a TJA1040 CAN transceiver, STM32 microcontroller which comes standard with an integrated CAN controller and a GTM900C GSM/GPRS module. The TJA1040 CAN transceiver plays a similar role to that of the MCP 2551. The transceiver is responsible to interpreting the signals of the CAN bus for processing by the STM32 microcontroller. GTM900C acts as a gateway allowing data to be transferred to the database and commands to be received from the monitoring station. The paper documents a design for interfacing with the CAN bus node and the GSM/GPRS module. The program executing on the CAN node handles initialization of the CAN bus node, transmission, and reception of data from the CAN bus. The software to also interface with the GSM/GPRS handles GPRS packet transmission between the server and the module.

Authors Renjun Li, Chu Liu and Feng Luo in their paper, *A Design for Automotive CAN Bus Monitoring System* propose and implement a CAN bus monitoring system for automotive applications[16]. The hardware supports two independent CAN communication channels and is compatible with both high-speed and low-speed CAN communication. The hardware also supports logging of CAN messages for post-analysis. Other features include the ability to send and receive CAN frames, large on-board RAM for temporary storage of CAN data and a listen-only mode, for monitoring the bus. The main components that support the system are LM2596 a step-down converter, which acts as the systems power supply unit. A 16-bit MC9S12DP256 Freescale semiconductor microcontroller, with integrated CAN controller. This microcontroller interfaces with PDIUSBD12 Phillips Semiconductor which serves as USB interface module and a TJA1050 CAN transceiver. The system interfaces with a vehicle via an ODB 2 to DB9 cable. In the implementation, the authors simulate a CAN node using Kvaser system. CAN communication is established between the prototype hardware and the Kvaser system and packet transfer is monitored via a developed PC application. The authors propose a feasible design as illustrated through implementation.

E. Ceuca, A. Tulbure, A. Taut, O. Pop, I. Farkas in the paper *Embedded System for Remote Monitoring of OBD Bus* present a design for monitoring vehicle speed via the vehicle's CAN bus network [17]. The transmission system is implemented with a GSM/GPRS hardware that reports the location of the vehicle in addition to equipping the monitoring system with the ability to transfer data to a server remotely. The CAN network monitoring hardware is based on the AVG 4000. The design connects to a target vehicle using a DB9 to OBD-II connector cable. The design allows real-time capture of vehicle data via and vehicle coordinates. The design allows the investigation of a correlation between vehicle location with vehicle performance. The use of the GSM/GPRS module is effective as the wide area network coverage ensures internet connectivity via GPRS connection. This ensures that data can transferred to the central database remotely. Additionally, the use of a battery ensures that satellite connection with the GSM/GPRS module persists even if the engine of the target vehicle is turned off.

2.3 Gaps and Improvements.

Salunkhe, Pravin P Kamble, Rohit Jadhavin successfully simulate the CAN network. The implementation allows data to be transferred between nodes on the network. Whereas the simulation works well, the authors fail to test, the system on an actual vehicle. This limits the application of the work done. In order to interact and receive vehicle data such as engine RPM and brake status, commands have to be issued to the vehicle nodes of interest using parameter identifiers (PIDs). PIDs allow external nodes connected to the CAN bus to request and receive data over the CAN bus. Also, authors HaiPing Sun, Hong Zeng, JiaLi Guo in their work fail to show any evidence of actual implementation. The depicted hardware system is not developed and thus not tested.

For authors, E. Ceuca, A. Tulbure, A. Taut, O. Pop, I. Farkas, their work focuses on investigating a correlation between vehicle location and performance. However, the implementation focuses more on vehicle tracking as supposed to monitoring OBD parameters. Little detail is provided to understand the underlying mechanism the OBD hardware monitor.

Finally, authors Renjun Li, Chu Liu and Feng Luo present a hardware system for monitoring the CAN network. However, in testing, the authors simulate CAN nodes using the Kvaser product as supposed to connecting the system to a real-life CAN network. For the purposes of this capstone implementation, an embedded system will be developed and tested. The hardware design will be well documented, and the prototype will be connected to a live CAN network. The hardware system will request CAN packets using PIDs to obtain real-time information of the target vehicle. Data received will be communicated to a webserver to be displayed on the web application and subsequently stored in a database.

Chapter 3: Design

3.1 Overview

After reviewing existing designs in the previous chapter, this chapter elaborates on the components and the design of the overall system. This chapter focuses on component selection, and the justification of choice. The chapter also discusses the requirements specification of the system elaborating on some use cases and illustrating how the various components come together.

3.2 Thesis Design Objective

- To design an embedded system that is capable of monitoring CAN bus traffic.
- To keep the cost of fabrication of the design at a minimum.
- To develop a hardware system capable of interpreting CAN packets.
- To develop abstracted view of the entire system.

3.3 System Overview

For this section, both the functional and non-functional requirements of the hardware system are described. Also, a use case for the overall system is illustrated.

3.3.1 Functional Requirements

In this suction, the capabilities and roles of the system are defined. The functional requirements of the system are stated below.

- The embedded system should monitor the CAN bus network for data frames.
- The hardware system should request for vehicle speed, engine rpm and coolant temperature.
- The embedded system should display vehicle speed, engine rpm and coolant temperature on the LCD.

- The system should upload the data frames of interest to the cloud via GPRS internet connection.
- The system should populate the database with vehicle data.
- The web application should display graphical vehicle performance over time.

3.3.2 Non-Functional Requirements

This section provides a rubric that guides the functional requirements stated above.

- Performance: The system should provide a user with vehicle performance data once connected to power source.
- Speed: The GSM module should provide internet connectivity via GPRS connection with tolerable latency.
- Usage: The system should function properly during periods of prolonged usage.
- Accuracy: The system will provide accurate vehicle performance data.

3.3.3 Use Case.

The principal actor is the owner and user of the hardware system.

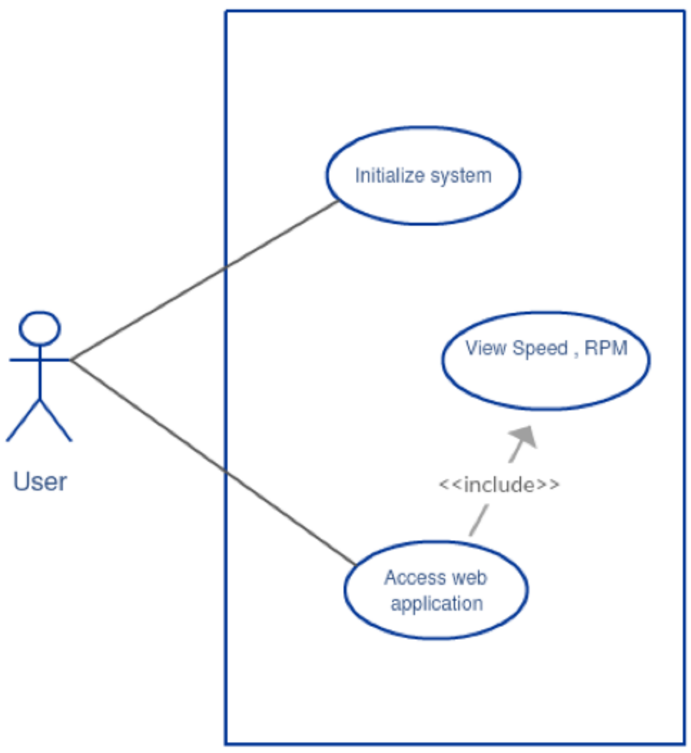


Figure 3.1 Use case of the overall system

3.3.4 Flowchart

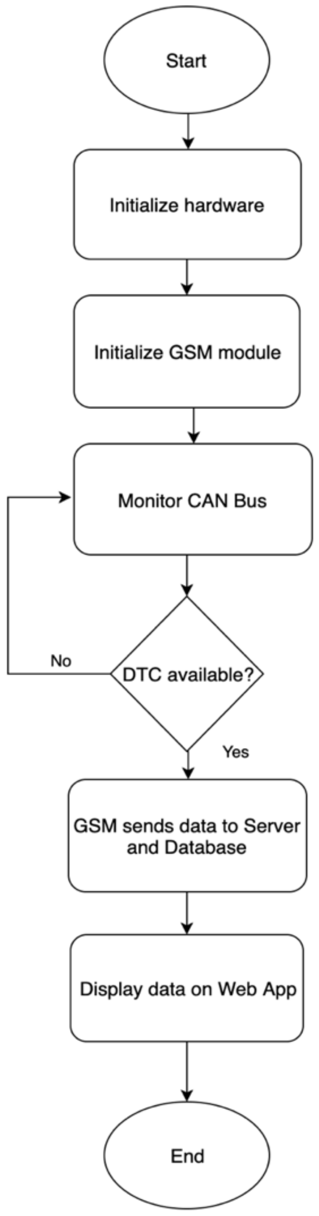


Figure 3. 2 Flowchart showing operation of the overall system

3.4 Component Selection

This section discusses the hardware selection criteria that was used in deciding on what components to include in the final system design. The literature review informed the available components to be considered for use. A Pugh matrix was used to select the final component that ended up in the hardware design.

3.4.1 Pugh Matrix

The Pugh matrix is a decision-making model that allows a designer to select from a group of alternatives based on prespecified criteria [18]. The tables below are the Pugh matrices constructed to select the key components of the prototype hardware.

KEY	
0	Equivalent to baseline
+	Better than
-	Worse than

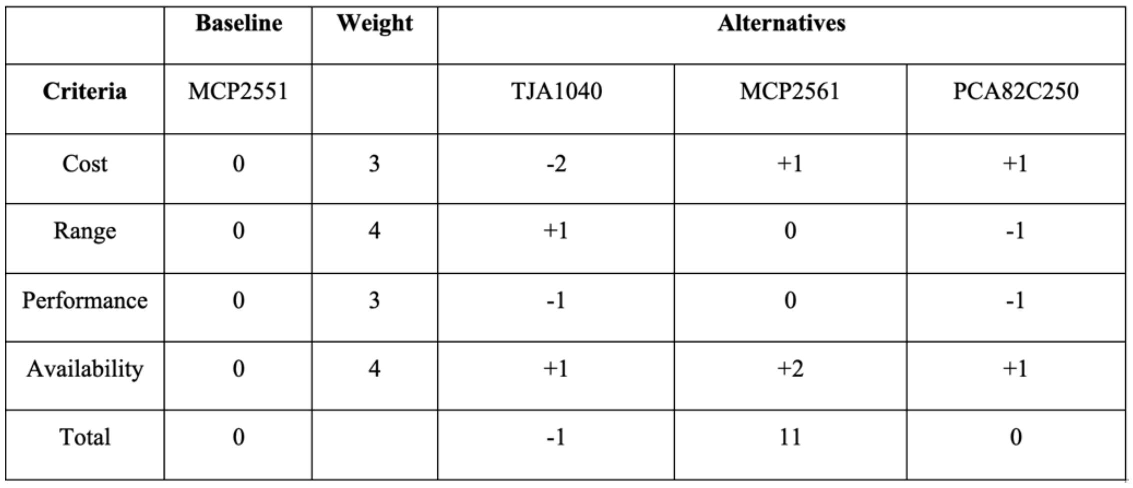
Table 3.1: A list of keys for interpreting the Pugh

	Baseline	Weight	Alternatives			
Criteria	PIC18F2480		ATMEGA328P	AT90CAN128	STM32F042	LPC1517
Cost	0	3	+1	-1	+1	+1
Power consumption	0	4	+1	-1	-1	0
Peripheral features	0	3	-1	+3	+1	+1
Performance	0	3	-1	+2	+2	+1
Availability	0	4	+3	+1	+1	+1
Total	0		13	12	12	10

Table 3.2: Pugh matrix showing the selection of microcontroller

	Baseline	Weight	Alternatives		
Criteria	Arduino GSM shield		SIM800L	ESP8266	HC-05
Cost	0	3	+1	-1	+1
Power consumption	0	3	-3	-1	0
Latency	0	4	0	+2	+1
Coverage	0	4	0	-1	-3
Availability	0	4	+3	+1	+2
Total	0		6	2	3

Table 3.3: Pugh matrix showing the selection of connectivity module



	Baseline	Weight	Alternatives		
Criteria	MCP2551		TJA1040	MCP2561	PCA82C250
Cost	0	3	-2	+1	+1
Range	0	4	+1	0	-1
Performance	0	3	-1	0	-1
Availability	0	4	+1	+2	+1
Total	0		-1	11	0

Table 3.4: Pugh matrix showing the selection of CAN controller

3.4.2 Hardware Components

In this section the selected hardware components and their functions are discussed.

Component	Role
Atmega 328-P	The Atmega328P belongs to the 8-bit CMOS logic family of AVR microcontrollers. It is based on the enhanced RISC Reduced Instruction Set Computer architecture (RISC). This microcontroller acts as the brain of the hardware system [19].
SIM800L GSM/GPRS	SIM800L is a GSM/GPRS module with capabilities that allow for GPRS internet connectivity. It comes in a package that makes it ideal for use. The module supports quadband frequency, making it ideal for long-range connectivity. The SIM800L module equips the hardware system with IoT capability, allowing data to be uploaded to the web [20].
MCP2515	The MCP2515 is a stand-alone CAN controller. The component acts as the interface between the CAN transceiver and the microcontroller. This component handles functions such as filtering unwanted messages reducing the overhead of the microcontroller [21]

MCP2561	The MCP2561 is a CAN transceiver that converts data frames on the CAN bus into differential voltage signals and differential voltage signals to their corresponding binary representations which make up a data frame. It interfaces with the CAN controller, and the physical two-wire CAN High and Low [22].
LM7805	The LM7805 is an integrated-circuit voltage regulator. This device is a DC-DC voltage regulator that can take a minimum supply input voltage of 7V and maximum input voltage of 25V giving an output of 5V at a maximum current of 1.5A [23].
AZ1084	The AZ1084 is a low voltage drop-out power regulator. This device is another DC-DC voltage regulator with a maximum dropout voltage of 1.5V at 5A load current. The GSM/GPRS module draws as much as 2A during pushing operations. The AZ1084 meets the requirements of the system [24]

Table 3. 5: A description of components and functions

3.4.3 Software

Software Package	Role
Arduino IDE	The Arduino IDE is used in programming the microcontroller of the hardware system. Using the IDE enables quick and easy implementation of sketches. These sketches are then uploaded unto the microcontroller
MQTT	MQTT is a machine-to-machine message-based protocol that allows devices to share information [25].
MySQL	Database for storing data.

Table 3. 6: A description of software applications.

3.5 Architectural Diagram

This section discusses at a high-level the subcomponents of the hardware prototype and how these components interact with one another.

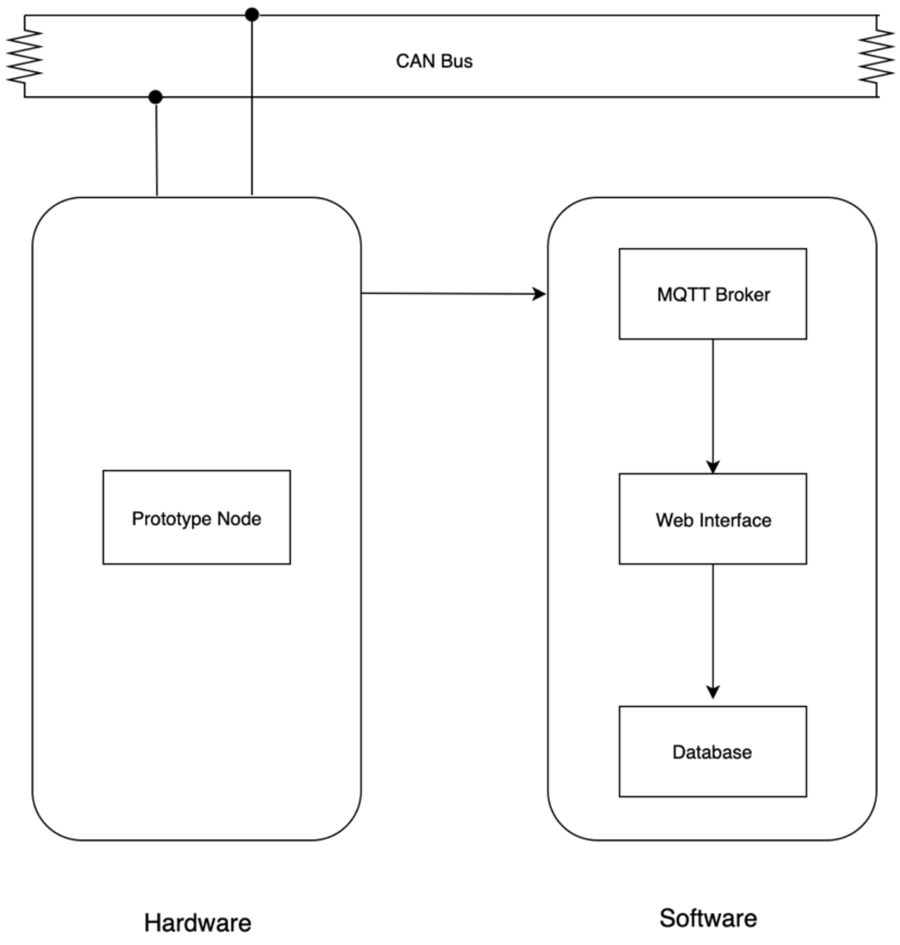


Figure 3. 3: High-level overview of the overall system

3.5.1 Contextual Diagram

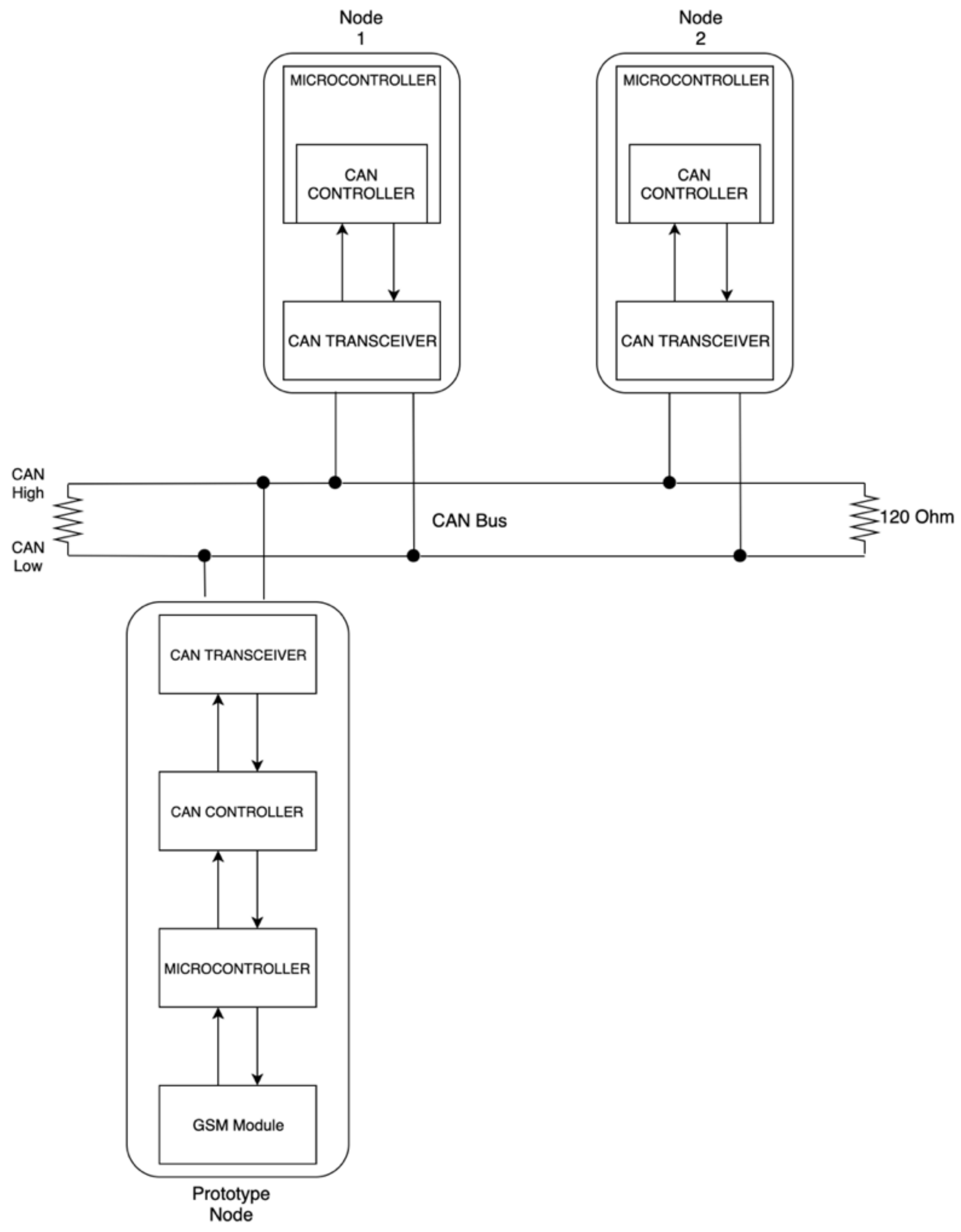


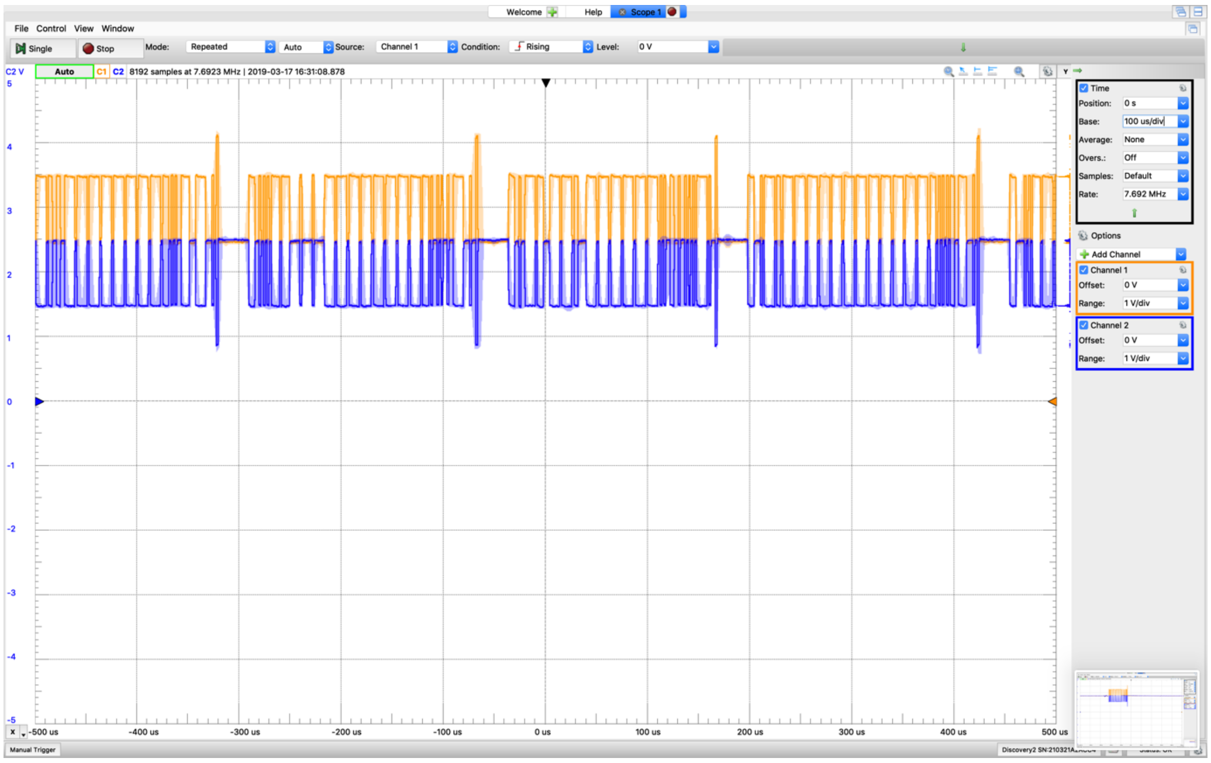
Figure 3.4: CAN bus architecture with prototype node

Chapter 4: Implementation.

This chapter documents the approaches in achieving the design objectives of the project. The chapter elaborates on the development process of the overall system.

4.1 Scoping CAN Bus

The CAN bus operates on two twisted pair wires CAN High (CANH) and CAN Low (CANL). The resting voltage levels of the respective signal lines are 2.5V for both CANH and CANL [26]. Measuring the CAN lines of the target vehicle (Honda Accord 2016), the voltage for the CANH signal line was 2.7V and 2.27V for CANL. These voltages were measured relative to signal ground line as supposed to chassis ground. This is because of the isolation of the signal ground line from noise. When a signal is transmitted over the CAN bus, the voltage on the CANH increases by 1V to 3.5V and the voltage on CANL also decreases by 1V to 1.5V. This method of communication is used for its robustness and greater immunity to electromagnetic interference (EMI) and crosstalk. The figure below shows a signal captured



Mode: Repeated | Auto | Source: Channel 1 | Condition: Rising | Level: 0 V

8192 samples at 7.6923 MHz | 2019-03-17 16:31:08.878

Time
Position: 0 s
Base: 100 us/div
Average: None
Overs.: Off
Samples: Default
Rate: 7.692 MHz

Channel 1
Offset: 0 V
Range: 1 V/div

Channel 2
Offset: 0 V
Range: 1 V/div

Figure 4. 1: Real-time capture of CAN signal from target vehicle using Analog Discovery 2 while scoping CANH (Scope Channel 1, Orange) and CANL (Scope Channel 2, Blue) signals of the target vehicle. This image was recorded after starting the engine of the target vehicle to initiate CAN bus communication.

4.2 Hardware Design

The hardware design required extensive reading in areas of embedded systems and design. After selecting the various components that were required to realize the design objectives., the datasheets of each component were obtained. The datasheets of each component had to be studied to understand how each component would be interfaced with one another. The datasheets also provided application circuits for each key component. The datasheets showed the value of passive components such as capacitors, resistors and crystal oscillators and how these components should be combined to ensure proper functioning of the integrated circuit (IC). A key learning point was interfacing the GSM module to the microcontroller. Since both devices operated at different voltage levels, there was the need to step-down the communication signals from the microcontroller to the GSM module. This was achieved by logic-level shifting operation.

The schematic design of the circuit was realized using Eagle CAD software. A Proteus simulation of the circuit was done concurrently. The Proteus simulation only featured the power supply sections of the schematic. This was due to the limited library components offered with the Proteus suite. The next logical step was to test the schematic connections on a breadboard. However, because only surface-mount components were provided, only a printed circuit board (PCB) realization was feasible. Due to the complex nature of the schematic, a single layer PCB was not feasible. As such, the board had to be shipped to China for printing with a turn-around time of two-weeks. The first prototype PCB measures 81.26 mm by 63.17 mm.

4.2.1 Circuit Schematic

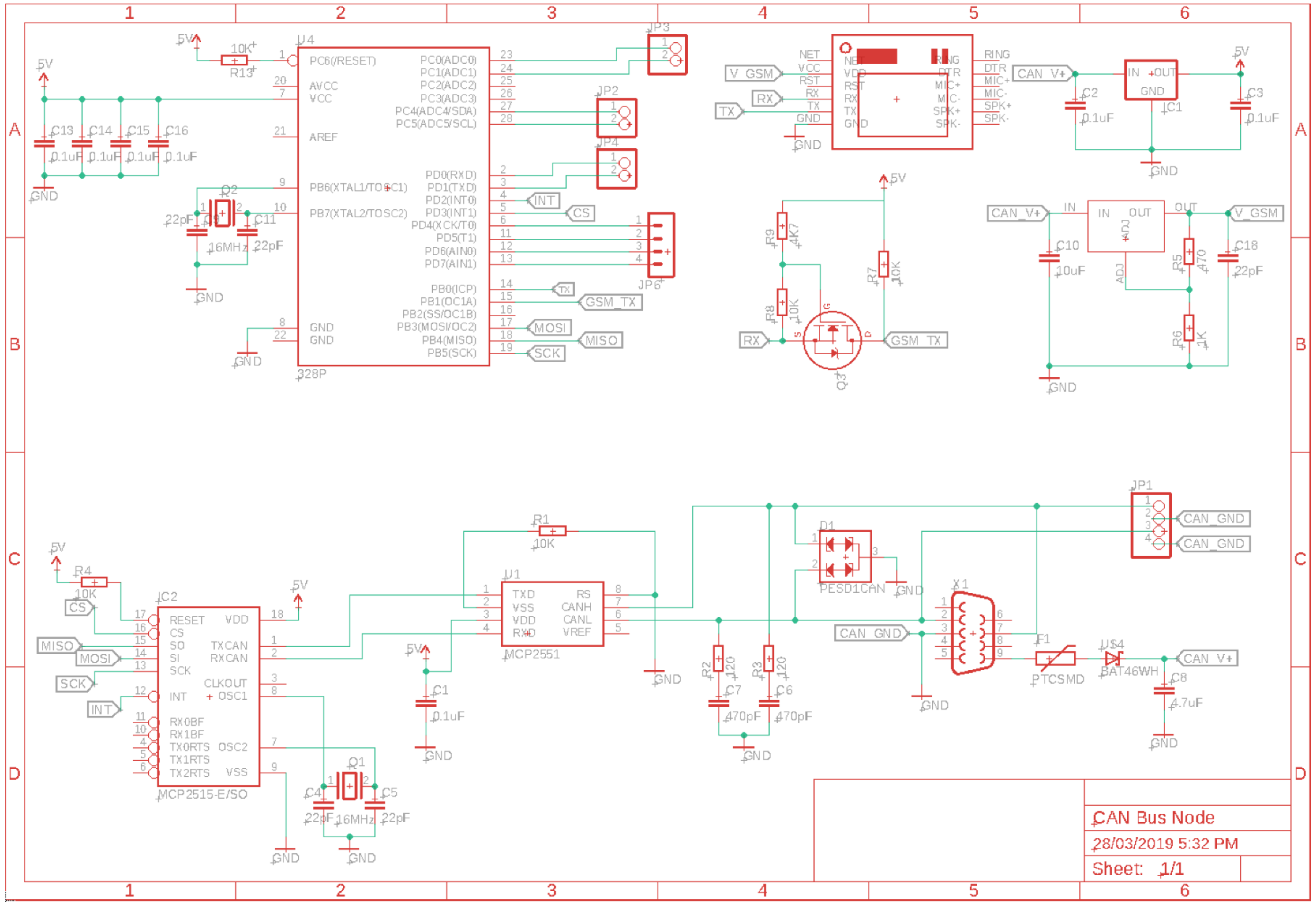


Figure 4.2: Schematic diagram of prototype node

4.2.2 PCB Design

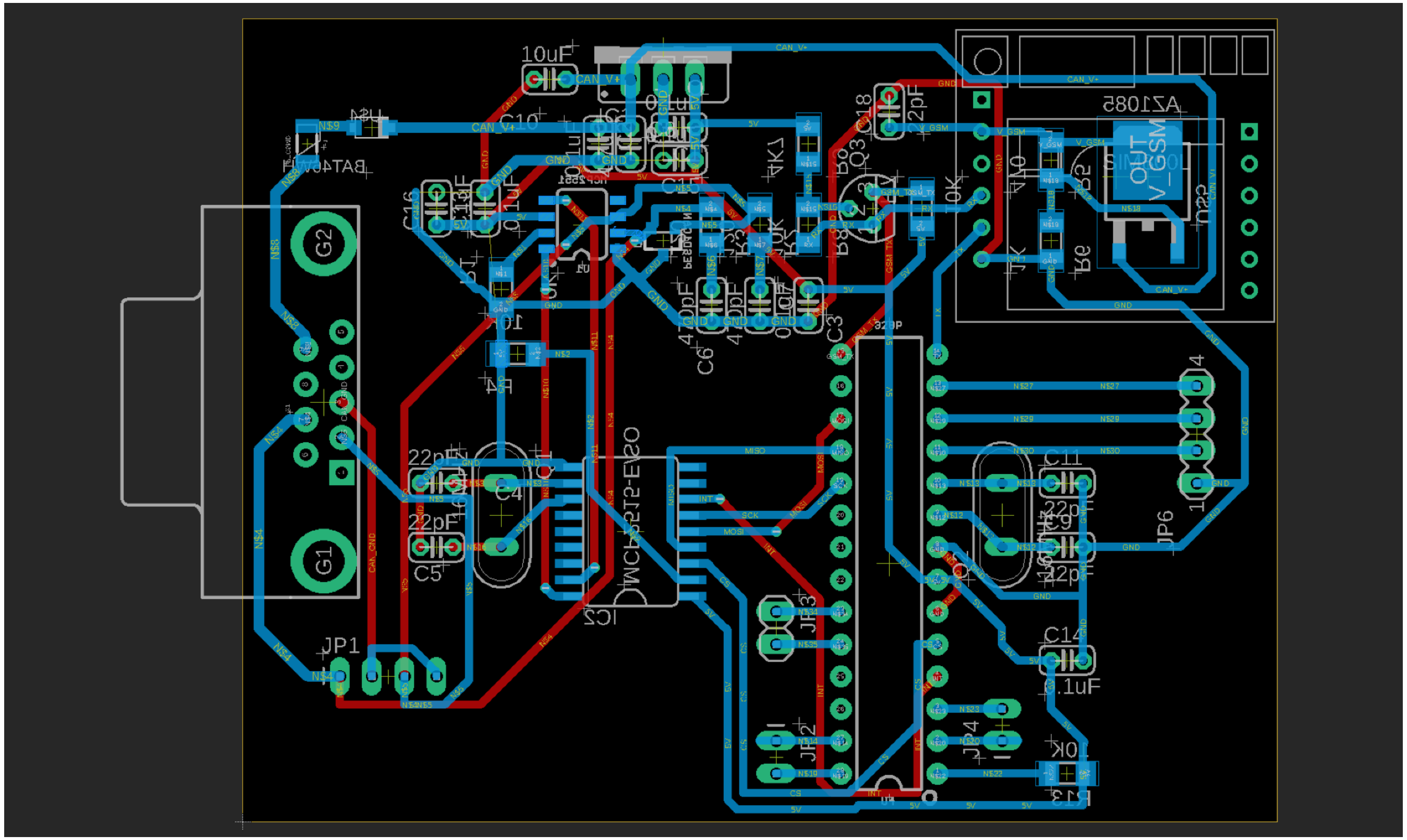


Figure 4. 3: PCB design showing board layers top(red) and bottom(blue)

After the schematic design, the next stage of the hardware design was arranging the various components on the PCB board. As can be seen above, the components were arranged to conform to the schematic design. This was done to ensure the shortest path between interconnected components. A process known as routing was used to define current paths between components. Since the PCB had two layers, vias were used to interconnect components on different layers of the board.

4.2.3 Soldering

The expected PCB after the PCB manufacturing process has been conducted is shown below. The diagram shows traces, vias and footprint of the various components that would be soldered onto the board.

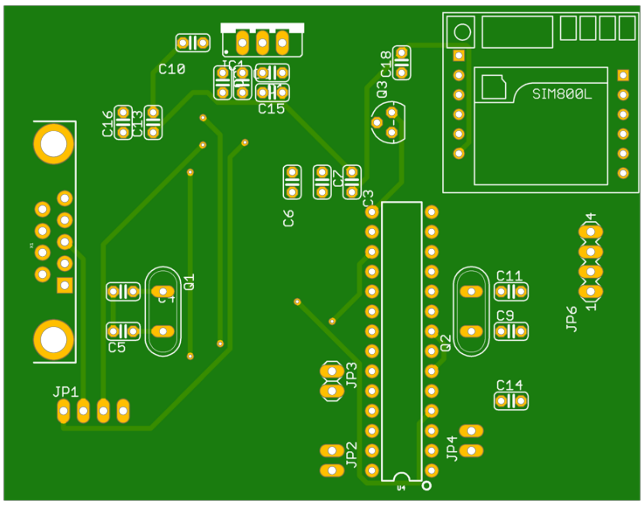


Figure 4. 5: Expected PCB layout (front)

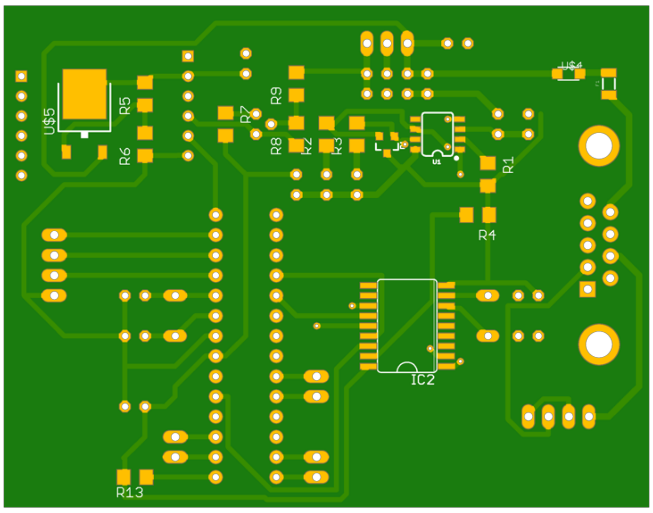


Figure 4. 4: Expected PCB layout (bottom)

Soldering required the use of lead paste and solid lead for holding the components in place on the PCB board. Using soldering iron, solid lead was used to solder Dual In-line Package (DIP) components. As for the lead paste, hot-air from a soldering station was used to adhere surface-mount components on to the PCB.

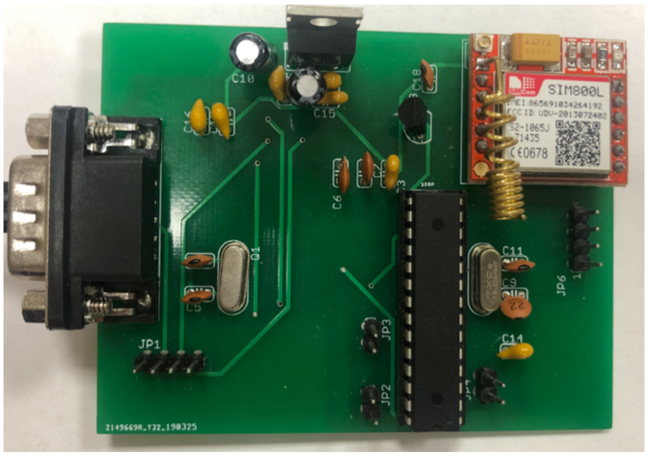


Figure 4. 7: PCB board after soldering (front)

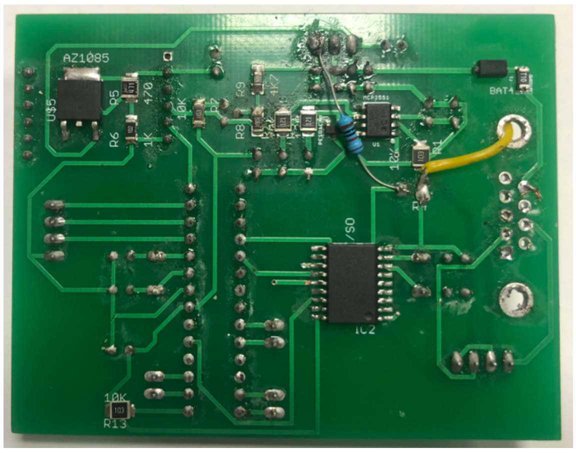


Figure 4. 6: PCB board after soldering (back)

After all components were mounted, the PCB board was tested to ensure proper functioning of each component. Continuity between components were tested between using a multimeter. It was observed that, there were two separate grounds. This challenge was rectified by soldering a wire to bridge both grounds. After testing continuity, the power section of the PCB was tested. As expected the LM7805 with the aid of decoupling capacitors stepped down the supply voltage to the expected 5V for distribution to other subcomponents. The AZ1084 also produced an output voltage of 4V for supply to the GSM module. However, in testing whether the GSM module was functioning properly, it was observed that, the GSM module was unable to maintain network reception regardless of the sim card inserted. Upon investigation, it was observed that the traces connecting the GSM module to its power supply were not thick enough to deliver the required current to drive the module.

The setup also included an LCD that acted as a display for the system. On the LCD, the vehicle parameters being requested would be printed. The I2C LCD requires only two pins for operation. The LCD was interfaced with the microcontroller using Analog pins 4 and 5, the

SDA and SCL pins of the microcontroller respectively. Analog pins 4 and 5 support the I2C communication protocol. The microcontroller communicated to the LCD via the I2C protocol.

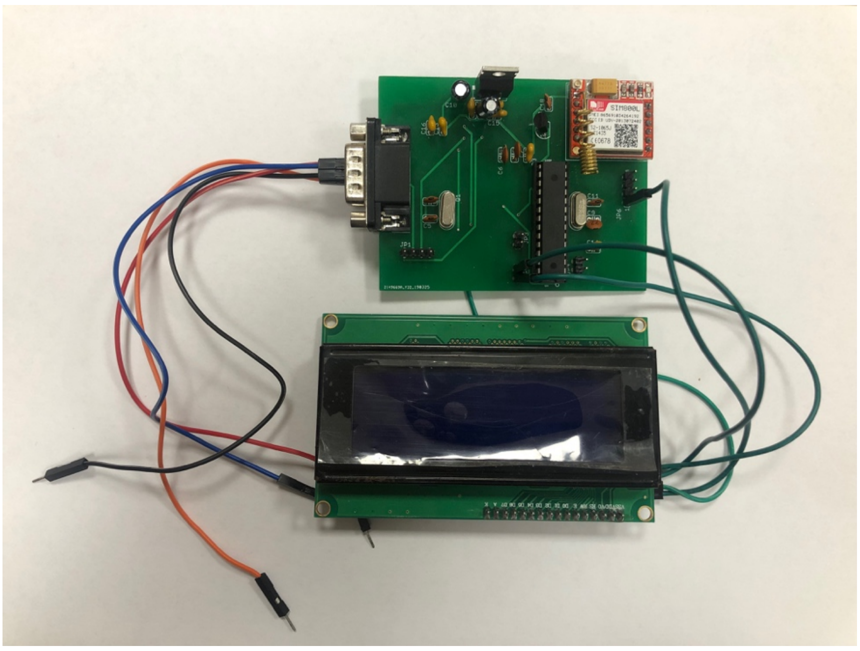


Figure 4.8: PCB interfaced with I2C LCD

4.4 Revised Design

After reviewing the first prototype, revisions were made to the design of the hardware. To keep in-line with the design objectives stated earlier, modifications were made to the existing design where necessary. The new design of the hardware system took into consideration the availability components and the simplification of the overall design. The new design features subcomponents in smaller package sizes to achieve a smaller board design. In the new design, the microcontroller comes in a smaller package (QFP). The choice of crystal oscillators used has integrated capacitors.

Also, the GSM module is placed as close to the power source as possible. Thick traces are used to ensure ample power is supplied to the GSM module. In the new design, provision is made for in-circuit programming to enable easy configuration and programming of the microcontroller. Additionally, a reset push-button is included to reset the microcontroller in case of any program goes out of control. Terminal blocks are used to replace the DB9-connector since a cable to interface the two connectors are unavailable. The final design measures 69.83 mm by 35.23 mm.

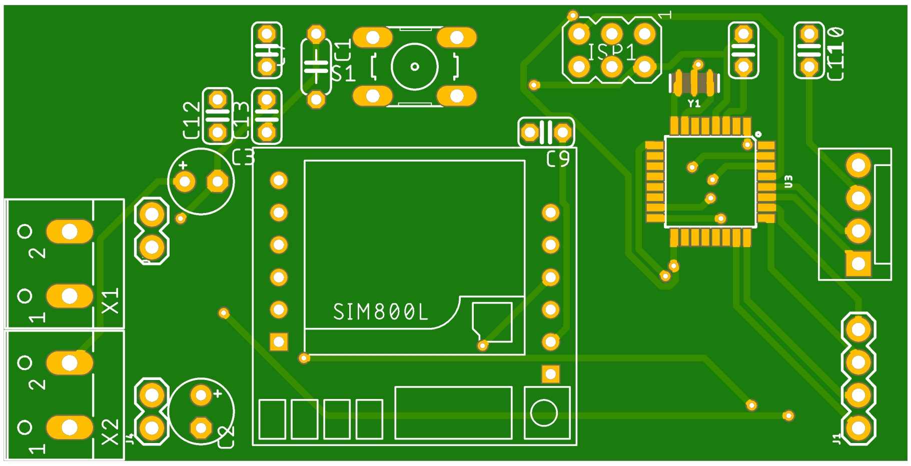


Figure 4.9: Expected PCB 2 design (front)

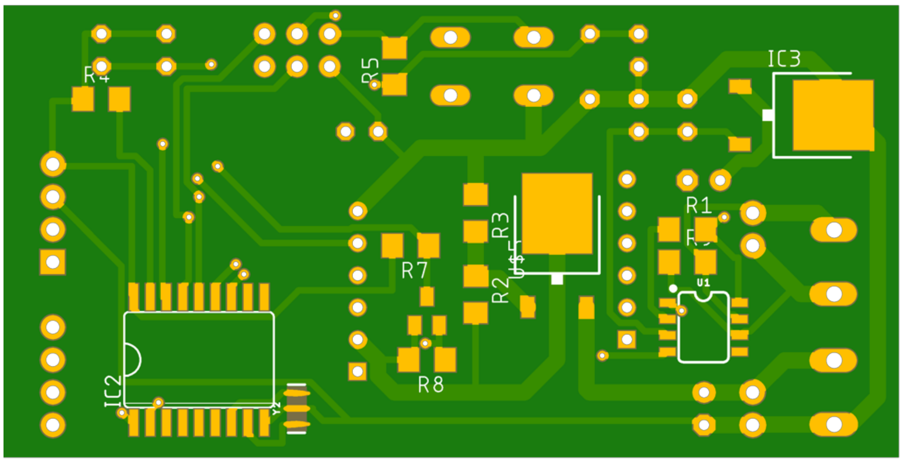


Figure 4.10: Expected PCB 2 design (bottom)

4.4.1 Redesigned Circuit Schematic

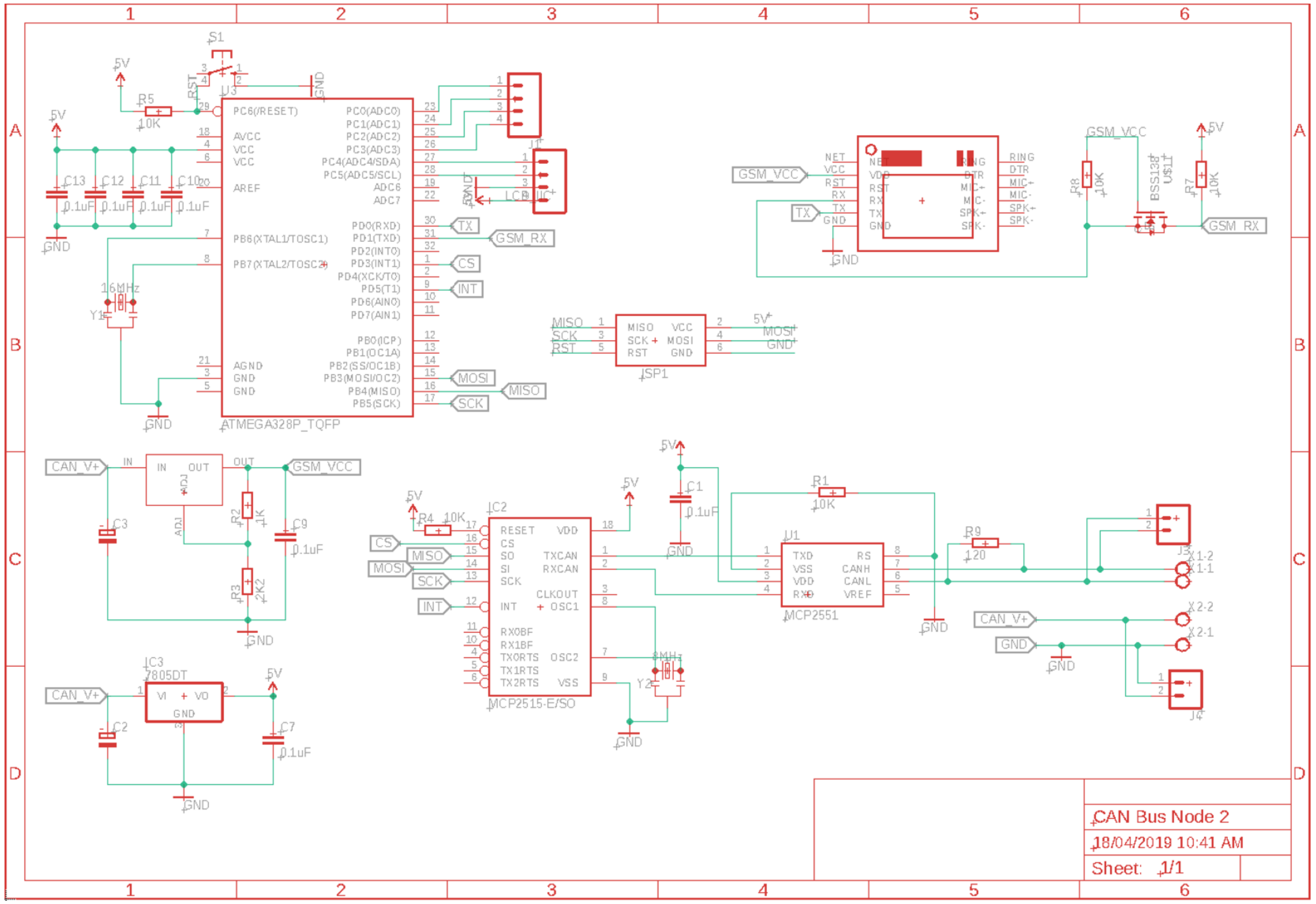


Figure 4. 11: Schematic diagram of revised prototype node

4.4.2 Redesigned PCB Design

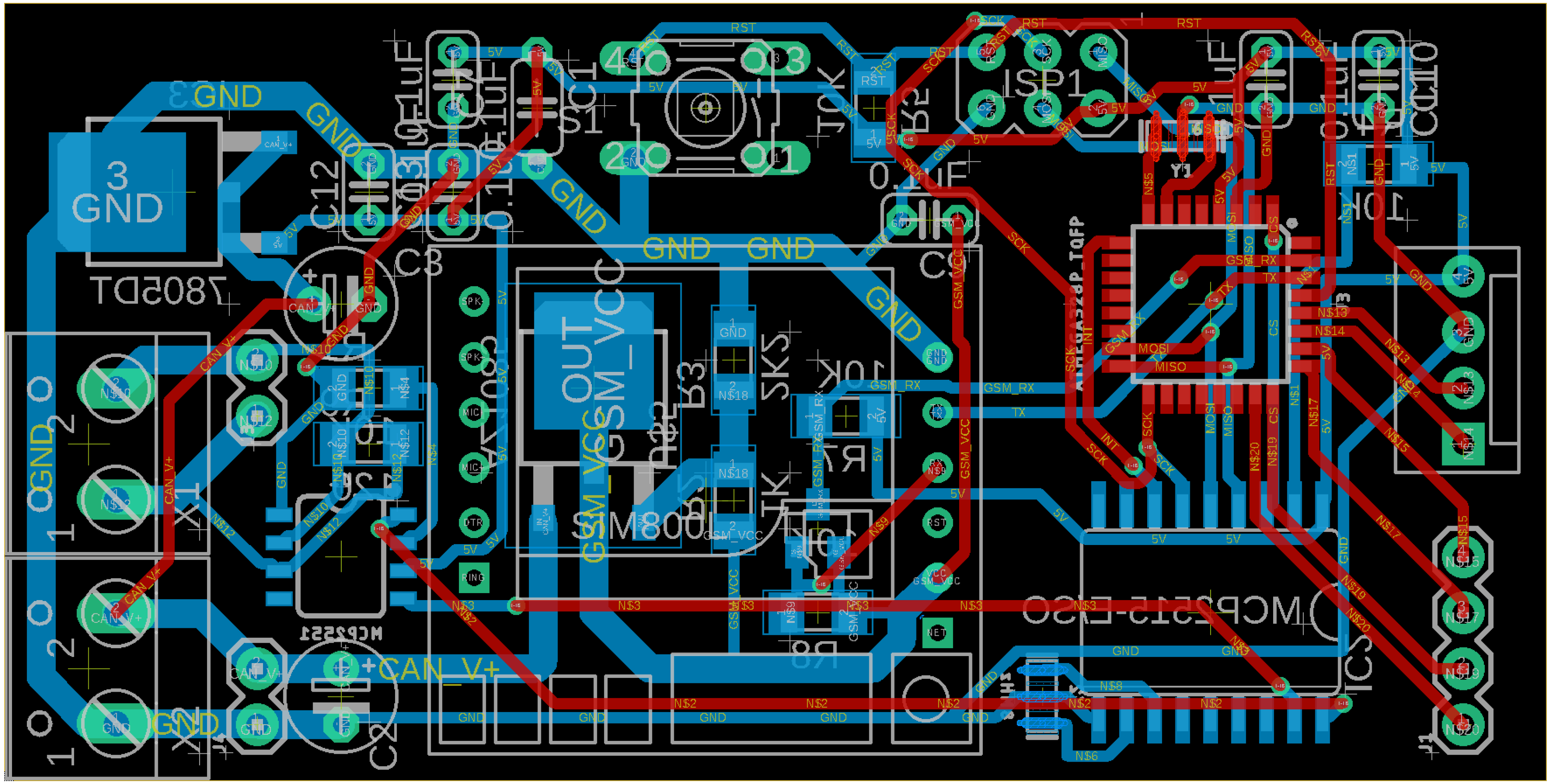


Figure 4.12: PCB layout design of revised prototype node

4.5 OBD-II Connector

All vehicles since 1998 are required to have an OBD-II connector. By law, the connector is required to be around the steering column [27]. The ODB-II specification defines this standardized hardware interface for use in OBD-II applications.

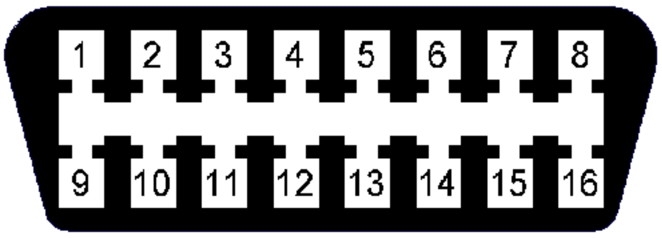


Figure 4.13: OBD-II connector

Source: Adapted from [28]

5: Signal ground	6: CAN High
14: CAN Low	16: Battery voltage (12V)

Table 4.1: Pinout for CAN bus connection via OBD-II connector

A cable connector is used to interface the OBD-II connector to the hardware system. Below is an image of the fabricated cable for interfacing the new hardware system to the OBD-II connector.

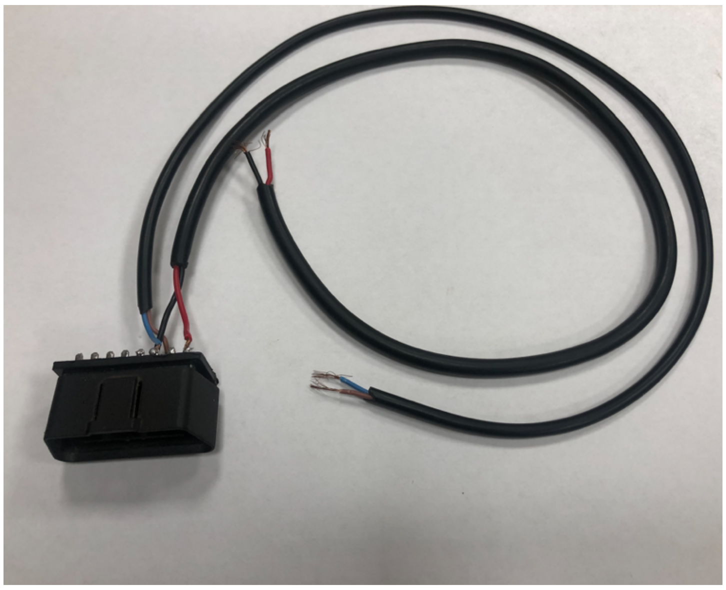


Figure 4.14: Fabricated OBD-II cable connector

4.6 Software

The application software used in programming the microcontroller is the Arduino IDE. The program implemented on the microcontroller executes the following tasks. The program initializes the CAN network with at a board rate of 500Kbps which is the standard data rate used in communicating over the CAN network. Since the CAN traffic is continually broadcasted, filters are set on the two registers of the CAN controller to allow only interested messages through to be processed by the microcontroller. A routine periodically checks for incoming messages. If any message is available, the message is stored in a buffer and its length and ID are printed to the LCD screen. The message is then converted to decimal and computed using one of the formulas to obtain the desired value.

4.6.1 Web Application

The web application is created from an already existing web template. Using credentials provided by Cloud MQTT, the platform is set up to receive dummy data being pushed via the GSM module unto the Cloud MQTT broker. An API call fetches this data and periodically updates the web application on refreshing the page. Since the online broker comes with an integrated database, the cloud MQTT broker handles database population.

Chapter 5: Results

This chapter describes testing and evaluation of the hardware and software system. The target vehicle on which the hardware system was tested is a Honda Accord 2016.

5.1 Detection of Parameters

In testing the prototype hardware, the system was connected to the datalink connector and the target vehicle was started to initiate CAN bus communication. There were several instances where the hardware failed to initialize. However, after successful initialization of the hardware, the data printed on the LCD screen was not as expected.

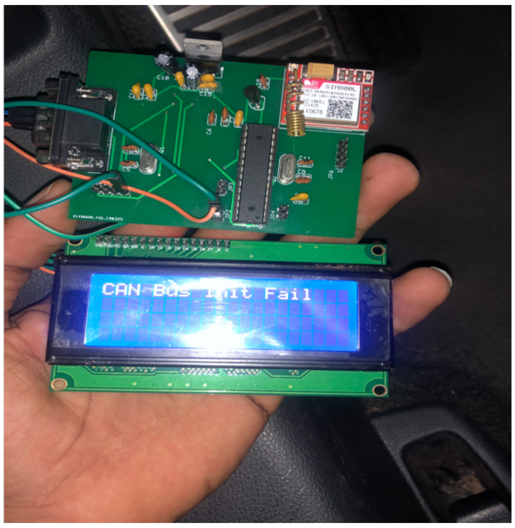


Figure 5. 1: CAN bus with failed initialization

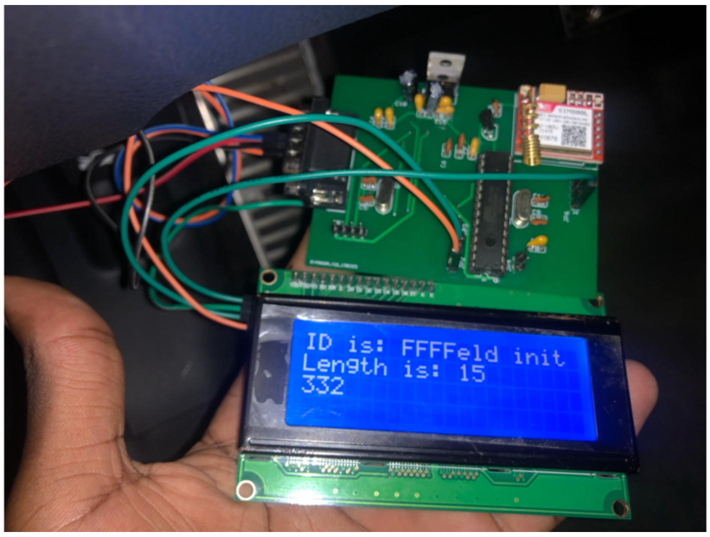
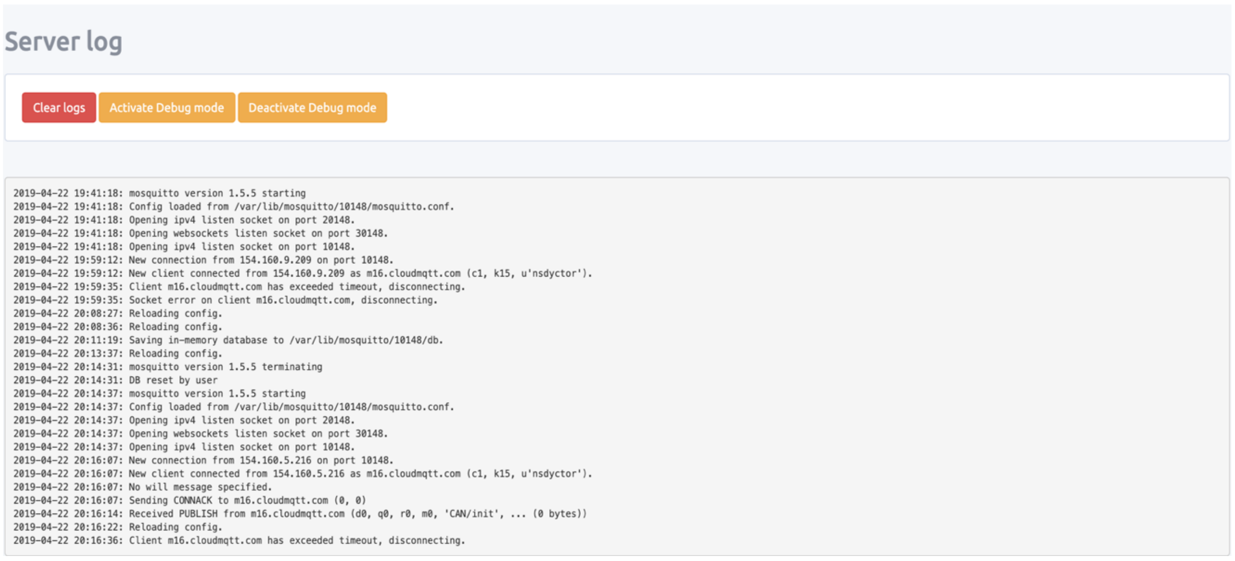


Figure 5. 2: CAN bus with successful initialization

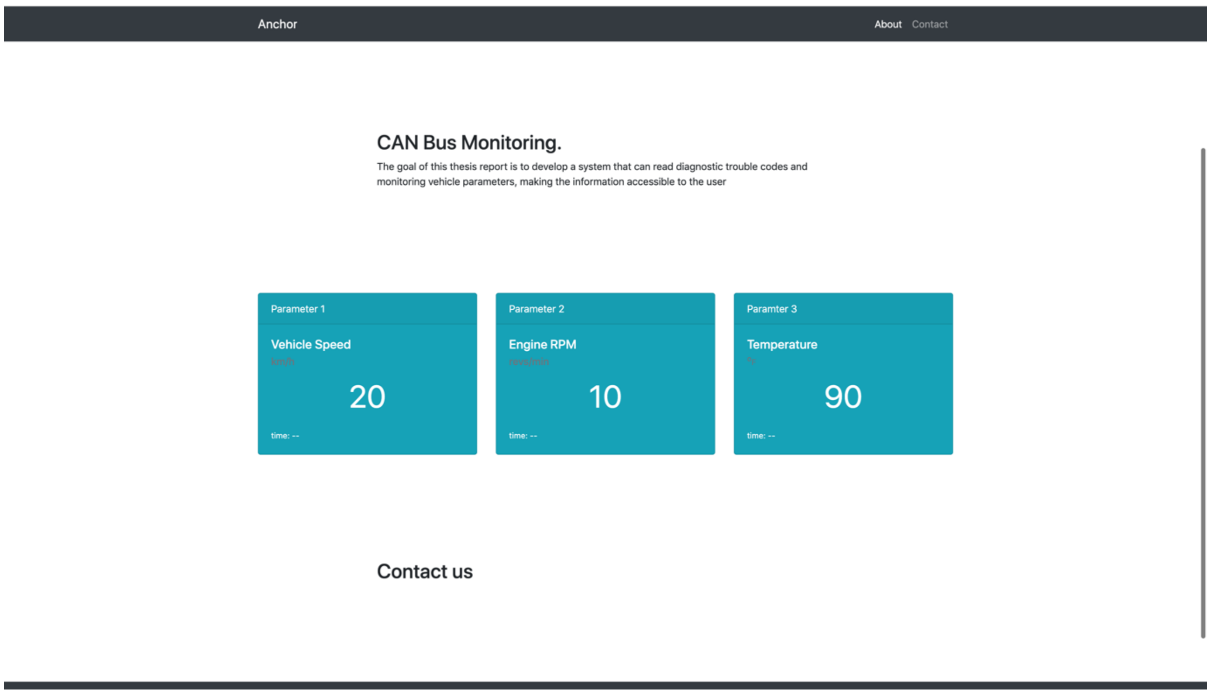
5.2 Data on Cloud

Due to the lack of correct data coupled with the insufficient power supplying the GSM module, the set up was unable to push data. The data received could not be pushed to the MQTT platform. To test that the web platform was receiving data, dummy data was pushed via a different GSM module to the MQTT cloud broker.



```
2019-04-22 19:41:18: mosquitto version 1.5.5 starting
2019-04-22 19:41:18: Config loaded from /var/lib/mosquitto/10148/mosquitto.conf.
2019-04-22 19:41:18: Opening ipv4 listen socket on port 20148.
2019-04-22 19:41:18: Opening websockets listen socket on port 30148.
2019-04-22 19:41:18: Opening ipv4 listen socket on port 10148.
2019-04-22 19:59:12: New connection from 154.160.9.209 on port 10148.
2019-04-22 19:59:12: New client connected from 154.160.9.209 as m16.cloudmqtt.com (c1, k15, u'nsdyctor').
2019-04-22 19:59:35: Client m16.cloudmqtt.com has exceeded timeout, disconnecting.
2019-04-22 19:59:35: Socket error on client m16.cloudmqtt.com, disconnecting.
2019-04-22 20:08:27: Reloading config.
2019-04-22 20:08:36: Reloading config.
2019-04-22 20:11:19: Saving in-memory database to /var/lib/mosquitto/10148/db.
2019-04-22 20:13:37: Reloading config.
2019-04-22 20:14:31: mosquitto version 1.5.5 terminating
2019-04-22 20:14:31: DB reset by user
2019-04-22 20:14:37: mosquitto version 1.5.5 starting
2019-04-22 20:14:37: Config loaded from /var/lib/mosquitto/10148/mosquitto.conf.
2019-04-22 20:14:37: Opening ipv4 listen socket on port 20148.
2019-04-22 20:14:37: Opening websockets listen socket on port 30148.
2019-04-22 20:14:37: Opening ipv4 listen socket on port 10148.
2019-04-22 20:16:07: New connection from 154.160.5.216 on port 10148.
2019-04-22 20:16:07: New client connected from 154.160.5.216 as m16.cloudmqtt.com (c1, k15, u'nsdyctor').
2019-04-22 20:16:07: No will message specified.
2019-04-22 20:16:07: Sending CONNACK to m16.cloudmqtt.com (0, 0)
2019-04-22 20:16:14: Received PUBLISH from m16.cloudmqtt.com (d0, q0, r0, m0, 'CAN/init', ... (0 bytes))
2019-04-22 20:16:22: Reloading config.
2019-04-22 20:16:36: Client m16.cloudmqtt.com has exceeded timeout, disconnecting.
```

Figure 5.4: Results of pushing data to MQTT cloud using GSM module



CAN Bus Monitoring.

The goal of this thesis report is to develop a system that can read diagnostic trouble codes and monitoring vehicle parameters, making the information accessible to the user

Figure 5.3: Interface for web application

Chapter 6: Conclusion

The hardware system developed in this project illustrates how technology can be used to supplement the skills of an existing and growing labor force. With this hardware system, it is possible to monitor real-time vehicle data. With such data, it is possible to proactively monitor vehicle performance. This enables one to predict and preempt unscheduled and unforeseen vehicle downtime. This is only possible as a result of the availability of vehicle data over the CAN bus. As cars are fitted with more and more sensors, varied data would be available over the bus. Allowing for much more comprehensive vehicle data and analysis. With the advent of machine learning, it would be possible to have a repository where models could be trained with aggregated vehicle data. This would mean vehicles could be more specific in diagnosing faults it notices.

6.1 Limitations

The designed hardware system cannot make sense of any other vehicle parameter aside the three parameters it was designed to monitor. Due to the use of the GSM module, latency may vary depending on the strength of the network reception. The hardware system relies only on the power supply of the vehicle. As a result, it is only functional while connected to the vehicle. The hardware system cannot retrieve DTCs stored in any ECU.

6.2 Challenges

In the execution of this project, there were several limitations that hindered the smooth execution of the project. The first of which is the need to outsource the PCB for milling and manufacturing. This was not only costly in terms of the time, but also the personal costs that went into having the boards printed and delivered. Another limitation that restricted the depth of research that could be done was the inaccessibility of certain document concerning the scope of the subject matter being investigated. ISO documents that defined vehicle standards in addition to other standardized codes were inaccessible. An example being the inability to request vehicle identification number (VIN) because the PID code for requesting this information is proprietary. This information affected the database design as in place of the VIN, the vehicle car number had to be used. The unavailability of DIP components to perform bread board testing prior to PCB design affected the ability to thoroughly test the hardware before moving to a PCB prototype.

6.3 Future Works

In spite of the limitations above, future works include enabling the hardware system to retrieve DTCs which are indicative of a given vehicle fault. Increasing the number of vehicle parameters being measured and thus expanding the functionality of the system. Logging GPS coordinates which could provide insightful details as to what point in a journey a fault first occurred. Additionally, a mobile application can be developed to replace the need for a web application. Also, the GSM module can inform a user via text message whenever a DTC is stored.

References

[1] C. A. Hesse and J. B. Ofosu, “COMPARATIVE ANALYSIS OF REGIONAL DISTRIBUTION OF THE RATE OF ROAD TRAFFIC CASUALTIES IN GHANA Key Words : Road traffic , Casualty and Accidents,” no. March 2014, p. 10, 2015.

[2] C. Roderick, “InfoSec Reading Room Developments in Car Hacking,” 2015.

[3] Valasek Chris and M. Charlie, “IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf,” 2013.

[4] C. Smith, The Car Hacker’s Handbook (2016).pdf. .

[5] Sewell Direct "A Brief Explanation of CAN Bus", Available:URL https://sewelldirect.com/learning-center/canbus-technology [Accessed 20 February 2019]

[6] Wojdyla Ben "How it Works: The Computer Inside Your Car", Available: URL https://sewelldirect.com/learning-center/canbus-technology [Accessed 20 February 2019]

[7] S. Corrigan, “Introduction to the Controller Area Network (CAN),” Texas Instruments, 2016.

[8] C. Valasek and C. Miller, “Adventures in Automotive Networks and Control Units,” Tech. White Pap., p. 99, 2013.

[9] A. Sawant, S. Dr. Lenina, and J. Dhananjay, “CAN , FlexRay , MOST versus Ethernet for Vehicular,” Int. J. Innov. Adv. Comput. Sci., vol. 7, no. 4, pp. 336–339, 2018.

[10] A. Hentschel, “Design of an information system for vehicle diagnostic trouble codes Computer Systems and Networks Design of an information system for vehicle diagnostic trouble codes,” 2013.

[11] A. Engineers, “SAE J1979: E/E Diagnostic Test Modes,” vol. 552, no. 1, p. 121, 2002.

[12] "OBD-II PIDs." Wikipedia. Wikimedia Foundation, 28 May 2013. URL: http://en.wikipedia.org/wiki/OBD-II_PID

[13] N. Navet, “Controller area network,” IEEE Potentials, vol. 17, no. 4, pp. 12–14, 1998.

[14] A. A. Salunkhe, P. P. Kamble, and R. Jadhav, “Design and implementation of CAN bus protocol for monitoring vehicle parameters,” 2016 IEEE Int. Conf. Recent Trends Electron. Inf. Commun. Technol. RTEICT 2016 - Proc., pp. 301–304, 2017.

[15] H. P. Sun, H. Zeng, and J. L. Guo, “Bus data acquisition and romote monitoring system based on CAN bus and GPRS,” 2011 Int. Conf. Consum. Electron. Commun. Networks, CECNet 2011 - Proc., pp. 1094–1097, 2011.

[16] R. Li, C. Liu, and F. Luo, “A Design for Automotive CAN Bus Monitoring System,” pp. 1–5, 2008.

[17] E. Ceuca, A. Tulbure, A. Taut, O. Pop, and I. Farkas, “Embedded System for Remote Monitoring of OBD Bus,” Proc. 36th Int. Spring Semin. Electron. Technol., pp. 305–308, 2013.

[18] Decision-making-Confidence "How To Use THe Pugh Matrix", Available:URL https://www.decision-making-confidence.com/pugh-matrix.html [Accessed 21 January 2019]

[19] Atmel Microchip, “8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash, ATmega328P datasheet,” [Revised April 2018].

[20] SIMCom, “SIM800L Hardware Design V1.0, SIM800L datasheet,” [Revised August 2013].

[21] Atmel Microchip,"Stand-alone CAN Controller with SPI Interface", “MCP2515 datasheet,” [Revised May 2014].

[22] Atmel Microchip,"High-speed CAN Transceiver", "MCP2561 datasheet,"[Revised April 2014].

[23] Diodes Incorporated, “Low Dropout Linear regulator”, “AZ1084 datasheet,” [Revised October 2013].

[24] Texas Instrument, “Positive Voltage Regulator”, “LM7805 datasheet”, May 1976 [Revised May 2003].

[25] MOTT.org "MQTT", Available:URL http://mqtt.org/ [Accessed 4 April 2019]

[26] VW, “Data transfer on CAN data bus II Drivetrain CAN data bus Convenience / infotainment CAN data bus.”

[27] I. Aris, M. F. Zakaria, S. M. Abdullah, and R. M. Sidek, “Development of OBD-II Driver Information System,” Int. J. Eng. Technol., vol. 4, no. 2, pp. 253–259, 2007.

[28] OBDII "The Connector", Available:URL http://www.obdii.com/connector.html [Accessed 8 April 2019]