Study program: Applied Electronics (ETC-ELAeng), 2024

Electronic Module for the Dashboard of a Formula Student Racing Car

Project objectives: The student must build from scratch an electronic module that will act as an interface between the pilot and the Formula Student race car it. The module will monitor the CAN bus of the car, from which it takes the following data points: oil temperature and water temperature, the status of the hybrid system as well as real-time data such: RPM and the current gear. It will also include an accelerometer on the same PCB. All the sampled data will be shown on an LCD module as well as a web interface which will be served by the HTTP server of the ESP32 microcontroller. The PCB will integrate an SD card that stores the data points in a file, which can be downloaded via the web interface.

A test module will be implemented on a second PCB for the student to showcase the functionality of the dashboard electronic module. It emulates the ECU of the Formula Student race car and will send via CAN the above-mentioned data. This module will feature the Raspberry Pi Pico microcontroller.

From a hardware perspective, the following components will be used:

- ESP32 WROOM 32 microcontroller as the brains of the dashboard electronic module
- SN65HVD230D CAN Transceiver that will act as an interface between ESP32 and the CAN bus
- LCD module from 4DSystems to show the data in a graphical interface to the pilot
- NeoPixel Stick - 8 x 5050 RGB LED that will be the shift bar, which signals the right moment to change the gear
- Panel mount buttons which will be the way of interacting with the dashboard module

Project fulfilment and results: To fulfill the above-mentioned requirements, PCB Design as well as advanced C/C++ programming techniques had to be used. The contributions for this project are not trivial, since the PCB of the dashboard had to be designed from scratch, as well as the PCB of the testing module. No development boards were used on the dashboard module. It has a schematic that integrates all the needed peripherals as well as the ESP32 SoC with its necessary programming peripherals. To achieve this, the schematic of the ESP32 WROOM 32 dev board was analyzed and reimplemented in the KiCAD EDA.

Besides this, the dashboard includes standalone power delivery circuitry, which includes a LDO regulator, that converts the 5V power rail into a 3V3 power rail which is used for the ESP32, SD card module, CP2102 programming chip, the Hybrid selector, and the CAN transceiver SN65HVD230DR. The 5V rail is converted from the 12V power rail of the Formula Student racing

car, by a buck converter. The circuit of the buck includes the components used in the reference design, given by its datasheet, for a 5V 3A output.

Other circuitry that needed to be implemented on the PCB were the MPU6050 accelerometer circuitry and the I2C lines that go to the ESP32 SoC, the SD card circuitry with all the necessary pull-up resistors as well as filtering for the 3V3 rail via ceramic capacitors of 1uF and 0.1uF. A lot of capacitors are used for filtering in the dashboard schematic to remove the noise induced by the buck. As a fail-over technique, the 3V3 rail can be supplied also by the LCD, which includes an LDO itself. This is achieved by placing on the PCB a solder jumper.

The LCD connector is a 30-pin ribbon cable connector. The pins of this connector were used as per the specification sheet of the 4DSystems company. Therefore, multiple LCD screens can be used with the same PCB as they all respect the same pinout of the ribbon connector.

The PCB is a 2 layer one, components being installed mostly on the front copper layer, the only exceptions being the LDO, because of its large footprint, and the automotive harness connector of the dashboard module, which needs to be oriented to the back of the PCB.

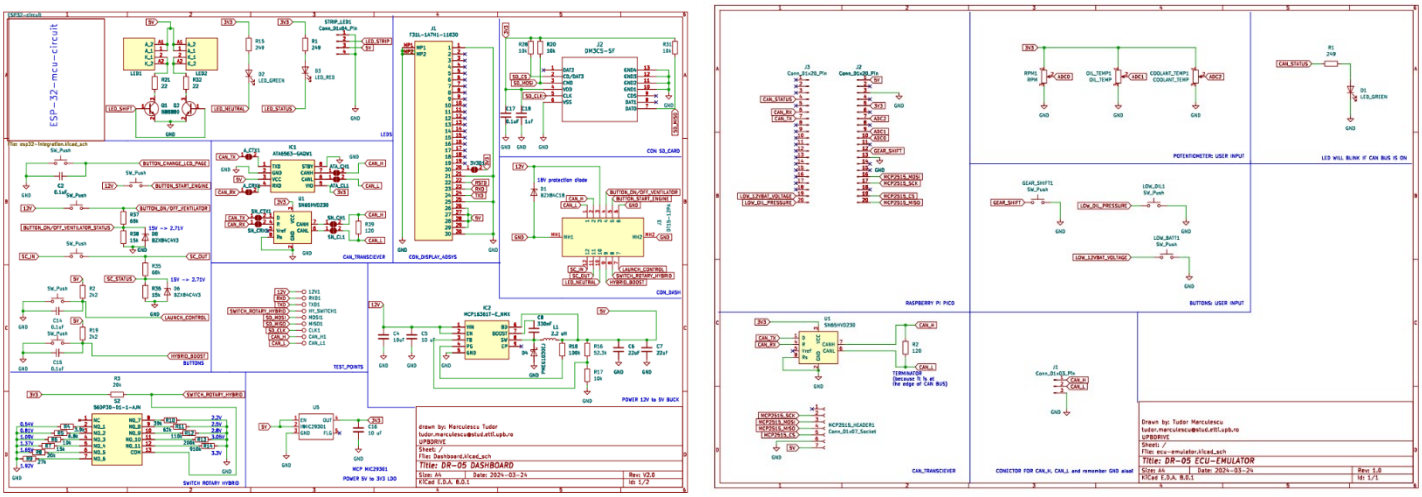

Fig. 1 Schematics of the dashboard module and test module

The PCB design of the test module is also based on a 2-layer approach, this time by using the PCB as an interconnection method between the Raspberry Pi Pico development board and the MCP2515 development board from Microchip, which includes a CAN controller that communicates via SPI with the Pico.

The PCB features 3 potentiometers that are used to emulate the RPM, water temperature and oil temperature signals from the ECU (Electronic Control Module), as well as 3 buttons that emulate a gear change, low oil pressure signal and low battery voltage signal. The low oil and low battery signals are used to showcase the function of the dashboard to display error messages on the LCD, in order to inform the pilot in case of catastrophic failures. The dashboard module and the test module will be connected via the CAN bus lines: CAN HIGH and CAN LOW, through which the test module will send CAN packets with the data acquired from the potentiometers and the buttons.

Do not think that this data is not converted into actual RPM and temperature values! Pico does all the data conversion in its firmware then formats the data into CAN packets which are send via the bus by the MCP2515 CAN controller.

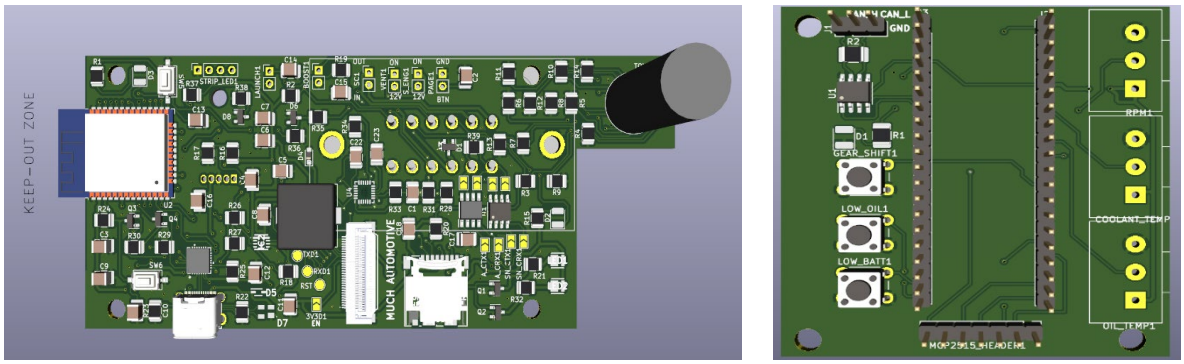

Fig. 2 3D View of the dashboard and test module

From a software point of view, the ESP32 one is complex, since it integrates a RTOS (Real Time Operating System). This allows the firmware to run on multiple threads on the same core of the microcontroller, and to also run parts of the firmware in parallel, by using multicore processing. In this way, CAN read and send tasks and LCD communication task, which are considered high priority, can be run completely parallel on 2 cores of the ESP32.

To achieve the highest performance possible, ESP32 was clocked to 240Mhz, which is the maximum supported by its Tensilica xTensa cores. Please note that this module is powered by a combustion engine, therefore efficiency is not considered, since it's already lost in combustion.

Espressif, the company which designed the ESP32 microcontroller, offers an IDF (Integrated Development Framework), written in C programming language, that includes a special flavor of FreeRTOS. It was adapted to run precisely on the hardware of Espressif. FreeRTOS is a renowned software solution for real-time applications in the automotive industry. If one uses this RTOS, he needs to break the code into multiple functions, which are called tasks.

Tasks are one of the most important concept of RTOS. They have different priorities, therefore when a high priority task is scheduled to be executed, it will preempt the lower priority ones. This allows the programmer to have certainty that high priority code will always be executed as scheduled by him. However, if no higher priority tasks need to be executed, lower priority ones are put into execution, therefore achieving the principle of multiple thread processing, since on one core, not only the high priority task is executed, assuming it will have some idle time, but also the lower priority ones. Multiple tasks had to be developed in order to achieve all the features proposed: accelerometer read task via I2C bus, SD card write task via SPI and VFS (Virtual File System) library, LCD communication task via UART2 IP, the HTTP server backend, etc. The html web interface was embedded in the binary that is flashed on the ESP32, therefore only the log file is stored on the SD card, not the index.html and its additional files used on the web interface.

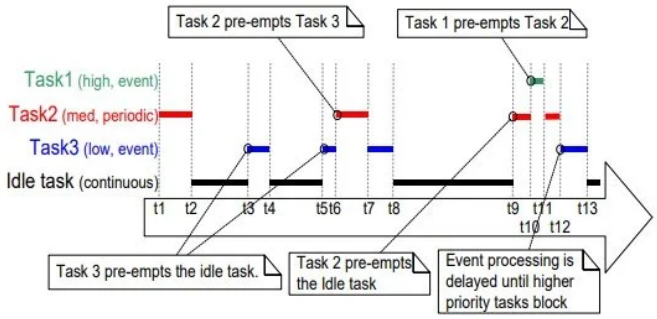

Fig. 3 Preemptive scheduling

To communicate with the NeoPixel LED bar, NRZ (Non-Return to Zero) protocol was used. The protocol was emulated with the RMT (ReMoTe control) library of the ESP32. The RMT is usually used to emulate TV remote control signals on the microcontroller's pins, however, it can be also integrated to respect the protocol used by all 8 LEDs of the shift bar, which communicate with the ESP32 only via 1 pin. Data sharing between cores and threads is done using Semaphores which are a way of excluding race conditions on the same memory location.

Raspberry Pi Pico was programmed using bare metal programming techniques in C++. It does not include any operating system, and only runs one thread of monolithic code. This proves to be problematic, in case the code will get as complex as the dashboard firmware, but for the test board use case, it's more than reasonable to not use RTOS. The main features are ADC reading of multiple potentiometers via ADC pins on the Pico development board and button state polling, the formatting of CAN packets. The packets are objects in C++, defined by a library developed for the MCP2515. In the dashboard firmware a common C data structure was used to store the display data, which is also integrated into the Pico C++ firmware to make the codebase more uniform and easier to understand. The CAN packet identifiers are also uniform between the dashboard and test module firmware as well as the packet formatting functions.

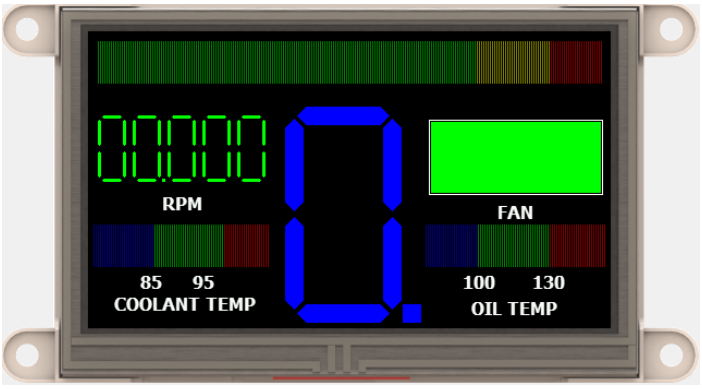

Fig. 4 Pilot interface

The dashboard was tested in an empiric method by trying all its functionalities and running it over an extended period. No crashes were detected during the test run time, therefore, it can be said that all the proposed student contributions were achieved, and the project development was a success.