Red Bird Racing EVRT Software Training

Project

Last updated: 2025-12-19

Authored by: CHEUNG Pui Ki (Planeson)

Contents

Contents 1
Instructions 2
Software Installation 2
Work 2
Submission 3
Foreword 3
Board setup 4
Tips 4
Part 1 5
Deadline 5
Description 5
INIT 5
STARTIN 5
BUZZIN 5
DRIVE 5
Part 2 7
Deadline 7
Description 7
BMS 7
Motor 7
Debug 7
Part 3 9
Deadline 9
Description 9
BONUS 10
Deadline 10
Description 10
Grading Scheme 11

Instructions

Software Installation

For making the CAN DBC table:

Get SavvyCAN:

https://www.savvycan.com/

For coding:

Get VSCode:

https://code.visualstudio.com/download

Get the PlatformIO Extension for VSCode:

https://docs.platformio.org/en/latest/integration/ide/vscode.html

For flashing your code onto an actual board:

https://github.com/gen-so/PROGISP-V1.72

For sending and logging CAN messages:

https://www.peak-system.com/PCAN-View.242.0.html?L=1

Work

Create a new project on the board "*ATMega328P/PA*" using the Arduino framework.

Upload your work by

1. Creating a new GitHub repository;
2. Uploading the **entire** folder by Git actions, e.g. via GitHub desktop or the Git extension in VSCode;
3. (For uploading .dbc) uploading the .dbc file to the root of the folder, i.e. directly add the file to your GitHub repository;
4. (For uploading documentation) uploading a .pdf (preferred) / .docx / .rtf file to the root of the folder, i.e. directly add the file to your GitHub repository. Note that this isn't best practice, but it will suffice for this project.

Submission

After pushing your work to GitHub, send a link to Carson Cheung (@Planeson, +852 9437 6620) via WhatsApp. I will grade your work and give feedback.

Please send a message each time you want another round of feedback, e.g. after fixing bugs with your code, or finishing a part of this project.

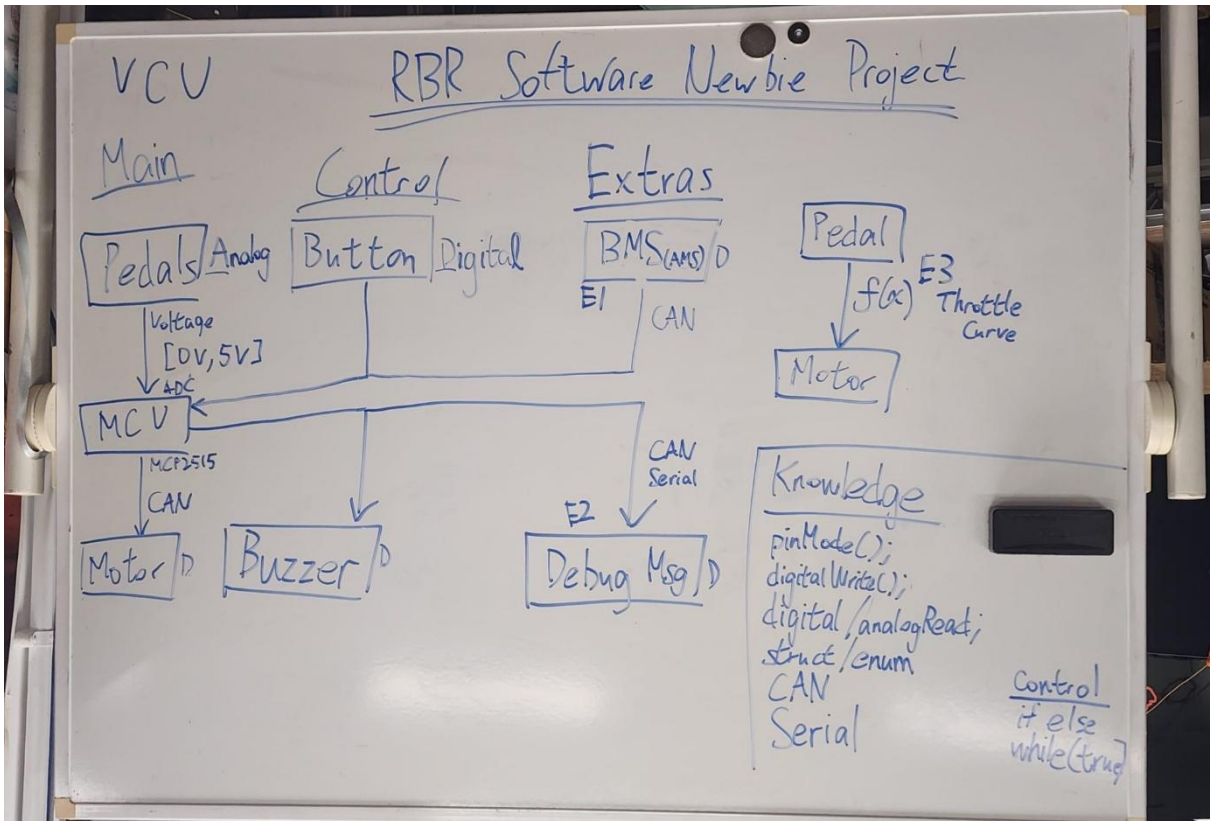
The scores of the project will be not be published, but note that the project scores directly affect your chance of entering the Software subteam, so you should try your best. If you have issues, e.g. deadlines, not knowing what to do, etc, please do reach out, we're here to help you, not to harm you.

Foreword

This project is a complete project from start to finish. We go from project requirements to a finished project, with minimal handholding in between. After this project, you should be able to tackle on any other embedded programming projects, barring any issues regarding specific field knowledge.

The main focus of the project is the VCU – vehicular control unit. It takes in the pedals and the start button, and communicates over CAN, with optional communication via Serial. The only non-CAN outputs are the buzzer, a brake light and a Drive mode LED.

A brief (incomplete) drawing can be found below.



Board setup

This is a reference set of pins for the VCU. Because of availability of hardware, the actual definition of pins may change. You are suggested to hold the pin numbers as a const that is easily modifiable, for instance, as an enum, or with #DEFINEs.

- The MCP2515 CAN controllers are at PB2 for motor, PB1 for BMS, and PB0 for debug.
- The pedals are analog inputs at PC0 for APPS_5V, PC1 for APPS_3V3, and PC3 for brakes.
- The Start button is at pin PC4.
- The brake light is located at PD2. The buzzer is at PD4. The drive mode LED is at PD3.
- The MCP2515's crystal frequencies are at 20MHz. You can safely assume all 3 MCP2515s share the same crystal frequencies.

While reference boards will be provided at the Red Bird lab at Hall VIII G01, I can't provide everyone with their own boards for now. An Arduino can help you complete part 1 and 3. You can ask me to help you test your code if you really can't make it to the lab.

Tips

- You are suggested to document your code early, e.g. before submission for each part. This helps you and us to better understand the code; the process of documentations also helps you catch logical errors and simplify logic flow. It will also make part 3 far easier.
- Start work early. While the code is some of the easiest code you may write in your university life (even COMP1023 is harder), the logic is not easy. You are given quite a bit of liberty in this project. You have to be the system analysist, system architect, as well as the programmer and documenter.
- You can choose an object-oriented OR a functional approach. While there are clear advantages to an object-oriented approach, a functional implementation should be easier to code. If you have no prior experience to object-oriented programming, you can give it a shot.
- Do NOT use dynamic memory; ALWAYS use static memory. You don't want to deal with random crashes, or allocating memory. Static memory is easy to use and deal with.
- You can still modify previous part(s) in the submission for the next parts, and remember you can resubmit as many times as you wish before the final deadline. The deadline for each part is to give you a concrete goal to achieve, with a concrete timeframe. You won't receive mark penalties for a poorly written implementation that is fixed later, so feel free to go back to old code to change it up.

Part 1

Deadline

2026-01-04, Sunday, 23:59:59 HKT.

Description

In the first phase, you will implement the basis of the VCU. This includes the input reading, output production, and the state transitions.

There are 4 states in the system: INIT, STARTIN, BUZZIN, and DRIVE.

INIT

The system starts in the INIT state. In this state, set the motor output to 0 for safety. To transition into the STARTIN state, the start button must be pushed, and the brakes must be depressed. The amount of depression needed from the brakes for it to be considered "depressed" should be set via a const variable.

STARTIN

In the STARTIN state, we continue to set the motor output to 0. If at any point the brakes are released OR if the button is not held down, we return to the INIT state. If and only if the brakes AND the start button had been held down for at least 2 seconds, we transition into the BUZZIN state.

BUZZIN

In the BUZZIN state, hold the buzzer on. We transition into the DRIVE state if 2 seconds has past. In the BUZZIN phase, the driver can release the brakes and the start button.

DRIVE

The DRIVE state is the main state of the VCU, and the part where you pay the most attention on. This state is clearly indicated by the fact that the drive mode LED is on. You should do the following:

1. Read pedal inputs.
2. Check if the 2 APPS are faulty.
    a. We define faulty as them having a difference of more than 10%
    b. Note that since the maximum of APPS_3V3 is only 3.3V, you need to scale APPS_3V3 and/or APPS_5V such that they are on a common basis for comparisons.
    c. If the pedals are faulty for more than 100ms, you must stop motor output and transition back into the INIT state. To make this easier to understand, we can say "the VCU thinks the pedal readings are faulty. For safety, we stop the car ASAP".

3. Calculate the torque output for the motor using the input from the pedals. You can simply scale linearly from the input range to the output range, though other ways of scaling is a BONUS task that is not necessary. The output torque value ranges from -32768 to 32767, where negative values make the motor spin backwards, and positive values make the motor spin forwards. You should include a const Boolean to enable flipping the motor directions at compile time. You should only ever output -32768 – 0, or 0 – 32767, depending on this Boolean.
4. Repeat.

Note that in all states, the brake lights should turn on if the brake reading exceeds your set threshold. You should set a reasonable threshold yourself.

Notice that we aren't outputing the torque values to anywhere yet. We implement CAN communications in part 2.

Part 2

Deadline

2026-01-18, Sunday, 23:59:59 HKT.

Description

In the second phase, you will implement CAN communications on the VCU.

As a brief reminder of what you need to do to communicate on CAN:

1. Create, then initialize all MCP2515 objects.
2. Send messages on CAN whenever you need

Implement the following CAN communications.

BMS

To keeps thing simple, you only need to implement the following for the BMS:

During the STARTIN phase, ***only*** transition into BUZZIN if we read, from the BMS MCP2515, CAN ID 0x186040F3 with message 0x……50., i.e. rx_msg.data[6]=0x50

Motor

You only need to implement sending a torque command to the motor. On the motor MCP2515, send the message as shown below:

CAN ID: 0x201

Message:

0	1	2
90	D1	F9
torque_cmd	torque_val	

Explanation: the first byte, 0x90, tells the motor this is a torque command. The third ***then*** second byte is the requested torque. F9D1 is just an example value. Note that the torque value is in little endian, i.e. we store 0x123456ABCD as CD AB 56 34 12. Read the training manual if you need an explanation.

Debug

On the debug MCP2515, send debug messages every cycle. While you can add extra ones, you need at least the following:

- Pedal inputs
- Car state (INIT, STARTIN, BUZZIN, DRIVE)
- Fault and difference of APPS readings (only if a fault occurs, send a fault message)

The structure of these CAN messages shall be decided by you. This includes the CAN ID, as well as the messages themselves. As this is data defined by you, provide a .DBC file to decode your own messages. The DBC file should handle ALL CAN outputs from the VCU. It should show each field, decoded into more meaningful results, like pedal reading, and car state in English, that’s to say, don’t simply name the message without explaining what the bytes mean.

Part 3

Deadline

2026-01-25, Sunday, 23:59:59 HKT.

Description

1. Write automatic PlatformIO-based tests for your code. You need test for the following 2 functions of the pedals.
    a. Fault detection
        - Check that the pedal fault triggers correctly, i.e. a fault is detected if the pedal readings are more than 10% different, but no fault is detected if the readings are less than 10% different. Note that being roughly 0.5% off is acceptable.
    b. Pedal-to-output mapping
        - Check that the pedal readings translate into torque values correctly. Make sure the outputs are in-range for possible input values, and that the torque values have the same sign (except 0).
2. Comment your code properly. Write doxygen-style comments such that doxygen can automatically generate documentations for all your files. You can refer to the training manual to learn how to write doxygen comments properly.
3. Write a technical handbook. You can refer to the training manual on how to do this, but to keep it brief, include things not typically included in a doxygen documentation, like how the system architecture works, the (rough) logic flow of the program, how to interpret the CAN messages, etc. Include maintenance notes where appropriate.

BONUS

Deadline

2026-01-25, Sunday, 23:59:59 HKT.

Description

You can get a maximum of 10% extra credits by completing the following:

Implement a better conversion of pedal reading to torque value.

Allow ways to map a “curve” from the pedal readings to the output, e.g. pedal reads 100, output 500, pedal reads 200, output 1500, pedal reads 300, output 3000

I suggest using a few fixed points, then linearly map pedal readings to output values using a linear function.

You MUST provide a plot to show how the pedal readings and the output values are correlated. Please include this file in your technical handbook.

Grading Scheme

The project is graded 0-100.

30 marks is given for correct implementation of the state transitions from part 1. This includes reading inputs, processing, as well as proper storage containers for variables.

30 marks is given for correct implementation of CAN communications from part 2. This includes initializing the MCP2515 objects, reading/writing messages correctly, and having a reasonable CAN message structure.

40 marks is given is for satisfactory completion of everything outside the core logic, i.e. part 3. Of the 40 marks total in this section, 15 marks is for correct implementation of testing functions for the pedals. 25 marks is for appropriate documentation, including in-code comments and the technnical handbook.

Maximum of 10 extra marks is given for satisfactory completion of BONUS.

For late submissions, the late penalty is a points penalty on the total score. The exact amount of points deducted is decided on a case by case basis, but if you have clear communication on why you are late, you may be granted late submission without penalty. Therefore, please tell us if you ever encounter difficulties; communicate is of paramount importance in a colaborative environment.

As a reminder, your submissions are only graded at the final deadline, i.e. 2026-01-25, Sunday, 23:59:59 HKT. Feel free to revisit old code and change it up.