FSAE Data Acquisition Corner Board

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

Ethan Jacobson

Spring, 2025

Technical Project Team Members

Jack Basinet

Jack Hebert

Casey Ladd

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Todd Delong, Department of Electrical and Computer Engineering

I. Statement of Work

Each team member has experience in both hardware and software development. Team debugging, parallel development, and collaboration were staples of our system design.

Jack Basinet:
In tandem with Ethan, I spearheaded hardware and PCB design. I led the creation of all the power and CAN systems for our device, performing intensive research, component selections, schematic design, PCB routing, and device testing. This work spread across multiple PCB iterations and design changes. Managerially, I took on meeting planning, time and deadline management, and contributed to part orders. Additionally, I designed our team logo.

Jack Hebert:
Alongside Casey, I primarily focused on building and testing the software. I specifically led the combination and debugging of the final program. I also added documentation to the code so that VME can easily adjust it for their future needs. Behind the scenes, I also handled bookkeeping for our group.

Ethan Jacobson:
As the electrical team lead of the client, Virginia Motorsports, I defined system-level requirements and planned the design around integrating the system with the car. I also communicated with the data acquisition team lead, Michael Clark, to ensure the project stayed up to date with changes to the car.

I designed the embedded Teensy 4.1 circuit and its integration into the CAN and power modules. I also managed the project's Altium symbol and component library and routed the transceiver breakout board, the first debug board, and the final six-layer embedded board.

Casey Ladd:
I focused on writing and testing the device software. Specifically, I performed CAN and Teensy communication research, wrote the data collection and transmission code, and supported hardware with design specifications. Managerially, I hosted, managed, and organized our communication platforms.

II. Table of Contents

III. List of Figures

IV. Abstract

This project is about updating and simplifying the wheel sensor data acquisition process for Virginia Motorsports Education's (VME) FSAE car (VM25). The customer competes in the Formula Society of Automotive Engineers (FSAE) competition each year and is iterating on the data acquisition hardware and software for the new car. The current system has each sensor in each wheel wired directly into the central data computer, which will be replaced by a single CAN bus data line connecting the four wheels. Each sensor will send its analog signal to its wheel's sensor board, which will be multiplexed into a digital signal and then sent out to the central computer via the CAN protocol.

V. Background

This project aims to develop a unified data system for VME's VM25, which will be implemented for the 2025 FSAE competition. The system will replace individual analog data wires with CAN bus transmission, streamlining data transfer between modules and the central data computer, an Nvidia Jetson. Specifically, this project focuses on developing the hardware and software to convert analog data into CAN bus signals, working in parallel with VME's data acquisition team as they develop the software for the Nvidia Jetson and select sensors for the car.

Fig. 1: VM24 in the paddocks at Michigan International Raceway

The CAN (Controller Area Network) bus protocol is commonly used in automotive applications for its high data rate, reliability, and flexibility for connecting a variety of electronic control units (ECUs) to a single data line. Its components are shown in figure 2.

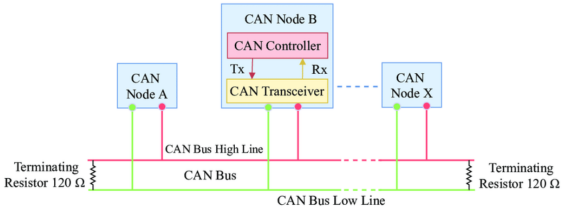


Fig. 2: Diagram of components of the CAN Bus

Currently, VME's CAN bus is used for transmitting wheel speed data and controlling the electric motor via the vehicle control unit (VCU). However, data from suspension linear potentiometers, brake temperature, and tire temperature sensors, critical for design validation, are not integrated into a unified system. This project will address this gap, significantly reducing wiring harness complexity by transmitting all data via the CAN bus.

The project builds on prior experience in Altium Designer, CAN bus interfacing, and coursework, including Intro to Embedded Systems, the ECE Fundamentals series, Computer Architecture, and Dependable Computing Systems.

VI. Project Description

A. Performance Objectives and Specifications

The board will connect to the VM25 harness, interfacing with CANH, CANL, 24V, and CAN ground wires. It supports connections for a wheel speed sensor, a linear potentiometer, and CAN 2.0 sensors for brake and tire temperatures. The board operates with the CAN FD protocol at 5.6 Mb/s, with the ability to scale down to 1 Mb/s to ensure compatibility with existing car systems. To minimize noise on the CAN lines, EMC measures such as bypass capacitors, choke filters, ESD diodes, and differential pair matching are incorporated.

The board must withstand power and communication faults during dynamic FSAE events and be compact,

resulting in a 3” x 1.125” footprint. It is able to tolerate EMI in order to ensure that there is no more than a 10% increase in CAN errors during idle operation and is vibration-resistant, utilizing SMD components and AEC-Q100-rated ICs [1]. Finally, it operates on the car’s 24V DC grounded low-voltage (GLV) system.

B. Functionality

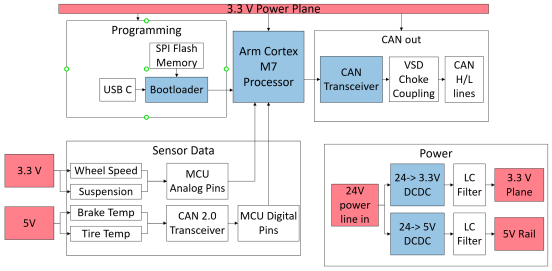


Fig. 3: Hardware design flow chart

The sensor data processing is done by embedding a Teensy 4.1 onto a custom PCB; this allows us to meet our space and signal integrity requirements, as opposed to socketing a Teensy 4.1 into a custom PCB with the power and CAN hardware.

The flow chart in figure 3 shows the system design of the board hardware. The modules involved are the MCU, programming, power, CAN out, and sensors in.

MCU and Programming Module Overview:

The MCU is a MIMXRT1062 series ARM Cortex M7 processor, programmed using a preprogrammed MKL02Z32 bootloader chip from PJRC. It uses a W25Q64JVXGIM chip for 64 Mb of serial memory storage and an external 24 MHz oscillator for timing, as shown in Figure 4.

The bootloader interfaces with the MCU’s JTAG pins to flash code and completes the hardware power-up sequence by asserting pin 3 high, enabling the MCU’s onboard DC-DC converter to supply SOC voltage. A tactile switch pulls PTB2 on the bootloader to ground to initiate programming mode, with LED D5 indicating this status.

Power Module Overview:

24V power is converted to usable levels by two independent DCDC regulators, one for 5V (figure 6), and one for 3.3V (figure 5). The 5V regulator powers the IR temperature sensors and initial power in the MCU power-on sequence.

During the power-on sequence, the MCU is able to activate the 3.3V DCDC in Figure 5 to supply power to GPIO, transceivers, memory, bootloader, and analog sensors. The TPS62177 is fixed at 3.3V output compared with the TPS62175 used for the 5V output (Figure 6) [2], [3].

Sensor Data and Software Module Overview:

The board interfaces with the CAN bus in the VM25 harness and the four sensors in the suspension assembly, as illustrated in Figure 7. Hirose GT32 series shielded automotive data cables were selected for both input and output connections, using 4-pin and 19-pin variants, respectively [4].

The 4-pin input connector carries CANH, CANL, 24V, ground wires. The 24V line is fused at 500mA and filtered with a 120-ohm ferrite. The board supports a suspension travel linear potentiometer, a hall effect wheel speed sensor, and two CAN 2.0 IR temperature sensors, which connect via the 19-pin shielded GT32 output connector.

Sensor data is processed by the MCU via the analog pins shown in Figure 8, which follow the same layout as the Teensy 4.1, enabling their use for analog-to-digital conversion and CAN bus communication. Linear potentiometer and wheel speed sensors connect to analog pins with ADC functionality, while the tire and brake temperature sensors, after passing through the CAN 2.0 receive transceivers, communicate with the MCU via one of the three CAN TX and RX ports labeled as CAN_B.

These sensors ([5], [6] transmit 8-byte CAN 2.0 signals to the transceivers. The MCU software integrates all sensor data into a single 64-byte CAN FD message. This

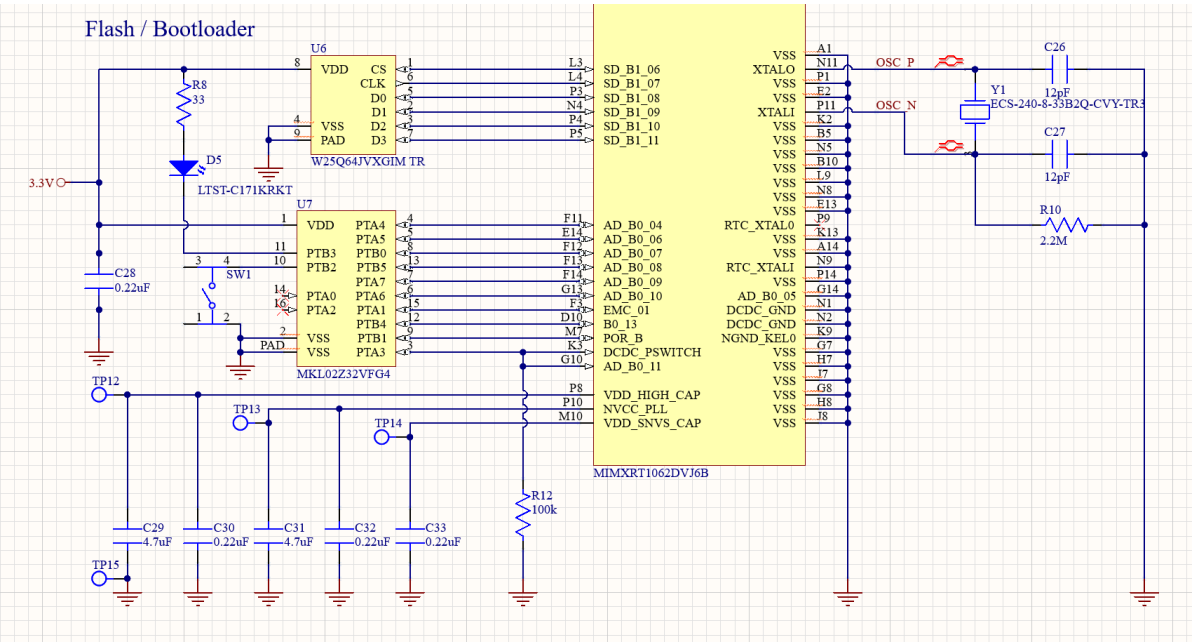


Fig. 4: MCU bootloader (U7), flash memory (U5) and external oscillator (Y1) circuits

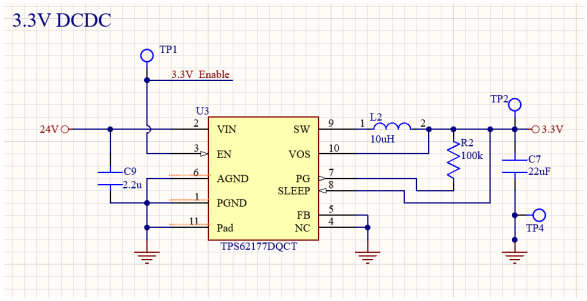


Fig. 5: 3.3V supply

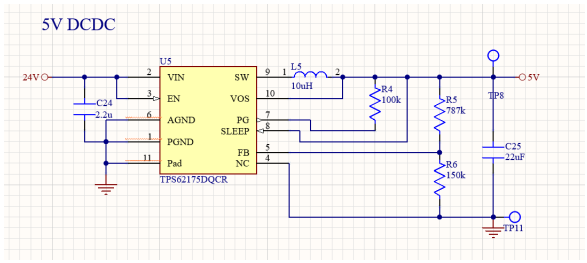


Fig. 6: 5V supply

consolidated data is sent using the TX and RX ports of another CAN interface, labeled CAN_A.

CAN Module Overview:

The two pairs of TX and RX data lines from the

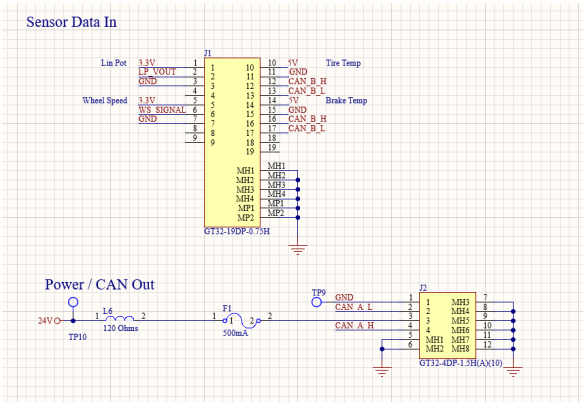


Fig. 7: Schematic of input and output connectors on board

MCU feed into the receiving and transmitting CAN transceivers, shown in Fig. 9. The CAN 2.0 input transceiver reads the sensor input CAN package on the differential pair, CAN_B_H and CAN_B_L, and drives the input TX and RX lines accordingly. The CAN FD output transceiver takes the combined data packages from the MCU and transmits them efficiently on the protected output differential pair: CAN_A_H and CAN_A_L. These input and output CAN_H and CAN_L lines are optionally terminated, depending on the client's implementation, protected from high-frequency spikes

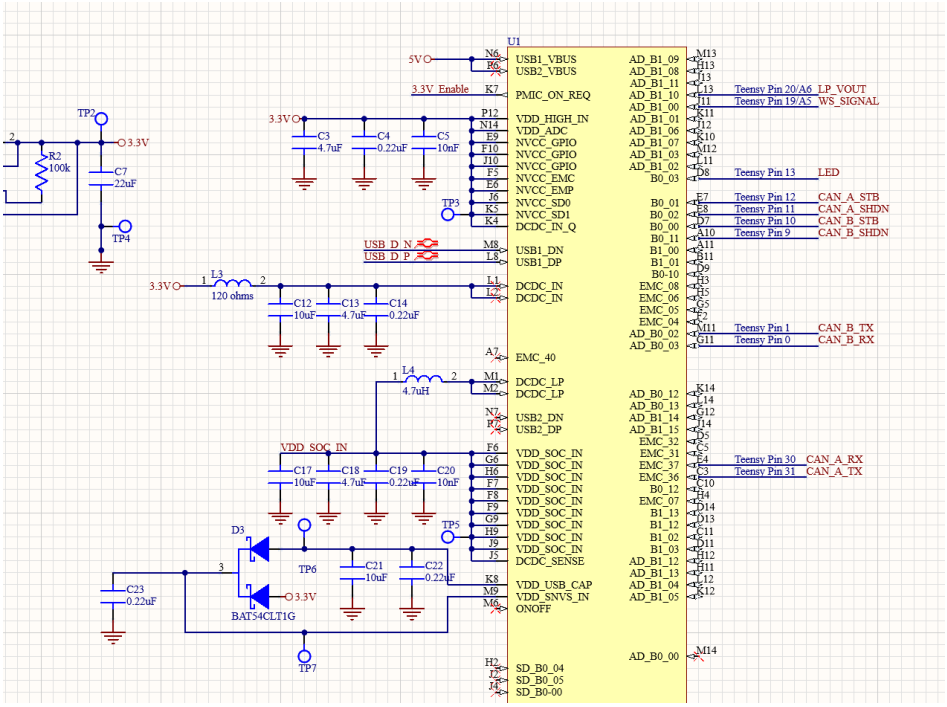


Fig. 8: ARM Cortex M7 Processor with Labeled Analog Data Pins

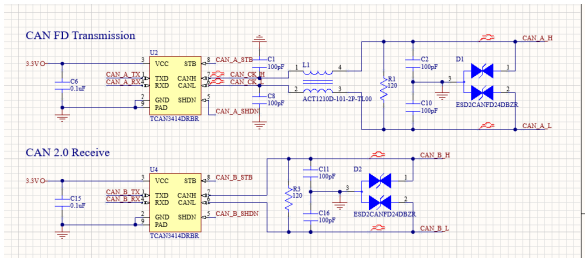


Fig. 9: CAN Transceivers

by bypass caps, and shielded from static discharge by TVS Diodes. Additionally, the CAN FD transmission is fed through a common mode choke to filter out electromagnetic interference and stabilize the higher data rate signals.

C. Technical Details of Design Process

The design process followed several stages: research, schematic development, test board development, test board debugging, final board design, and final board testing.

Power, transmission, and data reception research and schematic development closely followed product standards. The documentation for the voltage regulators used contained component selection recommendations and routing layouts. Bypass caps around the DCDCs stabilize input and output ripple, and the inductor and resistor in the control loop are device standard and limit ripple current. Additionally, with the 5V regulator, the pair of resistors (R5 and R6) set the output to 5V with a standard device reference voltage (V_{ref}) of 7.4V and the following formula [7]:

$$R_1 = R_2 \, (\frac{V_{out}}{V_{ref}} - 1)$$

Placing and routing custom on-board regulator circuits involved closely following device standards [7]. The component placement had to carefully follow recommendations to keep short current loops, clear power ground return paths, and strategic bypass capacitor placement to best protect the IC and provide a stable output. Routing needed beefy traces for the current flow, proper output/input ground pathing, and strategic via use.

The chosen industry-used CAN Transceivers were

selected for their 3.3V operating voltage and high bit rate potential [8]. Component selection for these devices was also driven by documentation. Bypass caps of 100pF paired with the CAN termination resistance of 120Ω filter out high-frequency noise with a low-frequency passband cutoff of around 10MHz:

$$\frac{1}{2\pi(120)(100 \cdot 10^{-12})} = 13.3MHz$$

This cutoff leaves enough passband for a maximum bitrate of 8Mbps or around 4MHz (one bit ≈ half period). Additionally, the caps are kept small to avoid slowing bus speed and adding to power consumption. The chosen ESD diodes are designed for CAN application. On the CAN FD transceiver, a common mode choke is also designed for CAN applications and added to the series for current stability and the noticeably higher bit rate [9], [10]. Lastly, CAN bus realization involved matched differential pair routing and tight component packing for line protection.

Of note, at the beginning of the project, we planned on only designing a single regulator and CAN transceiver. Further communication with the motorsports club revealed the necessity for 5V power for the on-car sensors and a second transceiver to read the temperature sensors at a lower bit rate. This change turned out to be helpful, as the bootloader power-up sequence also needed 5V [11], [2].

The MCU and its peripheral ICs were designed by referencing the schematics for the Teensy 4.1 [3], [12]. We chose to emulate this microcontroller because VME has chosen it as the main MCU to power VM25's battery management main board, dashboard, and vehicle control unit, and keeping microcontroller architecture consistent throughout the car will improve design flow across the car.

The Teensy 4.1 is based on Arm Cortex M7 architecture with a 600 Mz clock speed, 18 analog i/o pins, 55 digital i/o pins, 8 serial, 3 SPI, 3 I2C, and 3 CAN ports, of which one has CAN FD support. Its massive feature count combined with its small size make it perfect for motorsports applications. The board alone does not meet the performance requirements for the project since it lacks CAN transceivers to physically interface with the CAN bus, cannot run on 24V power, and does not include shielded data connections. Likewise, socketing a Teensy 4.1 onto a PCB with the power and transceivers is not sufficient since the 2.54mm headers are not AEC-Q100 compliant. Therefore, the project must embed the hardware of a Teensy 4.1 directly onto a custom PCB. Fortunately, the Teensy 4.1 schematic is open source and the programmed bootloader chip used to flash code to the MCU is sold by PJRC, this will eliminate issues with low level programming and reduce the complexity of the microcontroller design.

For each board implemented, surface mount components were exclusively chosen to maintain size constraints and utilize modern specialty components since few new through-hole components are manufactured that meet AEC-Q100 [1]. We chose to use pick-and-place reflow as our assembly method over a hot plate, hand soldering, or hot air soldering as it is the most reliable way to solder small components.

The first board we created was purchased through the Motorsports Club, doubling as a member education tool for SMD soldering and CAN transceiver operation. This shield board contained the TCAN3414 transceiver and its associated filtering circuits. Since the Teensy 4.1 lacks a built-in CAN transceiver, the external PCB provided CAN functionality for software development before transitioning to the final embedded Teensy board. This board contained some minor design flaws that were fixed in subsequent revisions of the main boards, which included utilizing ground pours instead of return traces to reduce noise in ground return loops and shunting bypass caps straight to the ground plane as recommended by the datasheet [8].

The second board we created (Figure 11 was an attempt to integrate the Teensy 4.1 into a PCB. Header pins were added for debugging, and a large four-layer PCB was used for ease of routing. Components were placed in a similar manner to the Teensy 4.1 to minimize

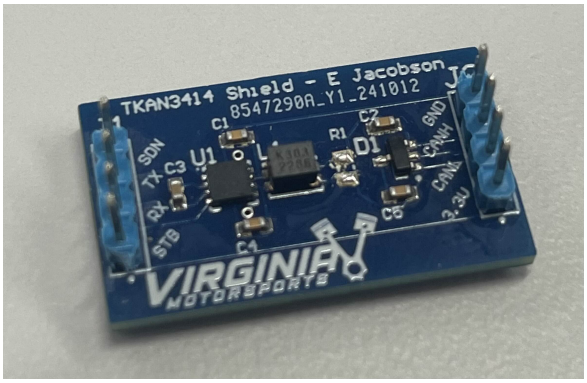


Fig. 10: TCAN3414 transceiver board used in software testing.

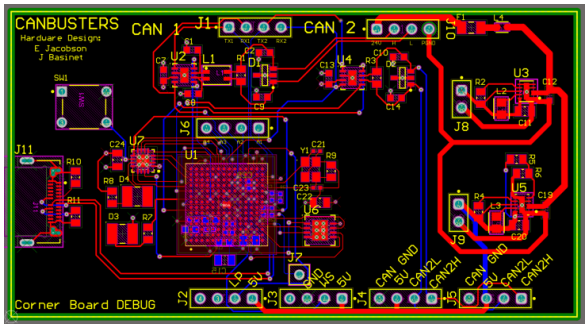


Fig. 11: CAD of the first iteration of embedded Teensy board

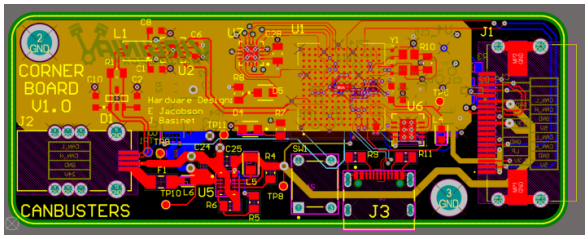


Fig. 12: CAD of final integrated MCU

rats nest overlap, and traces were routed primarily with top signal traces horizontally and bottom signal traces vertically.

A challenge of routing the board was dealing with the ball grid array (BGA) footprint of the MCU. Routing this was done by following traditional BGA fanout routing, where the outer two layers of balls are routed away from the chip in four directions and the inner layers use vias to escape on the bottom layer. The 3.3V and GND plane layers of the PCB greatly improved ease of routing, but switching to six layers with a third signal layer would improve component density. Following the RT1062 hardware design recommendations [2], 4 mil traces and 8/14 mil vias were used in the BGA footprint, with 3.78 mil minimum trace separation. Outside of the BGA footprint, 6 mil traces were used for dense signals, and 10 mil traces were used for sparse traces. Power traces used polygon pours and 20-30 mil traces.

The first debug board was dead on arrival because the MCU's SOC power was not connected properly, the can tx and rx were crossed, and the wrong CAN pins were used. These issues prevented the MCU from powering on correctly and sending CAN messages.

This setback changed the timeline, as the debug board could not be used for testing. This caused us to pivot to rush the final board, which was in progress, and develop a second debug board that socketed a Teensy 4.1 in case testing time ran out on the integrated MCU board.

The final board, shown in Figure 12, was reduced to 3" by 1.125" by using a 6-layer design and placing components on both sides. Routing details are in Appendix XVII-F. The layer stack comprises top signal, ground 1, power plane, intermediate low-speed signal, ground 2, and bottom signal. Two ground planes provided shielding for high-speed signals on the top and bottom layers from DCDC power layer switching fields but reduced available routing space.

The CAN 2.0 transceiver, 3.3V regulator, and MCU bypass capacitors were placed on the bottom side, with shielded GT32 connectors replacing debug header pins. Test points allowed measurement of regulator outputs and MCU internal logic for debugging the power-on sequence. The Altium differential pair routing tool ensured impedance-matched traces for CANH/CANL and USB D+/D-, with careful attention to avoid crossing traces unless a ground shield was present.

The second iteration of the debug board (figure 14) included the CAN transceivers, power hardware, and shielded connectors needed to meet our design goals but had socket connections for a Teensy 4.1 replacing

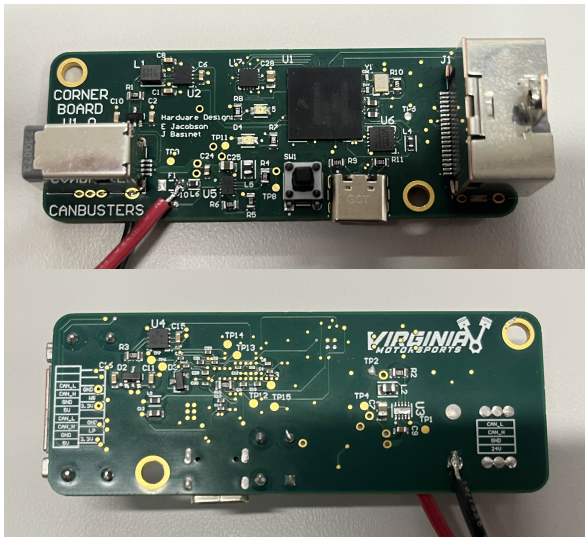


Fig. 13: Fully populated embedded MCU board

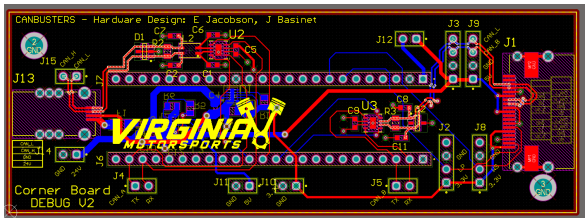


Fig. 14: CAD of debug board with socketed Teensy 4.1

the embedded MCU. This board was created to emulate the final board so software could be fully developed and demonstrated, but it does not meet our size and vibration resistance goal. The Teensy socketed into the board in figure 15 is on the top side of the board, but a mistake in switching the location of the socket headers required them to be soldered upside-down on the bottom of the board.

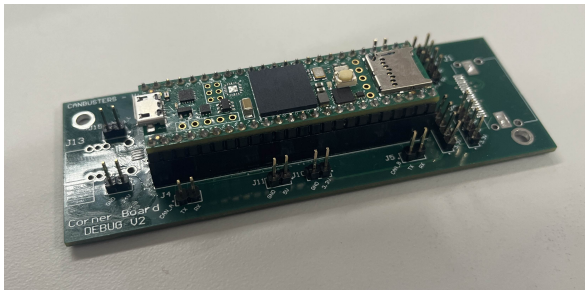


Fig. 15: Teensy 4.1 socketed into debug v2 board (backwards and upside down)

The software component of this project leverages the hardware and software compatibility of the Teensy 4.1 board. Since the MCU, bootloader, and flash memory are identical to the Teensy 4.1, all code is cross compatible between the embedded MCU and the Teensy 4.1.

Programming the Teensy 4.1 is facilitated by the Arduino IDE, complemented by the Teensyduino add-on, which provides access to numerous optimized libraries [3]. This compatibility streamlines development by allowing the reuse and adaptation of existing software designed for the Teensy 4.1.

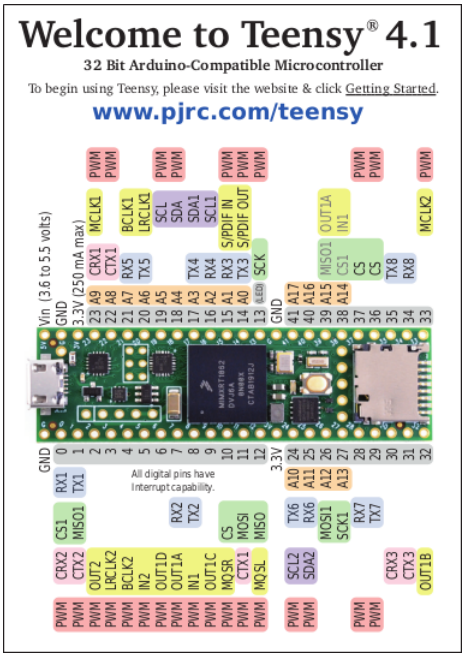


Fig. 16: Teensy Reference Card

The two main libraries that are used for the software component are the ADC library [13] and the FlexCAN library [14]. Using Figure 16, the pins labeled ”A#” indicate that they are capable of being used in the ADC library, and the pins labeled ”CTX#” and ”CRX#” indicate the pins used in the CAN bus library. CAN1 and CAN2 can only use CAN 2.0, while CAN3 can also use CAN FD.

For the program, the project incorporates an ADC that will read analog values synchronously on pins labeled A5 and A6. The project will incorporate two CAN buses, CAN2, which will use CAN 2.0, and CAN3, which will

use CAN FD. CAN2 will communicate asynchronously with TX/RX signals from the CAN 2.0 transceiver to read via interrupts, and CAN3 will communicate with the TX/RX signals from the CAN FD transceiver to write the final packaged signal.

Every Arduino program needs to have a setup and a loop function. The setup function initializes the ADC and the CAN buses and initializes the board to begin clocking. The loop function does not have direct functionality, but it verified that the program does not crash. The CAN 2.0 bus has a function called canSniff that runs after an interrupt is triggered due to the arrival of a CAN 2.0 message. Inside the function, the message is printed, including its index, flags, length, and contents. Next, the ADC reads the two pin's analog values and prints and saves the value. Lastly, the CAN FD bus writes a message to be sent out to the CAN FD transceiver. The CAN FD message incorporates the two possible CAN 2.0 messages that can be read as well as the two values collected by the ADC, making a message that is roughly 20 bytes long, and the message is printed for debugging. The full main program can be found in the appendix XVII-D.

D. Test Plans

Phase 1: Verifying Successful Reflow

The first step is to visually inspect IC solder joints using a microscope to check for bridged pins. If any bridged pins are found, they will be repaired with hot air rework. Due to equipment limitations, it is not possible to visually verify the soldering of the BGA layout MCU. Instead, its functionality will be confirmed using the power-on tests.

Phase 2: Power-On Testing

1) **Power Supply** Pin out the 4 pin gt32 connector and connect to 24V power supply.
2) **Voltage Regulation:** Verify the outputs of the DC-DC converters.

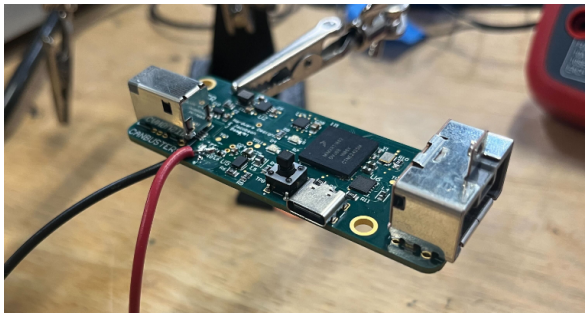


Fig. 17: Power on sequence test setup

   - Measure and confirm the 5V and 3.3V outputs are stable.
3) **MCU Power-On Sequence:** Verify the power-on sequence using the test points (TPs) detailed below. All measurements are relative to ground (e.g., GND TPs are the screw mounting holes, TP4, TP9, TP11, and TP15).
   Before proceeding with testing, press the bootloader switch and hold for 15 seconds to set the MKL02 fuses to bond with the MCU.
   a) Measure **VIn** at **TP8**, should be **5V**.
   b) Measure the **USB Voltage Regulator** output at **TP6**, should be **2.5V**.
   c) Verify the **SNVS Cap** voltage at **TP14**, should be **1.1V**.
   d) Check the **PLL and Analog Regulators**:
      - **TP12**: Should measure **2.5V**.
      - **TP13**: Should measure **1.1V**.
   e) Verify that the **3.3V Regulator On-Request Signal** is driven high (1-3.3V) at **TP1**.
   f) Ensure the **DCDC is Enabled** by the bootloader. Measure the pin at the via between the bootloader and **C28**.
   g) Check the **SoC Voltage** is enabled by the DCDC at **TP5**, should measure **1.15V**.

Bootloader Diagnostics

After completing the power-on tests, observe the behavior of the bootloader:

- **Red Error LED (D5):**
  - Smooth pulsing indicates that MCU is in boot-

loader mode.
  - Repeated blinking indicates a diagnostic error. Refer to the diagnostic error codes on PJRC's MKL02 bootloader page [3].

After verifying the hardware is operational:

1) Flash known good LED blink code by holding the bootloader switch for 15 seconds.
2) Observe the **Green LED (D4)**:
   - Regular blinking confirms successful operation.

Once the green LED blinks as expected, the MCU hardware is considered operational.

To verify can transceiver functionality, run code in software that drives the transceiver STB and SHDN pins low (teensy pins 9 through 12) and verify the following with a multimeter.

- Voltage between CANH and GND: 1.9V
- Voltage between CANL and GND: 1.9V

These measurements can be easily taken from the pins of the ESD protection diodes (D1 and D2)

Phase 3: Software Testing

Phase 3: ADC Testing

The functionality of the analog-to-digital converters (ADCs) is verified through a series of steps to ensure accurate and independent operation.

1) **Constant Voltage Input Testing**
   - Apply a stable voltage (e.g., 2.5V) to one ADC pin.
   - Measure the ADC output in software and confirm that it matches the expected input voltage.
2) **Variable Voltage Input Testing**
   - Apply a variable voltage signal (e.g., 0-3.3V) to the same ADC pin.
   - Confirm the ADC output tracks the variable input accurately over the full range.

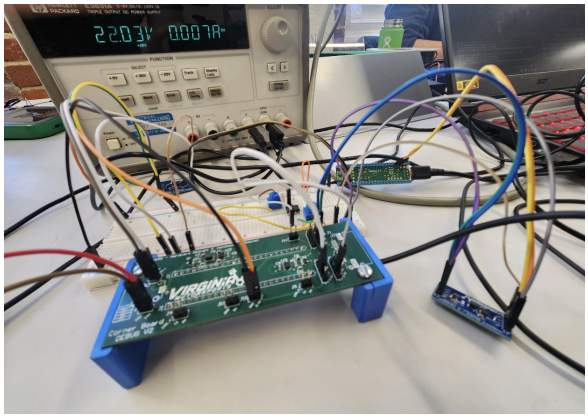


Fig. 18: Teensy 4.1 socket board setup for software testing

3) **Independent Channel Testing**
   - Apply distinct constant voltages (e.g., 1.5V and 2.5V) to two ADC pins.
   - Verify that each ADC channel independently reads the correct input voltage.
   - Repeat the test with variable voltages on both ADC pins to confirm simultaneous functionality.
4) **Full ADC Functionality Validation**
   - Combine constant and variable voltage inputs on multiple ADC channels.
   - Confirm that all ADC pins can independently and simultaneously read and process input voltages without interference.

This test plan is applicable to all three boards utilized in testing to various extents. The transceiver breakout boards only need the transceiver software test section, the debug V2 board (figure 18) tests everything except the MCU power-on sequence, and the final integrated board requires the entire test plan. In testing the regulators and final board, we discovered that the board was very sensitive to ESD and short circuits and had to replace several regulators throughout the testing process. Testing the initial embedded debug board before the final test plan was written revealed that our reflow process was flawed and that the SOC power on the MCU was not connected; this prompted revisions to the test plan to thoroughly inspect the MCU power on sequence and test

points were included in the final design to do so.

Testing the final board revealed an error in step 7, where the DCDC_PSWITCH pin had failed to be driven high. This required further investigation into the bootloader operation. Consulting with Paul Stoffregen, the creator of the Teensy 4.1, we created a revised test plan for bootloader operation.

First, a wire was soldered to TP2 to connect the 3.3V output to the virtual bench to measure the ripple voltage. This was to ensure that the 3.3V signal did not drop below 3V during the ripple, which would cause the bootloader chip to not enable the DCDC.

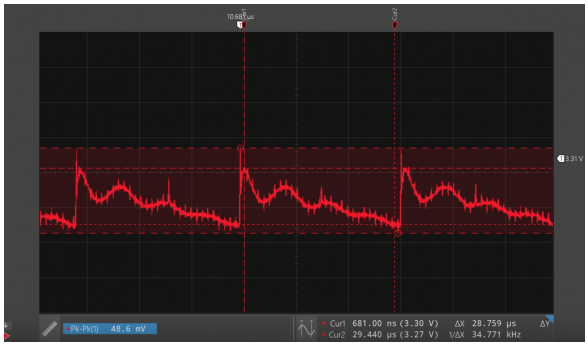


Fig. 19: 50mV pkpk DCDC ripple voltage

The voltage measured in figure 19 does not drop below 3.0V and ripples by 50mV peak to peak, which is within the stability margin of the bootloader code according to Paul Stoffregen. Next, the voltage at pin 10 (bootloader switch pin) of the bootloader was measured, which should be 3.3V. This signifies that the bootloader code is executed as the first lines of code pull pin 10 high. If the bootloader code is stuck at the DCDC pull high stage, then it will not flash the error LED, which is the observed behavior of the embedded MCU board. We measured 3.3V at pin 10 as expected, and also measured that pin 9 power on reset was high so the bootloader is not being held in a reset state. These results, along with Paul's approval of the MCU schematic, indicate a likely soldering error of the BGA chip, so two more boards will be manufactured after submission of this report to check the functionality of the final embedded MCU board. This result validated the team's decision to design a PCB that includes a socket for the Teensy 4.1 to demonstrate software and CAN hardware functionality in the event of embedded MCU failure.

VII. Physical Constraints

The design and manufacturing process faced several constraints, including documentation, software libraries, and project timelines. While most components were accompanied by comprehensive documentation and implementation guidelines, the MCU bootloader startup protocol was only discovered late in the process, complicating the MCU hardware design.

Although Teensy 4.1 provides a CAN design library, it was written in C++, a language unfamiliar to the team, which required additional learning and adaptation. Furthermore, part and PCB orders involved lead times of at least one week, delaying initial board assembly and significantly limiting the time available for redesigns, especially near the end of the semester.

Budget constraints were the most restrictive, as there was no room for professional assembly costs. Hand pick and place caused several delays as soldering mistakes and SMD rework had to be conducted on several boards, the worst culprit being the 196-pin BGA footprint MCU chip. The largest individual cost of the project was PCB orders, with the 6-layer final board costing over $70 for five from JLCPCB. Ordering the full BOM of the corner board with enough components to populate 3 boards and 2 debug boards cost over $130, as seen in Table I.

The tools used in this project included the software Altium Designer, Arduino IDE, and NI Diligent Waveforms, as well as production tools like soldering irons and heat guns, a 3D printer, multimeters, a power bench, a reflow oven, microscope, and breadboards. All PCB designs were performed on Altium, which is used in industry; however, platform limitations and learning processes restricted real-time collaboration and design fluency. Arduino IDE hosted all project software development. Soldering tools, reflow ovens, and microscopes helped populate all our PCBs. Waveforms, multimeters,

breadboards, and the power bench assisted in project testing.

Most of the tools and costs used in product development can be attributed to design and lack of experience. Thus, creating a production version would be extremely feasible. Bulk PCB and component orders bring down part costs, the PCB population is redundant and quick with simple tools, and integrating our project with FSAE vehicles is efficient and modifiable.

VIII. Societal Impact

The development of this board will primarily impact Virginia Motorsports (VME) and potentially other FSAE teams. While the hardware design remains closed-source, the software is publicly accessible on the VME GitHub repository. The board must function accurately to collect suspension data for component validation, but any malfunction poses no safety risk to drivers or spectators at the SAE electric competition in June. Safety-related design constraints are limited to proper mechanical fastening to the car, which will be managed by the customer.

The improved data collection enabled by this board will significantly benefit VME, as suspension data is critical for design validation and enhancement. Unlike mathematical models, real-world data provides a more comprehensive understanding of the suspension system, driving more effective, data-driven design processes. This will elevate the quality of VME's future cars and contribute to the professional growth of team members who graduate with refined engineering skills.

These alumni, entering various engineering sectors, will carry forward the principles of data-driven, experiential learning. This project thus has a broader sociotechnical impact, improving engineering practices and positively influencing industries and society by raising the standard of engineering in the field. Given VME alumni's presence across nearly every engineering sector, the ripple effects of this project will extend to a wide range of sociotechnical systems.

IX. External Standards

As this product interfaces with CAN and a motor vehicle, each part of our project had to comply with automotive and CAN conformance standards. Additionally, using integrated circuits, even without wireless communication, puts the project under FCC jurisdiction for unintentional electromagnetic emission.

AEC-Q100 standard for automotive IC chips [1]

FCC Regulations Part 15, Subpart B - Unintentional Radiator [15]

ISO 16845-1:2016: Road vehicles — Controller area network (CAN) conformance test plan — Part 1: Data link layer and physical signaling [16]

ISO 16845-2:2018: Road vehicles — Controller area network (CAN) conformance test plan — Part 2: High-speed medium access unit — Conformance test plan [17]

X. Intellectual Property Issues

Patents similar to our product include a CAN Message Filtering patent [18], a Local CAN Bus Clock Network patent [19], and a CAN Vehicle Data Acquisition Network Layout patent [20]. All of these patents describe data acquisition network improvements that impact vehicle performance. Judging by our research, our design, which mixes hardware and software, is likely patentable, but we believe open sourcing would be more in the spirit of FSAE.

If this design is ever monetized, we would likely utilize a CERN Open Hardware License [21] to promote open-source modification and use by other FSAE teams. Additionally, we would sell fully assembled boards and hardware kits. FSAE is a design competition, so teams generally prefer to design custom systems themselves. This board is applicable beyond FSAE, however, since it interfaces generically with analog and CAN sensors, which are found in production automobiles, robotics, and information networks.

XI. Timeline

The Gantt chart in Fig. 20 outlines our project timeline, including due dates and holidays. The three main parts of our project are the power and CAN system, the MCU hardware, and the software.

Because the interactions between the three systems are standardized, their research, design, and testing were all performed mostly in parallel. The series of PCB designs were designed once the main components were finalized. The MCU software design was written and tested on the teensy and breadboard with temporary power, inputs, and outputs (using the small transceiver PCB). The power system was tested independently with oscilloscopes. With the final debug PCBs populated, we tested the power system, MCU, transceiver, and simulated analog inputs altogether. Lastly, we connected the sensor board to a temporary wheel chassis instead of to the FSAE car itself. The project conclusion coincides with the last few steps of the project, and demos take place once everything has been completed.

XII. Costs

The cost split between components and boards was about 60% component cost and 40% PCB cost. BGA routing requires small, high-precision drill holes, six-layer PCBs, ENIG finish, and FR4-Tg155 surface finish. This results in a PCB that costs $70 for a minimum order of five, which quickly depletes the budget for iteration.

Given that this project was based on a partnership with the Virginia Motorsports Club, they purchased some items to help us stay within budget. Notably, they purchased transceiver boards that were used for early software testing, miscellaneous components, and automotive-grade connectors for the final board. These costs are not accounted for in the budget in Table I, but in total sum to about $150.

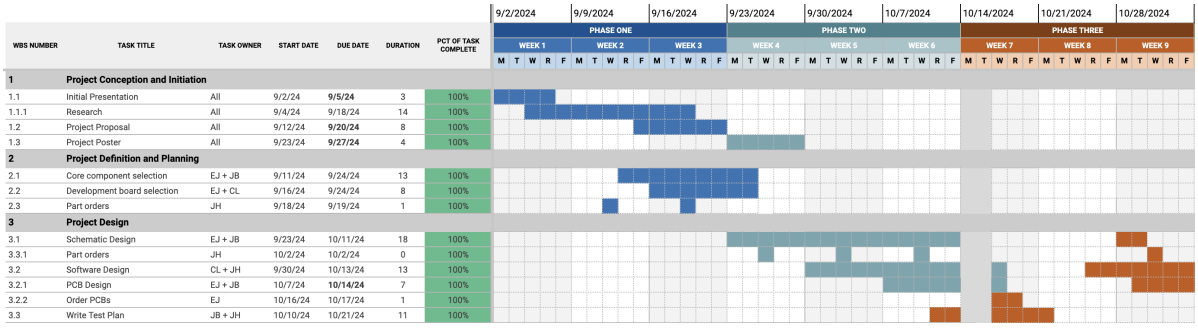
Scaling this project up to 10000 components would roughly halve the per unit price from roughly $36 to around $13. Automation would also greatly reduce the failure rate in building the boards. This is because the main source of failure in the boards was the delicate SMD components, like the CPU, being very difficult to pick and place by hand.

TABLE I: Project Costs using Capstone Budget

Parts Orders	Date	Amount
1	9/11/2024	$31.06
2	9/18/2024	0
3	9/25/2024	$0.00
4	10/2/2024	$50.99
5	10/9/2024	$22.34
6	10/16/2024	0
7	10/23/2024	$22.99
8	10/30/2024	$3.17
9	11/6/2024	$0.00
10	11/13/2024	$131.04
11	11/20/2024	
12	11/27/2024	
Board Orders		
1	10/21/2024	$77.11
2	11/12/2024	$112.36
3		
Other		
Bootloader Order	10/1/2024	$43.63
	TOTALS	
	Total Parts	$261.59
	Total Boards	$189.47
	Total Other	$43.63
	Total	**$494.69**
	Budget Remaining	$5.31

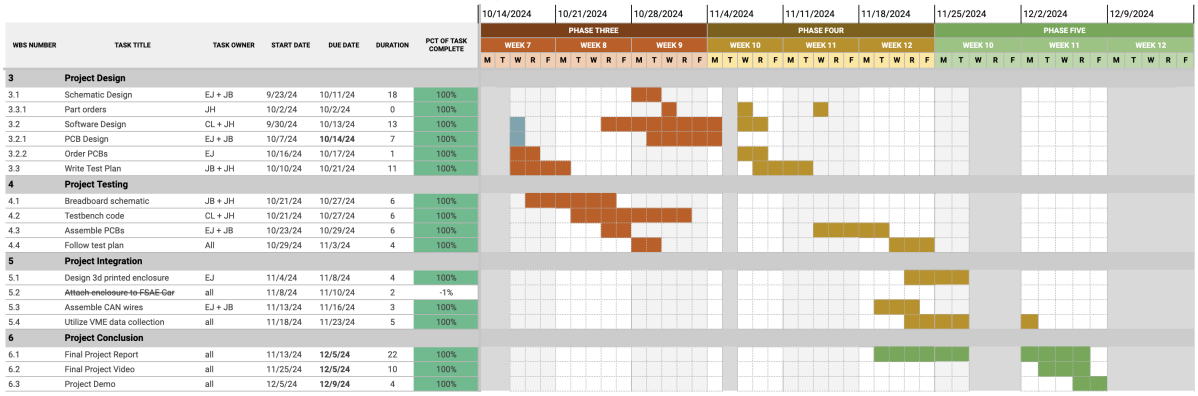
XIII. Final Results

Due to some issues with the DCDC component and the bootloader of the final board, the final test was conducted on the socketed debug board. The main setup of the test is shown in Figure 18. The testing board applies the power regulator and the CAN transceiver from the final board but replaces the Teensy 4.1 components with the actual Teensy board. The final test also includes a secondary Teensy board that will mimic the two temperature sensors that produce CAN 2.0 messages, as well as the Nvidia



WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Project Conception and Initiation**					
1.1	Initial Presentation	All	9/2/24	**9/5/24**	3	100%
1.1.1	Research	All	9/4/24	9/18/24	14	100%
1.2	Project Proposal	All	9/12/24	**9/20/24**	8	100%
1.3	Project Poster	All	9/23/24	**9/27/24**	4	100%
2	**Project Definition and Planning**					
2.1	Core component selection	EJ + JB	9/11/24	9/24/24	13	100%
2.2	Development board selection	EJ + CL	9/16/24	9/24/24	8	100%
2.3	Part orders	JH	9/18/24	9/19/24	1	100%
3	**Project Design**					
3.1	Schematic Design	EJ + JB	9/23/24	10/11/24	18	100%
3.3.1	Part orders	JH	10/2/24	10/2/24	0	100%
3.2	Software Design	CL + JH	9/30/24	10/13/24	13	100%
3.2.1	PCB Design	EJ + JB	10/7/24	**10/14/24**	7	100%
3.2.2	Order PCBs	EJ	10/16/24	10/17/24	1	100%
3.3	Write Test Plan	JB + JH	10/10/24	10/21/24	11	100%

(a) 9/2/2024 - 11/3/2024



WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
3	**Project Design**					
3.1	Schematic Design	EJ + JB	9/23/24	10/11/24	18	100%
3.3.1	Part orders	JH	10/2/24	10/2/24	0	100%
3.2	Software Design	CL + JH	9/30/24	10/13/24	13	100%
3.2.1	PCB Design	EJ + JB	10/7/24	**10/14/24**	7	100%
3.2.2	Order PCBs	EJ	10/16/24	10/17/24	1	100%
3.3	Write Test Plan	JB + JH	10/10/24	10/21/24	11	100%
4	**Project Testing**					
4.1	Breadboard schematic	JB + JH	10/21/24	10/27/24	6	100%
4.2	Testbench code	CL + JH	10/21/24	10/27/24	6	100%
4.3	Assemble PCBs	EJ + JB	10/23/24	10/29/24	6	100%
4.4	Follow test plan	All	10/29/24	11/3/24	4	100%
5	**Project Integration**					
5.1	Design 3d printed enclosure	EJ	11/4/24	11/8/24	4	100%
5.2	~~Attach enclosure to FSAE Car~~	all	11/8/24	11/10/24	2	-1%
5.3	Assemble CAN wires	EJ + JB	11/13/24	11/16/24	3	100%
5.4	Utilize VME data collection	all	11/18/24	11/23/24	5	100%
6	**Project Conclusion**					
6.1	Final Project Report	all	11/13/24	**12/5/24**	22	100%
6.2	Final Project Video	all	11/25/24	**12/5/24**	10	100%
6.3	Project Demo	all	12/5/24	**12/9/24**	4	100%

(b) 10/14/2024 - 12/9/2024

Fig. 20: Project schedules for the different phases of the project.

Jetson that will read the CAN FD messages, a breadboard that has two Square Trimming Potentiometer and a TLC272 LinCMOS Precision Dual Operational Amplifier to mimic linear potentiometer sensors for suspension and wheel speed, two additional VMS created, CAN transceivers to mimic the CAN bus that would be wired throughout the VM25, and a Hewlett Packard E3631A to produce the 24 Volt source mimicking the battery of the car.

The testing code that the second Teensy 4.1 has uploaded can be found in Appendix XVII-D. Using the final program of the main board and the secondary program, the final test has the following steps. The secondary board produces CAN 2.0 messages at some variable frequency that is controlled by the variable sleep cycles. The messages are sent to one of the pins labeled for the CAN 2.0 inputs and are read by the main Teensy asynchronously. The potentiometers and the op-amp unity gain buffer create signals that are sent to the WS and LP port pins on the main board and are also read by the main Teensy synchronously. The main program processes the data to create a unified CAN FD message that is sent through the main output of the main board. Lastly, the secondary board reads the CAN FD message.

An example of the result is shown in Fig. 21. The secondary board prints to the console the sample CAN 2.0 message that is sent to the main board and the CAN FD message sent out, and the main board prints out the CAN 2.0 message sent by the secondary board; the readings from the ADC analog input pins, and the CAD FD to be sent. The two messages are verified if they have the same ID, and the list of data is identical. The data printed is in hexadecimal. There are 2 IDs that the secondary message sends out: 0x7FF and 0x7FE, and this indication is how the two temperature sensors are mimicked. The CAN FD ID is 0x7FD.

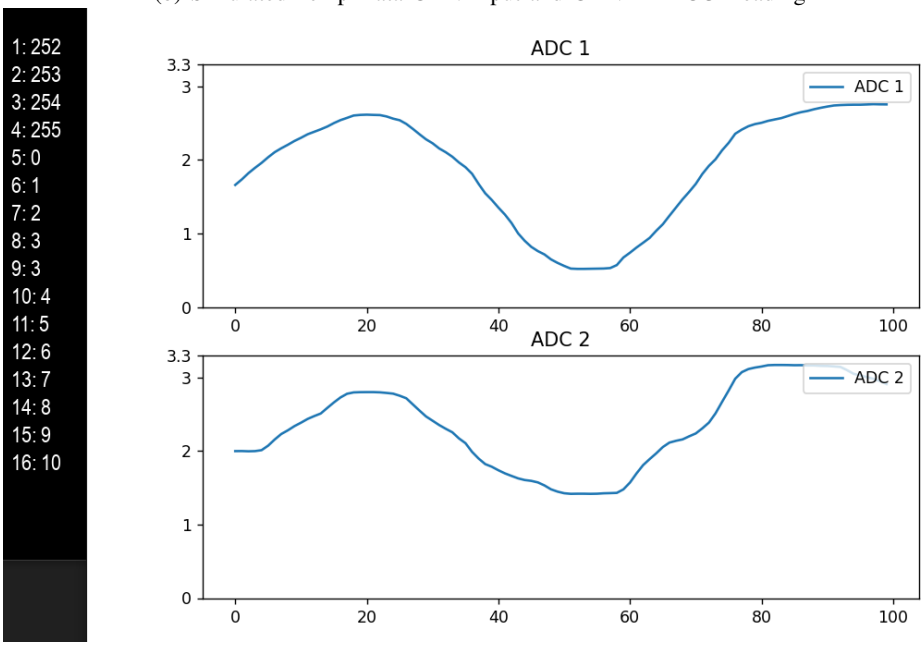
The CAN 2.0 write program creates a sequence of

```
CAN2 MB: 99  ID: 0x7FE  EXT: 0  LEN: 8 DATA: 21 22 23 24 25 26 27 28   TS: 24804
Pin: 19, value ADC0: 2000, value ADC0: 1.6117216117
Pin: 20, value ADC1: 753, value ADC1: 0.6068131868
CANFD MB: 0  ID: 0x7FD  EXT: 0  LEN: 64 DATA: 15 16 17 18 19 1A 1B 1C 1B 1C 1D 1E 1F 20 21 22 7 D0 2 F1   TS: 0

CAN2 MB: 99  ID: 0x7FE  EXT: 0  LEN: 8 DATA: 22 23 24 25 26 27 28 29   TS: 44915
Pin: 19, value ADC0: 1998, value ADC0: 1.6101098901
Pin: 20, value ADC1: 752, value ADC1: 0.6060073260
CANFD MB: 0  ID: 0x7FD  EXT: 0  LEN: 64 DATA: 16 17 18 19 1A 1B 1C 1D 1B 1C 1D 1E 1F 20 21 22 7 CE 2 F0   TS: 0

CAN2 MB: 99  ID: 0x7FF  EXT: 0  LEN: 8 DATA: 31 32 33 34 35 36 37 38   TS: 65026
Pin: 19, value ADC0: 1999, value ADC0: 1.6109157509
Pin: 20, value ADC1: 753, value ADC1: 0.6068131868
CANFD MB: 0  ID: 0x7FD  EXT: 0  LEN: 64 DATA: 16 17 18 19 1A 1B 1C 1D 1F 20 21 22 23 24 25 26 7 CF 2 F1   TS: 0
```

(a) Debug Board Data Out

```
CAN1 MB: 0  ID: 0x7FF  EXT: 0  LEN: 8 DATA: 85 86 87 88 89 90 91 92   TS: 0

CANFD MB: 0  ID: 0x7FD  EXT: 0  LEN: 64 DATA: 4B 4C 4D 4E 4F 50 51 52 55 56 57 58 59 5A 5B 5C 7 D0 2 EE   TS: 15577

CAN1 MB: 0  ID: 0x7FE  EXT: 0  LEN: 8 DATA: 78 79 80 81 82 83 84 85   TS: 0

CANFD MB: 0  ID: 0x7FD  EXT: 0  LEN: 64 DATA: 4E 4F 50 51 52 53 54 55 55 56 57 58 59 5A 5B 5C 7 CF 2 EE   TS: 35698
```

(b) Simulated Temp Data CAN Input and CAN FD ECU Reading



(c) Test Result Visualtization

Fig. 21: Final Test Results

8 numbers for every clock tick as shown with the incrementing values for every new message. The main program combines the two messages and the ADC readings together to create the CAN FD message, where the first 8 values are the same values from the CAN 2.0 message with ID 0x7FF, the next 8 values are the same values from the CAN 2.0 message with id 0x7FE, the next 2 values represent the numerical value of one of the ADC ports, and the last 2 values represent the numerical value of the other ADC port. The ID for the CAN FD message is 0x7FD, and the length of the data buffer is 20 bytes. The test code that was deployed on the secondary Teensy is under Appendix XVII-E.

The results of the final testing show that the software is sound and that the CAN and power systems are functional. Further testing is needed on the embedded MCU hardware but it is believed that improving the soldering

process will fix the board. We tentatively consider our main design goals met and will be confident once the embedded MCU is functional. We currently do not have a way of validating the EMC of the board until VM25 is complete and the corner boards can be tested with the tractive system running.

XIV. Engineering Insights

During this project, we learned a plethora of technical skills. Research honed our component selection and reference sourcing. Exploring new topics forced us to problem-solve using documentation, learn and apply knowledge quickly, lean on each other for help, and take moments to breathe. Honing our teamwork was an essential part of project success. That involved consistently checking one another's work and rampant communication, delegation, and flexibility.

Additionally, we learned a series of important lessons. When ordering USB-C connectors, accidentally copying the wrong part number resulted in ordering a 1000-component reel instead of 10 parts on cut tape. Poor inspection of data sheets and requirements resulted in ordering a handful of wrong parts. During testing, poor labeling of header connector polarity caused us to mount the Teensy Microcontroller on the debug board backward. Connecting the wrongly-populated board to power resulted in the destruction of our Teensy, fried regulators, and visible smoke. During the same mishap, accidentally touching the multimeter leads together caused an onboard capacitor to explode (Fig. 22). Lastly, on multiple occasions, not closely following spec and documentation recommendations resulted in several design and redesign delays.

The importance of proper equipment also proved evident. The original set of transceiver boards and the first revision of the debug board were soldered with the old reflow oven at Lacy Hall, which was just a resistive-heated toaster oven with old solder paste. The ICs all flowed incorrectly, proving that better equipment would be needed to properly assemble the SMD boards.

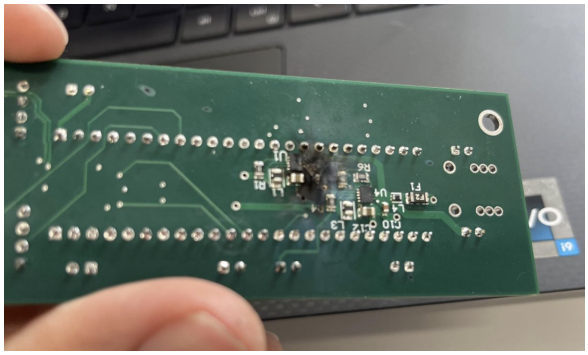


Fig. 22: Debug Board Capacitor Explosion

Fig. 23: Whizoo Controleo 3 reflow oven

We encouraged the motorsports club to purchase a proper reflow oven; the oven chosen was the Whizoo Controleo 3 oven shown in Figure 23. The oven is a modified toaster oven that includes an extra heating element, proper thermocouples, improved insulation, and PID control to follow manufacturer-recommended reflow profiles instead of direct heating in the oven. Some time was taken from design to assemble the reflow oven and dial-in reflow profiles.

For future capstone students, the advice we can share is that catastrophes and mistakes will happen, but learning to adapt and problem-solving are the most important parts of the project process.

XV. Future Work

Another iteration on the board is needed before final integration on VM25; this will be conducted over winter break and will include minor component changes and several additional features.

New features will include implementing the real-time clock (RTC) feature by including a 32.768 Hz oscillator and 20pF filter caps. This feature will improve DAQ by adding the ability to send accurate timestamps along with the transmitted data starting from the power-up time. Synching with actual time is possible by transmitting the actual clock via LoRa to the Nvidia Jetson and sending the time stamp to the corner boards over the CAN bus.

Next, the unused five pins on the 19-pin GT32 connector will be connected to the MCU to support another lin pot and one two-wire switch, giving further flexibility to the board's use cases.

Additionally, a hard-learned lesson in this project is that the smaller the IC, the more sensitive it is to shorts, reverse polarity, ESD, and other calamities. As of this writing, the DCDC kill count is set at five regulators. The diagnosis points to the inductor size in the regulator control loops. The chosen ICs contain short protection, but the recommended inductor sizes have saturation currents of around 1A. Our inductors, while they have proper inductance values, have max DC current ratings of only 350mA. Beefing up the size of these inductors should trigger the IC's short protection and keep the regulators stable in the case of a power short.

Small changes to the board will be conducted to further reduce its size. Switching to a smaller tactile switch will allow the board to shrink to 3" by 1" as desired in the performance requirements. Further optimization of component placement and routing can further compact the board, which is just limited by the width of the 19-pin GT32 connector.

XVI. References

References

[1] "Failure mechanism based stress test qualification for integrated circuits in automotive applications," 2023. [Online]. Available: http://www.aecouncil.com/Documents/AEC_Q100_Rev_J_Base_Document.pdf (visited on 09/20/2024).

[2] *Hardware development guide for the mimxrt1050 /mimxrt1060 processor*, MIMXRT105060HDUG, Rev 0, NXP Semiconductors, Aug. 2018.

[3] P. Stoffregen, *PJRC Store*. [Online]. Available: https://www.pjrc.com/store/ic_mkl02_t4.html (visited on 12/06/2024).

[4] *GT32_10p_1_5h_cl0782_0001_1_00_catalog_d49392_en-1927810*. [Online]. Available: https://www.mouser.com/datasheet/2/185/GT32_10P_1_5H_CL0782_0001_1_00_Catalog_D49392_en-1927810.pdf (visited on 12/06/2024).

[5] *Multichannel brake ir temperature sensor*, IRTS-V3, Izze Racing, 2023. [Online]. Available: https://www.izzeracing.com/products/ewExternalFiles/IZZE_IRTS_V3_BRAKE_Datasheet.pdf.

[6] *Multichannel brake ir temperature sensor*, IRTS-60-V3, Izze Racing, 2023. [Online]. Available: https://www.izzeracing.com/products/ewExternalFiles/IZZE_IRTS_V3_BRAKE_Datasheet.pdf.

[7] *Tps6217x 28-v, 0.5-a step-down converter with sleep mode*, TPS62175, TPS62177, Texas Instruments, Oct. 2012.

[8] *Tcan341x 3.3-v can fd transceivers with standby mode and ±58 v bus standoff*, TCAN3413, TCAN3414, Texas Instruments, Mar. 2023.

[9] Andre Ix, Hartmut Habben, and Caroline Volmari, "Rules and recommendations for in-vehicle CAN networks," NXP Semiconductors, Tech. Rep., 2012.

[10] *CAN-Bus: Designing CAN-Bus Circuitry*, en. [Online]. Available: https://resources.altium.com/p/can-bus-designing-can-bus-circuitry (visited on 12/06/2024).

[11] *Kinetis kl04 32 kb flash*, KL04P48M48SF1, Rev 4, Freescale Semiconductor, Inc., Mar. 2014.

[12] *I.mx rt1060 crossover processors for consumer products*, IMXRT1060CEC, Rev 4, NXP Semiconductors, Apr. 2024.

[13] P. Villanueva, *Teensy 4/3.x/LC ADC implementation*, original-date: 2013-10-20T15:01:07Z, Sep. 2024. [Online]. Available: https://github.com/pedvide/ADC (visited on 12/06/2024).

[14] A. Brewer, *FlexCAN (CAN 2.0 / CANFD) Library for Teensy 3.x and 4.0*, original-date: 2019-01-14T02:14:21Z, Nov. 2024. [Online]. Available: https://github.com/tonton81/FlexCAN_T4 (visited on 12/06/2024).

[15] *eCFR :: 47 CFR Part 15 Subpart B – Unintentional Radiators*. [Online]. Available: https://www.ecfr.gov/current/title-47/chapter-I/subchapter-A/part-15/subpart-B?toc=1 (visited on 09/20/2024).

[16] *ISO 16845-1:2016(en), Road vehicles — Controller area network (CAN) conformance test plan — Part 1: Data link layer and physical signalling*. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:16845:-1:ed-1:v1:en (visited on 09/20/2024).

[17] *ISO 16845-2:2018(en), Road vehicles — Controller area network (CAN) conformance test plan — Part 2: High-speed medium access unit — Conformance test plan*. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:16845:-2:ed-2:v1:en (visited on 09/20/2024).

[18] M. H. J. V. D. MAAS, "Controller area network (can) message filtering," en, EP3206361B1, Feb. 2019. [Online]. Available: https://patents.google.com/patent/EP3206361B1/en?q=(vehicle+bus+node)&oq=vehicle+can+bus+node&page=2 (visited on 12/07/2024).

[19] A. S. Vijayaraj and M. M. Berthold, "Slave Node For Can Bus Network," US 10277385 B1, May 2018. [Online]. Available: https://lens.org/135-023-500-962-124.

[20] O. Krieger and A. Meier, "Verfahren zur übertragung von daten über einen kommunikationskanal, entsprechend ausgelegte vorrichtung und kommunikationsschnittstelle sowie entsprechend ausgelegtes computerprogramm," de, EP3763091B1, Jul. 2022. [Online]. Available: https://patents.google.com/patent/EP3763091B1/en?q=(vehicle+bus+node)&oq=vehicle+can+bus+node&page=1 (visited on 12/07/2024).

[21] *Home — CERN Open Hardware Licence*. [Online]. Available: https://cern-ohl.web.cern.ch/home (visited on 12/06/2024).

XVII. Appendix

A. MCU schematic

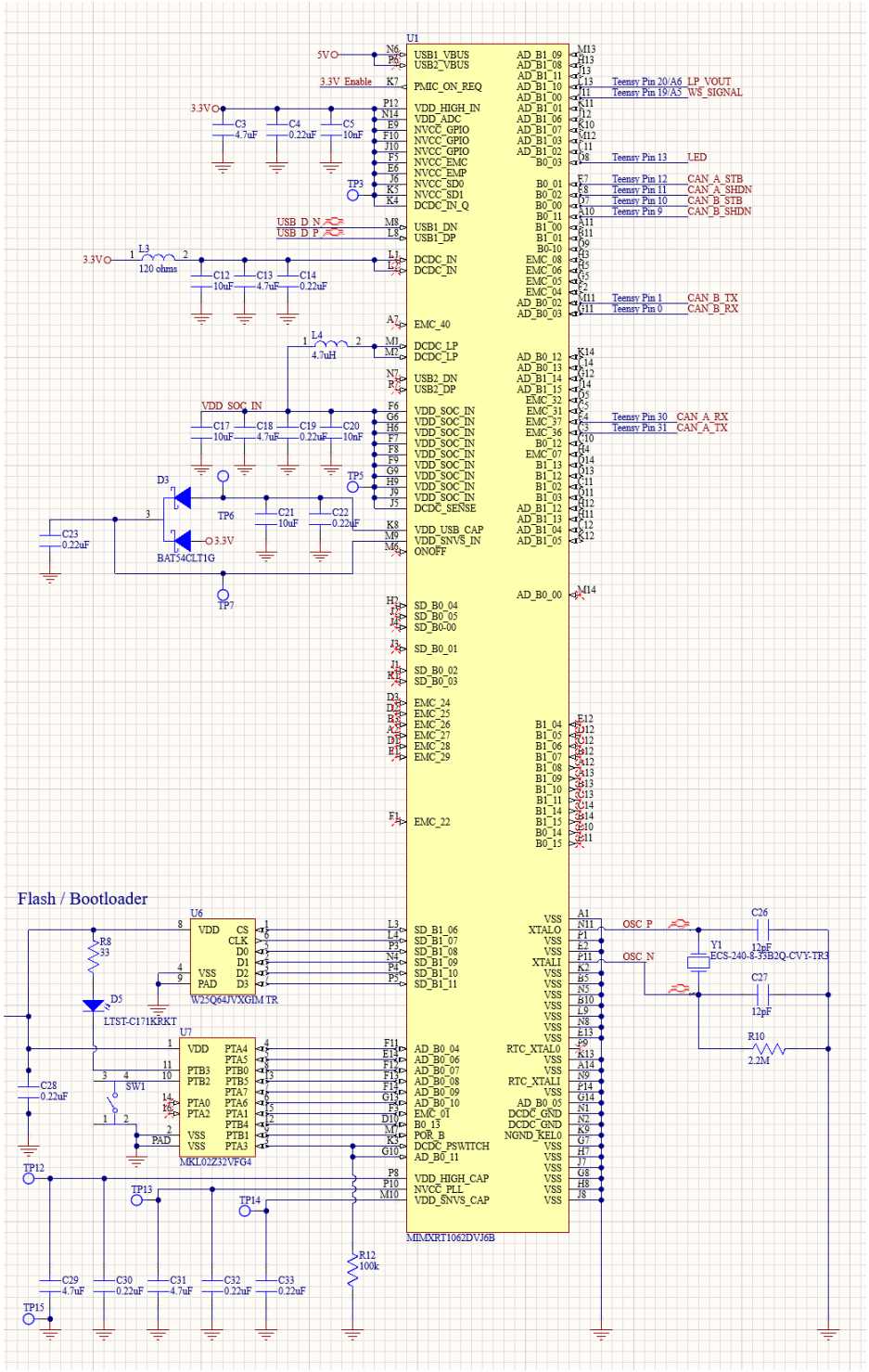


Fig. 24: Total MCU schematic

B. Transciever and connector schematic

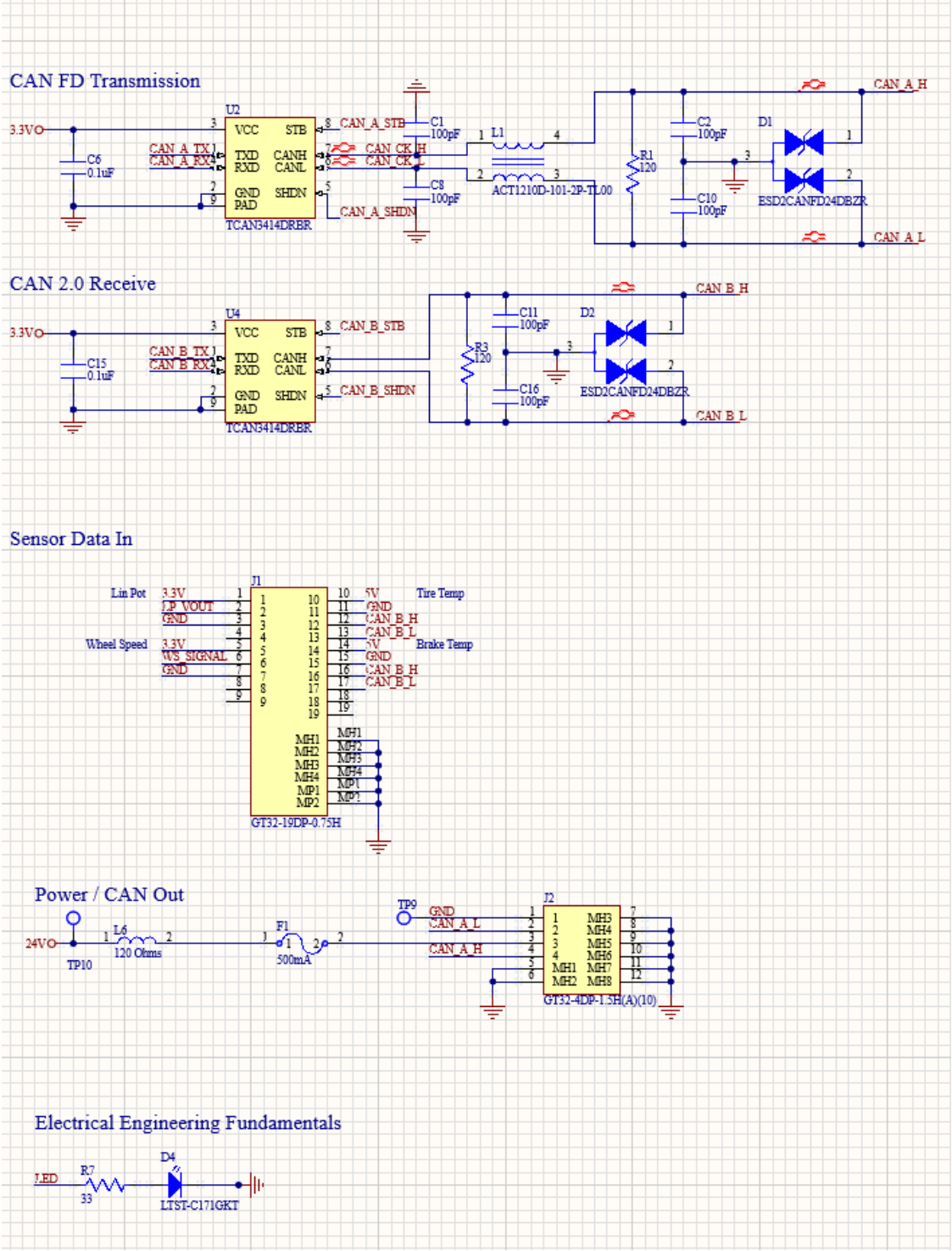


Fig. 25: Transciever, connector, and LED schematic

C. Final board BOM

TABLE II: Bill of Materials

Name	Designator	Quantity	Unit Price	Subtotal
CAP 0603 100pF	C1, C2, C8, C10, C11, C16	6	0.025	0.25
CAP 0402 4.7uF	C3, C13, C18, C29, C31	5	0.37	1.85
CAP 0402 0.22uF	C4, C14, C19, C22, C23, C30, C32, C33	8	0.024	0.24
CAP 0402 10nF	C5, C20	2	0.006	0.012
CAP 0603 0.1uF	C6, C15	2	0.018	0.036
CAP 0805 22uF	C7, C25	2	0.62	1.24
CAP 0603 2.2u	C9, C24	2	0.11	0.22
CAP 0603 10uF	C12, C17, C21	3	0.31	0.93
CAP 0402 12pF	C26, C27	2	0.1	0.2
CAP 0805 0.22uF	C28	1	0.016	0.016
ESD2CANFD24DBZR	D1, D2	2	0.35	0.7
BAT54CLT1G	D3	1	0.016	0.016
LTST-C171GKT	D4	1	0.198	0.198
LTST-C171KRKT	D5	1	0.19	0.19
1206L050/24WR	F1	1	0.868	0.868
GT32-19DP-0.75H	J1	1	3.72	3.72
GT32-4DP-1.5H(A)(10)	J2	1	1.52	1.52
USB4105-GF-A	J3	1	0.52	0.52
ACT1210D-101-2P-TL00	L1	1	1.64	1.64
MLZ2012M100WT000	L2, L5	2	0.11	0.22
MPZ1608S121ATAH0	L3, L6	2	0.1	0.2
MLZ1608E4R7MT000	L4	1	0.12	0.12
RES 0805 120	R1, R3	2	0.006	0.012
RES 0805 100k	R2, R4, R12	3	0.016	0.16
RES 0805 787k	R5	1	0.1	0.1
RES 0805 150k	R6	1	0.1	0.1
RES 0805 33	R7, R8	2	0.006	0.012
RES 0805 5.1k	R9, R11	2	0.016	0.16
RES 0805 2.2M	R10	1	0.1	0.1
1825967-2	SW1	1	0.22	0.22
MIMXRT1062DVJ6B	U1	1	12.24	12.24
TCAN3414DRBR	U2, U4	2	1.42	2.84
TPS62177DQCT	U3	1	1.7	1.7
TPS62175DQCR	U5	1	0.56	0.56
W25Q64JVXGIM TR	U6	1	1.18	1.18
MKL02Z32VFG4	U7	1	1.44	1.44
ECS-240-8-33B2Q-CVY-TR3	Y1	1	0.36	0.36

D. Corner Board Code

```
#include <ADC.h>
#include <ADC_util.h>

#include <FlexCAN_T4.h>

const int readPin = A5;
const int readPin2 = A6;

ADC *adc = new ADC();

FlexCAN_T4<CAN2, RX_SIZE_256, TX_SIZE_16> Can1;
FlexCAN_T4FD<CAN3, RX_SIZE_256, TX_SIZE_16> Can2;

void setup() {

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(readPin, INPUT_DISABLE);
  pinMode(readPin2, INPUT_DISABLE);

  Serial.println("Begin setup");
  Serial.begin(115200);

  adc->adc0->setAveraging(16);
  adc->adc0->setResolution(16);
  adc->adc0->setConversionSpeed(
      ADC_CONVERSION_SPEED::MED_SPEED);
  adc->adc0->setSamplingSpeed(
      ADC_SAMPLING_SPEED::MED_SPEED);

  adc->adc1->setAveraging(16);
  adc->adc1->setResolution(16);
  adc->adc1->setConversionSpeed(
      ADC_CONVERSION_SPEED::MED_SPEED);
  adc->adc1->setSamplingSpeed(
      ADC_SAMPLING_SPEED::MED_SPEED);

  Can1.begin();
  Can1.setBaudRate(1000000);
  //following 3 lines are what enables interrupts, IMPORTANT: does not work on a CAN FD inp
  Can1.enableFIFO();
  Can1.enableFIFOInterrupt();
```

```
  Can1.onReceive(canSniff);

  Can2.begin();
  Can2.setRegions(64);
  CANFD_timings_t config;
  Can2.setBaudRate(config);

  Serial.println("End setup");
}

int value; //raw ADC 1 output
int value2; //raw ADC 2 ouput
//to convert value to voltage, use "value * 3.3 / adc->adc0->getMaxValue()"
uint32_t firstID = 0x7FE; //ID of first CAN 2.0 signal, placed into CAN FD buffer from 0-7
uint32_t secondID = 0x7FF; //ID of second CAN 2.0 signal, placed into CAN FD buffer from 8-
uint8_t CANL[8]; //stores data of firstID
uint8_t CANR[8]; //stores data of secondID

//interrupt based on Can1 receiving a message
void canSniff(const CAN_message_t &msg) {

  Serial.print("CAN2 ");
  Serial.print("MB: "); Serial.print(msg.mb);
  Serial.print("  ID: 0x"); Serial.print(msg.id, HEX );
  Serial.print("  EXT: "); Serial.print(msg.flags.extended );
  Serial.print("  LEN: "); Serial.print(msg.len);
  Serial.print(" DATA: ");
  for ( uint8_t i = 0; i < 8; i++ ) {
    Serial.print(msg.buf[i]); Serial.print(" ");
  }
  Serial.print("  TS: "); Serial.println(msg.timestamp);

  //updates buffer corresponding to correct ID
  if (msg.id == firstID){
    for ( uint8_t j = 0; j < 8; j++ ){
      CANL[j] = msg.buf[j];
    }
  }else if (msg.id == secondID){
    for ( uint8_t j = 0; j < 8; j++ ){
      CANR[j] = msg.buf[j];
    }
  }
```

```
//ADC Reads
value = adc->adc0->analogRead(readPin);

Serial.print("Pin: ");
Serial.print(readPin);
Serial.print(", value ADC0: ");
Serial.print(value);
Serial.print(", value ADC0: ");
Serial.println(value * 3.3 / adc->adc0->getMaxValue(), DEC);

value2 = adc->adc1->analogRead(readPin2);

Serial.print("Pin: ");
Serial.print(readPin2);
Serial.print(", value ADC1: ");
Serial.print(value2);
Serial.print(", value ADC1: ");
Serial.println(value2 * 3.3 / adc->adc1->getMaxValue(), DEC);


//CAN FD write
Can2.events();
CANFD_message_t msgfd;
msgfd.brs = 0;
msgfd.id = 0x7FD;
msgfd.len = 64;

//writes both stored buffers to message buffer
for ( uint8_t j = 0; j < 8; j++ ){
  msgfd.buf[j] = CANL[j];
  msgfd.buf[j+8] = CANR[j];
}

//writes both ADC values to message buffer
msgfd.buf[16] = value>>8;
msgfd.buf[17] = value;
msgfd.buf[18] = value2>>8;
msgfd.buf[19] = value2;

Can2.write(msgfd);
Serial.print("CANFD ");
Serial.print("MB: "); Serial.print(msgfd.mb);
Serial.print("  ID: 0x"); Serial.print(msgfd.id, HEX );
```

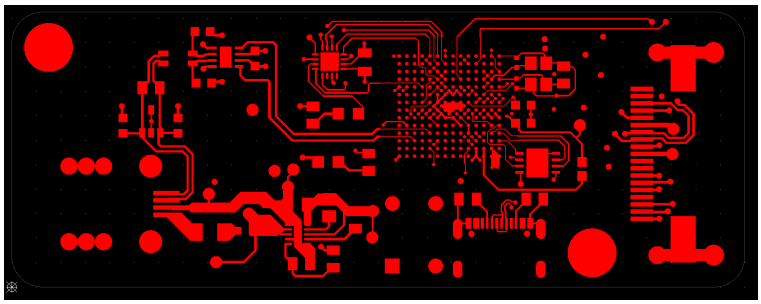
```
  Serial.print("  EXT: "); Serial.print(msgfd.flags.extended );
  Serial.print("  LEN: "); Serial.print(msgfd.len);
  Serial.print(" DATA: ");
  for ( uint8_t i = 0; i < 20; i++ ) {
    Serial.print(msgfd.buf[i], HEX); Serial.print(" ");
  }
  Serial.print("  TS: "); Serial.println(msgfd.timestamp);

  Serial.println();
}


void loop() {
  Can1.events();
}
```

E. ECU / CAN 2.0 Emulator code

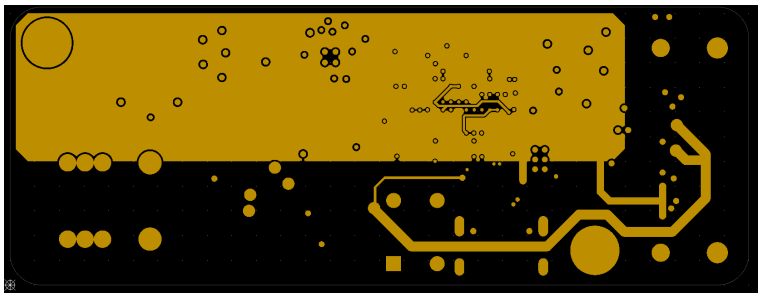
```
#include <FlexCAN_T4.h>

FlexCAN_T4<CAN1, RX_SIZE_256, TX_SIZE_16> Can0;
FlexCAN_T4FD<CAN3, RX_SIZE_256, TX_SIZE_16> Can2;

void setup() {

  Serial.println("Begin setup");
  Serial.begin(115200);

  Can0.begin();
  Can0.setBaudRate(1000000);

  Can2.begin();
  Can2.setRegions(64);
  CANFD_timings_t config;
  config.baudrateFD = 2000000;
  Can2.setBaudRate(config);

  Serial.println("End setup");
}

//to convert value to voltage, use "value * 3.3 / adc->adc0->getMaxValue()"
uint32_t firstID = 0x7FE; //ID of first CAN 2.0 signal, placed into CAN FD buffer from 0-7
uint32_t secondID = 0x7FF; //ID of second CAN 2.0 signal, placed into CAN FD buffer from 8-
uint32_t sleepCycles = 100000; //set how many clock cycles occur between writes
uint32_t cycleIndex = 0;
uint8_t offset = 0; //for testing, delete when no longer writing initial message

void loop() {
  if (cycleIndex == sleepCycles){
    cycleIndex=0;
    offset++;

    // CAN 2.0 write (for testing) (mimic temp readings)
    CAN_message_t msgwrite;
    int leftRight = random(0,2);
    if (leftRight == 0){
      msgwrite.id = firstID;
      for ( uint8_t i = 0; i < 8; i++ ) msgwrite.buf[i] = i + 1 + offset;
      Can0.write(msgwrite);
```

```
    }else if (leftRight == 1){
      msgwrite.id = secondID;
      for ( uint8_t i = 0; i < 8; i++ ) msgwrite.buf[i] = i + 9 + offset;
      Can0.write(msgwrite);
    }
    Serial.print("CAN1 ");
    Serial.print("MB: "); Serial.print(msgwrite.mb);
    Serial.print("  ID: 0x"); Serial.print(msgwrite.id, HEX );
    Serial.print("  EXT: "); Serial.print(msgwrite.flags.extended );
    Serial.print("  LEN: "); Serial.print(msgwrite.len);
    Serial.print(" DATA: ");
    for ( uint8_t i = 0; i < 8; i++ ) {
      Serial.print(msgwrite.buf[i]); Serial.print(" ");
    }
    Serial.print("  TS: "); Serial.println(msgwrite.timestamp);
    Serial.println();
  } else {
    cycleIndex++;
  }

  // CAN FD Read (for testing)
  CANFD_message_t msg;
  if(Can2.read(msg)){
    Serial.print("CANFD ");
    Serial.print("MB: "); Serial.print(msg.mb);
    Serial.print("  ID: 0x"); Serial.print(msg.id, HEX );
    Serial.print("  EXT: "); Serial.print(msg.flags.extended );
    Serial.print("  LEN: "); Serial.print(msg.len);
    Serial.print(" DATA: ");
    for ( uint8_t i = 0; i < 20; i++ ) {
      Serial.print(msg.buf[i], HEX); Serial.print(" ");
    }
    Serial.print("  TS: "); Serial.println(msg.timestamp);
    Serial.println();
  }
}
```

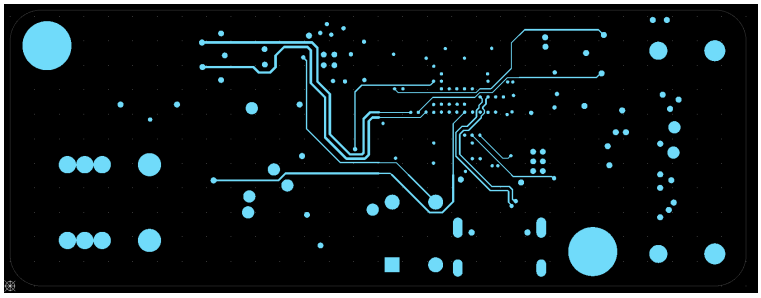
F. Final PCB Routing



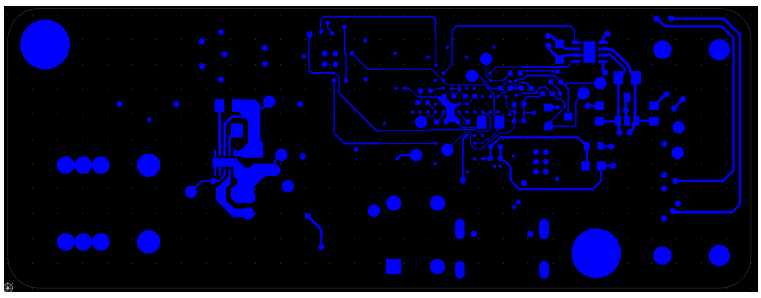
(a) Top Signal Layer



(b) Power Layer



(c) Intermediate Signal Layer



(d) Bottom Signal Layer

Fig. 26: Gerber files for final board PCB. Ground planes not shown