Analog Gauge Driver with CAN Interface

COMP 490 Final Report

December 16, 2025

Sam Anthony
Concordia University
Montréal Québec, Canada
Student ID: 40271987
Email: sam@samanthony.xyz

Supervised by Hovhannes Harutyunyan, PhD
Concordia University
Department of Computer Science and Software Engineering
Montréal Québec, Canada
Email: haruty@encs.concordia.ca

***Abstract*—An electronic device is developed that allows analog needle gauges to be retrofitted into a car's EMS (engine management system) while leveraging the capabilities of its CAN (controller area network) bus. The proceedings of the project are reported-on, including hardware design, firmware and software development, and testing. Finally, the outcomes of the project are assessed against the initial goals, with the overall conclusion being positive.**

CONTENTS

I. INTRODUCTION

Combustion engines, such as those used to power passenger cars, require precise control over their operation in order to run efficiently and reliably. Since the early 1970s, car engines have been electronically controlled by an EMS (engine management system) [37]. An EMS is an embedded system consisting of an ECU (electronic control unit), sensors, and actuators. The actuators include fuel injectors and spark plugs. The sensors measure crankshaft angle, intake manifold pressure, coolant temperature, and so on. The ECU features a microcontroller that uses feedback from these sensor data to operate the actuators, thus allowing the engine to run.

(a) (b)

Fig. 1. (a) Analog needle gauge [33] and; (b) skeuomorphic digital display emulating analog gauges [32].

Sensor data are used not only by the ECU, but also by a display system mounted in the cabin that allows the driver to monitor the engine's health. The display system is typically a set of gauges showing, for instance, engine speed, oil pressure, and coolant temperature, among other things.

The display system must visually encode sensor data and convey them to the driver. Each datum represents the instantaneous value of a continuous quantity—speed, temperature, pressure, etc. The data are most effectively represented by a graduated radial analog scale with the instantaneous value marked on said scale [28]. The graduated scale takes advantage of vernier acuity: our ability to discern slight misalignment between line segments [34], while the radial marker leverages the hypercolumnar acuity of vision: our ability to detect minute changes in angle of line segments [19]. Simply put, an analog needle gauge is the best way to display information to the driver. That is why even modern digital display systems are often skeuomorphs of analog gauges, as seen in Fig. 1.

As engine performance increases, the operating window where it will run reliably shrinks, necessitating even tighter control. This spurs demand for even more sensor data, both for the ECU—to precisely regulate the running of the engine—and for the driver, to monitor the engine's health and to ensure that it stays within its safe operating window.

This is especially true in racing, where the engine must be

Fig. 2. Analog gauges fitted to 1987 Nissan Skyline GTS-R Group A race car [2].

fine-tuned to its limits while remaining reliable for the duration of its life. Race teams often fit additional sensors and gauges to the car in order to monitor the health of the engine (Fig. 2).

All these sensor data must somehow be transmitted throughout the car; a computer network handles this task well. CAN (controller area network) [12] is ubiquitous: introduced by Bosch in the early 1990s and standardized by ISO 11898 [18], all cars sold in the United States are required to be equipped with a CAN bus [4].

Most modern digital display systems, such as the one shown in Fig. 1(b), come with a CAN interface, allowing them to display a practically unlimited array of information from whatever sensors may be equipped to the car. On the other hand, analog gauges have largely been abandoned by manufacturers, as digital panels can display myriad information at a fraction of the cost of an equivalent set of analog gauges. However, some, including myself, prefer a genuine analog gauge as opposed to a facsimile displayed on a screen. The trouble is, most analog gauges lack a CAN interface, with only a few companies making CAN-enabled analog gauges, mostly for industrial applications [38].

Those who wish to install an analog gauge typically do so by installing a sensor on the engine and running a wire directly to the gauge's analog input—this is the setup recommended by most gauge manufacturers [31]. While this does work, it is less than ideal for several reasons.

Firstly, running a wire through the engine bay leaves the signal vulnerable to EMI (electromagnetic interference) as it travels near noisy components like the ignition coils. This can cause signal integrity issues that can otherwise be avoided by transporting the signal via the CAN bus that the car is likely to have already—CAN uses a CRC (cyclic redundancy check) to provide data integrity.

Secondly, the EMS already features a host of sensors, and installing another for the gauge often leads to duplication. While this may appear to provide redundancy, it in fact does not. On the contrary, it reduces reliability and increases complexity by introducing additional failure points into the system. A properly engineered solution would integrate the gauge into the EMS, allowing them to share sensor data, thus reducing the complexity of the system.

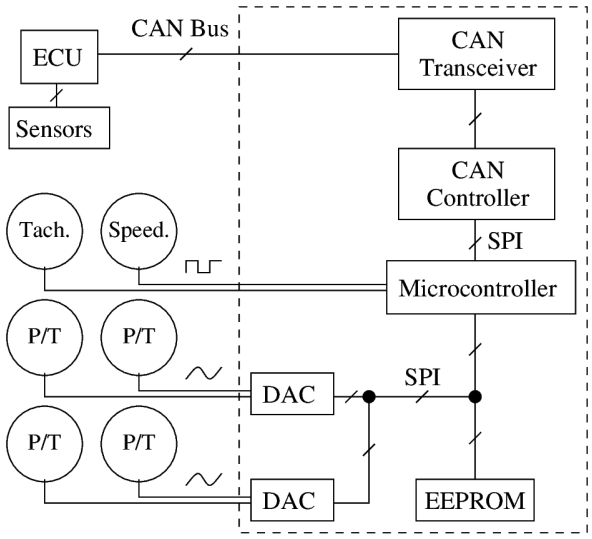

Fig. 3. System diagram.

This brings us finally to the subject of the report: a device that allows analog gauges to be retrofitted into a car while leveraging the capabilities of the CAN bus already present in the vehicle.

II. Project Overview

At a high level, the device acts as an interface between the car's EMS and the analog gauges in the cabin. It receives sensor data encoded in CAN frames from the ECU, and transforms them into signals that the gauges can understand: such as a variable-frequency square wave for a tachometer, or a 0–5V analog signal for a pressure gauge. Thus, it is referred to as an *analog gauge driver with a CAN interface*.

A block diagram of the system is shown in Fig. 3. The device has a CAN controller and transceiver for sending and receiving CAN frames. A microcontroller decodes and extracts sensor data from the frames, and communicates the data to the peripherals which drive the gauges. Two dual DACs (digital-to-analog converters) generate signals for the four analog outputs. These can drive up to four temperature or pressure gauges, or any gauge that accepts a 0–5V analog input. The microcontroller's timers produce two variable-frequency square waves which drive the tachometer and speedometer.

As it is intended to be a generic part that can be retrofitted to existing cars, the device must work with any combination of sensors, gauges, and CAN encoding schemes. Thus, it is user-programmable, allowing it to be installed in any system. The user-calibration is programmed via CAN and stored in non-volatile memory, namely an EEPROM (electrically erasable programmable read-only memory) chip.

The project consisted of four tasks that were completed concurrently over the course of the term. The timeline of the project is shown in Fig. 4. The tasks to be completed

were hardware design, firmware development, software development, and testing. Each of these topics are covered in the subsequent sections.

III. Hardware

The hardware is a PCB (printed circuit board) hosting a set of ICs (integrated circuits)—the microcontroller and peripherals listed in the previous section (Fig 3)—and some supporting passive components: resistors, capacitors, and an inductor.

A. Component selection

A car is a harsh environment. The device is subject to vibration, EMI (electromagnetic interference), and large variations in temperature. To increase reliability, AEC-certified parts were chosen wherever possible [1].

Microcontroller: The microcontroller is at the heart of the design. A Microchip PIC16F1459 was chosen for its simplicity, robustness, feature-set, and low cost[1] [29]. It is an 8-bit microcontroller with an SPI peripheral for communicating with the other ICs, and timers for generating the tachometer and speedometer signals. The PIC is a proven design that Microchip recommends for automotive applications. It is available in a DIP (dual in-line) package, making it convenient for breadboard prototyping (Fig. 5).

CAN controller and transceiver: The CAN controller handles the reception and transmission of CAN frames. It is complemented by a CAN transceiver which acts as a buffer between the controller's logic level signals and the differential signals on the bus. A Microchip MCP2515 controller [24] and MCP2561 transceiver [25] were chosen to fill these roles. The MCP2515 supports CAN 2.0B up to 1Mbps and it has an SPI interface for communicating with the PIC. Both these chips are available in DIP packages for breadboard prototyping.

EEPROM: The EEPROM stores the user-calibration that defines how sensor signals are encoded in CAN frames, as well as tables that map sensor readings to output signal values. A Microchip 25LC160C EEPROM was selected [3]. It has an SPI interface, and its 2KiB of space is sufficient to store the calibration.

DACs: Two dual-channel DACs (digital-to-analog converters) drive the four analog outputs of the board. Based on the characteristics of commonly-used pressure and temperature sensors [30], it was determined that a resolution of 15mV/step was required. Given the operating voltage of 5V, this meant that the DACs would need at least $5\text{V}/15\text{mV} \approx 333$ steps of resolution. Thus, an 8-bit DAC with 256 steps would have been insufficient, and so a 10-bit DAC was selected: namely a Microchip MCP4912[2] [26]. The MCP4912 is a dual-channel 10-bit DAC, so two of them are required to drive the board's four analog outputs.

B. Power supply

Standard automotive electrical systems operate at a nominal voltage of around 13.7V, but can swing between 9 and 16V. The voltage supply often has a strong pulsating component as well, known as ripple. The board's ICs require a stable 5V to operate reliably. Hence, the board's power supply is very robust to tolerate the wide input voltage range and to rectify the ripple.

The voltage drop $V_{\text{Drop}} = V_{\text{In}} - V_{\text{Out}}$ is $16\text{V} - 5\text{V} = 11\text{V}$ in the worst case. This ruled out the use of a linear regulator, since it would dissipate too much power—the power dissipation of a linear regulator is linear in V_{Drop}: $P = V_{\text{Drop}} \times I$. The load current was estimated to be 250mA at most [9], so a linear regulator would dissipate up to 2.75W. That amount of power from a single chip would be difficult to cool. Therefore, a linear regulator was deemed inadequate, necessitating the use of a switching regulator instead.

The downside of a switching regulator is that it introduces noise and ripple into the PDN (power distribution network). To isolate the other components, a two-stage PDN is used.

The first stage is the switching regulator itself, also known as a buck converter. It drops the voltage from the car's 9–16V down to an intermediate 7V level.

Just like a buck converter, switching digital ICs introduce noise into the PDN. Therefore, the second stage is split between two regulators in order to keep the analog and digital circuitry separate. It is composed of two ST L78M05AB [23] LDOs (low-dropout regulators) that bring the voltage from 7V down to the final 5V that the ICs require.

A diagram of the PDN is shown in Fig. 6.

The buck converter in the first stage is a Texas Instruments TPS5430 [36]. It is accompanied by some RC and LC networks that set the output voltage level and dampen the output ripple. Unfortunately, the passive component values were calculated incorrectly, which resulted in the buck converter outputting the wrong voltage. This mistake is discussed further in §VI.

C. PCB design and manufacture

The schematic (Fig. 7) and PCB (Figs. 8, 9, and 10) were designed in KiCad [22]. JLCPCB manufactured the PCB [21].

The board was laid out with PCB design best-practices in mind. It is a 4-layer design that uses a combination of surface-mount (SMD) and through-hole (THT) components[3]. The top and bottom layers are for signals, and the two middle layers are solid ground planes; power is routed on the bottom layer.

[1] The PIC16F1459 also features a USB (universal serial bus) interface. The original design, as shown in the proposal [5], used USB to program the user-calibration. However, later in the project, USB had to be dropped due to memory constraints; now the calibration is programmed via CAN. In a future revision of the board, the PIC16F1459 could be replaced with a chip lacking a USB interface in order to reduce cost.

[2] Perhaps it is worth noting that I have no particular affinity to Microchip as a company. It is purely a coincidence that all the chosen ICs ended up being made by them. It just so happens that they make good chips for this application.

[3] Not that mixing SMD and THT components is good practice—it is not. The board was only designed this way to allow the DIP parts that were used on the breadboard to be reused on the prototype PCB.

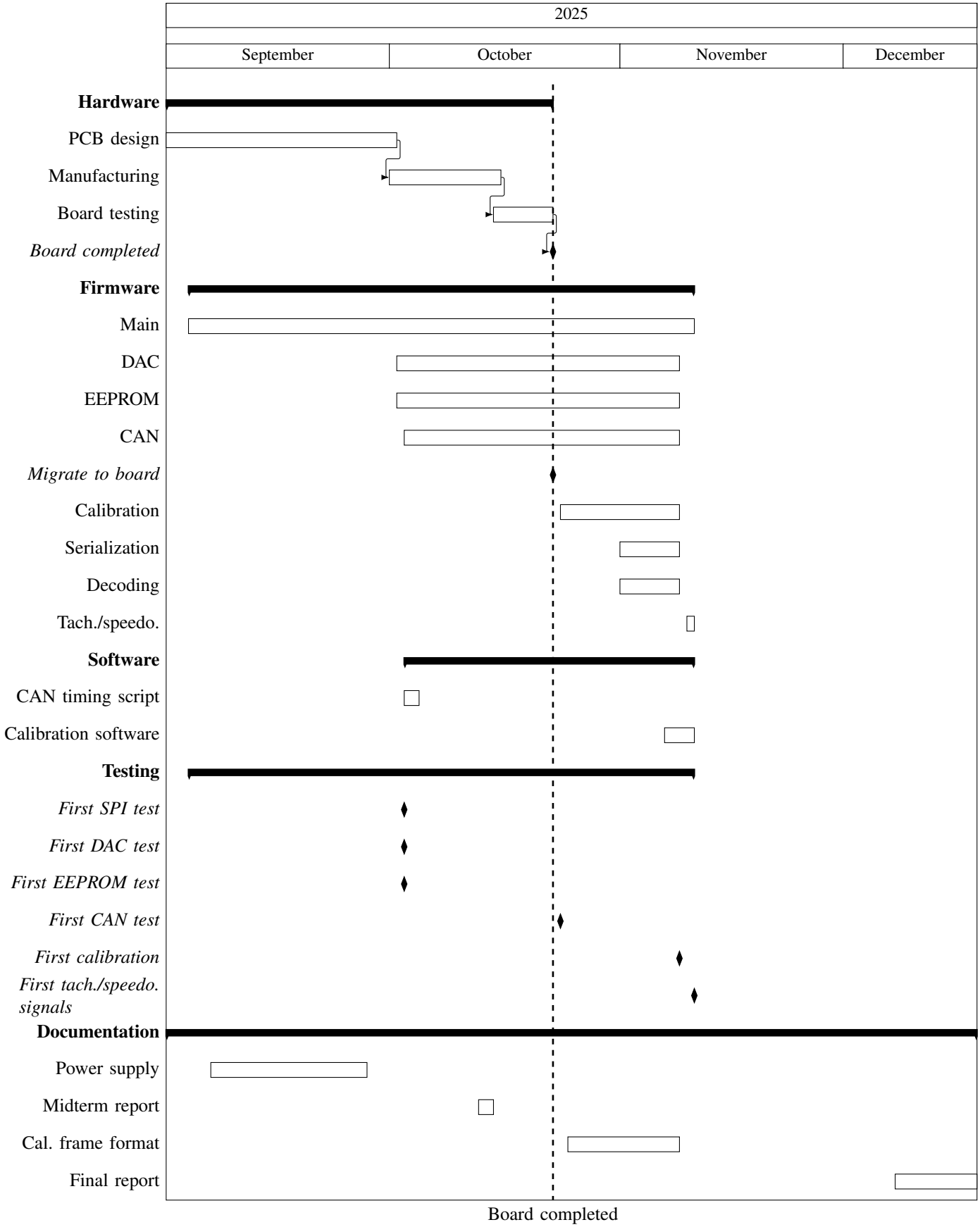

Fig. 4. Project timeline.

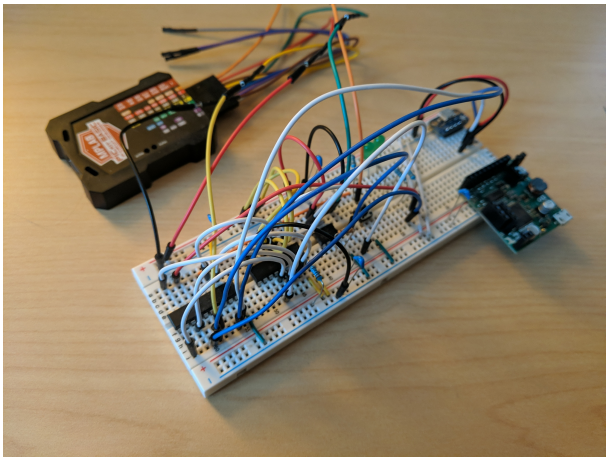

Fig. 5. Breadboard circuit for CAN testing.

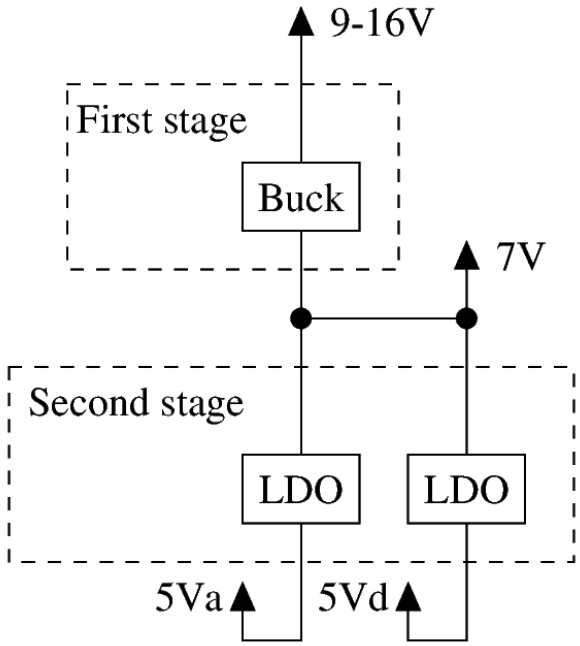

Fig. 6. Power distribution network.

All traces are microstripped above a solid ground plane to keep the fields tightly coupled and to give the current a low-impedance return path. All signal vias are accompanied by a ground via, again to keep the fields from spreading out in the dielectric. Traces are widely-spaced to reduce interference with one another. The noisy switching regulator is placed far away from the other components to reduce EMI—the sensitive analog signals and the DACs are on the opposite side of the board.

One may notice that the board sports a USB-B connector: a vestige of the original design which used USB for user-calibration, as opposed to CAN which it uses currently (see footnote 1). The connector is no longer required and will be removed in a future revision.

As mentioned above, most of the chosen ICs are available in DIP (through-hole) packages, and were used for prototyping on a breadboard (Fig. 5). They were then transplanted into the PCB once it had been manufactured (Fig. 10). This allowed firmware development to be carried out in parallel on the breadboard while waiting for the PCB to arrive, as the timeline in Fig. 4 shows.

IV. Firmware

Firmware is the program that runs on the board's microcontroller. It is responsible for interacting with the peripherals, decoding CAN frames, and transforming sensor data into output signal levels to drive the gauges. It is written in ISO C (C99) [20]. The program is compiled for the PIC16F1459 using Microchip's XC8 compiler [27]. In total, the firmware is 1601 lines of C code [6].

The firmware is divided into a set of C modules. Each peripheral (MCP2515, 25LC160C, MCP4912) has a corresponding module (`can`, `eeprom`, `dac`). Collectively, these are known as the HAL (hardware abstraction layer).

There are also higher-level modules that make use of the HAL. One example is the `table` module. The EEPROM stores several tables that are mappings from sensor-reading-values to output-signal-values. For example, one table maps engine speed (rpm) to tachometer pulse frequency. The `table` module makes use of the `eeprom` HAL module and provides a simple interface to read and manipulate these tables stored in the EEPROM.

Dependencies between the modules are shown in Fig. 11.

The `main` entry point of the firmware simply initializes the HAL and waits to receive an interrupt from either the CAN controller or a timer.

The ISR (interrupt service routine) handles the reception and decoding of CAN frames. Upon receiving a frame, the ISR decodes the sensor signal contained therein, and looks up the corresponding output signal value in the mapping table. It then directs one of the peripherals to generate the appropriate output signal accordingly. For instance, it may instruct one of the DACs to change the voltage being driven to a temperature/pressure gauge, or it may set the period of the tachometer wave.

The variable-frequency square waves that drive the tachometer and speedometer are generated using the PIC's built-in timer peripherals. Upon overflow, the timers trigger the ISR to toggle the appropriate GPIO (general-purpose input/output) pin. The waves' frequencies are controlled by setting the timers' periods.

A. Calibration

In order for the device to work with any combination of sensors, gauges, and CAN encoding schemes, it must be configurable by the end-user.

The user-calibration defines the mapping between sensor readings and output signal values that drive the gauges. It also defines how sensor readings are encoded in CAN frames.

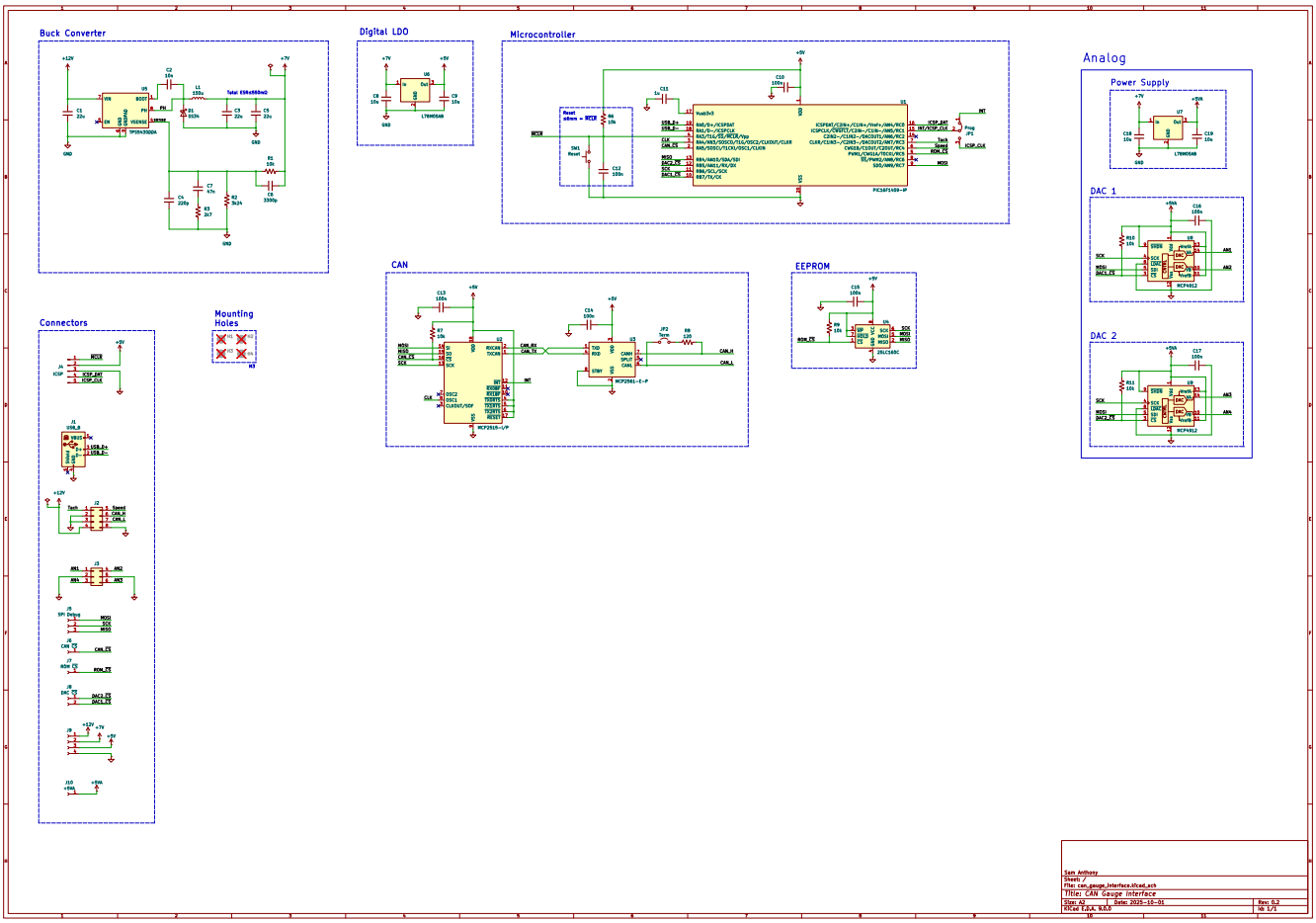

Fig. 7. Schematic.

The `signal` module defines a data structure that specifies the format of a sensor reading, or *signal* as it is known in the DBC format [14], within the payload of a CAN frame. This includes the CAN ID of the carrier frame, the size and offset of the signal within the frame's payload, and the byte order of the signal.

The format of each signal is stored in the EEPROM as part of the user-calibration. The calibration includes six signals: tachometer, speedometer, and the four analog channels.

The user-calibration is programmed onto the board via special CAN frames, called *calibration* or *control* frames. There are two control frames, each with a unique ID. The first is the *table control frame* which is used to read or write rows of the mapping tables. The second is the *signal control frame* which is used to read or write the encoding format of each signal. These control frames are defined in full by the *calfmt* document in the project's source repository [10].

When the device receives a control frame, the ISR decodes it and either, in the event of a write-request, updates the appropriate segment of the calibration in the EEPROM; or in the event of a read-request, reads the appropriate segment and responds by sending it back in a CAN frame. This allows the calibration to be flashed onto the board using write requests and verified using read requests. The calibration is sent to the device from a personal computer running the calibration software, which is discussed in the next section.

V. Software

In addition to the firmware, several other pieces of software were developed over the course of the project. In contrast with the firmware which runs on the device's microcontroller, these programs target a personal computer.

A. Calibration software

As described above, the user-calibration, stored in the board's EEPROM, allows the device to be configured for any combination of sensors, gauges, and CAN encoding schemes.

The *cal* program flashes the calibration from a personal computer onto the device via the CAN bus [7]. Cal takes a set of files containing the user-calibration as input, and outputs CAN frames that are sent to the device. These are the special *control* frames mentioned in §IV-A and defined fully in [10].

The calibration is represented by a DBC file [14] and a set of CSV files. The DBC file specifies the CAN encoding

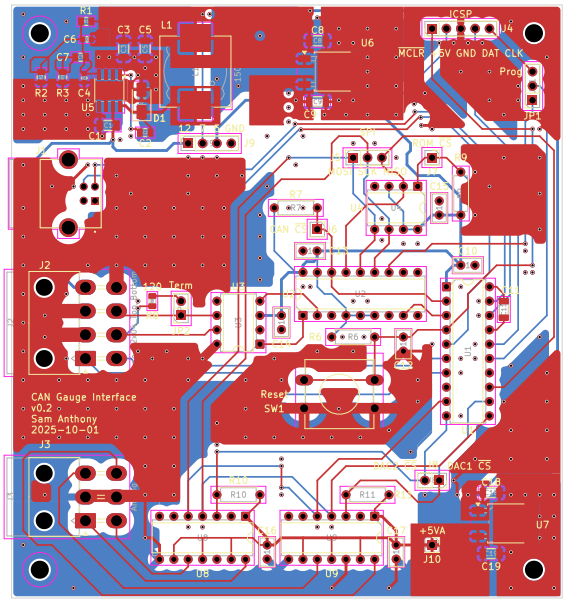

Fig. 8. PCB front and back copper pours.

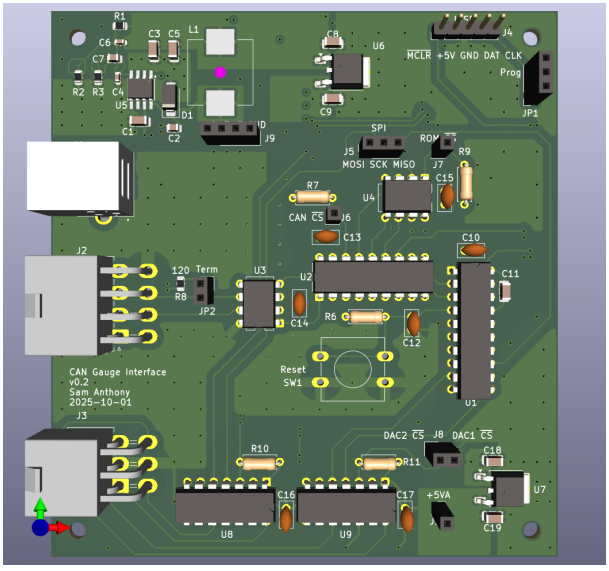

Fig. 9. 3D render of the PCB. Note the vestigial USB-B connector in the top left, to be removed in a future revision (see footnote 1).

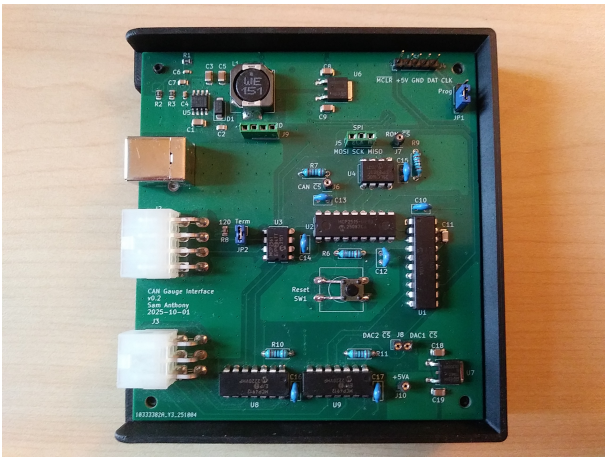

Fig. 10. Fully-assembled PCB sitting on 3D-printed stand.

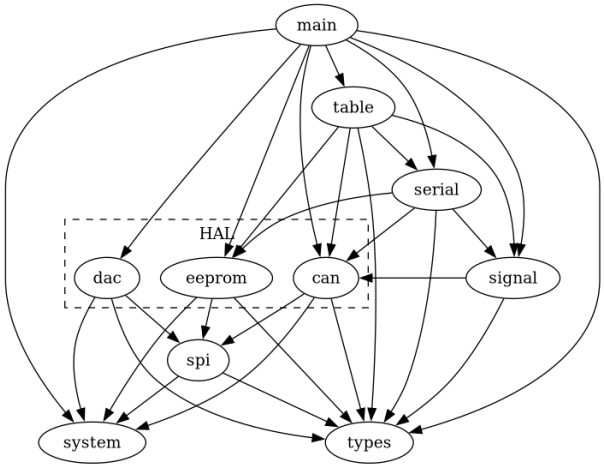

Fig. 11. Firmware module dependency graph.

scheme which defines how sensor readings, called *signals* in the DBC file, are encoded into CAN frames. The DBC file can specify one signal for each gauge, e.g., `EngineSpeed` for the tachometer, `EngineOilPressure` for one of the analog gauges, and so on, each with a corresponding CAN ID and encoding format. This allows the device to be configured for the CAN encoding scheme that the car's EMS uses.

The CSV files define the tables that map sensor-reading-values to output-signal-values. There is one table, and thus one CSV file, for each signal. This allows the device to be configured for the particular sensors and gauges installed in the car.

Cal is written in 674 lines of Go [35]. It uses the Linux kernel's SocketCAN [13] facility to access the CAN bus. Thus, its support is limited to Linux only. Ideally it should be ported to other operating systems in the future.

B. CAN bit timing calculation

The second program is a small script for calculating CAN bit timing parameters.

Each node in a CAN network has its own clock generator. The bit timing parameters can be configured individually at each node in order to achieve a common bit rate throughout the network even though the nodes' clock periods may be slightly out of sync [17].

The script, named `bittiming.py` [8], calculates a set of valid combinations of timing parameters for a given clock frequency and bit rate. It displays these parameter combinations in the form of configuration words for the MCP2515 CAN controller.

The configuration words are hard-coded in the firmware and are written to the MCP2515's configuration registers at startup time. Although the bit rate is currently a constant set at compile time, the firmware includes several sets of bit timing configuration words, one for each commonly-used bit rate as defined by CANopen [11]. This is to facilitate the addition of an automatic baud rate detection algorithm to the firmware at a later date.

A flaw in the board was discovered while calculating the bit timing parameters. The CAN controller requires an external oscillator. On the prototype board, the CAN controller's clock input is connected to the microcontroller's clock output which runs at 12MHz. 12MHz is not fast enough to run a bit rate of 800kbps or greater. Therefore, although the MCP2515 supports bit rates up to 1Mbps, the board is limited to 500kbps with the current clock setup. This can be remedied in a future revision by adding a dedicated 20MHz oscillator for the CAN controller.

VI. Testing

A suite of tests was developed to verify the functionality of the device. Each system test is a standalone firmware image designed to test a particular subsystem. For example, there are system tests for the CAN, DAC, and EEPROM modules, and for the tachometer and speedometer signal generation systems. There are also several unit tests to verify some of the more complex routines, such as the CAN decoding functions. The tests were used continually throughout the project to check the functionality of each firmware module as they were being developed, and to detect regression as changes were made and new modules added.

A. Equipment

Some special equipment was required for testing.

A USBtin USB-to-CAN interface enabled reading and writing CAN frames to the bus from a workstation computer [16]. It was used to monitor the bus during CAN system testing, and to flash the user-calibration onto the device.

An EspoTek Labrador is an inexpensive all-in-one oscilloscope, signal generator, power supply, and logic analyzer [15]. It proved invaluable during testing. The power supply provided 12V to the board. The logic analyzer was used to debug the SPI (serial peripheral interface) bus while developing the HAL firmware modules. And the oscilloscope was used to measure the tachometer and speedometer signals generated by the device.

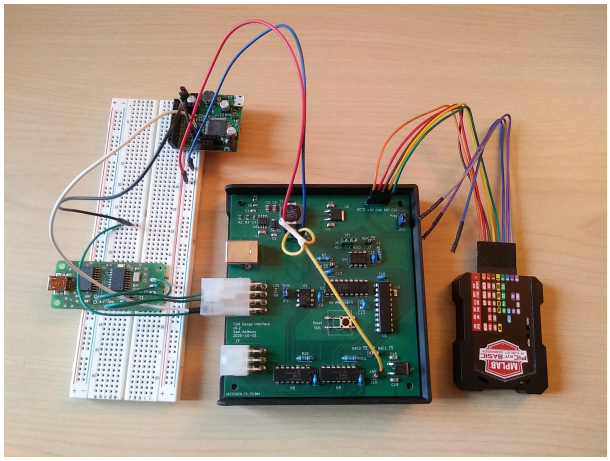

Fig. 12. Testing setup: EspoTek Labrador (top), USBtin (left), gauge driver (center), PIC programmer (right). Yellow wire jumps the 7V, 5V-analog, and 5V-digital rails, providing power to the board and circumventing the flaw in the first stage of the power supply.

Finally, a simple multimeter was used for continuity testing to verify the design of the circuit board, and to measure the voltages of the board's analog outputs during system testing.

B. Results

An issue with the power supply was uncovered while testing the circuit board. As mentioned in §III-B, the first stage of the power supply is supposed to output 7V. However, upon receiving the board, this rail measured only 5V, revealing a mistake in the calculation of the passive components surrounding the buck converter that control its output voltage. While this issue will have to be corrected in a future revision, it did not prevent the existing board from being used for the remainder of the project. Bypassing the second stage of the power supply by jumping the 7V and 5V rails, thus providing the ICs with the correct voltage, allowed the board to function well enough. This workaround can be seen in Fig. 12.

Overall, testing has been successful. Despite minor flaws in the board, the device is able to perform all of its duties. It is able to receive CAN frames from the bus, decode them, and generate signals to drive up to six analog gauges accordingly. It can generate two variable-frequency square waves to drive a tachometer and a speedometer, and four 0–5V analog signals to drive four pressure or temperature gauges. Furthermore, it can be configured for any combination of sensors, gauges, and CAN encoding schemes by flashing a user-calibration with the *cal* program.

VII. Conclusion

In conclusion, the project was a success; the device fulfills all of its requirements. What follows is a summary of the goals, proceedings, and outcomes of the project.

The goal was to design an electronic device that would allow analog gauges to be retrofitted into a car's EMS (engine management system) while leveraging the capabilities of the car's CAN (controller area network) bus. The device transforms

sensor reading data from the CAN bus into electric signals suitable for driving the gauges.

In the first phase of the project, the device's hardware was designed: a PCB (printed circuit board) hosting a set of ICs (integrated circuits) that, together as a system, perform the necessary functions of the device.

Firmware—software that runs on the device's microcontroller—was developed to operate the device's hardware. The firmware is responsible for interacting with the various peripherals on the board, decoding CAN frames, and transforming the sensor data contained therein into output signal values to drive the gauges.

In addition to the firmware, another piece of software was developed: a program for flashing a user-calibration onto the device. This program, called *cal*, reads the calibration from a set of files and writes it to the device via the CAN bus. The calibration allows the device to be configured for any combination of sensors, gauges, and CAN encoding schemes that the car's EMS may use.

Both the software and hardware of the device were tested continually over the course of the project. Suites of unit and system tests were developed to verify the device's subsystems and firmware modules. Hardware testing revealed some flaws in the board that will have to be fixed in a subsequent revision. However, they were not catastrophic, and development was able to proceed using the existing board.

Ultimately, testing confirmed that, despite minor flaws in the board, the device operates correctly and fulfills all of its requirements. It is able to receive CAN frames from the bus, decode them, and generate signals to drive up to six analog gauges. Furthermore, it can be configured for any combination of sensors, gauges, and CAN encoding schemes by flashing a user-calibration with the *cal* program. Hereby the project is considered a success.

References

[1] Automotive Electronics Council. URL: http://www.aecouncil.com/.

[2] *1987 Nissan Skyline 'Type HR31' GTS-R NISMO Group A*. Bonhams Cars. URL: https://cars.bonhams.com/auction/30558/lot/156/1987-nissan-skyline-type-hr31-gts-r-nismo-group-a-chassis-no-hr31-128388/#photos.

[3] *25LC160C. 16Kb 2.5V SPI Serial EEPROM with 16 Byte Page size*. Microchip Technology Inc. URL: https://www.microchip.com/en-us/product/25LC160C.

[4] U.S. Environmental Protection Agency. *CFR 40 §86.1806-05*.

[5] Sam Anthony. *Analog Gauge Driver with CAN Interface. COMP 490 Project Proposal*. report. Montréal Québec, Canada: Concordia University, 2025.

[6] Sam Anthony. *Analog Gauge Driver with CAN Interface*. URL: http://git.samanthony.xyz/can-gauge-interface.git.

[7] Sam Anthony. *Cal. Calibration Software for the Analog Gauge Driver with CAN Interface*. URL: http://git.samanthony.xyz/can-gauge-interface.git/tree/sw/cal.

[8] Sam Anthony. *CAN bit timing configuration script*. URL: http://git.samanthony.xyz/can-gauge-interface.git/tree/sw/bittiming/bittiming.py.

[9] Sam Anthony. *Power Budget Spreadsheet*. URL: doc/power/power_budget.ods.

[10] Sam Anthony. *User-Calibration CAN Frame Format*. URL: http://git.samanthony.xyz/can-gauge-interface.git/tree/doc/calfmt/?id=5e6a57011bdaf23e012eac28c04de3ebcbe65f9e.

[11] *CAN FAQ. CAN Bitrates*. URL: http://www.can-wiki.info/doku.php?id=can_faq:can_bitrates.

[12] *CAN Specification. Version 2.0*. Robert Bosch GmbH, 1991. URL: http://esd.cs.ucr.edu/webres/can20.pdf.

[13] The kernel development community. *SocketCAN — Controller Area Network*. The Linux Kernel. URL: https://docs.kernel.org/networking/can.html.

[14] *DBC File Format Documentation*. Version 1.0.5. Vector Informatik GmbH, Apr. 12, 2010.

[15] *EspoTek Labrador Board*. EspoTek. URL: https://espotek.com/labrador/.

[16] Thomas Fischl. *USBtin. USB to CAN Interface*. URL: https://www.fischl.de/usbtin/.

[17] Florian Hartwich and Armin Bassemir. "The Configuration of the CAN Bit Timing". In: *6th International CAN Conference*. Robert Bosch GmbH. Turin, Italy, Nov. 2–4, 1999.

[18] *History of CAN technology*. CAN in Automation. URL: https://www.can-cia.org/can-knowledge/history-of-can-technology.

[19] D. H. Hubel and T. N. Wiesel. "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex". In: *The Journal of Physiology* 160.1 (1962).

[20] *ISO/IEC 9899:1999. Programming Languages — C*. ISO/IEC, 1999.

[21] *JLCPCB. PCB Prototype & PCB Fabrication Manufacturer*. JCLPCB. URL: https://jlcpcb.com/.

[22] *KiCad. Schematic Capture & PCB Design Software*. KiCad Project. URL: https://www.kicad.org/.

[23] *L78M. Precision 500mA Regulators*. STMicroelectronics. URL: https://www.st.com/en/power-management/l78m.html.

[24] *MCP2515. Stand-Alone CAN Controller with SPI Interface*. Microchip Technology Inc. URL: https://www.microchip.com/en-us/product/MCP2515.

[25] *MCP2561. High-Speed CAN Transceiver*. Microchip Technology Inc. URL: https://www.microchip.com/en-us/product/MCP2561.

[26] *MCP4912. 10-Bit Dual Output DAC with SPI*. Microchip Technology Inc. URL: https://www.microchip.com/en-us/product/MCP4912.

[27] *MPLAB® XC8 Compiler*. Microchip Technology Inc. URL: https://www.microchip.com/en-us/tools-resources/develop/mplab-xc-compilers/xc8.

[28] Neil Panchal. *Skeuomorphic Hierarchy of a Needle Gauge*. July 17, 2025. URL: https://x.com/usgraphics/status/1945924815858311168.

[29] *PIC16F1459. 8-bit Microcontroller with USB*. Microchip Technology Inc. URL: https://www.microchip.com/en-us/product/PIC16F1459.

[30] *Pressure Sensor Combined PST-F 1*. Bosch Motorsport. URL: https://www.bosch-motorsport.de/content/downloads/Raceparts/Resources/pdf/Data%20Sheet_70496907_Pressure_Sensor_Combined_PST-F_1.pdf.

[31] *Professional Stepper-Motor Analogue Temperature Gauge. Installation Guide*. 2650-1289-00. Version B. Stack Ltd. URL: https://www.autometer.com/media/manual/2650-1289.pdf.

[32] *Stack Colour LCD Motorsport Display*. Stack Ltd. URL: https://www.stackltd.com/ST9918.html.

[33] *Stack ST400 & ST430 Professional Tachometers*. Stack Ltd. URL: https://www.stackltd.com/st400.html.

[34] Hans Strasburger, Jörg Huber, and David Rose. "Ewald Hering's (1899) On the Limits of Visual Acuity: A Translation and Commentary. With a Supplement on Alfred Volkmann's (1863) Physiological Investigations in the Field of Optics". In: *I-Perception* 9 (3 June 4, 2018). DOI: 10.1177/2041669518763675.

[35] *The Go Programming Language*. The Go Authors. URL: https://go.dev/.

[36] *TPS543x. Wide Input Range Step-Down Converter*. Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/tps5430.pdf.

[37] *Trends in the Semiconductor Industry. 1970s*. Semiconductor History Museum of Japan. URL: https://www.shmj.or.jp/english/trends/trd70s.html.

[38] *VDO Gauges and Accessories Catalog*. AUMOVIO Aftermarket GmbH. 2025. URL: https://vdo-instruments.com/.