2024 FSAE Electrical and Data Acquisition

James Keller

York College of Pennsylvania

8/8/2023

Contents

Introduction

As a computer engineering student, most of the work that falls within my skillset is in the intersection between data acquisition and the electrical subsystems of the car. Those of note that will be covered in this report being the data acquisition system, data visualization and dashboard design. Being one of two electrical majors on the team it is also my responsibility to be knowledgeable of the entire electrical system of the car. Thus, I must be able to troubleshoot issues that will inevitably arise and therefore will be discussing relevant portions of the electrical system as well. See Regan Ansel's report for more details regarding the electrical systems of the car.

Past Design

When approaching any engineering design problem, it is important to know about and comprehend existing solutions to said problem. Understanding and critiquing others' work is just as important as doing your own research as it will provide valuable insight into what to do and most importantly what not to do. Primarily, I wanted to dig through the 2023 team's tech reports. Examining the data acquisition system, it was broken into two parts. All engine sensors and wheel speed sensors were hooked into the engine control unit (ECU). The corresponding data was read out via controller area network bus (CAN) and was able to be viewed on a locally hosted website [1].

The other half of the system used a DAQ unit provided by DATA-Q (DI-2108) to record other sensors that were taken off before race day. Those being the shock potentiometers, steering angle sensors and accelerometer [2]. This setup is by far the easiest for overall data collection, however the team does not have the ability to view all sensors simultaneously and data from the DAQ can only be retroactively viewed.

As for the car's dashboard Zupko used a Neopixel stick and jewel to display tach, battery voltage, oil pressure and coolant temperature to the driver [3]. While cost-effective, this system carries no redundancies, meaning if the microcontrollers had any issues, then the operator would be driving blind [3].

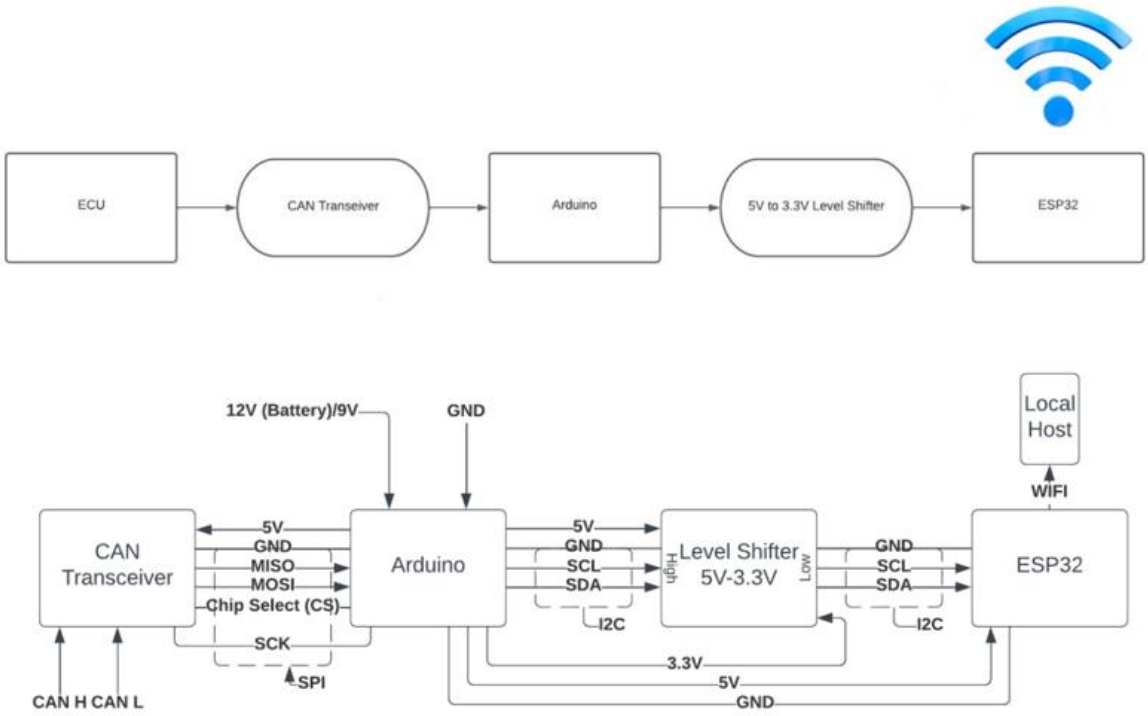


Figure 1: Dataflow of 2023 Team for Wireless transmission of Data off Car [1]

Design Goals and Specifications

The basis for the data acquisition system should be constructed around sensors chosen by the systems integration team and have the means to record all sensor readings. This system should not be concerned with what the data is, only that the data is properly recorded and passed along to some other end system. The end system in this case being some sort of data visualization software, preferably with live data viewing capabilities and the cars dashboard. The dashboard should only show indicators critical to the operational readiness of the vehicle and anything needed for the driver to operate the car effectively.

Critical criteria for final system:

- Value added data for the team is easily accessible
- All sensor data is retrievable from the same data path
- Have some level of redundancy
- Data contains appropriate level of resolution and is accurate
- Real time data viewing

The data collected will have a limited resolution dictated by the resolution per bit column found in figure 1. Due to the nature of how the CAN packets are formatted the data will be initially taken in as either an unsigned int, signed int or be left as raw Hexadecimal data. The data will only be converted to a float before calculations, before saving and storing said data or if the data is needed for dashboard indicators. This will prevent forcing less powerful MCUs to perform extra calculations than needed and put the load on more powerful boards. Also, this lowers the overall data throughput required as hex or integer data types are smaller than their decimal counterparts.

Table 1 below shows all the data that will be collected and which data it will be parsed as out of CAN. It also shows what the final data type will be when stored for data collection and visualization. Using table 1, the rate column in figure 2 and assuming a refresh rate of 50ms for the accelerometer we can calculate a final data rate required. The wireless communication will require a data rate at minimum of 6.5Kbps and a minimum of 4.6Kbps for CAN. These data low rates are easily achievable by todays MCUs and communication protocols. See appendix A5 for more detail on the calculations.

CAN ID (hex)	Name	Rate (ms)	Start Position	Length	Name	Units	Resolution per bit	Range	Type
0CFFF048	PE1	50	1-2	2 bytes	Rpm	rpm	1	0 to 30000	unsigned int
			3-4	2 bytes	TPS	%	0.1	0 to 100	signed int
			5-6	2 bytes	Fuel Open Time	ms	0.1	0 to 30	signed int
			7-8	2 bytes	Ignition Angle	deg	0.1	-20 to 100	signed int
0CFFF148	PE2	50	1-2	2 bytes	Barometer	psi or kpa	0.01	0-300	signed int
			3-4	2 bytes	MAP	psi or kpa	0.01	0-300	signed int
			5-6	2 bytes	Lambda	lambda	0.01	0-10	signed int
			7.1	1 bit	Pressure Type			0 - psi, 1-kPa	unsigned char
0CFFF248	PE3	100	1-2	2 bytes	Analog Input #1	volts	0.001	0 to 5	signed int
			3-4	2 bytes	Analog Input #2	volts	0.001	0 to 5	signed int
			5-6	2 bytes	Analog Input #3	volts	0.001	0 to 5	signed int
			7-8	2 bytes	Analog Input #4	volts	0.001	0 to 5	signed int
0CFFF348	PE4	100	1-2	2 bytes	Analog Input #5	volts	0.001	0 to 5	signed int
			3-4	2 bytes	Analog Input #6	volts	0.001	0 to 5	signed int
			5-6	2 bytes	Analog Input #7	volts	0.001	0 to 5	signed int
			7-8	2 bytes	Analog Input #8	volts	0.001	0 to 22	signed int
0CFFF448	PE5	100	1-2	2 bytes	Frequency 1	hz	0.2	0 to 6000	signed int
			3-4	2 bytes	Frequency 2	hz	0.2	0 to 6000	signed int
			5-6	2 bytes	Frequency 3	hz	0.2	0 to 6000	signed int
			7-8	2 bytes	Frequency 4	hz	0.2	0 to 6000	signed int
0CFFF548	PE6	1000	1-2	2 bytes	Battery Volt	volts	0.01	0 to 22	signed int
			3-4	2 bytes	Air Temp	C or F	0.1	-1000 to 1000	signed int
			5-6	2 bytes	Coolant Temp	C or F	0.1	-1000 to 1000	signed int
			7.1	1 bit	Temp Type			0 - F, 1 - C	unsigned char

Figure 2: CAN protocol reference full list in appendix A3 and A4 [8]

Table 1: Data Types Comparing Byte sizes for transferring Fata Through System

Data	Source of Collection	Parsed Before End System	Data Type When Communicating	Comm bytes	Final data Type	Final Bytes
RPM	PE3-8400A	Yes	Unsigned Int	2	Unsigned Int	2
Throttle Position	PE3-8400A	No	Hexadecimal	2	Float	4
Fuel Open Time	PE3-8400A	No	Hexadecimal	2	Float	4
Ignition Angle	PE3-8400A	No	Hexadecimal	2	Float	4
MAP	PE3-8400A	No	Hexadecimal	2	Float	4
Lambda	PE3-8400A	No	Hexadecimal	2	Float	4
Analog input 1 Oil Pressure	PE3-8400A	Yes	Signed Int	2	Float	4
*Analog input 2 - 7	PE3-8400A	No	Hexadecimal	12	Float	24
**Frequency 1 - 4	PE3-8400A	Yes	Signed Int	8	Float	16
Battery Voltage	PE3-8400A	Yes	Signed Int	2	Float	4
Coolant Temp	PE3-8400A	Yes	Signed Int	2	Float	4
G's X-Axis	ESP32	No	Float	4	Float	4
G's Y-Axis	ESP32	No	Float	4	Float	4
G's Z-Axis	ESP32	No	Float	4	Float	4

**Digital Wheel Speed Sensors
*Various 5V analog sensors (Shock Pots x4, O2 Sensor, Steering Angle)

While sensor specification should mostly be handled by the systems integration team, it's important to guide them in their decision making. Looking at table 2 all sensors should generally fall with the 0 – 5V output. This is to ensure compatibility with a wide range of DAQ systems including commonly used MCUs such as Arduinos. 5V power is also generally desired as it's commonly used from the ECU Pin M-16, the designated 5V sensor supply pin.

Lastly for data visualization the latency of the system should remain close to 2 seconds. Latency as in the time from first reading the CAN data to it being recorded or shown on an end system. This distinction is important as the CAN packets have a set rate that they are communicated via CAN bus. For example, battery voltage only updates at once a second at maximum (see appendix A3). This is to ensure that the data feels as close as possible to real time data so the team can actively monitor how the car is performing.

Table2: Car Auxiliary sensor specifications

Sensor	Measuring	Output Type	Input Voltage	Output Voltage
ADXL326	X, Y and Z acceleration	Analog	3.3 – 5V*	0 – 5V
Motec LPS 100	Shock Travel	Analog	0 – 5V	0 – 5V
Novotechnik SP2800	Steering Angle	Analog	0 – 5V	0 - 5V
Littlefuse 55075-02-00- A	Wheel Speed	Digital	5V**	N/A
***	Oil Pressure	Analog	0 – 5V	0 – 5V
***	O2	Analog	0 – 5V	0 – 5V

*ADXL326 has on board regulator for up to 5V input
** Requires high enough input voltage so ECU can properly differentiate between voltage logic levels
*** Not specifically listed in old tech reports but are hooked into 5V analog inputs on current car

Technical Research

To select components for the data acquisition system I first needed to familiarize myself more with the chosen ECU, CAN network technology and DI-2108 DAQ. Starting with CAN or controller area network. It is a bus standard commonly used in most vehicles on the road and in industrial applications [4]. It will allow collection of sensor data from the ECU at speeds up to 1 Mbit/s and has payloads of up to 8 bytes per frame [5]. Requiring only two wires, CAN high and CAN low, it will allow the broadcast of frames containing almost all the data the system will need to collect. However due to the nature of the bus topology requires a 120Ω resister to terminate both ends of the bus to eliminate signal reflections [4]. CAN carries many features such as error detection, automatic retransmission, remote data request and others that will not be used. Since our only concern is to sniff data off the bus, not send any messages, most of these features will never be leveraged.

Examining the ECU of choice, the PE3-8400A has serval nice features for data collection. Looking at appendix figures X and X the ECU has a total of eight auxiliary analog inputs and six digital inputs. The inputs allow for a range of at least 0 – 5V input. The ECU also has a dedicated PWM output for a tachometer on pin M-17. Most notably all auxiliary sensors and sensors required for the ECU to manage engine parameters can be read via its CAN bus output. Looking in the appendix (A3) provided by PE3 shows all CAN frame IDs and other pertinent info for the ECUs CAN bus. More specific settings for the CAN bus can be found in the ECUs manual [6].

Looking at the DI-2108 a pc based DAQ that can record up to 8 channels simultaneously [7]. The DI-2108 has 8 analog inputs, 3 configurable digital inputs and 4 digital utility digital inputs for other functions [7]. It also sports up to a 160,000 Hz sampling rate which is more than sufficient for any sensor on the car. While the 2023 used the DI-2108 for retroactive data

viewing by having the DAQ record data to a thumb drive as CSV files [2]. It is possible to connect a USB host device and send ASCII commands to the DI-2108 to stream data to the host [8]. This avenue of data collection will be explored in the iterative design section.

Lastly, examining the latency of the system requires some research as it's nearly impossible to calculate without running and testing the final system. The total latency of the system can be calculated:

$$System\ Latecny = Max\ CAN\ packet\ Rate + MCU\ Code\ time + LoRa\ Latecny$$

The CAN packet latency is unavoidable which is a max of 1 second (appendix A3) along with LoRa latency which is a function of distance [9]. The MCU processing time will be directly related to the complexity of the code and just how efficient it is written. Therefore, once the system is coded it must be tested for end-to-end latency to calculate and or improve overall latency for the system.

Iterative Design

For the first rounds of design, I took heavy inspiration from the 2023 team. The goal was to incorporate the idea of wireless communication of data off the car and further improve data collection capacity. To accomplish this, I moved to LoRa communication instead of Wi-Fi and attempted to incorporate the DI-2108 with serial communication. LoRa is a long-range communication protocol that is typically used in IoT applications. It is suitable for smaller data streams under 50 Kbps and runs on a 915 MHz bandwidth here in North America. To send LoRa data off the car the T-beam ESP32 an ESP32 development board with LoRa was chosen. It was

recommended by other students that have extensively used it at YCP Hackathons. The ESP32 will also serve as the control board for setting dash indicators such as the NeoPixel light bar.

To read CAN data and communicate with the DAQ a combo board was required, one with USB host capabilities and the ability to integrate a CAN controller. The RP2040, a Raspberry PI Pico rebranded by Adafruit has both a MCP2515 CAN controller hat and a version with a USB A host port. I2C communication will also be used for board-to-board communication due to its simplicity and speed. To both host a website and store the resulting collected data a full Raspberry PI is the perfect candidate and plenty of processing power. The Raspberry PI also has a matching LoRa bonnet that can be added for 915 MHz LoRa communication to the PI. As for the dash the main addition made was to add an analog tach to function as a backup for the driver. This is purely for redundancy purposes if an MCU were to fail.

When starting to assess the validity of the system I started by coding and testing the biggest unknown, the DI-2108 communication protocol. Through establishing connection and coding a basic CAN sniffer both on the RP2040 two issues became apparent. The DAQ samples data at such a rate (no lower than 4000 Hz) that there will be much more data from the DAQ than the ECU. Secondly the RP2040 must manage USB host functionality on one of its cores while the other handles the rest of the load. While all the data was processed, I also needed to then push that data to the ESP which proved to add too much latency to the process. So, for the final design a different approach was needed.

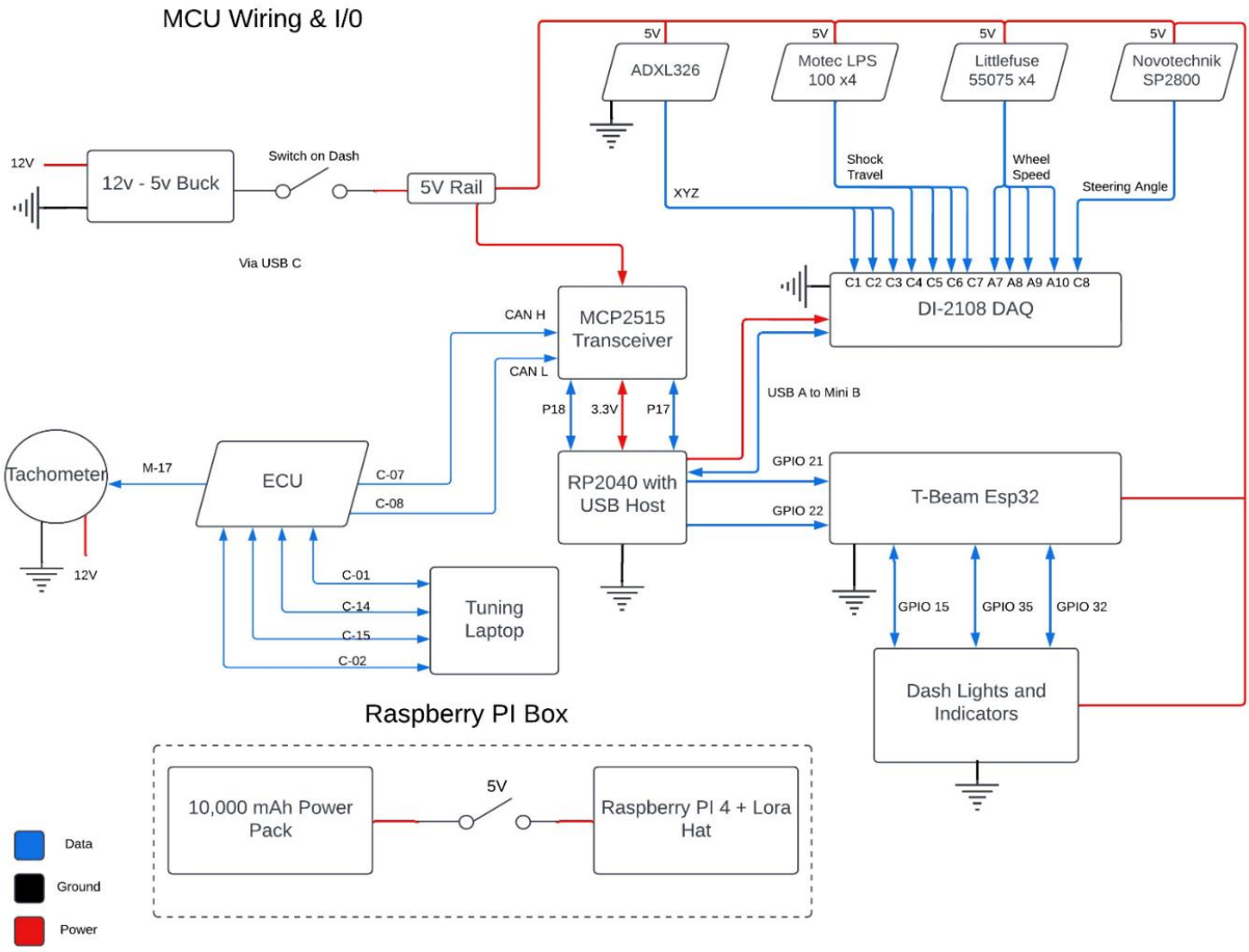


Figure 3: First DAQ system design and Layout

Final Design

The final outline of the DAQ system has several key features that differ from the first couple of designs. To free up space on the ECUs analog inputs the ADX326 accelerometer is moved to the ESP32. This frees three inputs and allows all other sensors to go only through the ECU and the CAN bus. The DI-2108 is completely removed and relegated to an optional expansion if multiple sensors need to be added. Since the legwork of testing and coding the communication is already done it is a good option to keep in our back pocket. The final flow of the system can be seen in figure 5.

Examining figure 4, it shows a basic SolidWorks sketch layout of the dash based on the frame team's assembly file. This was done to ensure indicators neatly fit in the space provided by the frame team. Lastly a backup gauge cluster which shows battery voltage, coolant temperature and oil pressure was added for system redundancy.

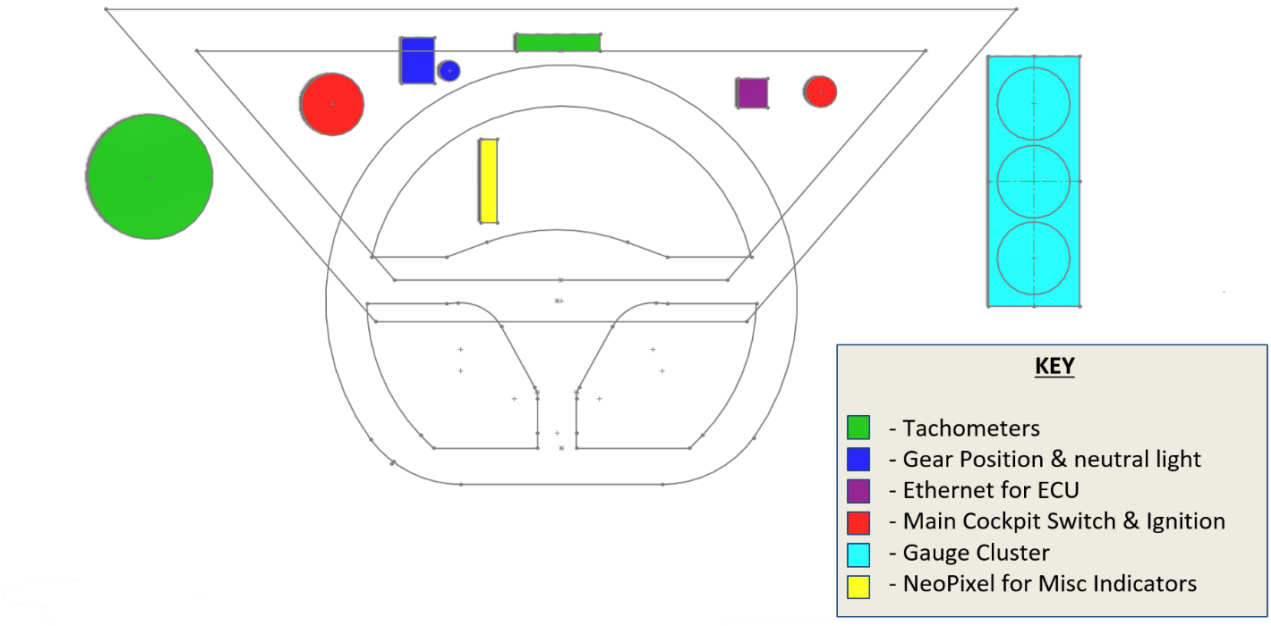


Figure 4: SolidWorks Layout of Dashboard

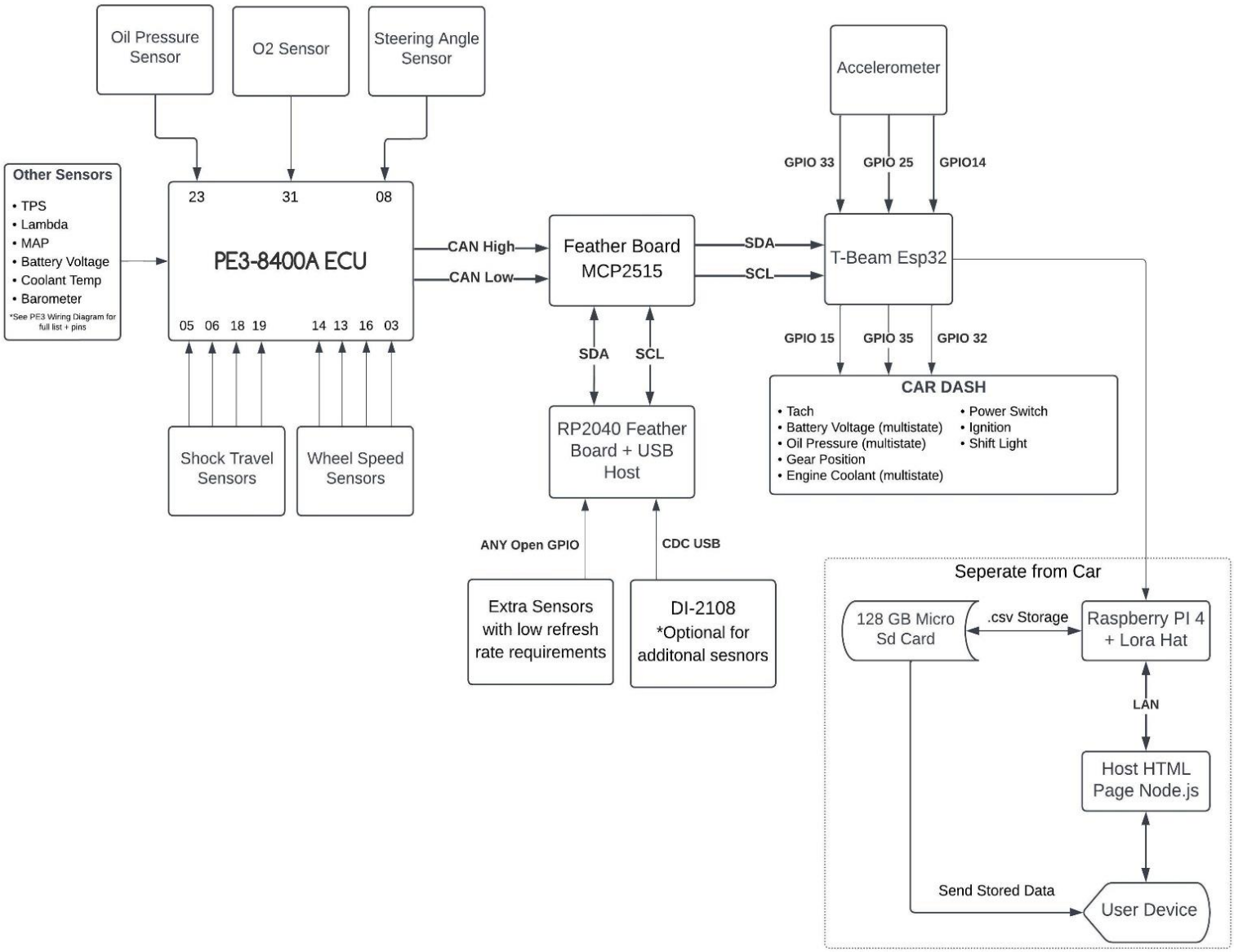


Figure 5: Final DAQ system data flow

General Engineering Design

Risk Assessment

The MCUs used in the data collection system are proven and reliable, however I cannot rule out electrical or unpredictable issues as a possibility. While data collection and monitoring are not the priority the dashboard function for the driver is. If a board were to fail the driver will have both an analog tachometer driven by the ECU and a neutral light. While not ideal it gives the driver enough info to drive the car without any of the electrical dash components. For diagnostics, the dash cluster will also function as their interdependent from the MCUs.

Standards and FSAE Rules

Most rules in the FSAE hand rule are not applicable aside from a couple regarding parts of the safety circuit and the dash of the car. As for CAN standards, since we are not integrating CAN with systems not of our design ISO standards do not apply. For coding, all code should be properly commented, white spaced and recorded to a GitHub repository for recordkeeping and readability purposes.

FSAE Rules of note:

- IC.9.1.3: The international electrical symbol must be placed near the main cockpit kill switch
- IC.9.4.1: *The main cockpit kill switch must b*e a push-pull or push-rotate emergency switch (pushing the button is the OFF position) and have a minimum diameter of 24mm
- IC.9.4.2: The kill switch must be easy for the driver to reach, adjacent to steering wheel and obstructed by any part of the vehicle

Accessibility and Ergonomics

When considering ergonomics and accessibility for the driver the only thing I'm concerned with is the dashboard and its limited controls. As shown in figure 2 the main cockpit shutoff and ignition are opposite and outside the area where the driver can easily bump them with their knuckles. This is in accordance with both FSAE rules and for the safety of the driver. The digital tach is positioned as high on the dash as possible so the driver can easily glance to it as needed without taking too much their attention away from the track.

Aesthetics

The DAQ system will be covered or otherwise hidden by the body of the car within the electrical enclosure. When considering aesthetics, the only visible part will be the dashboard of the car. Just as the 2023 team did, I plan on wrapping the dashboard in carbon fiber vinyl to match the rest of the car. Alongside the indicators pictured in figure 2 it should pull together a nice look for the cockpit.

System Fabrication

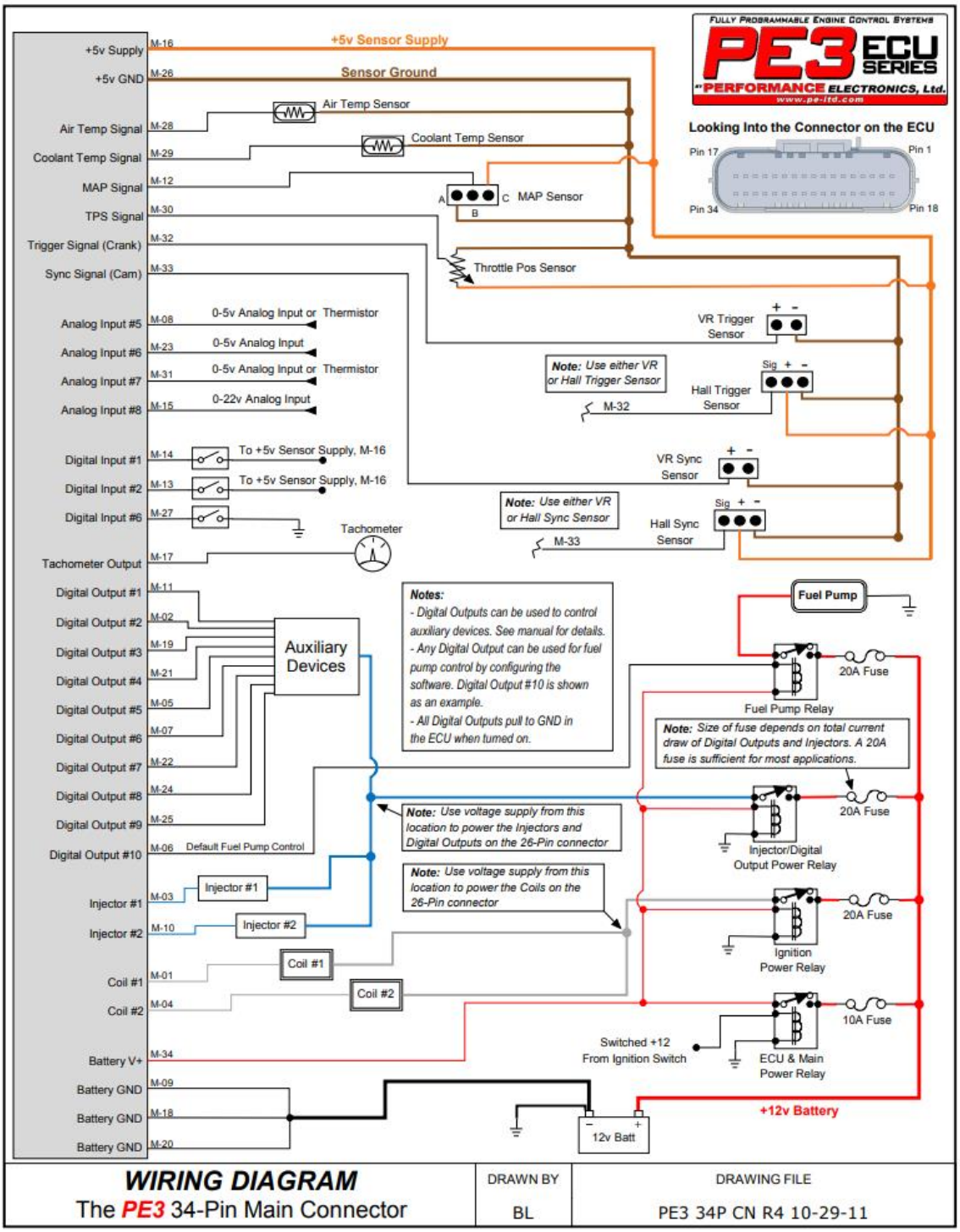
The overall construction of both the dash and the wiring of the MCUs should be elementary. The Dash being constructed out of MDF makes it easily modifiable and light compared to for example 6061 aluminum. The only concern will be mounts for the analog tachometer and gauge cluster. In which I will need aid from the frame team to design mounting brackets to properly secure them to the frame.

For the wiring and mounting of the MCUs the frame team has provided an electrical enclosure behind the dash like the 2023 team [3]. This will allow a simple compartment to mount the MCUs with standoffs and two terminal blocks for 5V rails and ground rail. As well as tucking away as many visible wires as possible to keep the cockpit both safe and aesthetic.

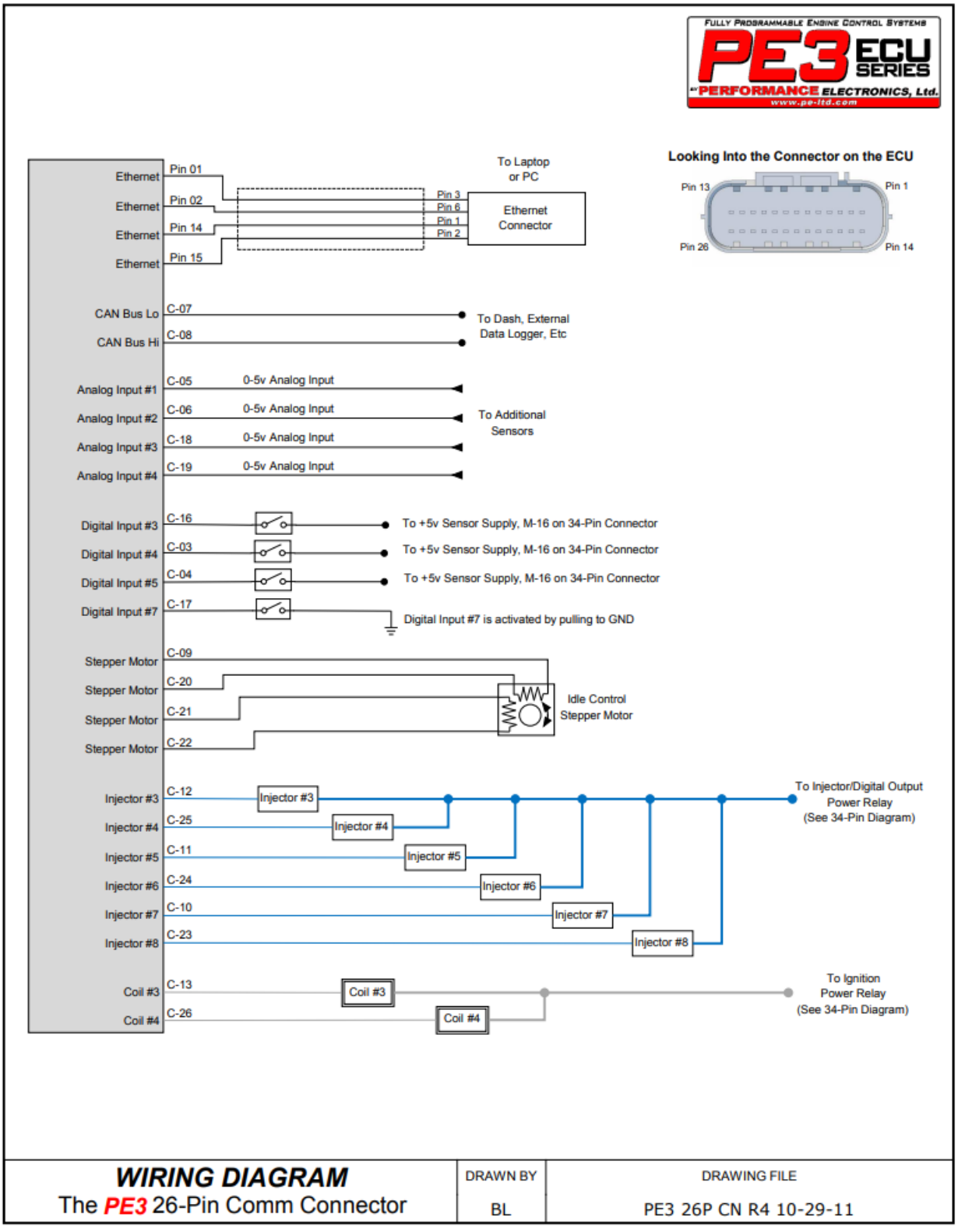
Conclusion

Moving forward to next fall and Co-op, it will be particularly important for me to continue coding the system as to move forward in the spring most of it needs to be complete. The focus of spring should be on final assembly, checkout of the system and aiding Regan in wiring the car. While the DAQ system is important for testing and validation of the cars performance and troubleshooting, its importance does not trump that of having a complete and functional car come competition.

Appendix



A1: Pg2 of Recommended Wiring for ECU from PE3



A2: Pg2 of Recommended Wiring for ECU from PE3

AN400 Rev C– Application Note
CAN Bus Protocol for PE3 Series ECUs
Release Date 12/20/16

Firmware/Software Version:	PE3 V3.04.01 and higher
Relevant Hardware:	All PE3 controllers with installed CAN Bus
Additional Notes:	This document defines the CAN based parameters that the PE3 is broadcasting for the firmware listed above. The PE3 ECU contains a 120 ohm termination resistor.

CAN Bus Details

- 250 kbps Rate
- Broadcast parameters are based on SAE J1939 standard
- All 2 byte data is stored [LowByte, HighByte]
  *Num = HighByte*256 + LowByte*
- Conversion from 2 bytes to signed int is per the following:
  *Num = HighByte*256+LowByte*
  if (Num>32767) then
  Num = Num - 65536
  endif

CAN ID (hex)	Name	Rate (ms)	Start Position	Length	Name	Units	Resolution per bit	Range	Type
0CFFF048	PE1	50	1-2	2 bytes	Rpm	rpm	1	0 to 30000	unsigned int
			3-4	2 bytes	TPS	%	0.1	0 to 100	signed int
			5-6	2 bytes	Fuel Open Time	ms	0.1	0 to 30	signed int
			7-8	2 bytes	Ignition Angle	deg	0.1	-20 to 100	signed int
0CFFF148	PE2	50	1-2	2 bytes	Barometer	psi or kpa	0.01	0-300	signed int
			3-4	2 bytes	MAP	psi or kpa	0.01	0-300	signed int
			5-6	2 bytes	Lambda	lambda	0.01	0-10	signed int
			7.1	1 bit	Pressure Type			0 - psi, 1-kPa	unsigned char
0CFFF248	PE3	100	1-2	2 bytes	Analog Input #1	volts	0.001	0 to 5	signed int
			3-4	2 bytes	Analog Input #2	volts	0.001	0 to 5	signed int
			5-6	2 bytes	Analog Input #3	volts	0.001	0 to 5	signed int
			7-8	2 bytes	Analog Input #4	volts	0.001	0 to 5	signed int
0CFFF348	PE4	100	1-2	2 bytes	Analog Input #5	volts	0.001	0 to 5	signed int
			3-4	2 bytes	Analog Input #6	volts	0.001	0 to 5	signed int
			5-6	2 bytes	Analog Input #7	volts	0.001	0 to 5	signed int
			7-8	2 bytes	Analog Input #8	volts	0.001	0 to 22	signed int
0CFFF448	PE5	100	1-2	2 bytes	Frequency 1	hz	0.2	0 to 6000	signed int
			3-4	2 bytes	Frequency 2	hz	0.2	0 to 6000	signed int
			5-6	2 bytes	Frequency 3	hz	0.2	0 to 6000	signed int
			7-8	2 bytes	Frequency 4	hz	0.2	0 to 6000	signed int
0CFFF548	PE6	1000	1-2	2 bytes	Battery Volt	volts	0.01	0 to 22	signed int
			3-4	2 bytes	Air Temp	C or F	0.1	-1000 to 1000	signed int
			5-6	2 bytes	Coolant Temp	C or F	0.1	-1000 to 1000	signed int
			7.1	1 bit	Temp Type			0 - F, 1 - C	unsigned char
0CFFF648	PE7	1000	1-2	2 bytes	Analog Input #5 - Thermistor	C or F	0.1	-1000 to 1000	signed int
			3-4	2 bytes	Analog Input #7 - Thermistor	C or F	0.1	-1000 to 1000	signed int
			5	1 byte	Version Major		1	0-255	unsigned char
			6	1 byte	Version Minor		1	0-255	unsigned char
			7	1 byte	Version Build		1	0-255	unsigned char
			8	1 byte	TBD				

A3: Pg1 of CAN protocol document from PE3

CAN ID (hex)	Name	Rate (ms)	Start Position	Length	Name	Units	Resolution per bit	Range	Type
0CFFF748	PE8	100	1-2	2 bytes	RPM Rate	rpm/sec	1	-10,000 to 10,000	signed int
			3-4	2 bytes	TPS Rate	%/sec	1	-3,000 to 3,000	signed int
			5-6	2 bytes	MAP Rate	psi/sec or kpa/sec	1	-3,000 to 3,000	signed int
			7-8	2 bytes	MAF Load Rate	g/rev/sec	0.1	-300 to 300	signed int
0CFFF848	PE9	100	1-2	2 bytes	Lambda #1 Measured	lambda	0.01	0 to 10	signed int
			3-4	2 bytes	Lambda #2 Measured	lambda	0.01	0 to 10	signed int
			5-6	2 bytes	Target Lambda	lambda	0.01	0 to 2.5	signed int
0CFFF948	PE10	100	1	1 byte	PWM Duty Cycle #1	%	0.5	0 to 100	unsigned char
			2	1 byte	PWM Duty Cycle #2	%	0.5	0 to 100	unsigned char
			3	1 byte	PWM Duty Cycle #3	%	0.5	0 to 100	unsigned char
			4	1 byte	PWM Duty Cycle #4	%	0.5	0 to 100	unsigned char
			5	1 byte	PWM Duty Cycle #5	%	0.5	0 to 100	unsigned char
			6	1 byte	PWM Duty Cycle #6	%	0.5	0 to 100	unsigned char
			7	1 byte	PWM Duty Cycle #7	%	0.5	0 to 100	unsigned char
			8	1 byte	PWM Duty Cycle #8	%	0.5	0 to 100	unsigned char
0CFFFA48	PE11	100	1-2	2 bytes	Percent Slip	%	0.1	-3000 to 3000	signed int
			3-4	2 bytes	Driven Wheel Rate of Change	ft/sec/sec	0.1	-3000 to 3000	signed int
			5-6	2 bytes	Desired Value	%	0.1	-3000 to 3000	signed int
0CFFFB48	PE12	100	1-2	2 bytes	Driven Avg Wheel Speed	ft/sec	0.1	0 to 3000	unsigned int
			3-4	2 bytes	Non-Driven Avg Wheel Speed	ft/sec	0.1	0 to 3000	unsigned int
			5-6	2 bytes	Ignition Compensation	deg	0.1	0 to 100	signed int
			7-8	2 bytes	Ignition Cut Percent	%	0.1	0 to 100	signed int
0CFFFC48	PE13	100	1-2	2 bytes	Driven Wheel Speed #1	ft/sec	0.1	0 to 3000	unsigned int
			3-4	2 bytes	Driven Wheel Speed #2	ft/sec	0.1	0 to 3000	unsigned int
			5-6	2 bytes	Non-Driven Wheel Speed #1	ft/sec	0.1	0 to 3000	unsigned int
			7-8	2 bytes	Non-Driven Wheel Speed #2	ft/sec	0.1	0 to 3000	unsigned int
0CFFFD48	PE14	100	1-2	2 bytes	Fuel Comp - Accel	%	0.1	0 to 500	signed int
			3-4	2 bytes	Fuel Comp - Starting	%	0.1	0 to 500	signed int
			5-6	2 bytes	Fuel Comp - Air Temp	%	0.1	0 to 500	signed int
			7-8	2 bytes	Fuel Comp - Coolant Temp	%	0.1	0 to 500	signed int
0CFFFE48	PE15	100	1-2	2 bytes	Fuel Comp - Barometer	%	0.1	0 to 500	signed int
			3-4	2 bytes	Fuel Comp - MAP	%	0.1	0 to 500	signed int
			5-6	2 bytes	-				
			7-8	2 bytes	-				
0CFFD048	PE16	100	1-2	2 bytes	Ignition Comp - Air Temp	deg	0.1	-20 to 20	signed int
			3-4	2 bytes	Ignition Comp - Coolant Temp	deg	0.1	-20 to 20	signed int
			5-6	2 bytes	Ignition Comp - Barometer	deg	0.1	-20 to 20	signed int
			7-8	2 bytes	Ignition Comp - MAP	deg	0.1	-20 to 20	signed int

A4: Pg2 of CAN protocol document from PE3

A5: Data Rate Calculations for CAN, LoRa and I2C

$$CAN\ Data\ Rate = \sum \frac{Bytes}{Second}\ of\ Each\ CAN\ ID$$

Using Table 1 and appendix A3 for packets sizes and rate:

$$CAN\ Data\ Rate = 3\left(\frac{8\ bytes}{0.1\ sec}\right) + \left(\frac{8\ bytes}{1\ sec}\right) + 2\left(\frac{8\ bytes}{0.05\ sec}\right) = 568\ \frac{bytes}{sec}$$

The result is converted to bits and rounded up to nearest hundred value:

$$CAN\ Data\ Rate = 568\frac{bytes}{sec} \times 8\ \frac{bits}{byte} = 4.6Kbps$$

LoRa data rate has addition of Accelerometer data see Table 1 and Figure 4:

$$LoRa\ Data\ Rate = CAN\ Data\ Rate + \left(\frac{8\ bytes}{0.05\ sec}\right) = 808\ bytes$$

The result is converted to bits and rounded up to nearest hundred value:

$$LoRa\ Data\ Rate = 808\frac{bytes}{sec} \times 8\ \frac{bits}{byte} = 6.5Kbps$$

Note: I2C data rate will equal CAN as total size of data does not change

Bill of Materials

Line	Part (w/ Hyperlink)	Description	Supplier Part #	Supplier	Weight (lbs.)	Cost Per Unit	Quantity	Total Cost
1	ESP32 T-Beam TTGO	Main MCU for Dash and sending Wireless data	B09VLFQQG4	Amazon	0.263	42.5	1	$ 42.50
2	MCP2515	CAN Transceiver for reading data from ECU	5709	Adafruit	0.008	12.5	1	$ 12.50
3	Power pack 10,000 mAh	Battery bank to power Raspberry PI	B07QXV6N1B	Amazon	0.47	21.99	1	$ 21.99
4	Raspberry Pi 4	MCU for Hosting HTML page and storing data	B07TC2BK1X	Amazon	0.11	99	1	$ 99.00
5	Raspberry Pi Lora Hat	Hat to add LORA capability to Raspberry pi	4074	Adafruit	0.024	32.5	1	$ 32.50
6	Arduino Micro	MCU for reading 5v sensor data	B00AFY2S56	Amazon	0.038	19.92	1	$ 19.92
7	Neopixel bar	Back up for Tachometer and other indicators	1426	Adafruit	0.006	5.95	2	$ 11.90
8	Neopixel lights	Used for other dash indicators	1312	Adafruit	0.006	7.95	2	$ 15.90
9	HDMI to micro HDMI	Display output for PI	1322	Adafruit	n/a	8.95	1	$ 8.95
10	12 to 5v Buck converter	To step down power for MCUs, other sensors and Neopixels	B01NALDSJ0	Amazon	0.033	9.99	1	$ 9.99
11	128gb Micro SD card	For Raspberry boot drive and data storage	B09X7DNF6G	Amazon	0.011	18.5	1	$ 18.50
12	5v to 3.3v level Shifter	For Arduino to ESP serial communication	5637	Adafruit	0.02	2.95	1	$ 2.95
13	3.3v to 5v level Shifter	For ESP to Neopixel logic	5649	Adafruit	0.02	2.95	1	$ 2.95
14	ADXL326	Accelerometer compatible with chosen MCUs	1018	Adafruit	0.003	17.95	1	$ 17.95
Total Cost								$ 317.50

References

[1] Engcapstone Share\\Formula SAE 2023\Tech Reports Spring23 Final \Formula_ODonnell_Wireless_Spr23_Final.pdf

[2] Engcapstone Share\\Formula SAE 2023\Tech Reports Spring23 Final \Formula_King_DataAcquisition_Spr23.pdf

[3] Engcapstone Share\\Formula SAE 2023\Tech Reports Spring23 Final \Formula_Zupko_Electrical_Sp23.pdf

[4] S. Corrigan, Introduction to the Controller Area Network (CAN) (rev. B),.

[5] B. Kelechava, "Controller Area Network (CAN) standards," The ANSI Blog, https://blog.ansi.org/2017/02/controller-area-network-can-standards-iso-11898/#gref.

[6] Performance electronics | where fire meets fuel, https://www.pe-ltd.com/assets/pe3-series-manual.pdf.

[7] DATAQ Instruments | The way PC-based instruments should be, https://www.dataq.com/resources/pdfs/manuals/di-2108-usb-daq-hardware-manual.pdf

[8] DATAQ Instruments | The way PC-based instruments should be, https://www.dataq.com/resources/pdfs/misc/Dataq-Instruments-Protocol.pdf

[9] R. Liang, L. Zhao, and P. Wang, "Performance Evaluations of lora wireless communication in building environments," Sensors (Basel, Switzerland), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7412434/.