Spring 2020

Zips Racing Electric CAN Communications

Andrew Jordan
The University of Akron, amj131@zips.uakron.edu

Adam Long
The University of Akron, ahl8@zips.uakron.edu

Susanah Kowalewski
The University of Akron, srk54@zips.uakron.edu

Rami Nehme
The University of Akron, ran49@zips.uakron.edu

Zips Racing Electric CAN Communication

Senior Design Project Final Report

Design Team 03

Andrew Jordan

Susanah Kowalewski

Adam Long

Rami Nehme

Advisor:

Huseyin Salis

April 24 2020

i. Table of Contents

ii. List of Figures

iii. List of Tables

Abstract

The CAN protocol has been a standard of electronic communication networks of automotive vehicles since the early 2000s due to its robust reliability in harsh environments. For the 2020 competition year, the Zips Racing Electric design team will be building an entirely new, fully-electric vehicle with CAN communication implemented rather than communicating via pure analog signals. Hardware and software can be utilized to read analog electrical signals from a source, such as accelerator and brake sensors, and encode them into a digital message that meets the CAN 2.0B communication protocol standard. Likewise, software can be used to extract data from CAN 2.0B messages, such as accumulator state of charge, which can then be sent to other subsystems, such as a dashboard display.

(AJ, SK, AL, RN)

1. Introduction

1.1. Need

The Zips Racing Electric team is building a brand new car for the 2020 competition season. In one year, the team designed and built a fully electric race car and competed against other universities for competitions in summer 2019.

Robust communication between electrical systems on the car is essential for reliable control. The 2019 vehicle used analog signaling to send the driver torque demand information from the pedals to the electronic control unit (ECU), which communicated with the motor controller. Electrical noise from the switching of the motor controller and the motor itself caused interference and disrupted the accelerator pedal signal. Signals such as brake pressure and battery cell temperatures were also impacted by noise. In addition, information such as the battery state of charge, motor temperature, and motor speed are monitored by independent electrical systems and were not communicated with the ECU, due to the limitations of discrete analog communication and wire harnessing. A digital communication scheme is needed on the new vehicle in order to facilitate data interfacing.

In addition, the 2019 vehicle had a rudimentary dashboard with no way for the driver to know the remaining state of charge in the traction battery. Feedback to the driver by including a display to indicate the battery state of charge will enable the driver to make full use of the battery during races and improve performance during a race.

(AJ, SK, AL, RN)

1.2. Objective

With a digital communication system based on the Controller Area Network (CAN) bus, data can be more easily shared between independent electrical systems and each system can make decisions based upon the data. This task includes a system to convert analog sensor data, such as the accelerator and brake pedal information, into CAN messages for other electronic modules on the vehicle to interpret.

Additionally, a dashboard with a display to indicate the high voltage battery state of charge will be included for driver feedback. The system will read and interpret CAN information about the state of charge and display the data.

(AJ, SK, AL, RN)

1.3. Background

Each year, the Society of Automotive Engineers (SAE) challenges students from universities around the world to “conceive, design, fabricate, develop, and compete” with small formula-style vehicles. Formula SAE cars are judged in a series of static and dynamic events, including technical inspection, cost analysis, engineering design, acceleration, autocross, and endurance. In 2012, SAE created a new class for electric only vehicles and the first competition was held in 2013 in Lincoln, Nebraska (FSAEOnline, 2019).

A team at the University of Akron built its first competitive electric car in the 2018-2019 academic year, with the stated goals of the team being to build a functioning car that passed the technical scrutineering by safety judges at the competition. Vehicles that do not pass technical inspection at competition are not allowed to race, and historically, the pass rate for electric teams is one third to one half, with pass rates near zero for first year teams. The newly-formed Zips Racing Electric team set out to build their first vehicle as a “rolling rulebook.” Safety, simplicity, and reliability were the main objectives for the team.

Nine months later, in April 2019, the Zips Racing Electric car was unveiled. With a custom 300 V lithium ion battery weighing 92 pounds, a liquid-cooled 80 kW motor and a three-phase inverter, the original electric race car from the University of Akron placed 6th out of 13 teams at its first competition in Ontario, Canada and placed 5th out of 25 teams two weeks later at competition in Lincoln, Nebraska (Formula SAE® Awards & Results, 2019).

The controls of the vehicle were communicated between electrical systems via analog signal levels. Electrical noise from the motor and inverter caused interference on the torque demand path and caused the torque output of the motor to be uneven. This effect was clearly visible from watching the car as it visibly and audibly jerked when driven. This interference was also seen in torque output as well, characterized by abrupt stops and starts at slow speeds. Along with noise, this lead to accelerated fatigue of parts on the car such as the axle, differential, and motor mounting points. A digital communication path along the vehicle, with proper shielding, is necessary to carry sensitive information and will improve performance overall.

A Controller Area Network (CAN) is a robust vehicle bus and communication protocol made for devices and microcontrollers to communicate without a host. Developed by Bosch in the 1980s, CAN is a message-based scheme, intended to reduce wiring inside of automobiles (Hartwich, F, 2014).

Unlike other buses, CAN is not a point-to-point electrical link, but instead is a multi-master bus. Standardized by ISO 11898, CAN is specified with a maximum bitrate of 1 Mbps for distances up to 40 m, dropping to 10 kpbs at the maximum specified bus length of 1 km. CAN is ideal for short broadcast type messages, limited to 8 bytes of data per packet, with data directed at no device in particular. Instead, any device on the bus can read any message transmission. (Horowitz, P., & Hill, W, 2015).

Electrically, CAN is a differential open-collector scheme, specified up to ±12 V common mode, allowing for noise robustness in the automotive environment (Road vehicles - Controller area network, 2016). CAN bus prescribes several error-detection mechanisms, including at the bit level and at the message level (Kvaser, Ed).

With a CAN communication bus, the Zips Racing Electric team will be well equipped to perform at future competitions with a more reliable race car.

(AJ, SK, AL, RN)

1.4. Marketing Requirements

1. Brake and acceleration pedal analog sensor values shall be converted to CAN messages and be outputted onto the CAN bus.
2. The software shall be capable of reading messages from the CAN bus.
3. The state of charge (SOC) reading from the battery management system (BMS) shall be displayed on the dashboard of the vehicle.

4. The product system should be power-efficient.
5. The product should provide redundancy in case of a single failure.

   (AJ, SK, AL, RN)

2. Engineering Analysis

2.1. Circuits

For analog to CAN conversion of the dual accelerator and brake sensors, circuit design is necessary. Hardware signal processing from the sensors includes scaling, offsetting, and filtering before the signal can be read properly by an analog-to-digital converter. In addition, detection for failure of the wiring for open and short circuit conditions must be included. For instance, the signal may have a typical output of 0.5 V-4.5 V and any reading outside of that range is considered to be erroneous and must be detectable.

As two independent redundant accelerator pedal position sensors are used, each sensor must have a different transfer function of pedal position to output voltage per FSAE 2020 rules. The sensors are to be used are linear potentiometers connected to the +5 V and GND rails, but an additional resistor between the rails is used for one of the sensors, thereby reducing the voltage across the sensor and causing the device to have a different transfer function. With different sensor outputs for the same pedal position allows for simple detection of wiring faults for any downstream devices.

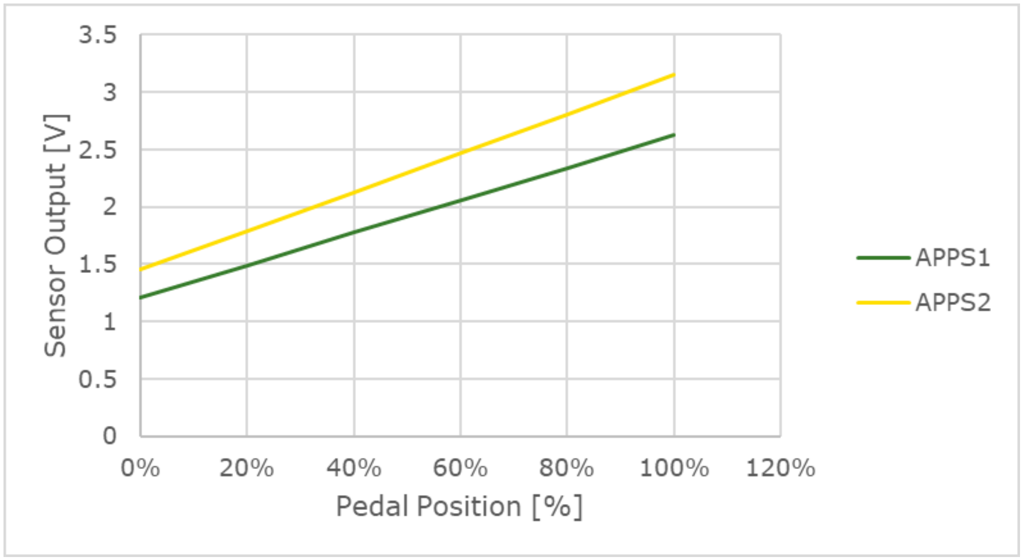


Figure 1: Accelerator pedal position sensor transfer function

For the dashboard, display driver hardware is needed to display information to the driver of the vehicle. The display should be easily visible, in direct sunlight, to the driver of the vehicle.

Both projects require stable power supply rails despite being powered from a varying battery voltage as the battery discharges. A power conversion circuit must provide a reliable output for accurate measurements.

Electrically, CAN bus is a differential open collector bus where the signals float nominally at 2.5 V. In order to successfully connect a microcontroller to the CAN bus to receive and transmit messages, a CAN transceiver is needed. The device should perform level shifting from the microcontroller outputs and inputs to the physical CAN bus. In addition, power output stages are needed in the driver in order to control the signals in the CAN bus, which may have lengthy wiring. The device will also require a stable power supply to operate.

(AJ, SK, AL, RN)

2.2. Communications

The CAN 2.0B protocol is a robust and reliable communication method widely used in the automotive industry defined by the International Standardization Organization. CAN is a broadcast, multi-master protocol with error detection. The protocol uses a twisted pair, differential sensing wires for its physical layer providing high resistance to EMI. Furthermore, the protocol transmits Cyclic Redundancy Check (CRC) bits with every message allowing for simple verification of data validity. The CAN protocol utilizes a 15-bit CRC polynomial ($x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + x^{0}$) division to calculate the CRC bits. The same algorithm is used by both the transmitter and receiver to verify that the message is error-free. In the event of a CRC mismatch/failure, the message is deemed corrupt and is thrown out.

CAN offers two message framing formats, a Standard Frame and an Extended Frame. The standard frame contains an 11-bit field for prioritization while the extended frame contains 22 bits for prioritization. Since this project will only transmit 3 message frames, the extra 11 priority bits available in the extended frame will not be needed, so the standard frame will be utilized for the design of the CAN bus in the electric vehicle. The IDE bit of the message will therefore be set to dominant (0) to indicate that this is a standard CAN message.

CAN allows for the identification of frame type, and also allows for prioritization of data on the bus based on a “Remote Transmission Request (RTR)” field. This is a single bit that is set to 0 for a dominant data frame, and 1 for a recessive remote data frame. As such, for the safety critical and time sensitive frames (the accelerator and brake position frames), the RTR value shall be set to a 0. Thus these critical frames will win arbitration on the CAN bus allowing their messages to go through. The data frame containing the BMS charge value will receive a 1 for its RTR value, as it is a non-critical sensor that outputs its value on a given interval. This is not necessary for the scope of this project, as there will only be 3 CAN messages and arbitration over the bus utilizing the RTR value will not occur in this small of a scale, but it is good engineering practice.

The 11-bit identifier serves dual purpose in the fact that it contains the priority of the message on the bus (for arbitration purposes) and identifies the messages for listening devices. Within the scope of this project, there will only be three messages on the bus. In the event that a frame has the same identifier (equal priority), the frame with a dominant (binary 0) RTR frame will win arbitration for the bus (Note for this design, no two messages will share the same identifier). The highest priority will be the CAN message containing the values of the accelerator sensor. The message with the second priority shall be the message containing the brake sensor values. Finally, the BMS sensor will output non-critical data on a given time interval, and thus will be assigned the lowest priority.

It should be noted that, the frames containing the accelerator and brake position will never require arbitration for priority, as they are sent out by the same microcontroller and it will only transmit messages when the bus is available.

For the frames containing the accelerator and brake position, the DLC (Data length field) will indicate that the message contains 4 bytes. Each sensor outputs 2 bytes worth of information, thus putting the information of both sensors onto one frame will require 4 bytes. Note that the accelerator and brake message frames are separate. Hence, 1 message will contain the values for both accelerator sensors, and another message will contain the values of both brake sensors. This architecture has been chosen to facilitate speed and safety. While both the accelerator and brake position could be placed into a single CAN message, in the event of a CRC failure, the entire message is thrown out and the ECU will not have values for either the accelerator or brake sensor. By splitting up the accelerator and brake position into two messages, this provides redundancy as a CRC failure for either the brake or accelerator sensor will only throw out the data for that respective message. Splitting the information up into 4 messages (1 sensor per message), is not practical either for this purpose as it adds complexity, creating more room for error, which is to be avoided in a safety-critical system. The frames will be transmitted by the microcontroller on a 10ms timing interval provided by the internal clock of the microcontroller. The packaging of the CAN message along with application layer encoding is handled by the CAN module on the microcontroller.

On transmission, the SOF (start of frame) bit will indicate to all devices on the CAN bus that the bus is in use and no longer idle. This "wake-up" feature will be utilized by the dashboard microcontroller to trigger an interrupt service routine (ISR), to scan the CAN message on the bus. All messages on the bus are placed into a buffer on the dashboard microcontroller. The software then scans the identifier bits for the identifier associated with the BMS. CAN messages that do not have this identifier are thrown out of the buffer. After receiving a frame from the BMS, the software will perform a CRC calculation on the frame, and look for a CRC mismatch. In the event of a mismatch, the message is thrown out. The software will then go to the first character of the data field. Normally for variable length messages (messages with the same identifier that may have different data lengths every transmission) the software should read the DLC character to identify how many data bits are contained in the message and then read that many bits from the data. However, since the BMS will always output the same number of bits for the battery charge percentage, the DLC bit does not need to be read and the data length can be hard-coded since it is constant. This saves on read cycles, the need to use vectors (a static array can be used instead), and saves iteration cycles and memory, thus optimizing the efficiency of the code.

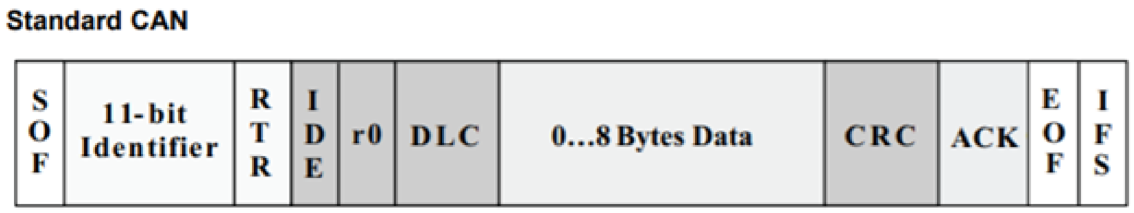


Figure 2: Standard CAN Message, (Corrigan, S)

In order to function properly, CAN bus requires termination resistors at each end of the bus. Due to the nature of the bus, a transmission line is created. These resistors prevent reflections on the line and allow the signals to be clearly read. A resistance of 120 Ω on each end of the CAN network is sufficient for successful operation. (AJ)

(AJ, SK, AL, RN)

2.3. Electromechanics

Measuring accelerator pedal position requires electrical sensors with moving parts. These sensors interface with the pedal assembly and must be mechanically robust to avoid damage. Common sensor types include rotary position sensors and linear potentiometer sensors as potentiometers.

Transducers for monitoring brake pressure are mounted on the brake lines. These sensors should be rated for the maximum anticipated pressure in the system during hard braking. Pressure in the brake lines is directly related to the force exerted on the brake pedal and the diameter of the master cylinder. The hydraulic pressure during hard braking will reach approximately 1500 psi, but the pressure can peak at 2000 psi. Therefore, any brake pressure transducer must be rated to read at least 1500 psi with a burst pressure specification of 2000 psi or more. Sensors that read pressure are strongly affected by the ambient temperature. Therefore, any sensor monitoring the brake pressure should include temperature compensation.

(AJ, SK, AL, RN)

2.4. Embedded Systems

For the purposes of analog to digital signal conversion and digital to CAN signal conversion, a microcontroller will be used. This microcontroller will be a 16-bit processor that contains an onboard analog-to-digital converter that can be used to more easily perform signal conditioning and conversion. The processor will support up to 36 analog inputs to allow scalability of the system, as more sensors are added onto the vehicle. The software architecture supports the use of high-level C code through the Microchip MPLAB IDE. This allows greater focus on algorithm development and enables structured programming. This enables the code to be more “modular” for the purposes of sensory input and data output. Furthermore, the XC16 compiler used for the microprocessor and MPLAB IDE are free to use, thus this cost factor also weighed into the decision to select this processor.

The microcontroller also features a CAN module with 32 buffers for data transmission. This is to allow for the output and input data to be synchronized to ensure that information is coherent with each other. Additionally, the CAN module allows for straightforward management and encoding of CAN messages per the CAN2.0 standards.

The design of the analog to CAN signal conversion system requires the use of one or more microcontrollers to handle the bulk of the signal conversion process. Once analog signals are received, ADC hardware and specialized software algorithms will interpret those signals and convert them into CAN 2.0B messages. These messages will then need to be broadcast to the CAN bus.

The dashboard display system will use also utilize a microcontroller in order to interface to an LCD 7-segment display to the CAN network. The CAN 2.0B message from the accumulator is broadcast to the CAN bus. Once this message is received by the microcontroller in the dashboard system, the message will be decoded by software to receive the state of charge. The microcontroller then needs to send HI/LO encoded signals to hardware addresses on the 7-segment display to show the accumulator state of charge.

(AJ, SK, AL, RN)

3. Engineering Requirements Specifications

Table 1: Engineering Requirements

Marketing Requirement	Engineering Requirement	Justification
1	The analog to CAN signal conversion system will accept signals in the range of 0 V to 5 V DC.	Most sensors today typically have outputs in the range of 0 V to 5 V DC.
4	The analog to CAN signal conversion system will operate from a battery supply voltage in the range of 9 V to 15 V.	Battery voltage in the range of 9 V to 15 V is the voltage supplied to the conversion system by the car.
1	The analog to CAN signal conversion system will convert signals to digital messages that meet the CAN 2.0B communication protocol standards.	CAN is the standard system for automotive vehicles. CAN 2.0B allows the needs of the car to be met without adding unnecessary complication.
2	The dashboard display system will accept a CAN 2.0B message from the electric car's accumulator.	A CAN 2.0B message will be supplied from the ECU to the conversion system, so the conversion system must be able to read it.
3	The dashboard display system will display the accumulator state of charge, as a percentage ranging from 0% to 99%.	The accumulator state of charge is vital information to the driver and displaying it as a percentage is easy to read quickly.
5	The accelerator pedal will have two separate position sensors with different transfer functions	Required by 2020 SAE rules for safety purposes.

	for their circuits.	
5	The system will be able to detect open or short circuit readings from the accelerator pedal and brake pedal sensor circuits.	Required by 2020 SAE rules for safety purposes.
1	The analog to CAN signal conversion system will output CAN messages onto the CAN bus which must be readable by the ECU.	These signals will go to the ECU which will tell the car to 'go' and therefore must be readable for the car to work as intended.
3	The dashboard display system's display will be readable in bright sunlight and visible to a driver sitting approximately three feet away.	Car driving will only occur during the day, so the display will need to be visible in bright sunlight.
4	The CAN conversion and dashboard display systems will each consume less than 1 ampere of current.	The system should not consume excess power from the battery. Lower power will also keep component temperatures within reasonable limits.
1	The dual accelerator sensors and brake pressure sensors will be sampled at an iteration rate of at least 10 Hz (100 ms).	The ECU will be configured per SAE rules to fault and shut the car off if it does not receive CAN messages after a specific amount of time.

Marketing Requirements:

1. Brake and acceleration pedal analog sensor values shall be converted to CAN messages and be outputted onto the CAN bus.
2. The software shall be capable of reading messages from the CAN bus
3. The state of change reading from the battery management system (BMS) shall be displayed on the dashboard of the vehicle.
4. The product system should be power-efficient.
5. The product should provide redundancy in case of a single failure.

4. Engineering Standards Specification

Table 2: Engineering Standards Applied

Application	*Standard*	*Use*
Safety	Formula SAE 2020 Version 2.0	FSAE 2020 Rules from SAE dictate how the race competition operates and gives safety requirements about the vehicle, such as have dual accelerator sensors and fault detection in the event of lost CAN messages.
Communications	ISO 11898-1, ISO 11898-2	ISO 11898 describes the CAN 2.0B bus protocol and physical layer. The standard is used to determine the voltages used in communication and the structure of the data frames sent in the bus to ensure compatibility between all devices connected on the vehicle.
Programming Languages	ANSI x3.159-1989	ANSI C or C89 details how the C programming language should behave. Using a standard for the programming language ensures the firmware will compile successfully and function in a known manner.
Ingress Protection	ANSI/IEC 60529	The IP Code outlines how components or products should tolerate dust and water. Having adequate protection from the environment is necessary in the automotive industry.

5. Accepted Technical Design

5.1. Hardware Design

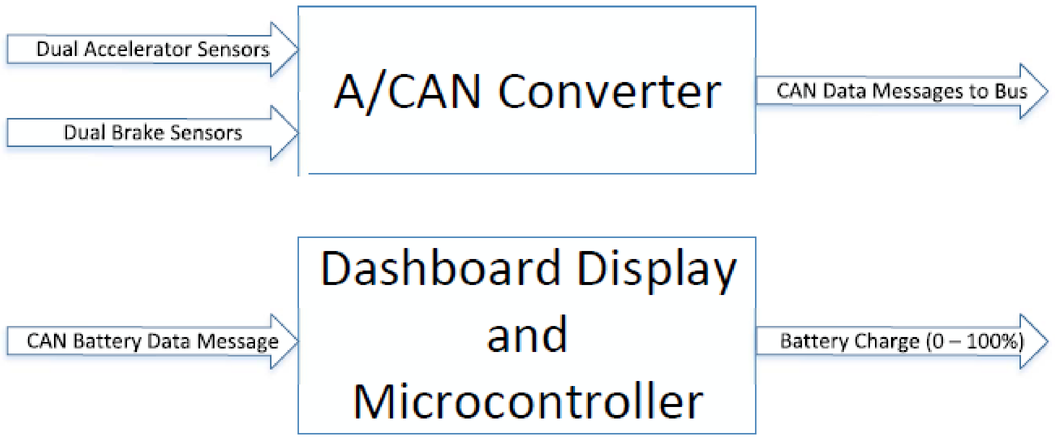


Figure 3: Hardware Level 0 Diagram

Table 3: A/D CAN Converter Hardware Functional requirements - Level 0

Module	Analog-to-CAN Converter
Inputs	Dual redundant accelerator pedal position sensors Dual brake pressure transducers for front and rear hydraulic circuits
Outputs	Direct connection to physical CAN bus Converted CAN messages
Functionality	The A/CAN Converter will accept analog electrical signals from dual accelerator sensors and dual brake sensors and convert them into digital CAN signals. These signals can later be used by other components of the vehicle.

Table 4: Dashboard Display Hardware Functional requirements - Level 0

Module	CAN-enabled Dashboard Display
Inputs	CAN bus data (accumulator)
Outputs	Display high-voltage battery's state of charge

Functionality	The dashboard will accept CAN data messages from the accumulator and display the remaining charge left on the battery. The charge will be displayed as a percentage between 0% and 99%.

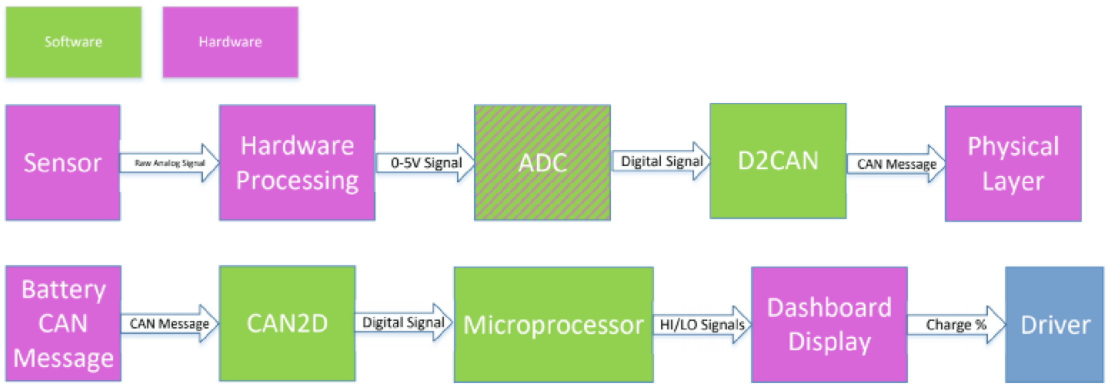


Figure 4: Hardware Level 1 Diagram

Table 5: Sensor - Hardware Functional Requirements - Level 1

Module	Sensor
Inputs	5 V input, power on voltage
Outputs	Dual redundant accelerator pedal position sensors output 0 - 5 V Dual brake pressure transducers for front and rear hydraulic circuits output 0 - 5 V
Functionality	The accelerator pedal sensors will report a voltage proportional to the pedal position. The brake pressure sensors will report a voltage proportional to the pressure in the brake lines.

Table 6: Hardware Processing - Hardware Functional Requirements - Level 1

Module	Hardware Processing
Inputs	0 - 5 V from accelerator pedal and brake pressure sensors
Outputs	0 - 5 V signal
Functionality	Reduces noise on the 0 - 5 V signals coming from the sensors by filtering.

Table 7: Microcontroller (A/CAN System) - Hardware Functional Requirements - Level 1

Module	Microcontroller (ADC and D2CAN)
Inputs	5 V input, power on voltage 0 - 5V analog signals from sensors
Outputs	CAN signal containing reported sensor information
Functionality	Performs analog-to-digital and digital-to-CAN conversion.

Table 8: Physical Layer - Hardware Functional Requirements - Level 1

Module	Physical Layer
Inputs	CAN signal containing sensor information
Outputs	CAN physical message containing sensor information
Functionality	Configures CAN signal to appropriate levels for transmitting and puts the message on the CAN Bus.

Table 9: Battery CAN Message - Hardware Functional Requirements - Level 1

Module	Battery CAN Message
Inputs	N/A
Outputs	CAN message, from accumulator, containing SOC
Functionality	Connector for accumulator message to transfer to the PCB

Table 10: Microcontroller (Dashboard System) - Hardware Functional requirements - Level 1

Module	Microcontroller (CAN2D, Microprocessor)
Inputs	5 V input, power on voltage CAN signal containing accumulator SOC
Outputs	Digital signal containing accumulator SOC
Functionality	Performs CAN to digital conversion.

Table 11: Dashboard Display - Hardware Functional requirements - Level 1

Module	Dashboard Display
Inputs	High/Low voltage signal inputs to seven-segment display
Outputs	Accumulator voltage percentage ranging from 0 - 99% displayed on seven-segment display
Functionality	Displays accumulator SOC.

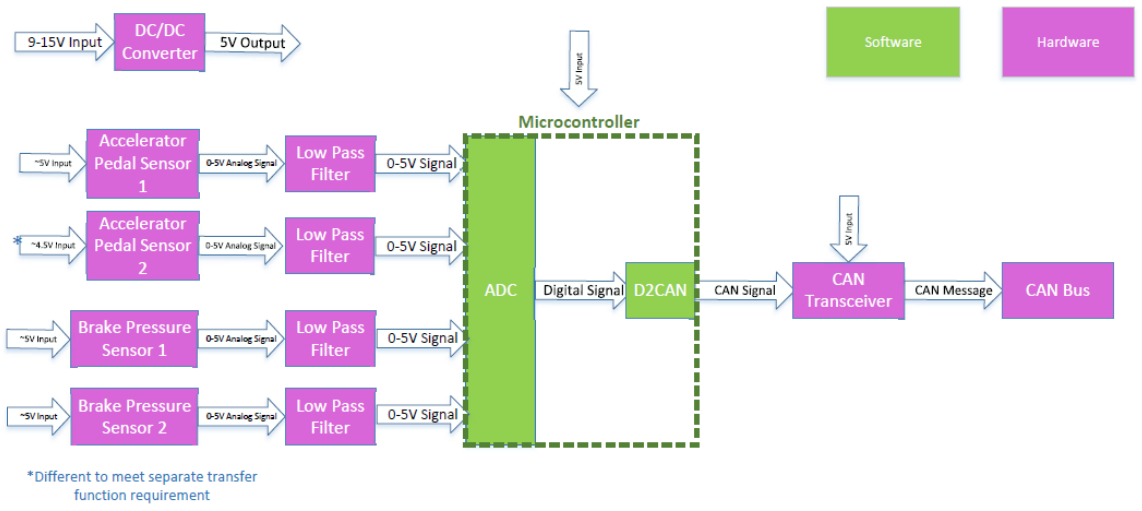


Figure 5: Hardware Level 2 Diagram (Analog to CAN System)

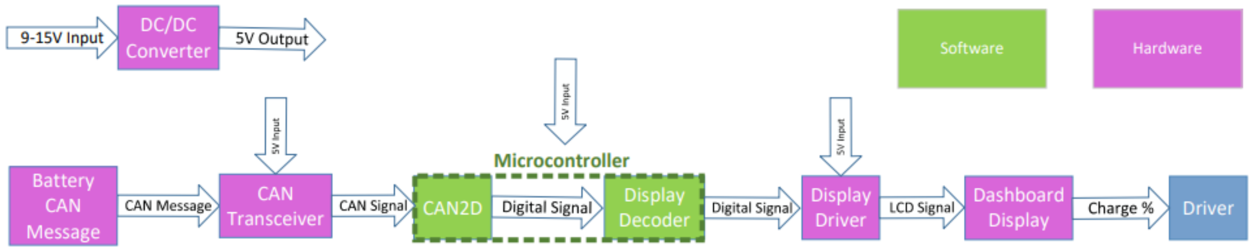


Figure 6: Hardware Level 2 Diagram (Dashboard Display System)

Table 12: Accelerator Pedal Sensor 1 & 2 - Hardware Functional requirements - Level 2

Module	Accelerator Pedal Sensor 1 & 2
Inputs	5 V input, power on voltage
Outputs	Dual redundant accelerator pedal position sensors output 0 - 5 V

Functionality	The accelerator pedal sensors will report a voltage proportional to the pedal position.

Table 13: Brake Pressure Sensor 1 & 2 - Hardware Functional requirements - Level 2

Module	Brake Pressure Sensor 1 & 2
Inputs	5 V input, power on voltage
Outputs	Dual brake pressure transducers for front and rear hydraulic circuits output 0-5 V
Functionality	The brake pressure sensors will report a voltage proportional to the pressure in the brake lines.

Table 14: Low Pass Filter - Hardware Functional requirements - Level 2

Module	Low Pass Filter
Inputs	0 - 5 V from accelerator pedal and brake pressure sensors
Outputs	0 - 5V signal
Functionality	Reduces noise on the 0 - 5 V signals coming from the sensors by filtering, constructed as an RC filter.

Table 15: Microcontroller (A/CAN System) - Hardware Functional requirements - Level 2

Module	Microcontroller (ADC, D2CAN)
Inputs	5 V input, power on voltage 0 - 5V analog signals from sensors
Outputs	CAN signal containing reported sensor information
Functionality	Performs analog to digital and digital to CAN conversion.

Table 16: CAN Transceiver (A/CAN System) - Hardware Functional requirements - Level 2

Module	CAN Transceiver
Inputs	5 V input, power on voltage CAN signal containing reported sensor information

Outputs	CAN message containing sensor information
Functionality	Configures CAN signal to appropriate levels for transmitting.

Table 17: CAN Bus - Hardware Functional requirements - Level 2

Module	CAN Bus
Inputs	CAN message containing sensor information
Outputs	CAN message to the ECU from the CAN Bus
Functionality	Stores and transports car information in CAN messages.

Table 18: DC/DC Converter (A/CAN and Dashboard Systems) - Hardware Functional requirements - Level 2

Module	DC/DC Converter
Inputs	9 - 15 V range
Outputs	5 V, regulated
Functionality	Convert the voltage received from the secondary car battery to a steady 5 V supply.

Table 19: Battery CAN Message - Hardware Functional requirements - Level 2

Module	Battery CAN Message
Inputs	CAN message, from accumulator, containing SOC
Outputs	CAN message, from accumulator, containing SOC
Functionality	Connector for accumulator message to transfer to the PCB

Table 20: CAN Transceiver (Dashboard System) - Hardware Functional requirements - Level 2

Module	CAN Transceiver
Inputs	5 V input, power on voltage CAN message containing accumulator SOC
Outputs	CAN signal containing accumulator SOC

Functionality	Configures CAN message to appropriate levels for reading by the Microcontroller.

Table 21: Microcontroller (Dashboard System) - Hardware Functional requirements - Level 2

Module	Microcontroller (CAN2D, Display Converter)
Inputs	5 V input, power on voltage CAN signal containing accumulator SOC
Outputs	Digital signal containing accumulator SOC
Functionality	Performs CAN to digital conversion.

Table 22: Display Driver - Hardware Functional requirements - Level 2

Module	Display Driver
Inputs	5 V input, power on voltage Digital signal containing accumulator SOC
Outputs	LCD signal containing accumulator SOC
Functionality	Sends signal to seven-segment LCD display to display accumulator SOC.

Table 23: Dashboard Display - Hardware Functional requirements - Level 2

Module	Dashboard Display
Inputs	High/Low voltage signal inputs to seven-segment display
Outputs	Accumulator voltage percentage ranging from 0 - 99% displayed on seven-segment display
Functionality	Displays accumulator SOC.

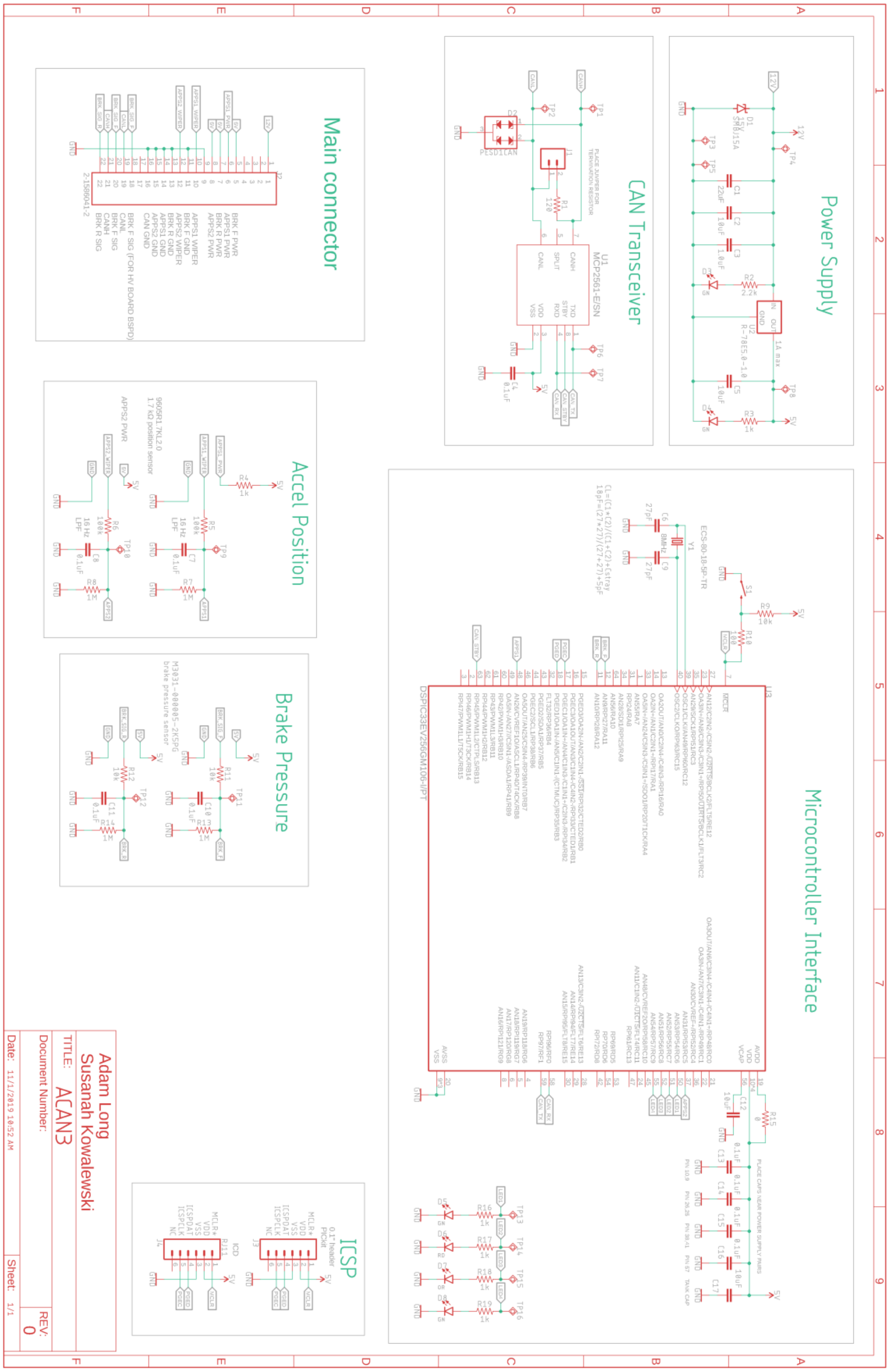


Figure 7: A/CAN PCB Schematic – Full

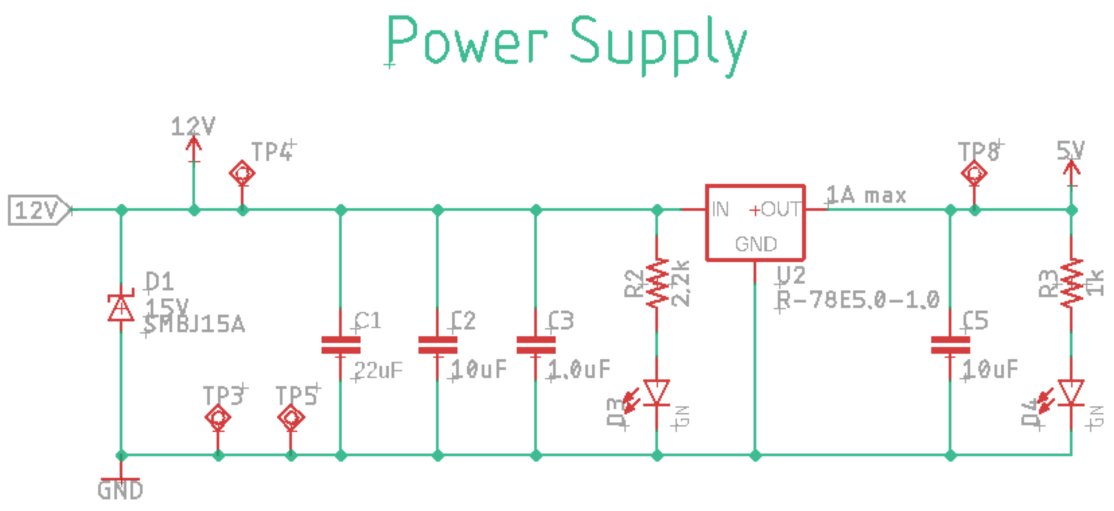


Figure 8: A/CAN PCB Schematic – Power Supply

Table 24: A/CAN PCB Schematic – Power Supply

Designator & P/N	U2 R-78E5.0-1.0
Inputs	9 – 15 V range
Outputs	5 V regulated
Functionality	Convert the voltage received from the secondary car battery to a steady 5 V supply.

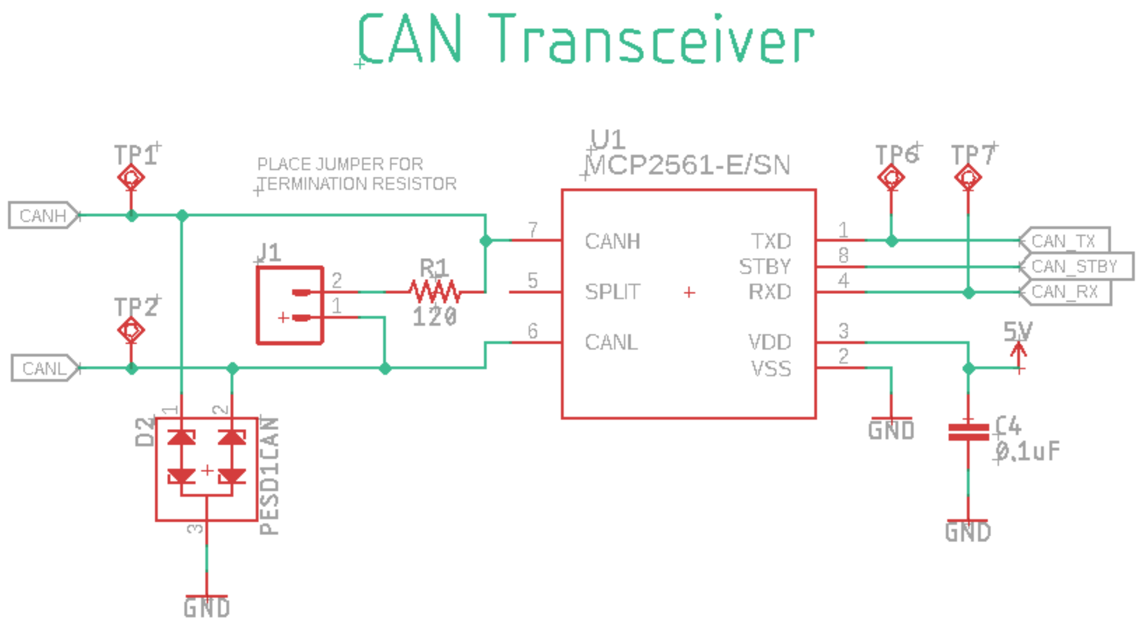
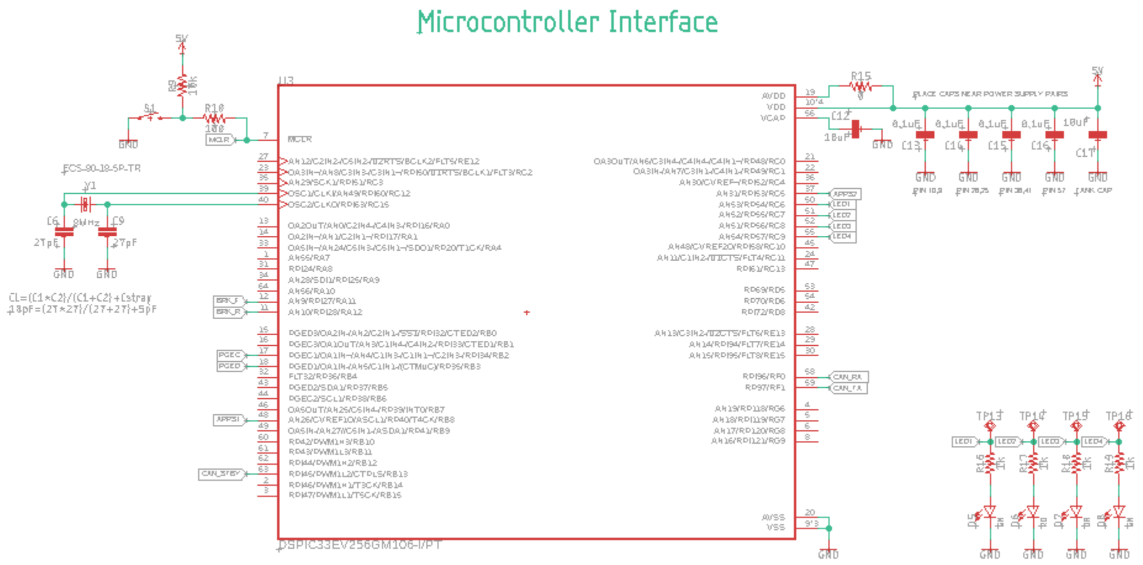


Figure 9: A/CAN PCB Schematic – CAN Transceiver

Table 25: A/CAN PCB Schematic – CAN Transceiver

Designator & P/N	U1 MCP2561-E/SN
Inputs	5 V input, power on voltage CAN signal containing reported sensor information
Outputs	CAN message containing sensor information
Functionality	Configures CAN signal to appropriate levels for transmitting



Microcontroller Interface
U3
DSPIC33EV256GM106-I/PT
PLACE CAPS NEAR POWER SUPPLY PAIRS
CL=(C1*C2)/(C1+C2)+Cstray
18pF=(27*27)/(27+27)+5pF

Figure 10: A/CAN PCB Schematic – Microcontroller

Table 26: A/CAN PCB Schematic – Microcontroller

Designator & P/N	U3 DSPIC33EV256GM106-I/PT
Inputs	5V input, power on voltage 0-5V analog signals from sensors
Outputs	CAN signal containing reported sensor information
Functionality	Performs analog-to-digital and digital-to-CAN conversion.

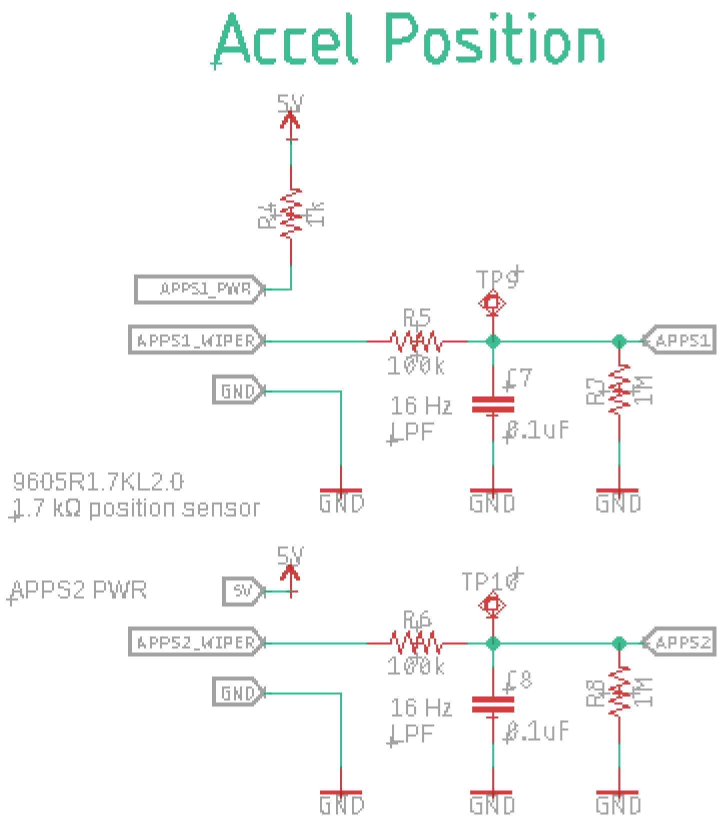


Figure 11: A/CAN PCB Schematic – Accelerator Sensor Filter

Table 27: A/CAN PCB Schematic – Accelerator Sensor Filter

Designator & P/N	R5/C7 & R6/C8 RMCF0603FG100K & CL10F104ZB8NNNC
Inputs	0-5V from accelerator pedal sensors
Outputs	0-5V signal
Functionality	Reduces noise on the 0-5V signals coming from the sensors by filtering, constructed as an RC filter.

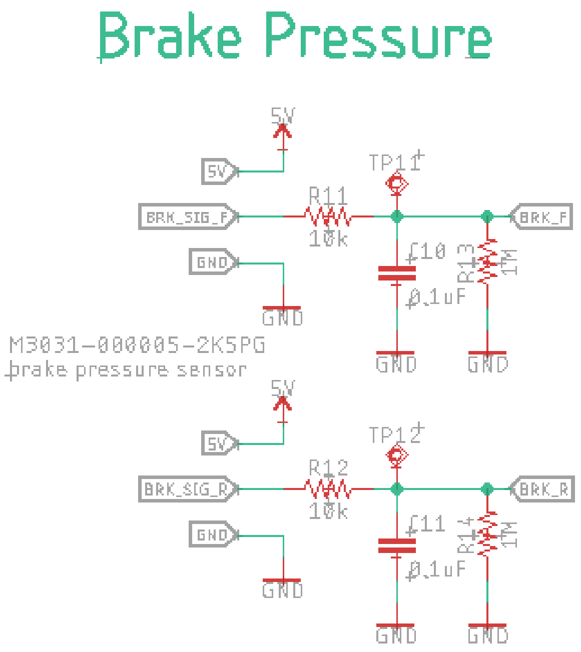


Figure 12: A/CAN PCB Schematic – Brake Sensor Filter

Table 28: A/CAN PCB Schematic – Brake Sensor Filter

Designator & P/N	R11/C10 & R12/C11 RMCF0603FT10K0 & CL10F104ZB8NNNC
Inputs	0-5V from brake sensors
Outputs	0-5V signal
Functionality	Reduces noise on the 0-5V signals coming from the sensors by filtering, constructed as an RC filter.

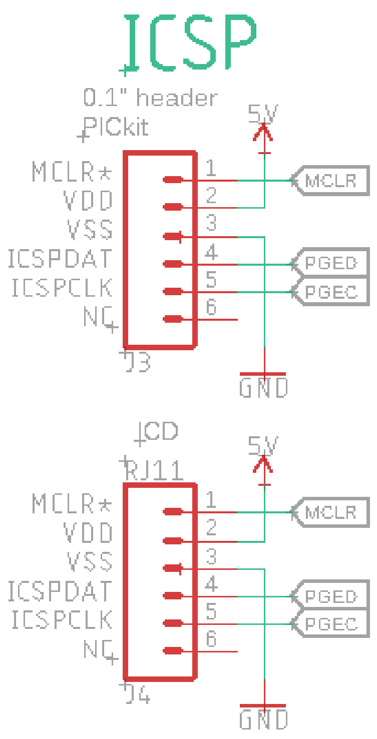


Figure 13: A/CAN PCB Schematic – PIC Programming Headers

Table 29: A/CAN PCB Schematic – In Circuit Serial Programming Connectors

Designator & P/N	J3 & J4 M20-9990646 & A-2004-1-4-N-R
Inputs	External programming connector
Outputs	Programming signals to the PIC
Functionality	Connectors to upload program and debug PIC microcontroller

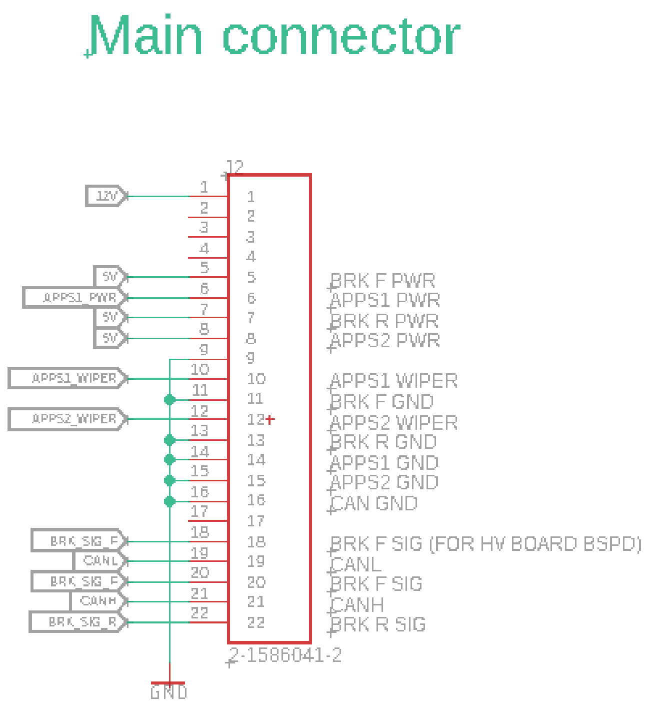


Figure 14: A/CAN PCB Schematic – Main Connector

Table 30: A/CAN PCB Schematic – Main Connector

Designator & P/N	J2 2-1586041-2
Inputs	0-5V from accelerator pedal and brake sensors 9 – 15 V range
Outputs	0-5V from accelerator pedal and brake sensors
Functionality	Connector to deliver power and sensors signal to the board

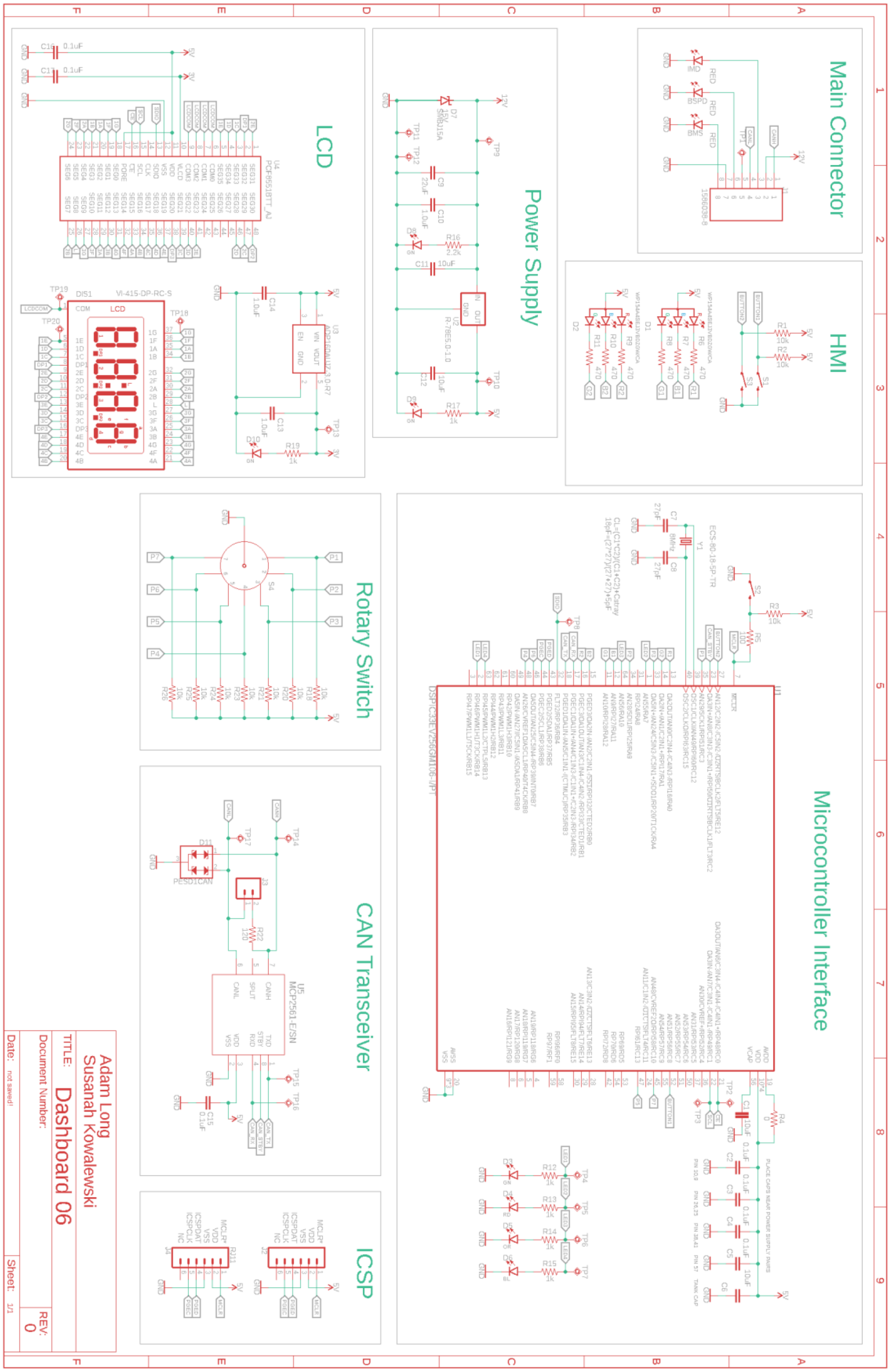


Figure 15: Dashboard PCB Schematic – Full

Power Supply

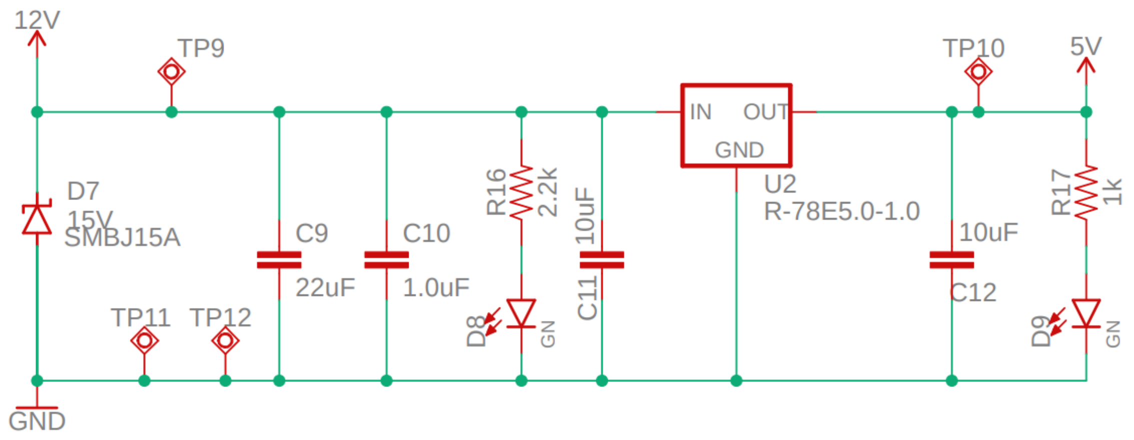


Figure 16: Dashboard PCB Schematic – Power Supply

Table 31: Dashboard PCB Schematic – Power Supply

Designator & P/N	U2 R-78E5.0-1.0
Inputs	9 – 15 V range
Outputs	5V regulated
Functionality	Convert the voltage received from the secondary car battery to a steady 5V supply.

CAN Transceiver

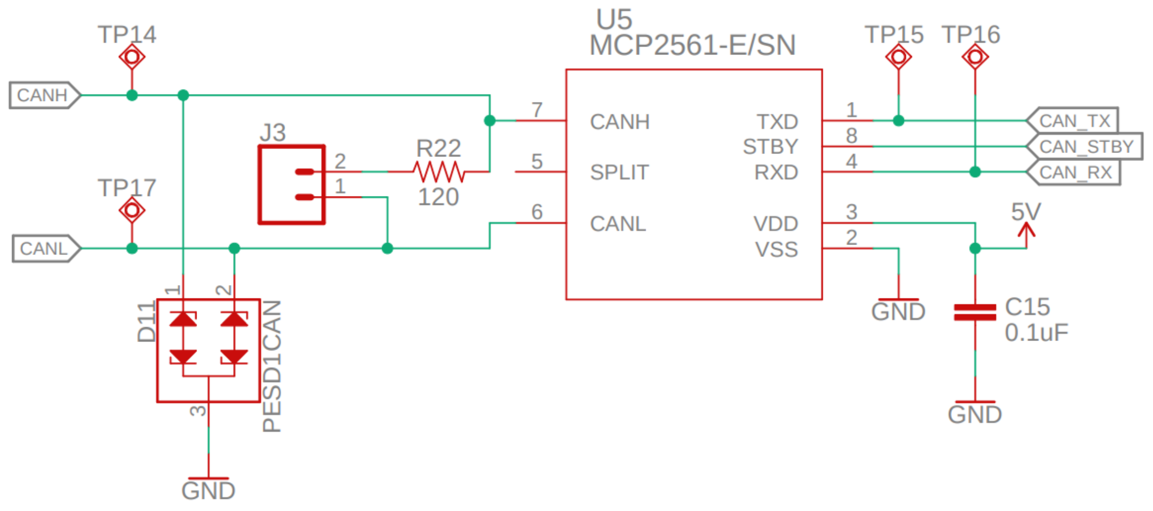


Figure 17: Dashboard PCB Schematic – CAN Transceiver

Table 32: Dashboard PCB Schematic – CAN Transceiver

Designator & P/N	U5 MCP2561-E/SN
Inputs	5V input, power on voltage CAN signal containing reported accumulator SOC information
Outputs	CAN message containing accumulator SOC information
Functionality	Configures CAN signal to appropriate levels for reading

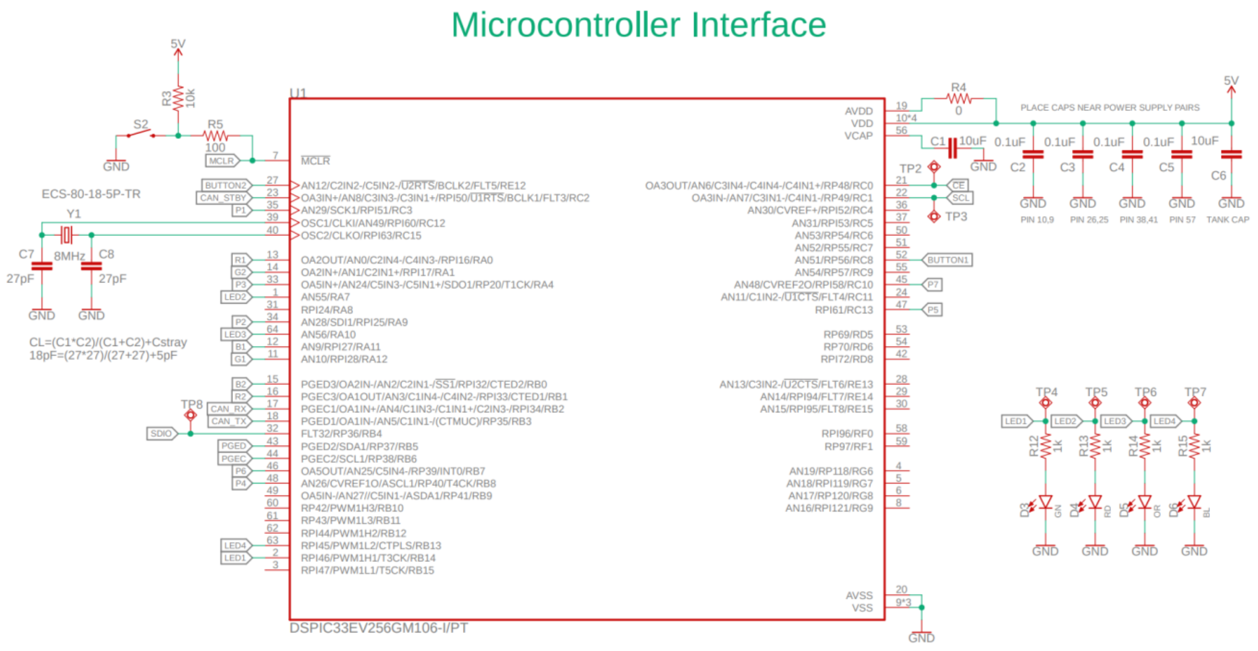

Microcontroller Interface
U1
DSPIC33EV256GM106-I/PT
PLACE CAPS NEAR POWER SUPPLY PAIRS
ECS-80-18-5P-TR
CL=(C1*C2)/(C1+C2)+Cstray
18pF=(27*27)/(27+27)+5pF

Figure 18: Dashboard PCB Schematic – Microcontroller

Table 33: Dashboard PCB Schematic – Microcontroller

Designator & P/N	U1 DSPIC33EV256GM106-I/PT
Inputs	5V input, power on voltage CAN signal containing accumulator SOC
Outputs	Digital signal containing accumulator SOC for LCD
Functionality	Performs CAN to digital conversion.

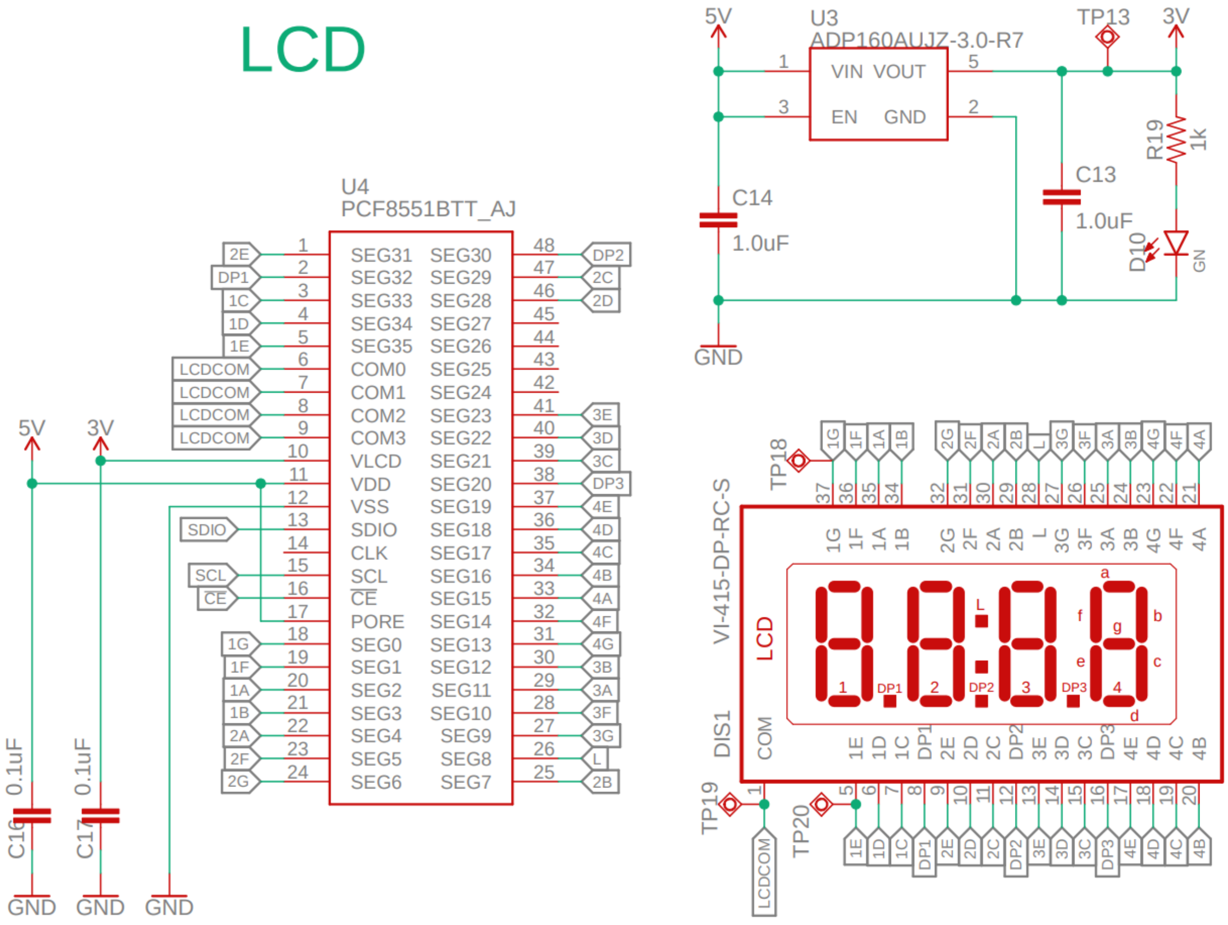


Figure 19: Dashboard PCB Schematic – LCD and Driver

Table 34: Dashboard PCB Schematic – LCD

Designator & P/N	DIS1 VI-415-DP-RC-S
Inputs	Digital signal containing accumulator SOC
Outputs	Accumulator SOC displayed in seven-segment numbers
Functionality	Displays accumulator SOC

Table 35: Dashboard PCB Schematic – LCD Driver

Designator & P/N	U4 PCF8551BTT/AJ
Inputs	3 V and 5 V input, power on voltages

	CAN signal containing accumulator SOC
Outputs	Signals containing accumulator SOC for LCD
Functionality	Changes digital signal to appropriate signal for LCD

Table 36: Dashboard PCB Schematic – DC/DC Converter

Designator & P/N	U3 ADP160AUJZ-3.0-R7
Inputs	5 V input
Outputs	3 V, regulated
Functionality	Steps 5 V down to 3 V

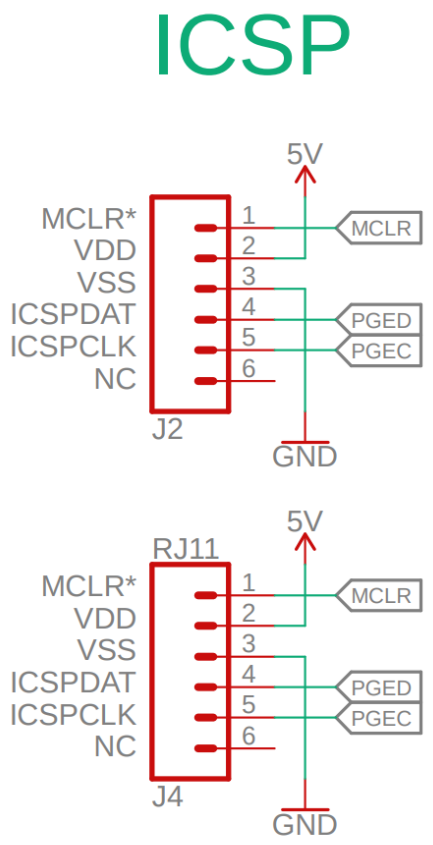


Figure 20: Dashboard PCB Schematic – PIC Programming Headers

Table 37: Dashboard PCB Schematic – In Circuit Serial Programming Connectors

Designator & P/N	J2 & J4 M20-9990646 & A-2004-1-4-N-R
Inputs	External programming connector
Outputs	Programming signals to the PIC
Functionality	Connectors to upload program and debug PIC microcontroller

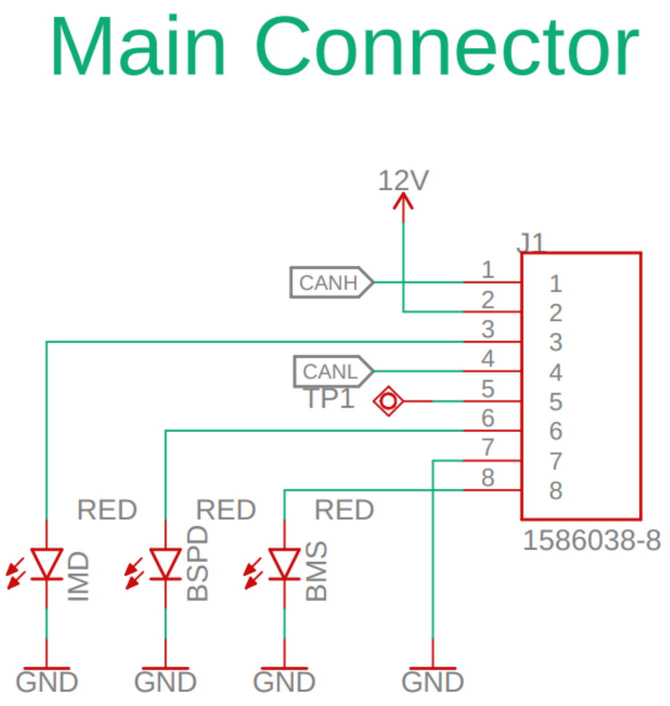


Figure 21: Dashboard PCB Schematic – Main Connector

Table 38: Dashboard PCB Schematic – Main Connector

Designator & P/N	J1 1586041-8
Inputs	CAN signal containing accumulator SOC 9 – 15 V range
Outputs	CAN signal containing accumulator SOC
Functionality	Connector to deliver power and CAN signal to the board

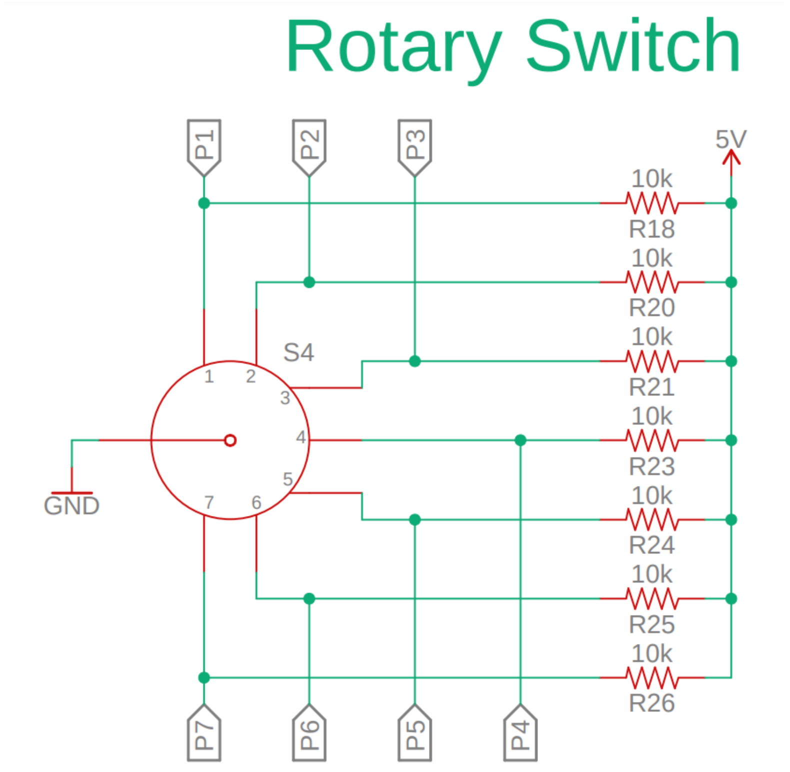


Figure 22: Dashboard PCB Schematic – Rotary Switch

Table 39: Dashboard PCB Schematic – Rotary Switch

Designator & P/N	S4 RM107772BCB
Inputs	User turning the knob
Outputs	High signals to desired microcontroller digital input
Functionality	Toggles between modes of operation and display screens on LCD

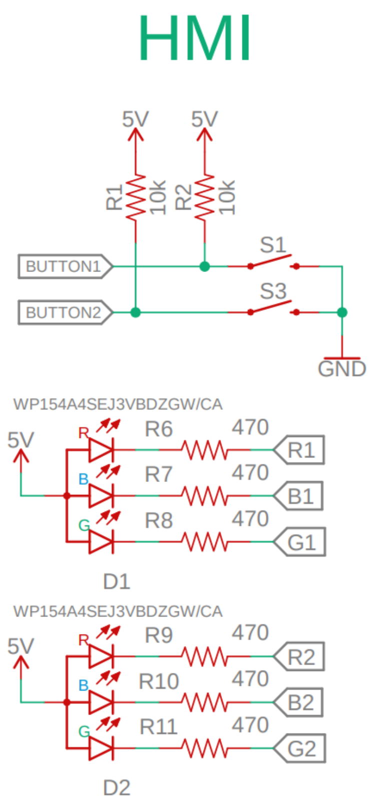


Figure 23: Dashboard PCB Schematic – Human Interface LEDs and Buttons

Table 40: Dashboard PCB Schematic – LEDs

Designator & P/N	D1, D2 WP154A4SEJ3VBDZGW/CA
Inputs	Microcontroller controls cathodes of LEDs
Outputs	Light emitted (red, green, or blue)
Functionality	Indicate various modes and statuses for driver

Table 41: Dashboard PCB Schematic – Buttons

Designator & P/N	S1, S3 PB6B2FM7M4CAL00
Inputs	User pressing button
Outputs	Signals from switches S1 and S3 to microcontroller
Functionality	Allows driver to control states of vehicle and start up car

5.2. Software Design

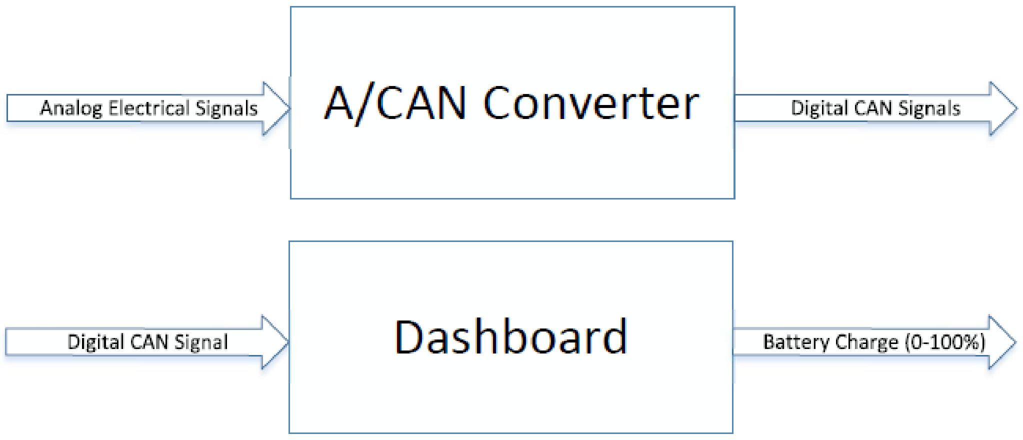


Figure 24: Software Level 0 Diagram

Table 42: A/D CAN Converter Functional Software Requirements - Level 0

Module	Analog-to-CAN Converter
Inputs	- Dual redundant accelerator pedal position sensors - Dual brake pressure transducers for front and rear hydraulic circuits
Outputs	- Direct connection to physical CAN bus - Converted CAN messages
Functionality	The A/CAN Converter will accept analog electrical signals from dual accelerator sensors and dual brake sensors and convert them into digital CAN signals. These signals can later be used by other components of the vehicle.

Table 43: Dashboard Display Functional Software Requirements - Level 0

Module	CAN-enabled Dashboard Display
Inputs	- CAN bus data (accumulator)
Outputs	- Display high-voltage battery's state of charge
Functionality	The dashboard will accept CAN data messages from the accumulator and display the remaining charge left on the battery.

	The charge will be displayed as a percentage between 0% and 99%.

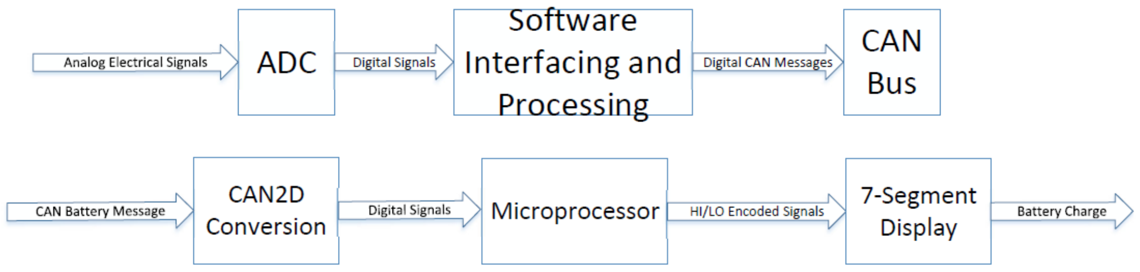


Figure 25: Software Level 1 Diagram

Table 44: A/D CAN Converter Functional Software Requirements - Level 1

Module	Analog-to-CAN Converter
Inputs	- Dual redundant accelerator pedal position sensors - Dual brake pressure transducers for front and rear hydraulic circuits
Outputs	- Direct connection to physical CAN bus - Converted CAN messages
Functionality	The A/CAN Converter will accept analog electrical signals from dual accelerator sensors and dual brake sensors and convert them into digital CAN signals. These signals can later be used by other components of the vehicle.

Table 45: Dashboard Display Functional Software Requirements - Level 1

Module	CAN-enabled Dashboard Display
Inputs	- CAN bus data (accumulator)
Outputs	- Display high-voltage battery's state of charge
Functionality	The dashboard will accept CAN data messages from the accumulator and display the remaining charge left on the battery. The charge will be displayed as a percentage between 0% and 99%.

Main MCU – Main Process Loop

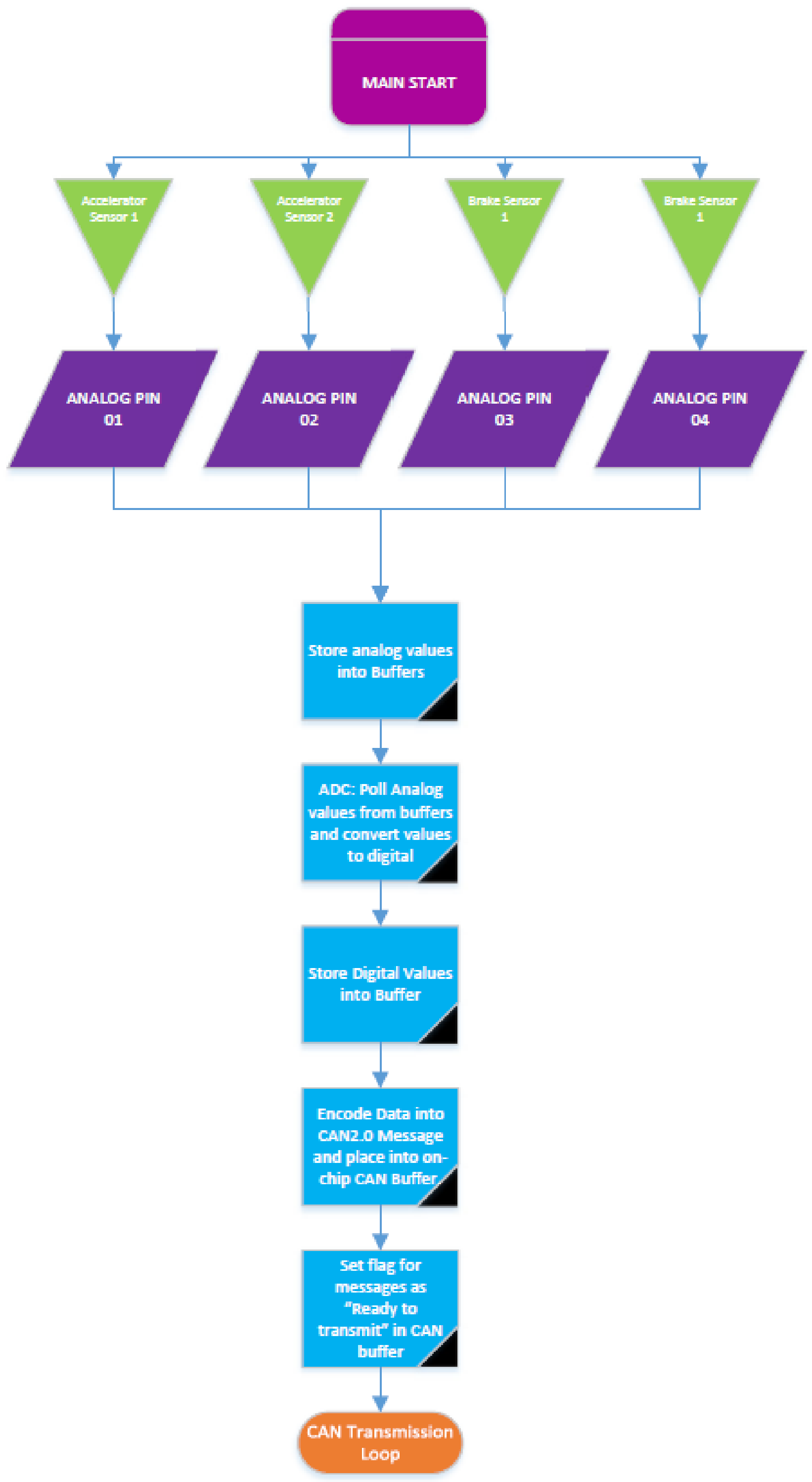

MAIN START
Accelerator Sensor 1
Accelerator Sensor 2
Brake Sensor 1
Brake Sensor 1
ANALOG PIN 01
ANALOG PIN 02
ANALOG PIN 03
ANALOG PIN 04
Store analog values into Buffers
ADC: Poll Analog values from buffers and convert values to digital
Store Digital Values into Buffer
Encode Data into CAN2.0 Message and place into on-chip CAN Buffer
Set flag for messages as "Ready to transmit" in CAN buffer
CAN Transmission Loop

Figure 26: Software Level 2 Diagram - Main Process Loop (RN)

Main MCU – CAN Transmission Loop

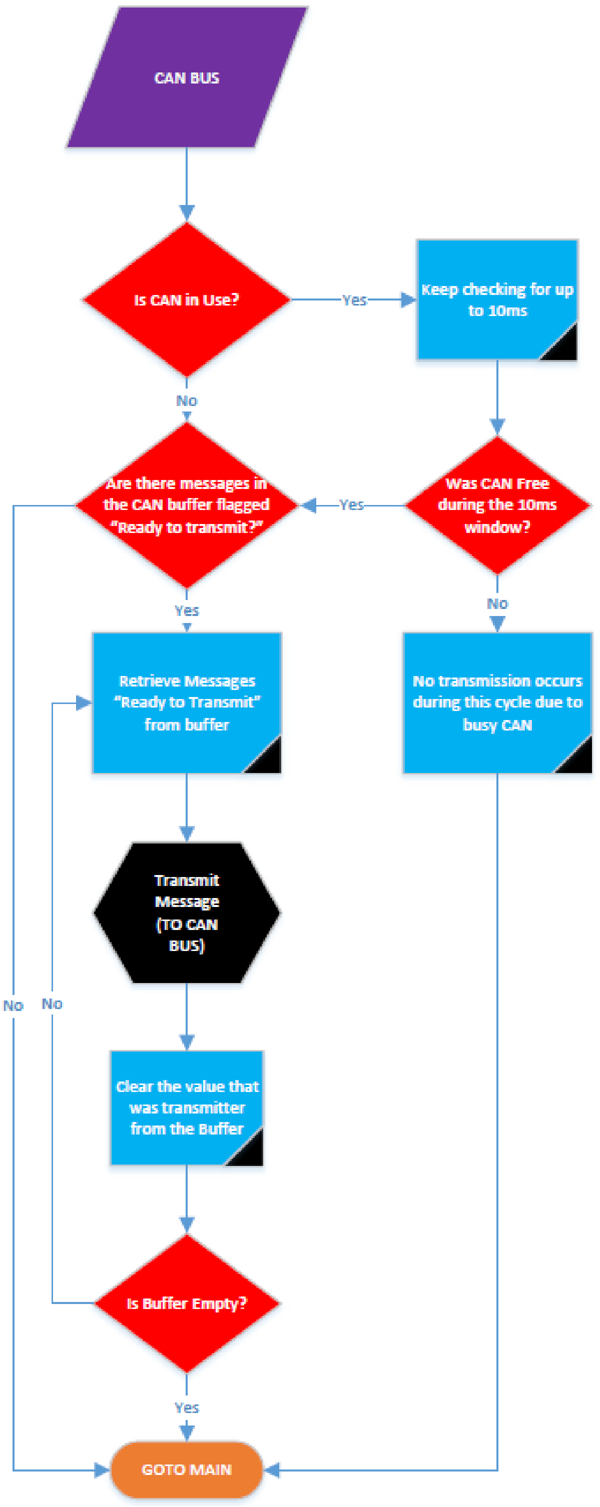

CAN BUS
Is CAN in Use?
Yes
Keep checking for up to 10ms
No
Are there messages in the CAN buffer flagged "Ready to transmit?"
Yes
Was CAN Free during the 10ms window?
Yes
No
Retrieve Messages "Ready to Transmit" from buffer
No transmission occurs during this cycle due to busy CAN
Transmit Message (TO CAN BUS)
No
No
Clear the value that was transmitter from the Buffer
Is Buffer Empty?
Yes
GOTO MAIN

Figure 27: Software Level 2 Diagram - CAN Transmission Loop (RN)

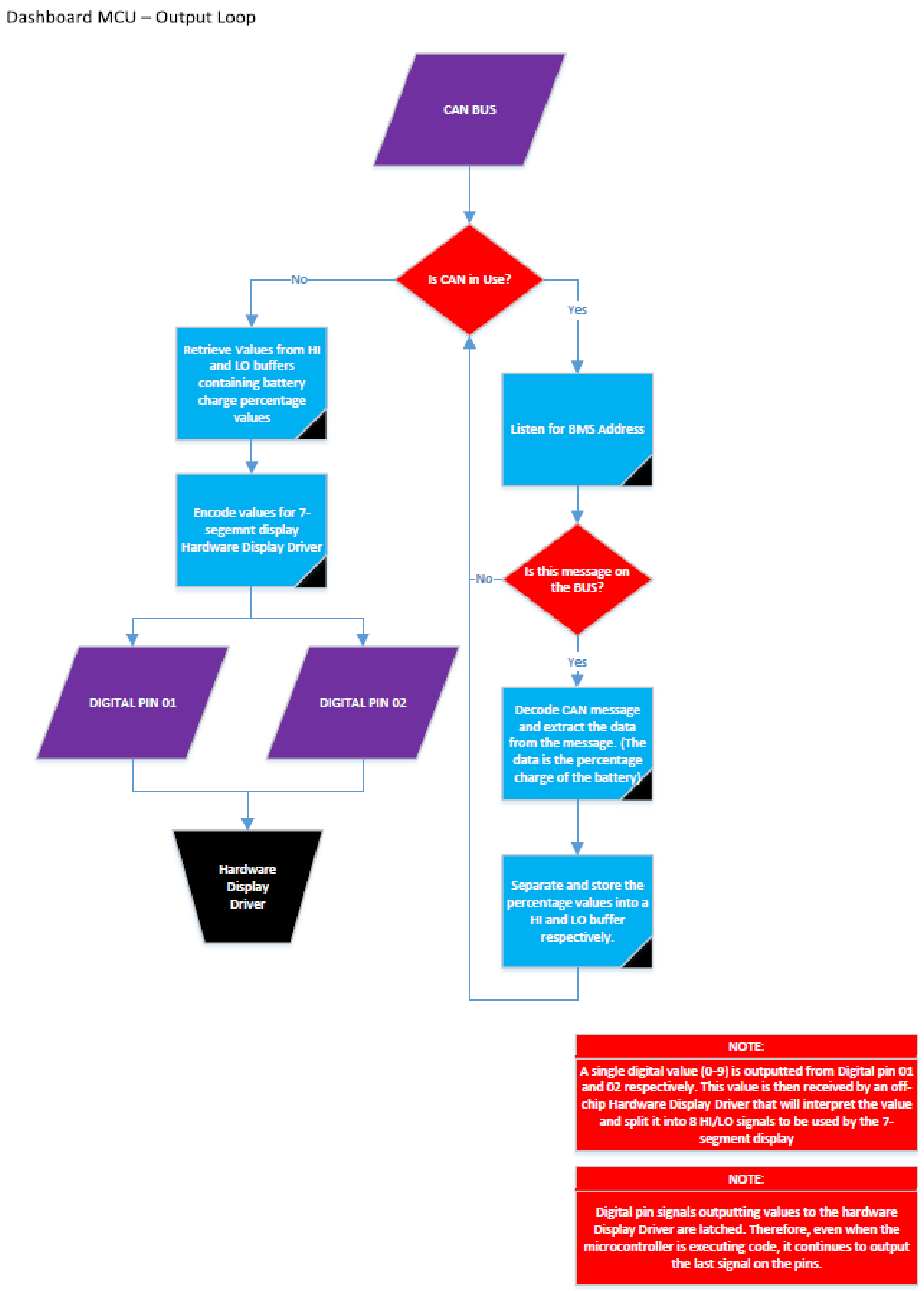

Dashboard MCU – Output Loop
CAN BUS
Is CAN in Use?
No
Yes
Retrieve Values from HI and LO buffers containing battery charge percentage values
Encode values for 7-segemnt display Hardware Display Driver
DIGITAL PIN 01
DIGITAL PIN 02
Hardware Display Driver
Listen for BMS Address
Is this message on the BUS?
No
Yes
Decode CAN message and extract the data from the message. (The data is the percentage charge of the battery)
Separate and store the percentage values into a HI and LO buffer respectively.
NOTE:
A single digital value (0-9) is outputted from Digital pin 01 and 02 respectively. This value is then received by an off-chip Hardware Display Driver that will interpret the value and split it into 8 HI/LO signals to be used by the 7-segment display
NOTE:
Digital pin signals outputting values to the hardware Display Driver are latched. Therefore, even when the microcontroller is executing code, it continues to output the last signal on the pins.

Figure 28: Software Level 2 Diagram - Dashboard MCU Output Loop (RN)

```
#define FCY 40000000UL
#include <libpic30.h>
#include "mcc_generated_files/system.h"
#include "mcc_generated_files/ecan1.h"
#include "mcc_generated_files/adc1.h"
#include "mcc_generated_files/can1.h"
#include "mcc_generated_files/pin_manager.h"

#define CH_APPS1 0x1A  // Accelerator sensor 1 ADC channel AN26
#define CH_APPS2 0x1F  // Accelerator sensor 2 ADC channel AN31
#define CH_BRK_F 0x0A  // Front brake sensor ADC channel AN10
#define CH_BRK_R 0x09  // Rear brake sensor ADC channel AN09

/************ INITIAL DEMONSTRATION MAIN ************/

int main(void)
{
// setup for arbitrary data
    bool Transmit_Success = false;  // bool to test for transmission success

    SYSTEM_Initialize();            // initialize dsPIC33 system
    CAN1_TransmitEnable();          // enable CAN transmission

    uCAN_MSG TestMessage;           // declare CAN message variable

// Transmit arbitrary message
    // pack CAN message with arbitrary data
    TestMessage.frame.dlc = 0x04;              // data length code (bytes)
    TestMessage.frame.idType = CAN1_FRAME_STD; // CAN1_FRAME_STD = 0x04 = 4
    TestMessage.frame.id = 0x0A;               // unique identifier
    TestMessage.frame.data0 = 0xFF;            // data
    TestMessage.frame.data1 = 0xFF;
    TestMessage.frame.data2 = 0xFF;
    TestMessage.frame.data3 = 0xFF;
    TestMessage.frame.data4 = 0xFF;
    TestMessage.frame.data5 = 0xFF;
    TestMessage.frame.data6 = 0xFF;
    TestMessage.frame.data7 = 0xFF;

// Test to see if CAN message transmission was successful
    while(1)
    {
       // transmit CAN message and assign status to bool
        Transmit_Success = CAN1_transmit(CAN_PRIORITY_HIGH, &TestMessage);
        if(Transmit_Success)
        {
            // if the transmission was successful, turn on LED2
            LED2_Toggle();      // XOR with value to determine if LED is on/off
        }
        __delay_ms(500);    // delay 500 ms = 0.5 s
    }
    return 1;
}
```

Figure 29: Basic CAN message data packing and transmission (main loop)

```
#ifndef _CAN1_H
#define _CAN1_H

#include "can_types.h"

#ifdef __cplusplus  // Provide C++ Compatibility

    extern "C" {

#endif

void CAN1_Initialize(void);

bool CAN1_receive(uCAN_MSG *recCanMsg);

bool CAN1_transmit(CAN_TX_PRIOIRTY priority,
                                   uCAN_MSG *sendCanMsg);

bool CAN1_isBusOff();

bool CAN1_isRXErrorPassive();

bool CAN1_isTXErrorPassive();

uint8_t CAN1_messagesInBuffer();

void CAN1_sleep();

void CAN1_TransmitEnable();

void CAN1_ReceiveEnable();

#ifdef __cplusplus  // Provide C++ Compatibility

    }

#endif

#endif  //_CAN1_H
```

Figure 30: can1.h Header File

```
#include "can1.h"
#include "ecan1.h"
#include "dma.h"

#define CAN1_TX_DMA_CHANNEL DMA_CHANNEL_0
#define CAN1_RX_DMA_CHANNEL DMA_CHANNEL_2

/* Valid options are 4, 6, 8, 12, 16, 24, or 32. */
#define CAN1_MESSAGE_BUFFERS         32

#define CAN1_TX_BUFFER_COUNT 1

typedef struct __attribute__((packed))
{
    unsigned priority                   :2;
    unsigned remote_transmit_enable     :1;
    unsigned send_request               :1;
    unsigned error                      :1;
    unsigned lost_arbitration           :1;
    unsigned message_aborted            :1;
    unsigned transmit_enabled           :1;
} CAN1_TX_CONTROLS;

static unsigned int can1msgBuf [CAN1_MESSAGE_BUFFERS][8] __attribute__((aligned(32 * 8 * 2)));

static void CAN1_DMACopy(uint8_t buffer_number, uCAN_MSG *message);
static void CAN1_MessageToBuffer(uint16_t* buffer, uCAN_MSG* message);

/* Null weak implementations of callback functions. */
void __attribute__((weak, deprecate("This callback ECAN1_CallbackBusOff() call will removed 
later"))) CAN1_CallbackBusOff(void)
{
    ECAN1_CallbackBusOff();
}

void __attribute__((weak, deprecate("This callback ECAN1_CallbackTxErrorPassive() call will 
removed later"))) CAN1_CallbackTxErrorPassive(void)
{
    ECAN1_CallbackTxErrorPassive();
}

void __attribute__((weak, deprecate("This callback ECAN1_CallbackRxErrorPassive() call will 
removed later"))) CAN1_CallbackRxErrorPassive(void)
{
    ECAN1_CallbackRxErrorPassive();
}

void __attribute__((weak, deprecate("This callback ECAN1_CallbackMessageReceived() call will 
removed later"))) CAN1_CallbackMessageReceived(void)
{
    ECAN1_CallbackMessageReceived();
}


void __attribute__((__interrupt__, no_auto_psv)) _C1Interrupt(void)
{

    if (C1INTFbits.ERRIF)
    {

        if (C1INTFbits.TXBO == 1)
        {
            CAN1_CallbackBusOff();
            C1INTFbits.TXBO = 0;
        }

        if (C1INTFbits.TXBP == 1)
        {
            CAN1_CallbackTxErrorPassive();
            C1INTFbits.TXBP = 0;
        }
        if (C1INTFbits.RXBP == 1)
        {
            CAN1_CallbackRxErrorPassive();
            C1INTFbits.RXBP = 0;
        }

        /* Call error notification function */
        C1INTFbits.ERRIF = 0;

    }

    if(C1INTFbits.RBIF)
    {
        C1INTFbits.RBIF = 0;

        /* Notification function */
        CAN1_CallbackMessageReceived();
    }

    if(C1INTFbits.WAKIF)
    {
        C1INTFbits.WAKIF = 0;
    }

    IFS2bits.C1IF = 0;
}

void CAN1_Initialize(void)
{
    // Disable interrupts before the Initialization
    IEC2bits.C1IE = 0;
    C1INTE = 0;

    // set the CAN{instance}_initialize module to the options selected in the User Interface

    /* put the module in configuration mode */
    C1CTRL1bits.REQOP = CAN_CONFIGURATION_MODE;
    while(C1CTRL1bits.OPMODE != CAN_CONFIGURATION_MODE);

    /* Set up the baud rate*/
    C1CFG1 = 0x03;     //BRP TQ = (2 x 4)/FCAN; SJW 1 x TQ;
    C1CFG2 = 0x41A8;   //WAKFIL enabled; SEG2PHTS Freely programmable; SEG2PH 2 x TQ; SEG1PH 6 x
TQ; PRSEG 1 x TQ; SAM Once at the sample point;
    C1FCTRL = 0xC001;  //FSA Transmit/Receive Buffer TRB1; DMABS 32;
    C1FEN1 = 0x01;     //FLTEN8 disabled; FLTEN7 disabled; FLTEN9 disabled; FLTEN0 enabled;
FLTEN2 disabled; FLTEN10 disabled; FLTEN1 disabled; FLTEN11 disabled; FLTEN4 disabled; FLTEN3
disabled; FLTEN6 disabled; FLTEN5 disabled; FLTEN12 disabled; FLTEN13 disabled; FLTEN14 disabled;
FLTEN15 disabled;
    C1CTRL1 = 0x00;    //CANCKS FOSC/2; CSIDL disabled; ABAT disabled; REQOP Sets Normal
Operation Mode; WIN Uses buffer window; CANCAP disabled;

    /* Filter configuration */
    /* enable window to access the filter configuration registers */
    /* use filter window*/
    C1CTRL1bits.WIN=1;

    /* select acceptance masks for filters */
    C1FMSKSEL1bits.F0MSK = 0x0; //Select Mask 0 for Filter 0

    /* Configure the masks */
    C1RXM0SIDbits.SID = 0x7ff;
    C1RXM1SIDbits.SID = 0x0;
    C1RXM2SIDbits.SID = 0x0;

    C1RXM0SIDbits.EID = 0x0;
    C1RXM1SIDbits.EID = 0x0;
    C1RXM2SIDbits.EID = 0x0;

    C1RXM0EID = 0x00;
    C1RXM1EID = 0x00;
    C1RXM2EID = 0x00;

    C1RXM0SIDbits.MIDE = 0x0;
    C1RXM1SIDbits.MIDE = 0x0;
    C1RXM2SIDbits.MIDE = 0x0;

    /* Configure the filters */
    C1RXF0SIDbits.SID = 0x123;

    C1RXF0SIDbits.EID = 0x0;

    C1RXF0EID = 0x00;

    C1RXF0SIDbits.EXIDE = 0x0;

    /* Non FIFO Mode */
    C1BUFPNT1bits.F0BP = 0x1; //Filter 0 uses Buffer1

    /* clear window bit to access ECAN control registers */
    C1CTRL1bits.WIN=0;

    /* CAN1, Buffer 0 is a Transmit Buffer */
    C1TR01CONbits.TXEN0 = 0x1; // Buffer 0 is a Transmit Buffer
    C1TR01CONbits.TXEN1 = 0x0; // Buffer 1 is a Receive Buffer
    C1TR23CONbits.TXEN2 = 0x0; // Buffer 2 is a Receive Buffer
    C1TR23CONbits.TXEN3 = 0x0; // Buffer 3 is a Receive Buffer
    C1TR45CONbits.TXEN4 = 0x0; // Buffer 4 is a Receive Buffer
    C1TR45CONbits.TXEN5 = 0x0; // Buffer 5 is a Receive Buffer
    C1TR67CONbits.TXEN6 = 0x0; // Buffer 6 is a Receive Buffer
    C1TR67CONbits.TXEN7 = 0x0; // Buffer 7 is a Receive Buffer

    C1TR01CONbits.TX0PRI = 0x0; // Message Buffer 0 Priority Level
    C1TR01CONbits.TX1PRI = 0x0; // Message Buffer 1 Priority Level
    C1TR23CONbits.TX2PRI = 0x0; // Message Buffer 2 Priority Level
    C1TR23CONbits.TX3PRI = 0x0; // Message Buffer 3 Priority Level
    C1TR45CONbits.TX4PRI = 0x0; // Message Buffer 4 Priority Level
    C1TR45CONbits.TX5PRI = 0x0; // Message Buffer 5 Priority Level
    C1TR67CONbits.TX6PRI = 0x0; // Message Buffer 6 Priority Level
    C1TR67CONbits.TX7PRI = 0x0; // Message Buffer 7 Priority Level

    /* clear the buffer and overflow flags */
    C1RXFUL1 = 0x0000;
    C1RXFUL2 = 0x0000;
    C1RXOVF1 = 0x0000;
    C1RXOVF2 = 0x0000;

    /* configure the device to interrupt on the receive buffer full flag */
    /* clear the buffer full flags */
    C1INTFbits.RBIF = 0;

    /* put the module in normal mode */
    C1CTRL1bits.REQOP = CAN_NORMAL_OPERATION_MODE;
    while(C1CTRL1bits.OPMODE != CAN_NORMAL_OPERATION_MODE);

    /* Enable CAN1 Interrupt */
    IEC2bits.C1IE = 1;

    /* Enable Receive interrupt */
    C1INTEbits.RBIE = 1;

    /* Enable Error interrupt*/
    C1INTEbits.ERRIE = 1;


}

void CAN1_TransmitEnable()
{
    /* setup channel 0 for peripheral indirect addressing mode
    normal operation, word operation and select as Tx to peripheral */
    /* DMA_PeripheralIrqNumberSet and DMA_TransferCountSet would be done in the
    DMA */

    /* setup the address of the peripheral CAN1 (C1TXD) */
    DMA_PeripheralAddressSet(CAN1_TX_DMA_CHANNEL, &C1TXD);

    /* DPSRAM start address offset value */
    DMA_StartAddressASet(CAN1_TX_DMA_CHANNEL, (uint16_t)(&can1msgBuf));

    /* enable the channel */
    DMA_ChannelEnable(CAN1_TX_DMA_CHANNEL);
}

void CAN1_ReceiveEnable()
{
    /* setup DMA channel for peripheral indirect addressing mode
    normal operation, word operation and select as Rx to peripheral */

    /* setup the address of the peripheral CAN1 (C1RXD) */
    /* DMA_TransferCountSet and DMA_PeripheralIrqNumberSet would be set in
    the DMA_Initialize function */

    DMA_PeripheralAddressSet(CAN1_RX_DMA_CHANNEL, &C1RXD);

    /* DPSRAM start address offset value */
    DMA_StartAddressASet(CAN1_RX_DMA_CHANNEL, (uint16_t)(&can1msgBuf) );

    /* enable the channel */
    DMA_ChannelEnable(CAN1_RX_DMA_CHANNEL);
}

bool CAN1_transmit(CAN_TX_PRIOIRTY priority, uCAN_MSG *sendCanMsg)
{
    CAN1_TX_CONTROLS * pTxControls = (CAN1_TX_CONTROLS*)&C1TR01CON;
    uint_fast8_t i;
    bool messageSent = false;

    if(CAN1_TX_BUFFER_COUNT > 0)
    {
        for(i=0; i<CAN1_TX_BUFFER_COUNT; i++)
        {
            if(pTxControls->transmit_enabled == 1)
            {
                if (pTxControls->send_request == 0)
                {
                    CAN1_MessageToBuffer( &can1msgBuf[i][0], sendCanMsg );

                    pTxControls->priority = priority;

                    /* set the message for transmission */
                    pTxControls->send_request = 1;

                    messageSent = true;
                    break;
                }
            }

            pTxControls++;
        }
    }
    return messageSent;
}

bool CAN1_receive(uCAN_MSG *recCanMsg)
{
    /* We use a static buffer counter so we don't always check buffer 0 first
     * resulting in potential starvation of later buffers.
     */
    static uint_fast8_t currentDedicatedBuffer = 0;
    uint_fast8_t i;
    bool messageReceived = false;
    uint16_t receptionFlags;

    receptionFlags = C1RXFUL1;

    if (receptionFlags != 0)
    {
        /* check which message buffer is free */
        for (i=0 ; i < 16; i++)
        {
            if (((receptionFlags >> currentDedicatedBuffer ) & 0x1) == 0x1)
            {
               CAN1_DMACopy(currentDedicatedBuffer, recCanMsg);

               C1RXFUL1 &= ~(1 << currentDedicatedBuffer);

               messageReceived = true;
            }

            currentDedicatedBuffer++;

            if(currentDedicatedBuffer >= 16)
            {
                currentDedicatedBuffer = 0;
            }

            if(messageReceived == true)
            {
                break;
            }
        }
    }

    return (messageReceived);
}

bool CAN1_isBusOff()
{
    return C1INTFbits.TXBO;
}

bool CAN1_isRXErrorPassive()
{
    return C1INTFbits.RXBP;
}

bool CAN1_isTXErrorPassive()
{
    return (C1INTFbits.TXBP);
}

uint8_t CAN1_messagesInBuffer()
{
    uint_fast8_t messageCount;
    uint_fast8_t currentBuffer;
    uint16_t receptionFlags;

    messageCount = 0;

    /* Check any message in buffer 0 to buffer 15*/
    receptionFlags = C1RXFUL1;
    if (receptionFlags != 0)
    {
        /* check whether a message is received */
        for (currentBuffer=0 ; currentBuffer < 16; currentBuffer++)
        {
            if (((receptionFlags >> currentBuffer ) & 0x1) == 0x1)
            {
                messageCount++;
            }
        }
    }
    return (messageCount);
}

void CAN1_sleep(void)
{
    C1INTFbits.WAKIF = 0;
    C1INTEbits.WAKIE = 1;

    /* put the module in disable mode */
    C1CTRL1bits.REQOP = CAN_DISABLE_MODE;
    while(C1CTRL1bits.OPMODE != CAN_DISABLE_MODE);

    //Wake up from sleep should set the CAN module straight into Normal mode
}

static void CAN1_DMACopy(uint8_t buffer_number, uCAN_MSG *message)
{
    uint16_t ide=0;
    uint16_t rtr=0;
    uint32_t id=0;

    /* read word 0 to see the message type */
    ide=can1msgBuf[buffer_number][0] & 0x0001U;

    /* check to see what type of message it is */
    /* message is standard identifier */
    if(ide==0U)
    {
        message->frame.id=(can1msgBuf[buffer_number][0] & 0x1FFCU) >> 2U;
        message->frame.idType = CAN_FRAME_STD;
        rtr=can1msgBuf[buffer_number][0] & 0x0002U;
    }
    /* message is extended identifier */
    else
    {
        id=can1msgBuf[buffer_number][0] & 0x1FFCU;
        message->frame.id = id << 16U;
        message->frame.id += ( ((uint32_t)can1msgBuf[buffer_number][1] & (uint32_t)0x0FFF) << 6U
);
        message->frame.id += ( ((uint32_t)can1msgBuf[buffer_number][2] & (uint32_t)0xFC00U) >>
10U );
        message->frame.idType = CAN_FRAME_EXT;
        rtr=can1msgBuf[buffer_number][2] & 0x0200;
    }
    /* check to see what type of message it is */
    /* RTR message */
    if(rtr != 0U)
    {
        /* to be defined ?*/
        message->frame.msgtype = CAN_MSG_RTR;
    }
    /* normal message */
    else
    {
        message->frame.msgtype = CAN_MSG_DATA;
        message->frame.data0 =(unsigned char)can1msgBuf[buffer_number][3];
        message->frame.data1 =(unsigned char)((can1msgBuf[buffer_number][3] & 0xFF00U) >> 8U);
        message->frame.data2 =(unsigned char)can1msgBuf[buffer_number][4];
        message->frame.data3 =(unsigned char)((can1msgBuf[buffer_number][4] & 0xFF00U) >> 8U);
        message->frame.data4 =(unsigned char)can1msgBuf[buffer_number][5];
        message->frame.data5 =(unsigned char)((can1msgBuf[buffer_number][5] & 0xFF00U) >> 8U);
        message->frame.data6 =(unsigned char)can1msgBuf[buffer_number][6];
        message->frame.data7 =(unsigned char)((can1msgBuf[buffer_number][6] & 0xFF00U) >> 8U);
        message->frame.dlc =(unsigned char)(can1msgBuf[buffer_number][2] & 0x000FU);
    }
}

static void CAN1_MessageToBuffer(uint16_t* buffer, uCAN_MSG* message)
{
    if(message->frame.idType == CAN_FRAME_STD)
    {
        buffer[0]= (message->frame.id & 0x000007FF) << 2;
        buffer[1]= 0;
        buffer[2]= message->frame.dlc & 0x0F;
    }
    else
    {
        buffer[0]= ( ( (uint16_t)(message->frame.id >> 16 ) & 0x1FFC ) ) | 0b1;
        buffer[1]= (uint16_t)(message->frame.id >> 6) & 0x0FFF;
        buffer[2]= (message->frame.dlc & 0x0F) + ( (uint16_t)(message->frame.id << 10) & 0xFC00);
    }

    buffer[3]= ((message->frame.data1)<<8) + message->frame.data0;
    buffer[4]= ((message->frame.data3)<<8) + message->frame.data2;
    buffer[5]= ((message->frame.data5)<<8) + message->frame.data4;
    buffer[6]= ((message->frame.data7)<<8) + message->frame.data6;
}
```

Figure 31: can1.c Source File

```
#ifndef _ECAN1_H
        #define _ECAN1_H

        #include "can_types.h"

        #warning deprecate ("\nThis will be removed in future MCC releases. \nUse can_types.h
file CAN message type identifiers instead. ")
        /* ECAN message type identifiers */
        #define CAN1_MSG_DATA    0x01
        #define CAN1_MSG_RTR     0x02
        #define CAN1_FRAME_EXT 0x03
        #define CAN1_FRAME_STD 0x04
        #define CAN1_BUF_FULL  0x05
        #define CAN1_BUF_EMPTY 0x06

        typedef union {
            struct {
                uint32_t id;
                uint8_t idType;
                uint8_t msgtype;
                uint8_t dlc;
                uint8_t data0;
                uint8_t data1;
                uint8_t data2;
                uint8_t data3;
                uint8_t data4;
                uint8_t data5;
                uint8_t data6;
                uint8_t data7;
            } frame;
            unsigned char array[16];
        } uCAN1_MSG __attribute__((deprecate ("\nThis will be removed in future MCC releases.
\nUse can_types.h file uCAN_MSG instead. ")));

        /* Operation modes */
        typedef enum
        {
                CAN1_NORMAL_OPERATION_MODE = 0,
                CAN1_DISABLE_MODE = 1,
                CAN1_LOOPBACK_MODE = 2,
                CAN1_LISTEN_ONLY_MODE = 3,
                CAN1_CONFIGURATION_MODE = 4,
                CAN1_LISTEN_ALL_MESSAGES_MODE = 7
        }ECAN1_OP_MODES __attribute__((deprecated ("\nThis will be removed in future MCC
releases. \nUse can_types.h file CAN_OP_MODES instead. ")));

        typedef enum{
            ECAN1_PRIORITY_HIGH = 0b11,
            ECAN1_PRIORITY_MEDIUM = 0b10,
            ECAN1_PRIORITY_LOW = 0b01,
            ECAN1_PRIORITY_NONE = 0b00
        } ECAN1_TX_PRIOIRTY __attribute__((deprecated ("\nThis will be removed in future MCC
releases. \nUse can_types.h file CAN_TX_PRIOIRTY instead. ")));

        #ifdef __cplusplus  // Provide C++ Compatibility

            extern "C" {

        #endif

        void ECAN1_Initialize(void) __attribute__((deprecate ("\nThis will be removed in future
MCC releases. \nUse CAN1_Initialize instead. ")));

        bool ECAN1_receive(uCAN1_MSG *recCanMsg) __attribute__((deprecate ("\nThis will be
removed in future MCC releases. \nUse CAN1_receive instead. ")));

        bool ECAN1_transmit(ECAN1_TX_PRIOIRTY priority,
                                            uCAN1_MSG *sendCanMsg) __attribute__((deprecate
("\nThis will be removed in future MCC releases. \nUse CAN1_transmit instead. ")));

        bool ECAN1_isBusOff() __attribute__((deprecate ("\nThis will be removed in future MCC
releases. \nUse CAN1_isBusOff instead. ")));

        bool ECAN1_isRXErrorPassive() __attribute__((deprecate ("\nThis will be removed in future
MCC releases. \nUse CAN1_isRXErrorPassive instead. ")));

        bool ECAN1_isTXErrorPassive() __attribute__((deprecate ("\nThis will be removed in future
MCC releases. \nUse CAN1_isTXErrorPassive instead. ")));

        uint8_t ECAN1_messagesInBuffer() __attribute__((deprecate ("\nThis will be removed in
future MCC releases. \nUse CAN1_messagesInBuffer instead. ")));

        void ECAN1_sleep() __attribute__((deprecate ("\nThis will be removed in future MCC
releases. \nUse CAN1_sleep instead. ")));

        void ECAN1_TransmitEnable() __attribute__((deprecate ("\nThis will be removed in future
MCC releases. \nUse CAN1_TransmitEnable instead. ")));

        void ECAN1_ReceiveEnable() __attribute__((deprecate ("\nThis will be removed in future
MCC releases. \nUse CAN1_ReceiveEnable instead. ")));

        /* Null weak implementations of callback functions. */
        void ECAN1_CallbackBusOff(void) __attribute__((deprecate("\nThis will be removed in
future MCC releases. \nUse CAN1_CallbackBusOff instead. ")));
        void ECAN1_CallbackTxErrorPassive(void) __attribute__((deprecate("\nThis will be removed
in future MCC releases. \nUse CAN1_CallbackTxErrorPassive instead. ")));
        void ECAN1_CallbackRxErrorPassive(void) __attribute__((deprecate("\nThis will be removed
in future MCC releases. \nUse CAN1_CallbackRxErrorPassive instead. ")));
        void ECAN1_CallbackMessageReceived(void) __attribute__((deprecate("\nThis will be removed
in future MCC releases. \nUse CAN1_CallbackMessageReceived instead. ")));

        #ifdef __cplusplus  // Provide C++ Compatibility

            }

        #endif

        #endif  //_ECAN1_H
```

Figure 32: ecan.h Header File

```
#include "ecan1.h"
#include "can1.h"
#include "dma.h"

/* Null weak implementations of callback functions. */
void __attribute__((weak, deprecate("\nThis will be removed in future MCC releases. \nUse
CAN1_CallbackBusOff instead. "))) ECAN1_CallbackBusOff(void){}
void __attribute__((weak, deprecate("\nThis will be removed in future MCC releases. \nUse
CAN1_CallbackTxErrorPassive instead. "))) ECAN1_CallbackTxErrorPassive(void){}
void __attribute__((weak, deprecate("\nThis will be removed in future MCC releases. \nUse
CAN1_CallbackRxErrorPassive instead. "))) ECAN1_CallbackRxErrorPassive(void){}
void __attribute__((weak, deprecate("\nThis will be removed in future MCC releases. \nUse
CAN1_CallbackMessageReceived instead. "))) ECAN1_CallbackMessageReceived(void){}

void ECAN1_Initialize(void)
{
    CAN1_Initialize();
}

void ECAN1_TransmitEnable()
{
    CAN1_TransmitEnable();
}

void ECAN1_ReceiveEnable()
{
    CAN1_ReceiveEnable();
}

bool ECAN1_transmit(ECAN1_TX_PRIOIRTY priority, uCAN1_MSG *sendCanMsg)
{
    return CAN1_transmit((CAN_TX_PRIOIRTY) priority, (uCAN_MSG *) sendCanMsg);
}

bool ECAN1_receive(uCAN1_MSG *recCanMsg)
{
    return CAN1_receive((uCAN_MSG *) recCanMsg);
}

bool ECAN1_isBusOff()
{
    return CAN1_isBusOff();
}

bool ECAN1_isRXErrorPassive()
{
    return CAN1_isRXErrorPassive();
}

bool ECAN1_isTXErrorPassive()
{
    return CAN1_isTXErrorPassive();
}

uint8_t ECAN1_messagesInBuffer()
{
    return CAN1_messagesInBuffer();
}

void ECAN1_sleep(void)
{
    CAN1_sleep();
}
```

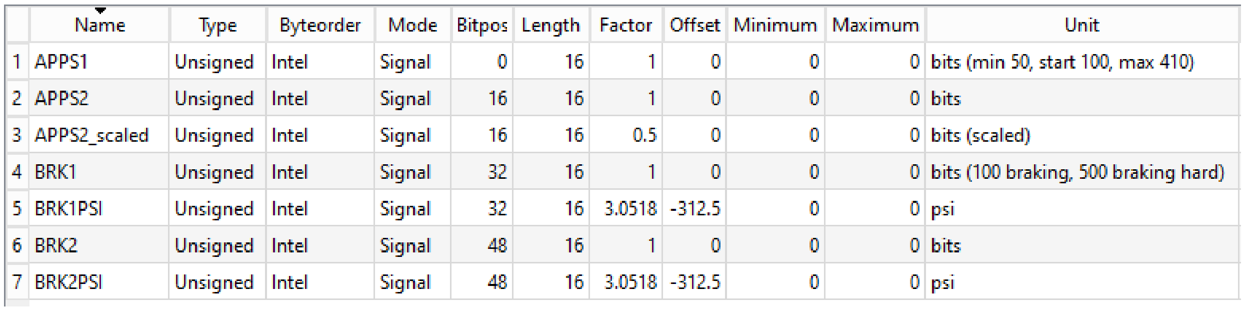
Figure 33: ecan.c Source File

6. Design Demonstration

Due to the unexpected closure of the university for the COVID-19 pandemic, the final demonstration of this project was cancelled. However, the team gave a midterm demonstration and showed that the engineering requirements had been met.

6.1. A/CAN Demonstration

For the A/CAN system, two of the primary requirements is that the board must convert analog signals into digital values. These values must be broadcast over the CAN bus. As the actual linear position sensors for the accelerator pedal and the brake pressure sensors had not yet been installed on the vehicle, the team verified this requirement by using two dual gang potentiometers to generate analog voltage levels. A CAN-USB converter allowed the received data to be displayed on a computer terminal. Using a CAN database file (*.dbc), the team could decode the messages in real time. Values were shown on the screen corresponding to the measured position of the potentiometers.



	Name	Type	Byteorder	Mode	Bitpos	Length	Factor	Offset	Minimum	Maximum	Unit
1	APPS1	Unsigned	Intel	Signal	0	16	1	0	0	0	bits (min 50, start 100, max 410)
2	APPS2	Unsigned	Intel	Signal	16	16	1	0	0	0	bits
3	APPS2_scaled	Unsigned	Intel	Signal	16	16	0.5	0	0	0	bits (scaled)
4	BRK1	Unsigned	Intel	Signal	32	16	1	0	0	0	bits (100 braking, 500 braking hard)
5	BRK1PSI	Unsigned	Intel	Signal	32	16	3.0518	-312.5	0	0	psi
6	BRK2	Unsigned	Intel	Signal	48	16	1	0	0	0	bits
7	BRK2PSI	Unsigned	Intel	Signal	48	16	3.0518	-312.5	0	0	psi

Figure 34: CAN database file editor for decoding ACAN messages

```
BO_ 5 ACAN: 8 ACAN
        SG_ APPS1 : 0|16@1+ (1,0) [0|0] "bits (min 50, start 100, max 410)"  ECU
        SG_ APPS2_scaled : 16|16@1+ (0.5,0) [0|0] "bits (scaled)"  ECU
        SG_ APPS2 : 16|16@1+ (1,0) [0|0] "bits"  ECU
        SG_ BRK1 : 32|16@1+ (1,0) [0|0] "bits (100 braking, 500 braking hard)"  ECU
        SG_ BRK2 : 48|16@1+ (1,0) [0|0] "bits"  ECU
        SG_ BRK1PSI : 32|16@1+ (3.0518,-312.5) [0|0] "psi"  ECU
        SG_ BRK2PSI : 48|16@1+ (3.0518,-312.5) [0|0] "psi"  ECU
```

Figure 35: CAN database file code for decoding ACAN messages

As many automotive sensors have analog outputs up to 5 V, the team demonstrated that the ACAN system could tolerate and read signals up to 5 V by turning the potentiometers fully in one direction and viewing the output on the computer monitor.

Another engineering requirement for the ACAN system is that open circuit in the sensor lines would be detected. The pedals are always preloaded by torsion springs and have mechanical stops before the end of the sensor range, so the sensors should never read below approximately 0.5 V, and any result lower than that threshold can be considered erroneous by the ECU. Pull down resistors of 1 MΩ bring the sensor output to zero volts in open circuit conditions, such as damage to a wire in the vehicle harness. The resistor value was selected larger enough not to significantly load the upstream RC lowpass filter. During the demonstration, the senior design coordinator cut one sense wire to the ACAN board and the subsequent CAN messages for that sensor reported 0 bits, as expected.

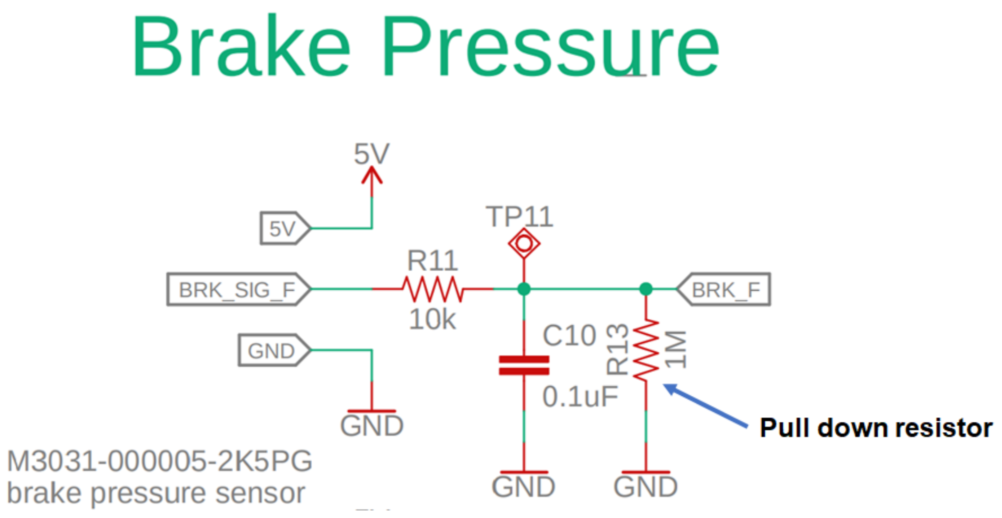


Figure 36: Location of sensor pull down resistor

FSAE rules require that both accelerator pedal position sensors must have unique transfer functions. This means for the same pedal position, the sensors report different outputs, such as having different output voltages. The team accomplished this by including a 1 kΩ resistor, R4, in the path for one of the sensors. For the same position of the dual gang potentiometer, the measured output voltages were different. The following figure shows a portion of the output recorded during the demonstration as the potentiometer shaft was rotated. The potentiometer used was 1 kΩ, so with R4, the value of APPS1 was expected to be half that of APPS2, as shown.

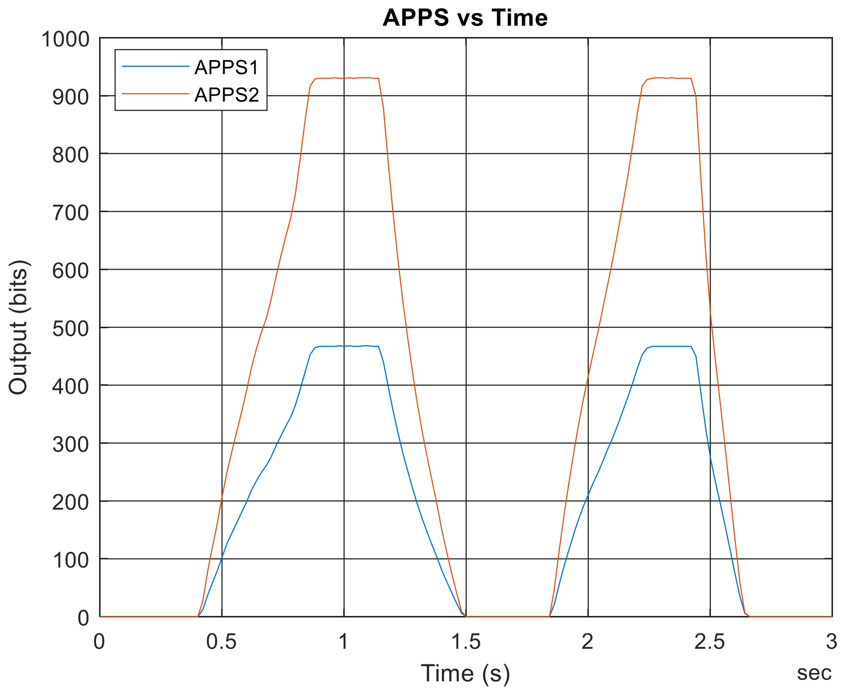


Figure 37: ACAN system output during demonstration

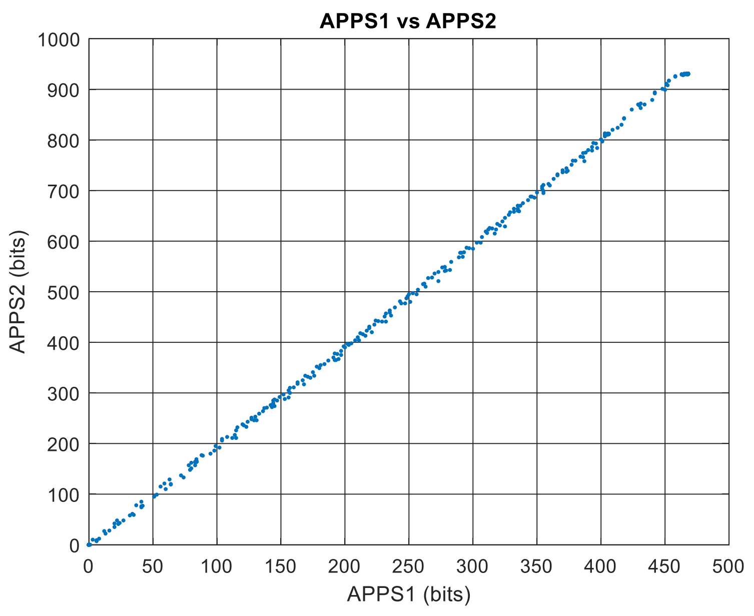


Figure 38: APPS1 and APPS2 reading plotted against each other

For quick response to driver action, the ACAN system must record and output data at a high rate. The requirement was that the system would output the reading of the sensors at a rate of at least 10 Hz. The team met this goal with an approximate rate of 50 Hz, depending on the CAN bus load due to message arbitration.

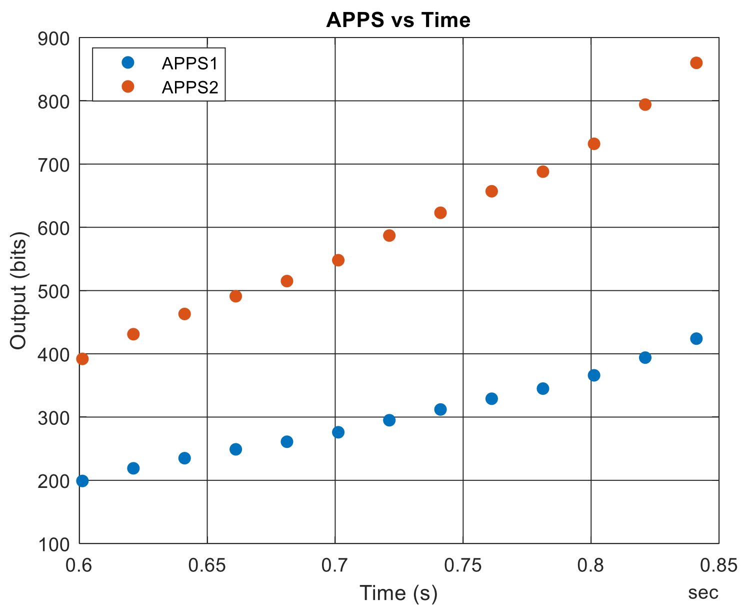


Figure 39: ACAN readings showing timing between messages

Table 46: Timing between ACAN system messages

Time (s)	APPS1 (bits)	APPS2 (bits)
0.601	199	392
0.621	219	431
0.641	235	463
0.661	249	491
0.681	261	515
0.701	276	548

The systems on the vehicle are powered by nominally 12 V LiFePO4 battery. This battery can vary over the course of a race from 15 V fully charged to about 9 V fully discharged. Therefore, one requirement of the system is to operate with any input voltage in this range. The board is powered with regulated 5 V from U2, R-78E5.0-1.0, a buck DC/DC converter with an input range 8 - 22 V. The team showed this requirement was met by powering the ACAN board from a lab power supply and adjusting the output voltage while the board was operating. CAN messages were still sent during the test and the message data was unaffected.

6.2. Dashboard Demonstration

The dashboard on the electric race car was designed to accept CAN messages from the BMS in about the accumulator SOC. The dashboard then decodes these messages and displays the information to the driver via a 7-segment display. As the BMS was not yet ready during the demonstration, the dashboard was programmed to accept the ID of the CAN message for the ACAN system. The dashboard then extracted and displayed the data for the first accelerator pedal position sensor.

In the following figure, the second two digits in the figure were configured to report linear speed of the vehicle. The LEDs above the display include two RGB LEDs controlled by the dashboard microcontroller. These include various states and statuses for the driver. The ECU sends various CAN messages telling the dashboard which LEDs and colors to illuminate.

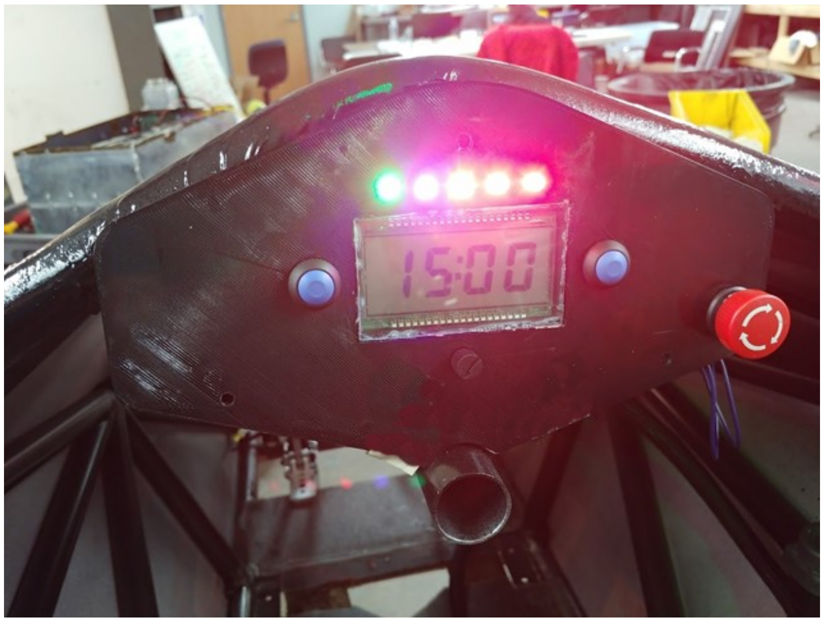


Figure 40: Dashboard configured to display ACAN result

The last requirement our team established for the dashboard is that the display must be legible from a distance of 3’ and in bright sunlight. During the demonstration, the team used a 4’ measuring ruler and a flashlight and showed that the display was easily readable from greater than 3’. Each character on the display is nearly ¾” tall and the VI-415-DP-RC-S liquid crystal display features a rear polarizer, reflecting incoming light, for clear viewing in bright light.

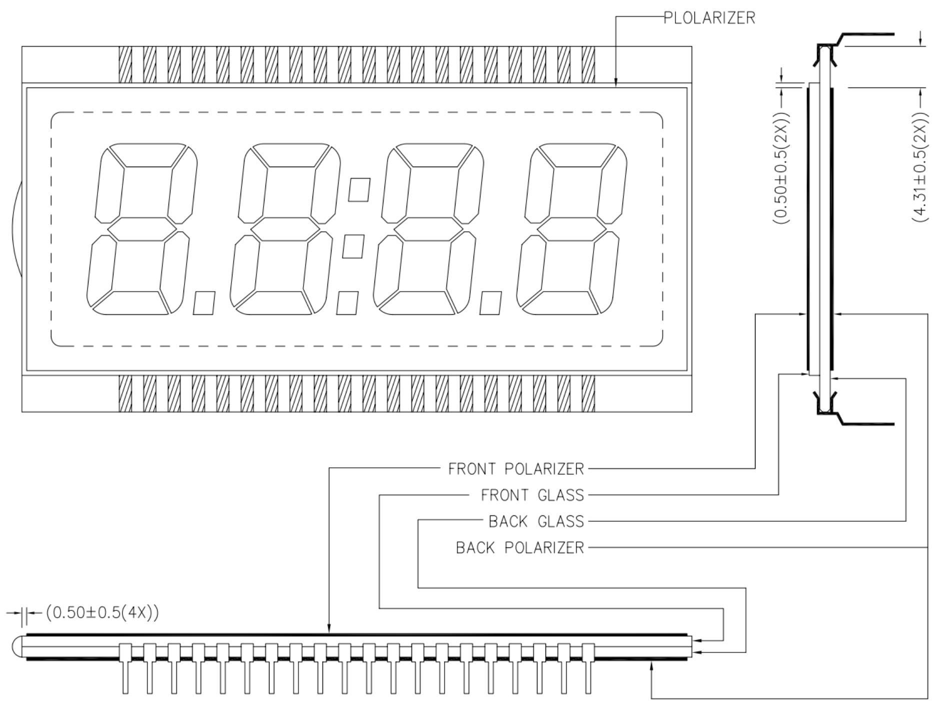


Figure 41: Dashboard display internal construction (dimensions are mm)

6.3. Current Consumption

Any battery-powered system has a limited amount of energy available. A requirement of for the project was that each PCB would draw less than 1 A of current at 12 V input. Both devices powered together during the demonstration required 49 mA, successfully meeting the goal.

7. Mechanical Sketch

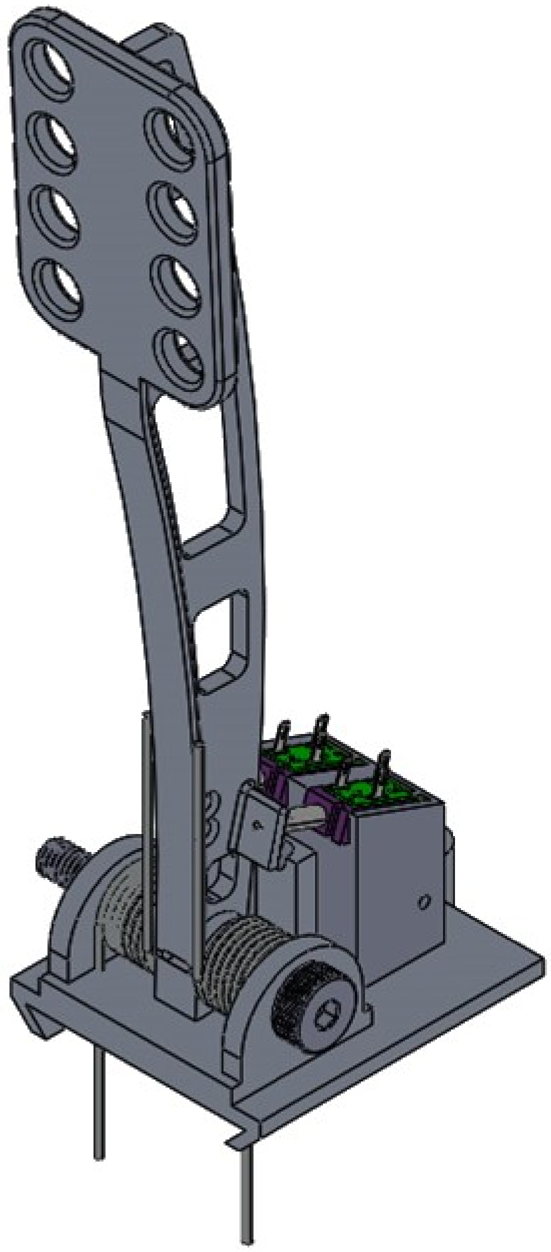


Figure 42: Accelerator pedal and dual sensor assembly

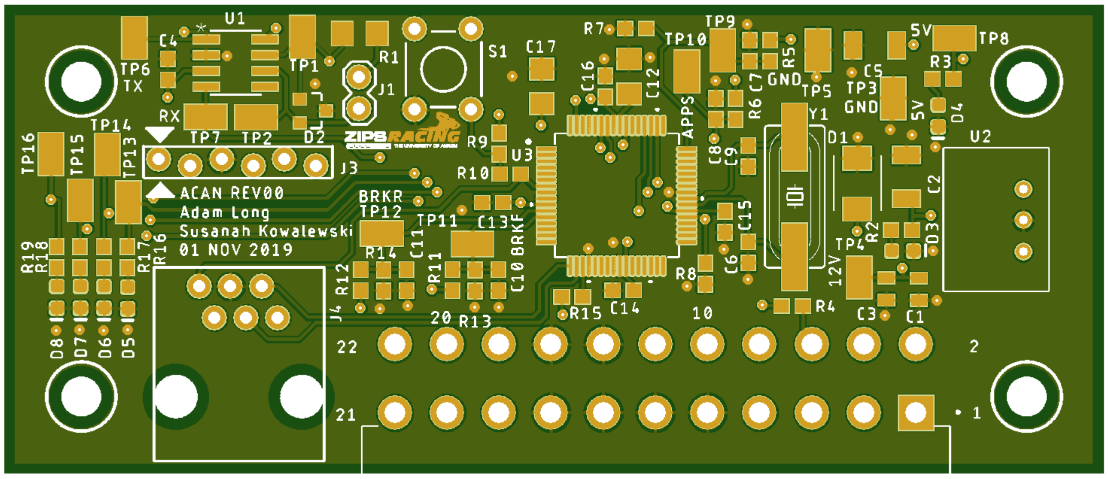


Figure 43: A/CAN converter system PCB sketch

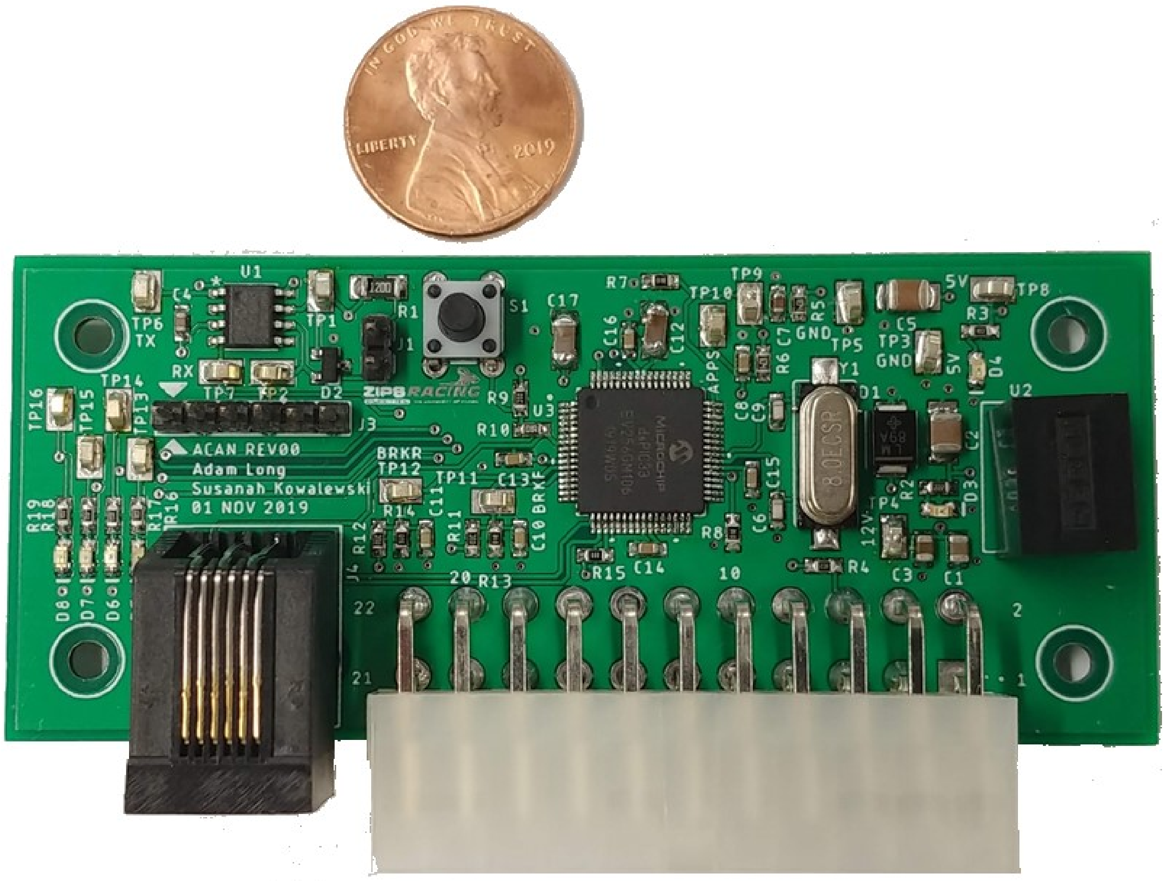


Figure 44: A/CAN converter PCB assembly

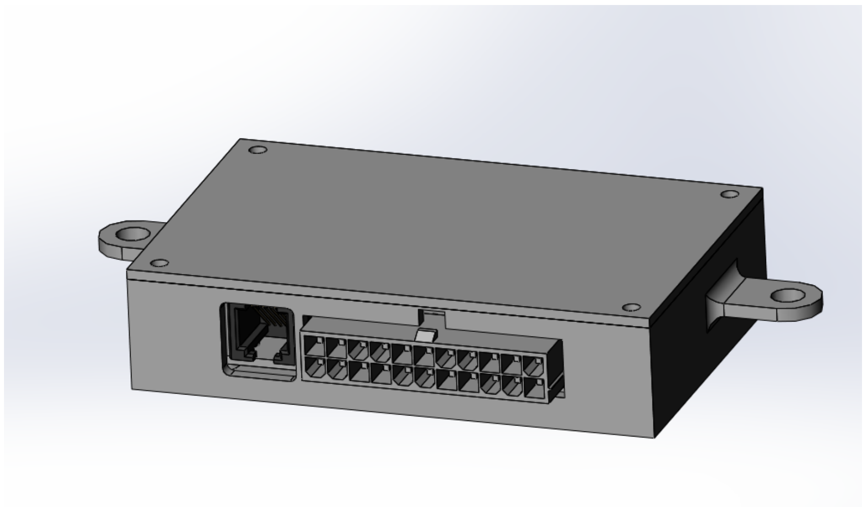


Figure 45: A/CAN converter CAD model in 3D-printed enclosure

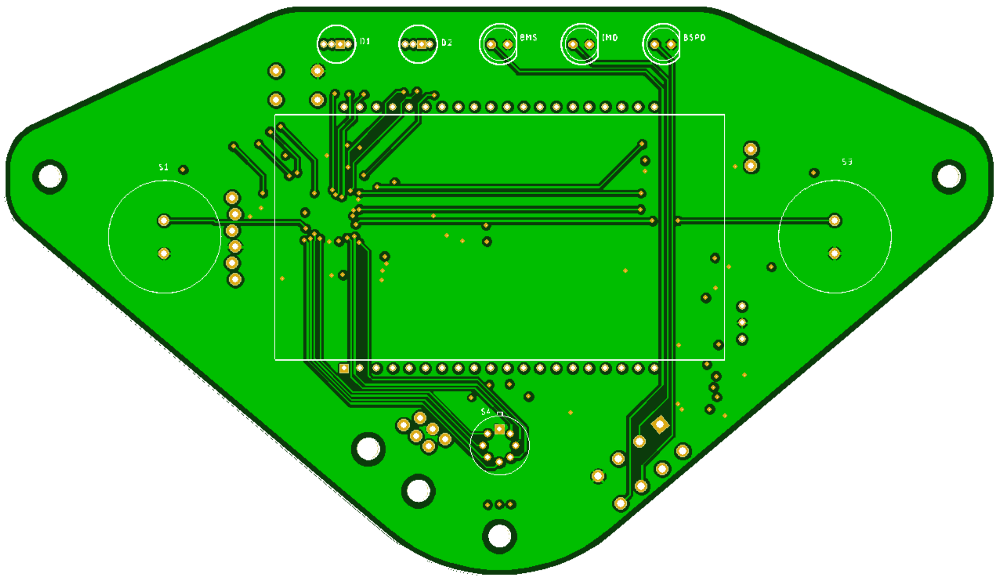


Figure 46: Dashboard PCB sketch

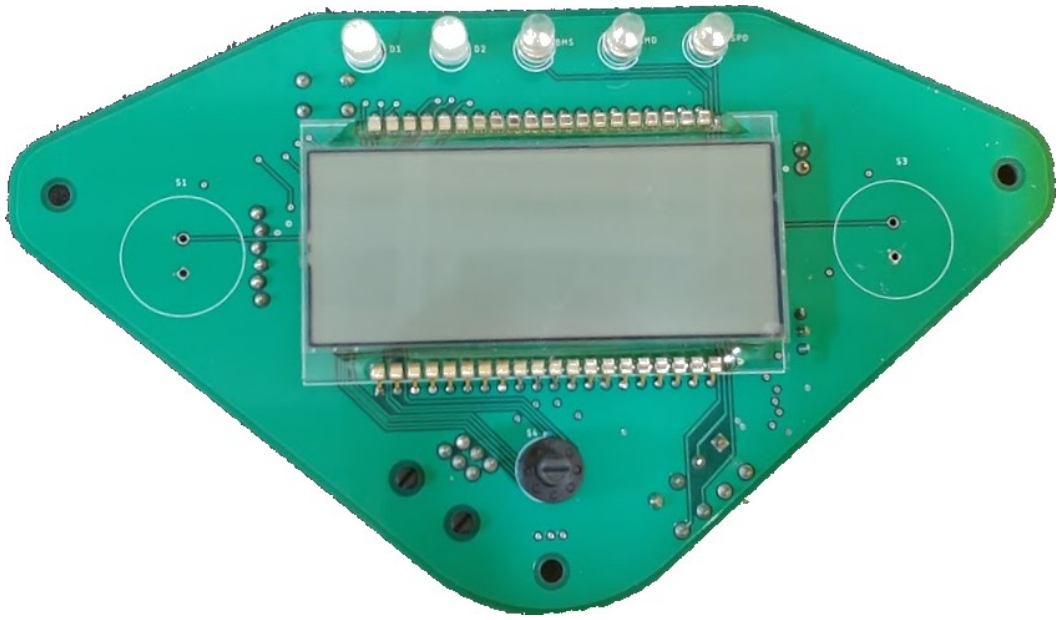


Figure 47: Dashboard PCB assembly, front side

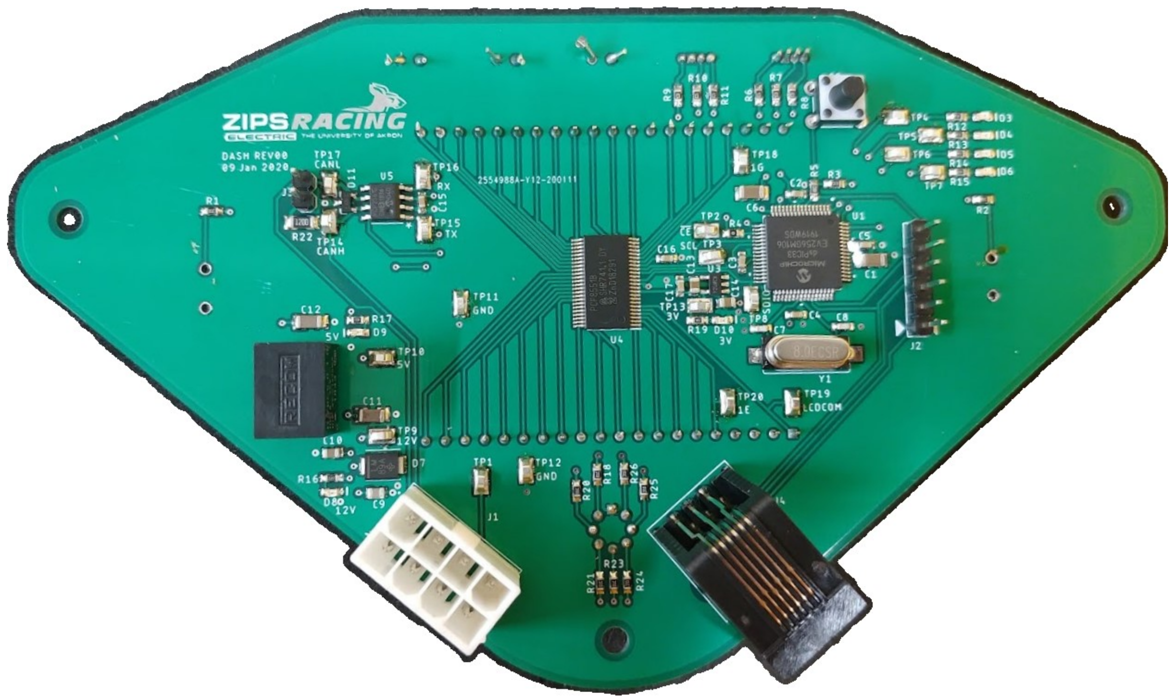


Figure 48: Dashboard PCB assembly, back side

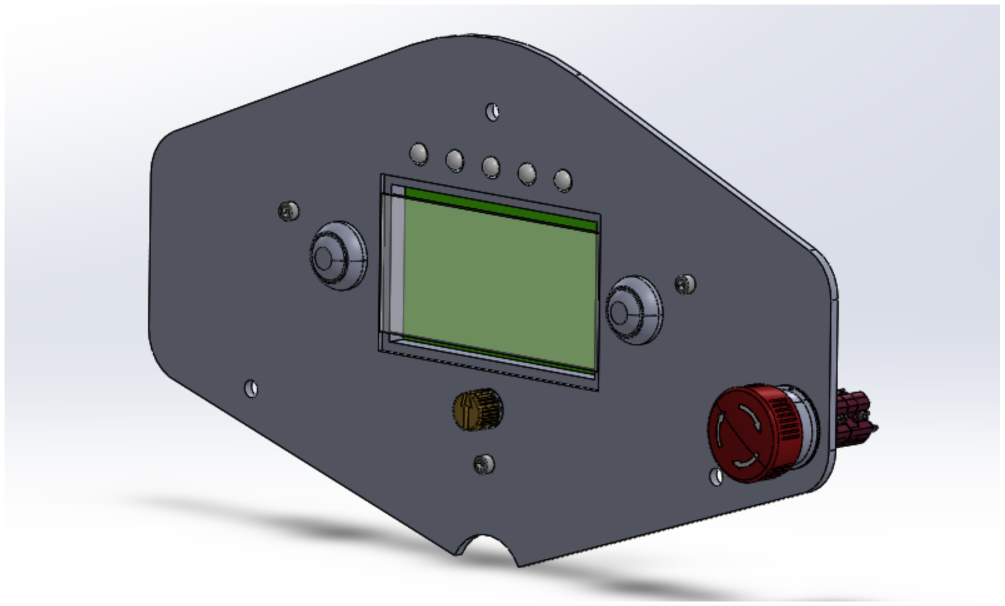


Figure 49: Dashboard CAD model mounted to 3D-printed cover

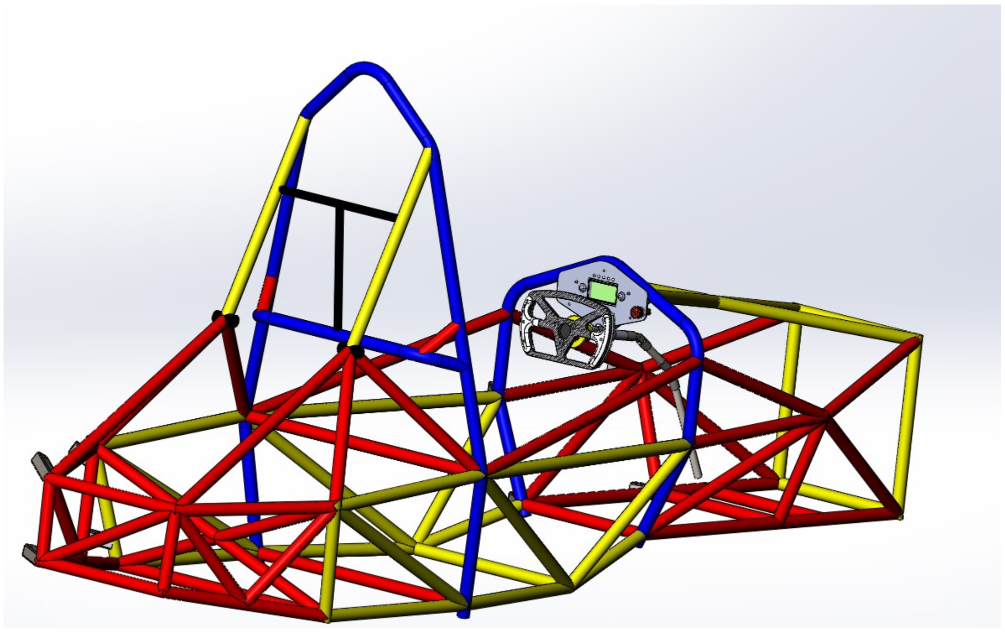


Figure 50: Dashboard CAD model installed on chassis

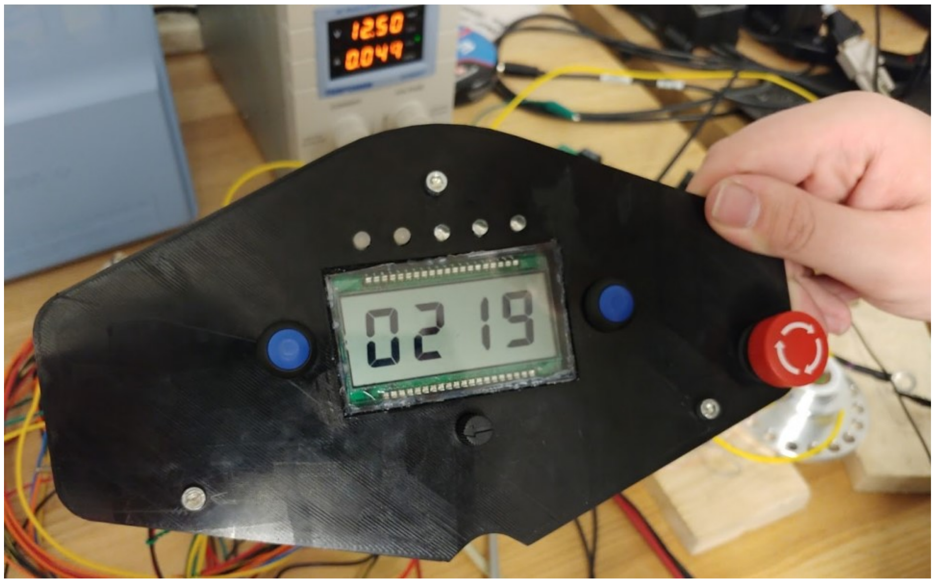


Figure 51: Dashboard powered up displaying test data

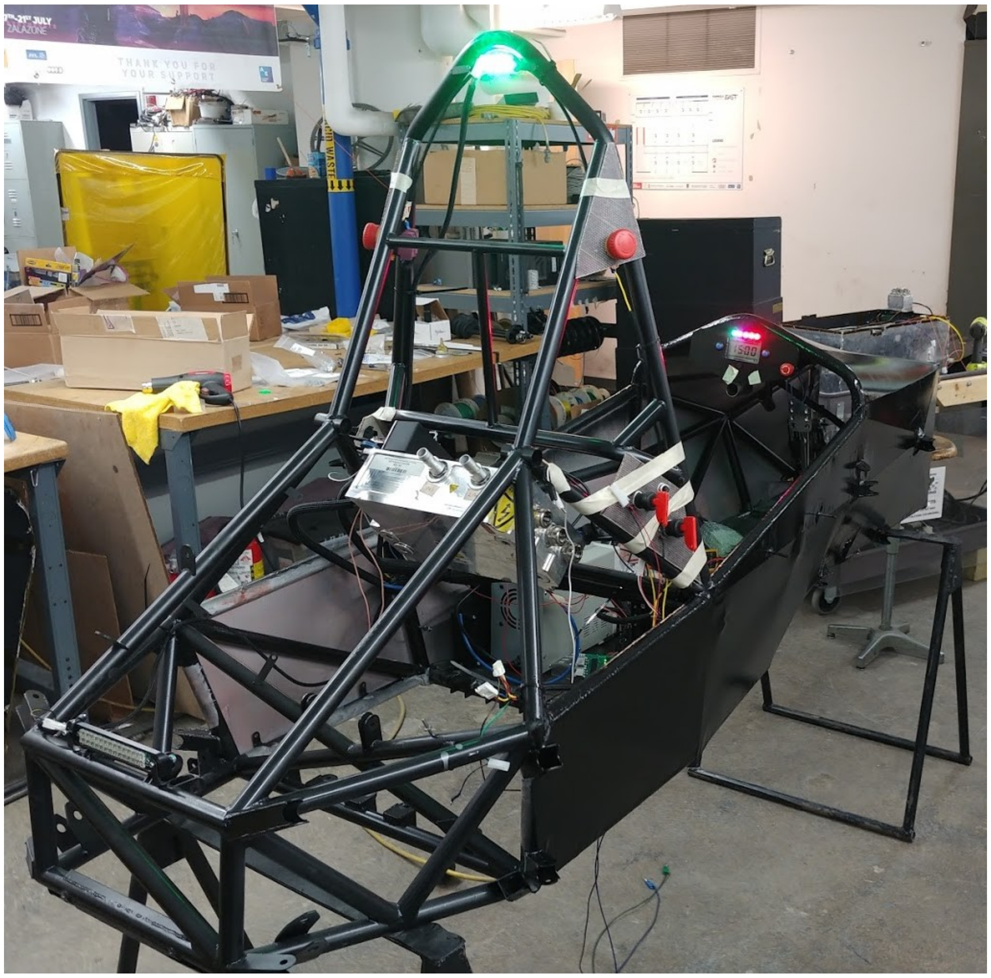


Figure 52: Car progress showing dashboard illuminated and other components installed

8. Team Information

Andrew Jordan, CpE, ESI, Archivist

Susanah Kowalewski, EE, ESI, Hardware Manager

Adam Long, EE, ESI, Team Manager

Rami Nehme, CpE, ESI, Software Manager

9. Parts List

9.1. A/CAN Bill of Materials

Table 47: A/CAN BOM

Manufacturer Part Number	Reference Designator	Qty	Description
CL21A226MOCLRNC	C1	1	CAP CER 22UF 16V X5R 0805
GRT31CR61H106ME01L	C12, C17	2	CAP CER 10UF 50V X5R 1206
CL32A106KATLNNE	C2, C5	2	CAP CER 10UF 25V X5R 1210
CL21B105KAFNNNE	C3	1	CAP CER 1UF 25V X7R 0805
CL10F104ZB8NNNC	C4, C7, C8, C10, C11, C13, C14, C15, C16	9	CAP CER 0.1UF 50V Y5V 0603
CC0603JRNPO9BN270	C6, C9	2	CAP CER 27PF 50V C0G/NPO 0603
SMBJ15A	D1	1	TVS DIODE 15V 24.4V DO214AA
PESD1CAN,215	D2	1	TVS DIODE 24V 70V SOT23
LTST-C190KGKT	D3, D4, D5, D8	4	LED GREEN CLEAR CHIP SMD
LTST-C190KRKT	D6	1	LED RED CLEAR CHIP SMD
LTST-C191KFKT	D7	1	LED ORANGE CLEAR SMD
M20-9990246	J1	1	CONN HEADER VERT 2POS 2.54MM
2-1586041-2	J2	1	CONN HEADER R/A 22POS 4.2MM
M20-9990646	J3	1	CONN HEADER VERT 6POS 2.54MM
A-2004-1-4-N-R	J4	1	CONN MOD JACK 6P6C R/A UNSHLD
RMCF1206FT120R	R1	1	RES 120 OHM 1% 1/4W 1206
RMCF0603FG100R	R10	1	RES 100 OHM 1% 1/10W 0603
CR0603-J/-000ELF	R15	1	RES SMD 0 OHM JUMPER 1/10W 0603
RMCF0603FT2K20	R2	1	RES 2.2K OHM 1% 1/10W 0603

RMCF0603FT1K00	R3, R4, R16, R17, R18, R19	6	RES 1K OHM 1% 1/10W 0603
RMCF0603FG100K	R5, R6	2	RES 100K OHM 1% 1/10W 0603
RMCF0603FT1M00	R7, R8, R13, R14	4	RES 1M OHM 1% 1/10W 0603
RMCF0603FT10K0	R9, R11, R12	3	RES 10K OHM 1% 1/10W 0603
1825910-6	S1	1	SWITCH TACTILE SPST-NO 0.05A 24V
5015	TP1, TP2, TP3, TP4, TP5, TP6, TP7, TP8, TP9, TP10, TP11, TP12, TP13, TP14, TP15, TP16	16	PC TEST POINT MINIATURE
MCP2561-E/SN	U1	1	IC TRANSCEIVER HALF 1/1 8SOIC
R-78E5.0-1.0	U2	1	DC DC CONVERTER 5V 5W
DSPIC33EV256GM106-I/PT	U3	1	IC MCU 16BIT 256KB FLASH 64TQFP
ECS-80-18-5P-TR	Y1	1	CRYSTAL 8.0000MHZ 18PF SMD

9.2. Dashboard Display Bill of Materials

Table 48: Dashboard BOM

Part Number	**Reference Designator**	**Qty**	**Description**
XLM2CRK20W	BMS, BSPD, IMD	3	LED RED CLEAR 5MM OVAL T/H
GRT31CR61H106ME01L	C1, C6, C11, C12	4	CAP CER 10UF 50V X5R 1206
CL21B105KAFNNNE	C10, C13, C14	3	CAP CER 1UF 25V X7R 0805
CL10F104ZB8NNNC	C2, C3, C4, C5, C15, C16, C17	7	CAP CER 0.1UF 50V Y5V 0603
CC0603JRNPO9BN270	C7, C8	2	CAP CER 27PF 50V C0G/NPO 0603
CL21A226MOCLRNC	C9	1	CAP CER 22UF 16V X5R 0805
WP154A4SEJ3VBDZGW/CA	D1, D2	2	LED RGB DIFFUSED T-1 3/4 T/H
PESD1CAN,215	D11	1	TVS DIODE 24V 70V SOT23
LTST-C190KGKT	D3, D8, D9, D10	4	LED GREEN CLEAR CHIP SMD
LTST-C190KRKT	D4	1	LED RED CLEAR CHIP SMD
LTST-C191KFKT	D5	1	LED ORANGE CLEAR SMD
LTST-C190TBKT	D6	1	LED BLUE CLEAR CHIP SMD

SMBJ15A	D7	1	TVS DIODE 15V 24.4V DO214AA
VI-415-DP-RC-S	DIS1	1	LCD MOD 4 DIG 4 X 1 REFLECTIVE
1586038-8	J1	1	CONN HEADER VERT 8POS 4.2MM
M20-9990646	J2	1	CONN HEADER VERT 6POS 2.54MM
M20-9990246	J3	1	CONN HEADER VERT 2POS 2.54MM
A-2004-1-4-N-R	J4	1	CONN MOD JACK 6P6C R/A UNSHLD
RMCF0603FT10K0	R1, R2, R3, R18, R20, R21, R23, R24, R25, R26	10	RES 10K OHM 1% 1/10W 0603
RMCF0603FT1K00	R12, R13, R14, R15, R17, R19	6	RES 1K OHM 1% 1/10W 0603
RMCF0603FT2K20	R16	1	RES 2.2K OHM 1% 1/10W 0603
RMCF1206FT120R	R22	1	RES 120 OHM 1% 1/4W 1206
CR0603-J/-000ELF	R4	1	RES SMD 0 OHM JUMPER 1/10W 0603
RMCF0603FG100R	R5	1	RES 100 OHM 1% 1/10W 0603
RMCF0603FT470R	R6, R7, R8, R9, R10, R11	6	RES 470 OHM 1% 1/10W 0603
PB6B2FM7M4CAL00	S1, S3	2	PB OFF/ON FC BLU M4 TERM. IP68
1825910-6	S2	1	SWITCH TACTILE SPST-NO 0.05A 24V
RM107772BCB	S4	1	SWITCH ROTARY 7POS 500MA 24V
5015	TP1, TP2, TP3, TP4, TP5, TP6, TP7, TP8, TP9, TP10, TP11, TP12, TP13, TP14, TP15, TP16, TP17, TP18, TP19, TP20	20	PC TEST POINT MINIATURE
DSPIC33EV256GM106-I/PT	U1	1	IC MCU 16BIT 256KB FLASH 64TQFP
R-78E5.0-1.0	U2	1	DC DC CONVERTER 5V 5W
ADP160AUJZ-3.0-R7	U3	1	IC REG LINEAR 3V 150MA TSOT5

PCF8551BTT/AJ	U4	1	IC DRVR 7 SEGMENT 48TSSOP
MCP2561-E/SN	U5	1	IC TRANSCEIVER HALF 1/1 8SOIC
ECS-80-18-5P-TR	Y1	1	CRYSTAL 8.0000MHZ 18PF SMD

10. Budget

The original budget of the team was $600. The items purchased by the team, from the team fund, and their associated costs are shown in the following table. The Zips Racing Electric team also had money available for the electrical system of the car, so some of the project items were purchased by the ZRE team. The team accomplished the project with $29.12 remaining in the design team fund, and stayed under the amount allowed from the ZRE team.

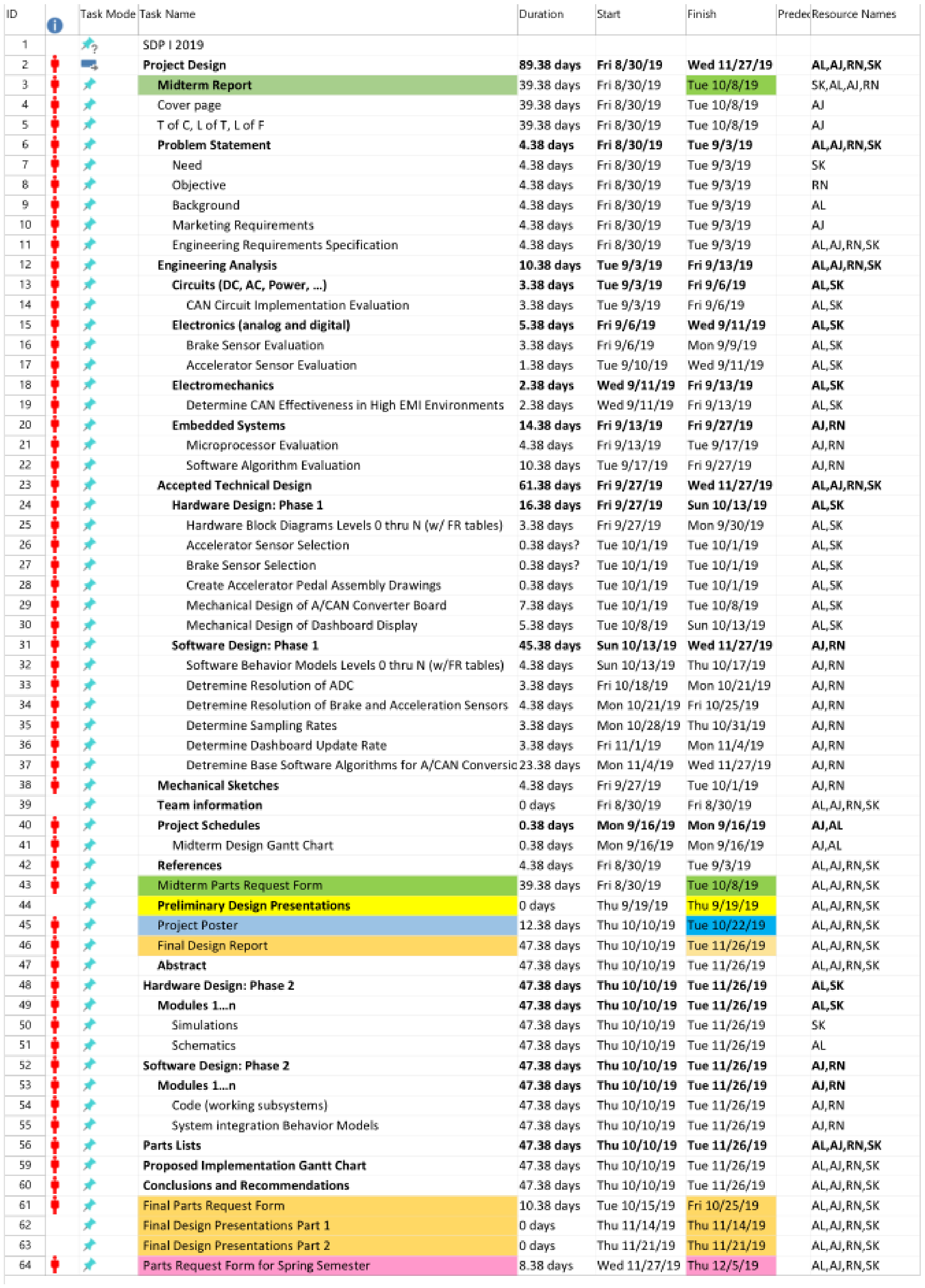
Table 49: Material Budget Information

Qty.	**Part Num.**	**Description**	**Unit Cost**	**Total Cost**
4	9605R1.7KL2.0	Resistive Sensor Linear Position Shaft Solder Lug	29.01	116.04
4	M3031-000005-2K5PG	Pressure Sensor 2500PSI (17236.89kPa) Vented Gauge Male - 1/4" (6.35mm) NPT 0.5 V ~ 4.5 V Cylinder	71.06	284.24
5	DSPIC33EV256GM106 -I/PT	dsPIC dsPIC™ 33EV Microcontroller IC 16-Bit 70 MIPs 256KB (85.5K x 24) FLASH 64-TQFP (10x10)	4.26	21.30
5	MCP2561-E/SN	1/1 Transceiver Half CANbus 8-SOIC	0.90	4.50
5	PESD1CAN,215	70V Clamp 3A (8/20µs) Ipp Tvs Diode Surface Mount TO-236AB	0.47	2.35
5	SMBJ15A-E3/52	24.4V Clamp 24.6A Ipp Tvs Diode Surface Mount DO-214AA (SMBJ)	0.35	1.75
5	ECS-80-18-5P-TR	8MHz ±30ppm Crystal 18pF 60 Ohms HC-49/US	0.77	3.85
4	MRJR-3360-01	Jack Modular Connector 6p6c (RJ11, RJ12, RJ14, RJ25) 90° Angle (Right) Unshielded	10.09	40.36
1	967-009-CAP	IP67 9 POS CONNECTOR COVER	8.96	8.96
1	160-000-209R002	DUST CAP 9POS FEMALE W/LANYARD	36.45	36.45

1	PN-1322-C	BOX PLSTC GRAY/CLR 4.53"LX2.56"W	2.13	2.13
1	G1301/8 OR007	SELF WRAP 1/8" X 50' ORANGE	18.19	18.19
1	PX0842/B	ADAPTER USB B RCPT TO USB A RCPT	12.00	12.00
1	1656356	INSERT 6P6C JACK COUPLER	6.43	6.43
1	PX0733	CONN SEALING COVER BLACK	10.14	10.14
3	956-009-010R011	CONN BACKSHELL 9POS 180DEG BLACK	0.73	2.19
			Total	**$570.88**

11. Project Schedule

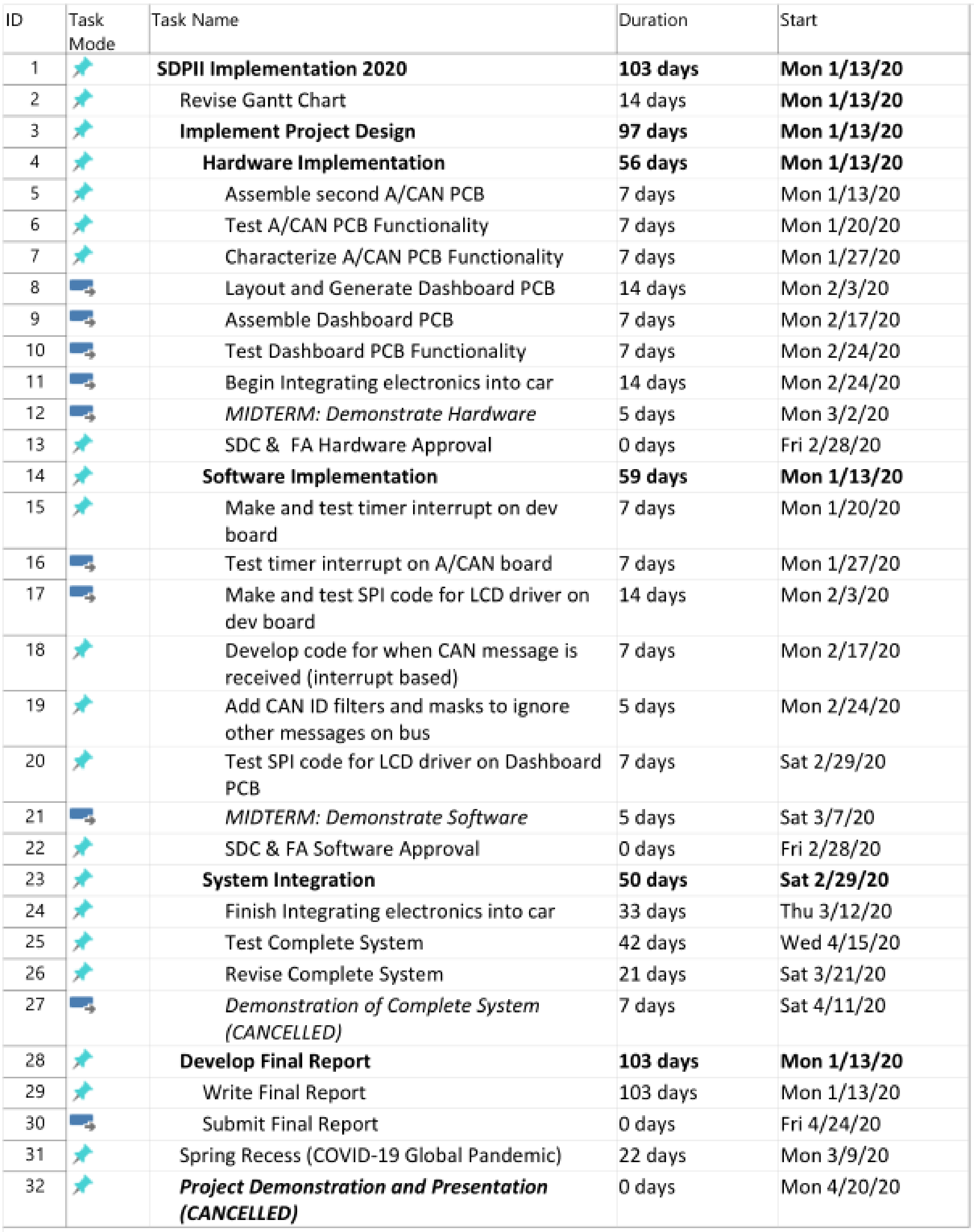
11.1. Fall 2019



ID	ℹ	Task Mode	Task Name	Duration	Start	Finish	Predec	Resource Names
1		📌?	SDP I 2019					
2	●	▬	Project Design	89.38 days	Fri 8/30/19	Wed 11/27/19		AL,AJ,RN,SK
3	●	📌	Midterm Report	39.38 days	Fri 8/30/19	Tue 10/8/19		SK,AL,AJ,RN
4	●	📌	Cover page	39.38 days	Fri 8/30/19	Tue 10/8/19		AJ
5	●	📌	T of C, L of T, L of F	39.38 days	Fri 8/30/19	Tue 10/8/19		AJ
6	●	📌	Problem Statement	4.38 days	Fri 8/30/19	Tue 9/3/19		AL,AJ,RN,SK
7	●	📌	Need	4.38 days	Fri 8/30/19	Tue 9/3/19		SK
8	●	📌	Objective	4.38 days	Fri 8/30/19	Tue 9/3/19		RN
9	●	📌	Background	4.38 days	Fri 8/30/19	Tue 9/3/19		AL
10	●	📌	Marketing Requirements	4.38 days	Fri 8/30/19	Tue 9/3/19		AJ
11	●	📌	Engineering Requirements Specification	4.38 days	Fri 8/30/19	Tue 9/3/19		AL,AJ,RN,SK
12	●	📌	Engineering Analysis	10.38 days	Tue 9/3/19	Fri 9/13/19		AL,AJ,RN,SK
13	●	📌	Circuits (DC, AC, Power, ...)	3.38 days	Tue 9/3/19	Fri 9/6/19		AL,SK
14	●	📌	CAN Circuit Implementation Evaluation	3.38 days	Tue 9/3/19	Fri 9/6/19		AL,SK
15	●	📌	Electronics (analog and digital)	5.38 days	Fri 9/6/19	Wed 9/11/19		AL,SK
16	●	📌	Brake Sensor Evaluation	3.38 days	Fri 9/6/19	Mon 9/9/19		AL,SK
17	●	📌	Accelerator Sensor Evaluation	1.38 days	Tue 9/10/19	Wed 9/11/19		AL,SK
18	●	📌	Electromechanics	2.38 days	Wed 9/11/19	Fri 9/13/19		AL,SK
19	●	📌	Determine CAN Effectiveness in High EMI Environments	2.38 days	Wed 9/11/19	Fri 9/13/19		AL,SK
20	●	📌	Embedded Systems	14.38 days	Fri 9/13/19	Fri 9/27/19		AJ,RN
21	●	📌	Microprocessor Evaluation	4.38 days	Fri 9/13/19	Tue 9/17/19		AJ,RN
22	●	📌	Software Algorithm Evaluation	10.38 days	Tue 9/17/19	Fri 9/27/19		AJ,RN
23	●	📌	Accepted Technical Design	61.38 days	Fri 9/27/19	Wed 11/27/19		AL,AJ,RN,SK
24	●	📌	Hardware Design: Phase 1	16.38 days	Fri 9/27/19	Sun 10/13/19		AL,SK
25	●	📌	Hardware Block Diagrams Levels 0 thru N (w/ FR tables)	3.38 days	Fri 9/27/19	Mon 9/30/19		AL,SK
26	●	📌	Accelerator Sensor Selection	0.38 days?	Tue 10/1/19	Tue 10/1/19		AL,SK
27	●	📌	Brake Sensor Selection	0.38 days?	Tue 10/1/19	Tue 10/1/19		AL,SK
28	●	📌	Create Accelerator Pedal Assembly Drawings	0.38 days	Tue 10/1/19	Tue 10/1/19		AL,SK
29	●	📌	Mechanical Design of A/CAN Converter Board	7.38 days	Tue 10/1/19	Tue 10/8/19		AL,SK
30	●	📌	Mechanical Design of Dashboard Display	5.38 days	Tue 10/8/19	Sun 10/13/19		AL,SK
31	●	📌	Software Design: Phase 1	45.38 days	Sun 10/13/19	Wed 11/27/19		AJ,RN
32	●	📌	Software Behavior Models Levels 0 thru N (w/FR tables)	4.38 days	Sun 10/13/19	Thu 10/17/19		AJ,RN
33	●	📌	Detremine Resolution of ADC	3.38 days	Fri 10/18/19	Mon 10/21/19		AJ,RN
34	●	📌	Detremine Resolution of Brake and Acceleration Sensors	4.38 days	Mon 10/21/19	Fri 10/25/19		AJ,RN
35	●	📌	Determine Sampling Rates	3.38 days	Mon 10/28/19	Thu 10/31/19		AJ,RN
36	●	📌	Determine Dashboard Update Rate	3.38 days	Fri 11/1/19	Mon 11/4/19		AJ,RN
37	●	📌	Detremine Base Software Algorithms for A/CAN Conversio	23.38 days	Mon 11/4/19	Wed 11/27/19		AJ,RN
38	●	📌	Mechanical Sketches	4.38 days	Fri 9/27/19	Tue 10/1/19		AJ,RN
39		📌	Team information	0 days	Fri 8/30/19	Fri 8/30/19		AL,AJ,RN,SK
40	●	📌	Project Schedules	0.38 days	Mon 9/16/19	Mon 9/16/19		AJ,AL
41	●	📌	Midterm Design Gantt Chart	0.38 days	Mon 9/16/19	Mon 9/16/19		AJ,AL
42	●	📌	References	4.38 days	Fri 8/30/19	Tue 9/3/19		AL,AJ,RN,SK
43	●	📌	Midterm Parts Request Form	39.38 days	Fri 8/30/19	Tue 10/8/19		AL,AJ,RN,SK
44		📌	Preliminary Design Presentations	0 days	Thu 9/19/19	Thu 9/19/19		AL,AJ,RN,SK
45	●	📌	Project Poster	12.38 days	Thu 10/10/19	Tue 10/22/19		AL,AJ,RN,SK
46	●	📌	Final Design Report	47.38 days	Thu 10/10/19	Tue 11/26/19		AL,AJ,RN,SK
47	●	📌	Abstract	47.38 days	Thu 10/10/19	Tue 11/26/19		AL,AJ,RN,SK
48	●	📌	Hardware Design: Phase 2	47.38 days	Thu 10/10/19	Tue 11/26/19		AL,SK
49	●	📌	Modules 1...n	47.38 days	Thu 10/10/19	Tue 11/26/19		AL,SK
50	●	📌	Simulations	47.38 days	Thu 10/10/19	Tue 11/26/19		SK
51	●	📌	Schematics	47.38 days	Thu 10/10/19	Tue 11/26/19		AL
52	●	📌	Software Design: Phase 2	47.38 days	Thu 10/10/19	Tue 11/26/19		AJ,RN
53	●	📌	Modules 1...n	47.38 days	Thu 10/10/19	Tue 11/26/19		AJ,RN
54	●	📌	Code (working subsystems)	47.38 days	Thu 10/10/19	Tue 11/26/19		AJ,RN
55	●	📌	System integration Behavior Models	47.38 days	Thu 10/10/19	Tue 11/26/19		AJ,RN
56	●	📌	Parts Lists	47.38 days	Thu 10/10/19	Tue 11/26/19		AL,AJ,RN,SK
59	●	📌	Proposed Implementation Gantt Chart	47.38 days	Thu 10/10/19	Tue 11/26/19		AL,AJ,RN,SK
60	●	📌	Conclusions and Recommendations	47.38 days	Thu 10/10/19	Tue 11/26/19		AL,AJ,RN,SK
61	●	📌	Final Parts Request Form	10.38 days	Tue 10/15/19	Fri 10/25/19		AL,AJ,RN,SK
62		📌	Final Design Presentations Part 1	0 days	Thu 11/14/19	Thu 11/14/19		AL,AJ,RN,SK
63		📌	Final Design Presentations Part 2	0 days	Thu 11/21/19	Thu 11/21/19		AL,AJ,RN,SK
64	●	📌	Parts Request Form for Spring Semester	8.38 days	Wed 11/27/19	Thu 12/5/19		AL,AJ,RN,SK

Figure 53: Fall 2019 Team Schedule

11.2. Spring 2020



ID	Task Mode	Task Name	Duration	Start
1	📌	**SDPII Implementation 2020**	**103 days**	**Mon 1/13/20**
2	📌	Revise Gantt Chart	14 days	**Mon 1/13/20**
3	📌	**Implement Project Design**	**97 days**	**Mon 1/13/20**
4	📌	**Hardware Implementation**	**56 days**	**Mon 1/13/20**
5	📌	Assemble second A/CAN PCB	7 days	Mon 1/13/20
6	📌	Test A/CAN PCB Functionality	7 days	Mon 1/20/20
7	📌	Characterize A/CAN PCB Functionality	7 days	Mon 1/27/20
8	➡	Layout and Generate Dashboard PCB	14 days	Mon 2/3/20
9	➡	Assemble Dashboard PCB	7 days	Mon 2/17/20
10	➡	Test Dashboard PCB Functionality	7 days	Mon 2/24/20
11	➡	Begin Integrating electronics into car	14 days	Mon 2/24/20
12	➡	*MIDTERM: Demonstrate Hardware*	5 days	Mon 3/2/20
13	📌	SDC & FA Hardware Approval	0 days	Fri 2/28/20
14	📌	**Software Implementation**	**59 days**	**Mon 1/13/20**
15	📌	Make and test timer interrupt on dev board	7 days	Mon 1/20/20
16	➡	Test timer interrupt on A/CAN board	7 days	Mon 1/27/20
17	➡	Make and test SPI code for LCD driver on dev board	14 days	Mon 2/3/20
18	📌	Develop code for when CAN message is received (interrupt based)	7 days	Mon 2/17/20
19	📌	Add CAN ID filters and masks to ignore other messages on bus	5 days	Mon 2/24/20
20	📌	Test SPI code for LCD driver on Dashboard PCB	7 days	Sat 2/29/20
21	➡	*MIDTERM: Demonstrate Software*	5 days	Sat 3/7/20
22	📌	SDC & FA Software Approval	0 days	Fri 2/28/20
23	📌	**System Integration**	**50 days**	**Sat 2/29/20**
24	📌	Finish Integrating electronics into car	33 days	Thu 3/12/20
25	📌	Test Complete System	42 days	Wed 4/15/20
26	📌	Revise Complete System	21 days	Sat 3/21/20
27	➡	*Demonstration of Complete System (CANCELLED)*	7 days	Sat 4/11/20
28	📌	**Develop Final Report**	**103 days**	**Mon 1/13/20**
29	📌	Write Final Report	103 days	Mon 1/13/20
30	➡	Submit Final Report	0 days	Fri 4/24/20
31	📌	Spring Recess (COVID-19 Global Pandemic)	22 days	Mon 3/9/20
32	📌	***Project Demonstration and Presentation (CANCELLED)***	0 days	Mon 4/20/20

Figure 54: Spring 2020 Team Schedule

12. Conclusions and Recommendations

With a CAN communication bus, the Zips Racing Electric team will be well equipped to perform at future competitions with a more reliable race car. With a digital communication system based on the Controller Area Network (CAN) bus, data can be more easily shared between independent electrical systems and each system can make decisions based upon the data. This data will be more reliable than last year due to the upgrade from analog communication, to noise resistant CAN communication. This will be a more robust system to accurately measure sensor data, which in turn will provide for smoother operation of the car. The system will also be replicable so that more sensors can be added to the CAN network, without hassle, in the future. All of this will improve car operation this year and help to set up Zips Racing Electric for success.

(AJ, SK, AL, RN)

13. References

13.1. Scholarly References

1. FSAEOnline.com. (2019). Retrieved July 21, 2019, from https://www.fsaeonline.com/
2. History of Formula SAE. (2019). Retrieved July 21, 2019, from https://www.fsaeonline.com/page.aspx?pageid=c4c5195a-60c0-46aa-acbf-2958ef545b72
3. Technical Specs. (2019). Retrieved July 23, 2019, from https://uakron.edu/engineering/beyond-the-classroom/zips-racing/tech-specs
4. Formula SAE® Awards & Results. (2019). Retrieved July 23, 2019, from https://www.sae.org/attend/student-events/formula-sae-michigan/awards-results
5. Horowitz, P., & Hill, W. (2015). The Art of Electronics. New York, NY, USA: Cambridge University Press.
6. Road vehicles - Controller area network (CAN) - High-speed medium access unit (Tech. No. ISO 11898-2:2016). (2016).

7. Kvaser (Ed.). (n.d.). CAN Protocol Tour. Retrieved July 28, 2019, from https://www.kvaser.com/can-protocol-tutorial/#/tab-1398106847865-4-6

8. Corrigan, S. (2016, May). Introduction to the Controller Area Network (CAN. Retrieved October 23, 2019, from http://www.ti.com/lit/an/sloa101b/sloa101b.pdf.

13.2. Patents

1. Hartwich, F. (2014). U.S. Patent No. EP2712123A1. European Patent Office. Standard CAN implementation tolerating CAN FD frames

2. Leyva, R. (1999). U.S. Patent No. US6587968B1. United States Patent Office. CAN bus termination circuits and CAN bus auto-termination methods

3. Thomson, T. (1993). U.S. Patent No. US5600782A. United States Patent Office. CAN interface with enhanced fault confinement

14. Appendices

14.1. Glossary

A/CAN – Analog to CAN
Accumulator – High voltage, lithium ion battery pack that provides tractive power to the vehicle
A/D – Analog to Digital
APPS – Accelerator Pedal Position Sensor
BMS – Battery Management System
CAD – Computer-Aided Design
CAN – Control Area Network
ECU – Electronic Control Unit
FSAE – Formula Society of Automotive Engineers
IDE – Integrated Development Environment
IDE bit – Identifier Extended bit, indicates whether CAN message will have standard (11 bit) or extended (29 bit) identifier
SOC – State of Charge
ZRE – Zips Racing Electric