Red Bird Racing EVRT Software Training

Homework 3

Last updated: 2025-11-06

Authored by: CHEUNG Pui Ki (Planeson)

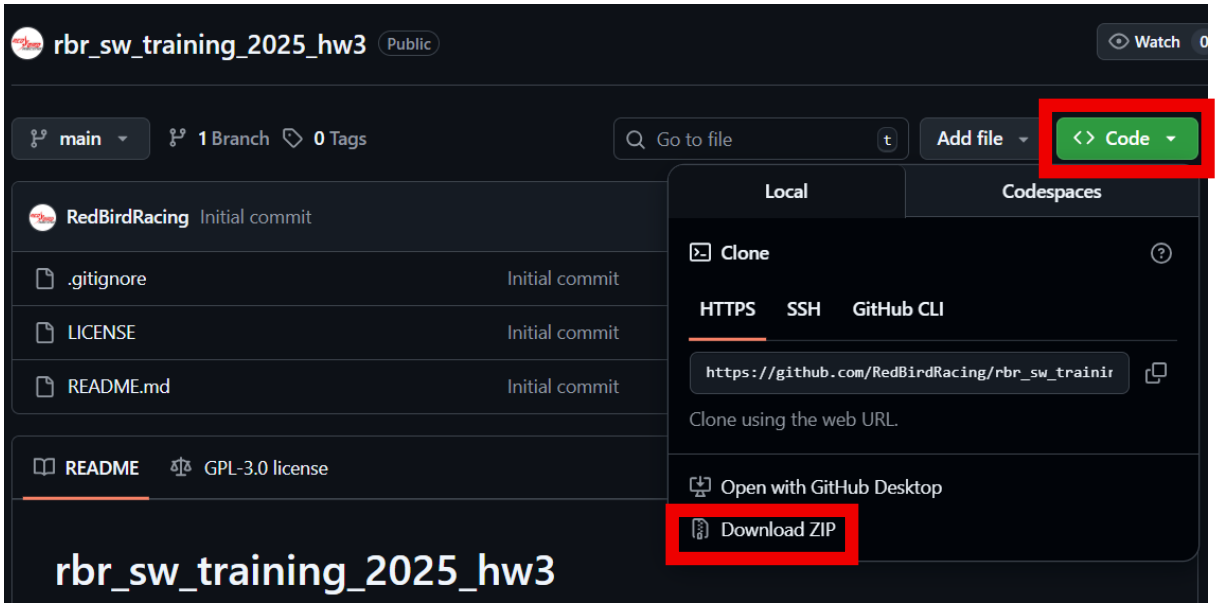
Deadline: 2025-11-15 23:59:59 HKT (**Saturday** Night)

Contents

Instructions

Go to the GitHub page of the files, go to Code and download as ZIP.

https://github.com/RedBirdRacing/rbr_sw_training_2025_hw3



Create your own repository and upload q1.cpp and q2.dbc to complete homework 3.

Foreword

This homework is meant to complement the notes (updated on 2025-11-03) by giving you more practical experience working with embedded systems. You will write parts of a program on an MCU, then you will work with the output given from the MCU.

Q1 – Programming

Introduction

In this question, you will be writing a simple embedded program. Edit, then submit q1.cpp; The goal of the program is to send an analog reading of a dial on a CAN bus. Complete the todos and you should arrive at a working program.

This code is intended for the atmega328p, and is written on the Arduino platform. On the Arduino platform, code is devided into 2 main sections – setup() and loop(). In setup(), you implement setup procedures, like setting the pin modes and initializing values. After setup() is run, loop() runs forever. You put the main business logic here, which is usually reading inputs, processing, then outputing. You can consider setup() and loop() like the following pseudocode.

```
int main(){
    setup();
    while (true) {
        loop();
    }
    return 0; // impossible to reach
}
```

Tasks

We only describe the functions to use, you should figure what parameters to use, as well as what to name objects/variables.

As a minor hint, you can complete all todos in a single line.

Todo 1

Complete the setup tasks. This includes:

- Creating an MCP2515 object at PIN_PB2
    - `MCP2515 canName(CS_PIN);`
- Setting pin mode on input and output
    - `pinMode(PIN_NUM, INPUT);`
- Initializing the MCP2515 object, with the baud rate set at 500kbps
    - `canName.reset();`
    - `canName.setBitrate(CAN_BAUDRATE, MCP_XTAL_FREQ);`
    - `canName.setNormalMode();`

Todo 2

Complete the CAN reading logic. This includes:

- Creating the receiving CAN frame
    - `struct can_frame frameName;`

- Checking if we received a CAN frame
  - `if (canName.readMessage(&frameName) == MCP2515::ERROR_OK) {`
- Checking the contents of the CAN frame
  - `if (frameName.can_id == 0x123 && frameName.data[0] == 0x98) {`
  - Check the CAN ID, then check the content. You can know the length of the payload via the .dlc member, and you access the payload via the data array.
  - For this question, check that the ID is 0x420, and the first byte is 0x69.

Todo 3

Complete the logic for reading and sending the analog value over CAN bus. This includes:

- Reading the analog value from the PIN_PC0
  - `analogRead(PIN_NUM)`
- Sending it over the CAN bus using the MCP2515 object
  - `canName.sendMessage(&frameName);`
- Having an LED at PIN_PB2 be on during this process (assume active high)
  - `digitalWrite(PIN_NUM, HIGH);`

Submit the completed file to complete Q1.

Skeleton Code (same as GitHub)

```
/*
File: q1.cpp
Triggered when a specific CAN frame is received
    Analog read PIN_PC0,
    Send the reading over CAN.
While this happens, make an LED turn on to indicate activity.
    This LED is connected to PIN_PD2.
The CAN frames are received and sent via an MCP2515 CAN controller.
    It is connected via SPI with CS on PIN_PB2.
    The crystal is 20MHz and the CAN speed is 500kbps.
*/

// include the necessary libraries
#include <Arduino.h>
#include <mcp2515.h>

// create MCP2515 object
/* todo 1 */

void setup() {
    // set the pinMode on the input pin
    /* todo 1 */
    // set the pinMode on the LED pin
    /* todo 1 */
    // initialize the CAN controller at 500kbps
    /* todo 1 */
}

void loop() {
    // create CAN frame objects
    // make one for receiving, another for transmitting
    struct can_frame txFrame;
    /* todo 2 */

    // check if a CAN frame is received
    if (/* todo 2 */) {
        // check if the received frame is of ID 0x420 and first data byte is 0x69
        if (/* todo 2 */) {
            // turn on the LED to indicate activity
            /* todo 3 */

            // read the analog value from the input pin and store it as a variable
            /* todo 3 */;

            // prepare the CAN frame to send
            txFrame.can_id = 0x690; // replace with desired send ID
            txFrame.can_dlc = 2; // data length code
            txFrame.data[0] = (analogValue >> 8) & 0xFF; // high byte
            txFrame.data[1] = analogValue & 0xFF; // low byte

            // send the CAN frame
            /* todo 3 */

            // turn off the LED after sending
            /* todo 3 */
        }
    }
}
```

Hints

- Since embedded code are meant for a specific platform, you cannot even compile without the toolchain. You can try compiling with the Arduino IDE if you have it, by first renaming the file to use a .ino extension. Note that without the mcp2515, you can't really see if related functions work.
- You may come to hall VIII test your code (under my assistance, I will help to compile then flash the code) during my office hours:

Mon	1040-1310; 1520-2400;
Tue	1300-1430;
Thu	1300-1430; 1820-2400;
Fri	1040-1510;

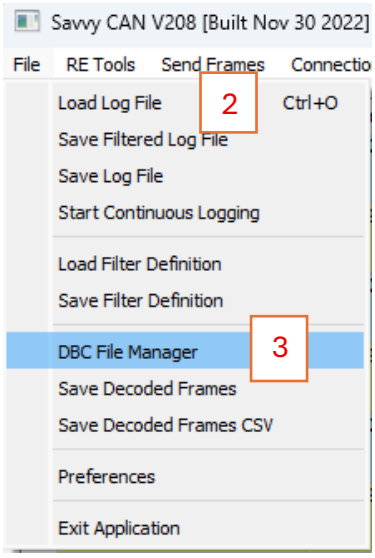
- Read the instructions carefully, they are meant to help you, not distract you.
- Avoid using AI tools directly, since it is eliminating thinking from your working process. You can use it to help you think, but don't ask it to solve the questions for you directly.
- The questions should be so straightforward that copilot *autocomplete* should be able to finish the code for you. If you want to learn well, I suggest disabling copilot autocomplete for homework questions.

Q2 – CAN .dbc table

Introduction

In this question, you will be writing a .dbc to interpret CAN frames in a more visual way. You will want to download SavvyCAN for this.

You will be reading the trace of the CAN bus from Q1 and creating a .dbc to interpret the frames.



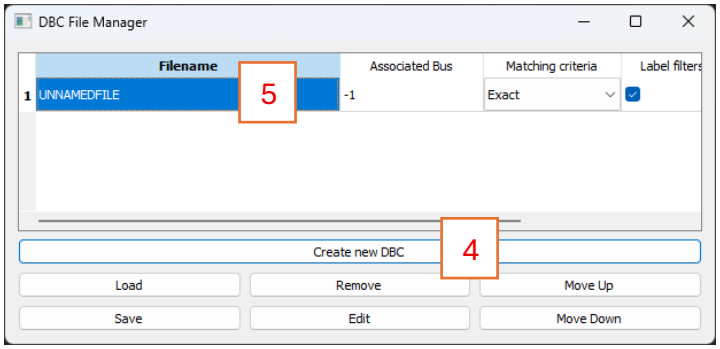
Steps

1. Install and start SavvyCAN

2. Download and load the given trace file|
   ([File] > [Load Log File])

3. Start the DBC file manager
   ([File] > [DBC File Manager])

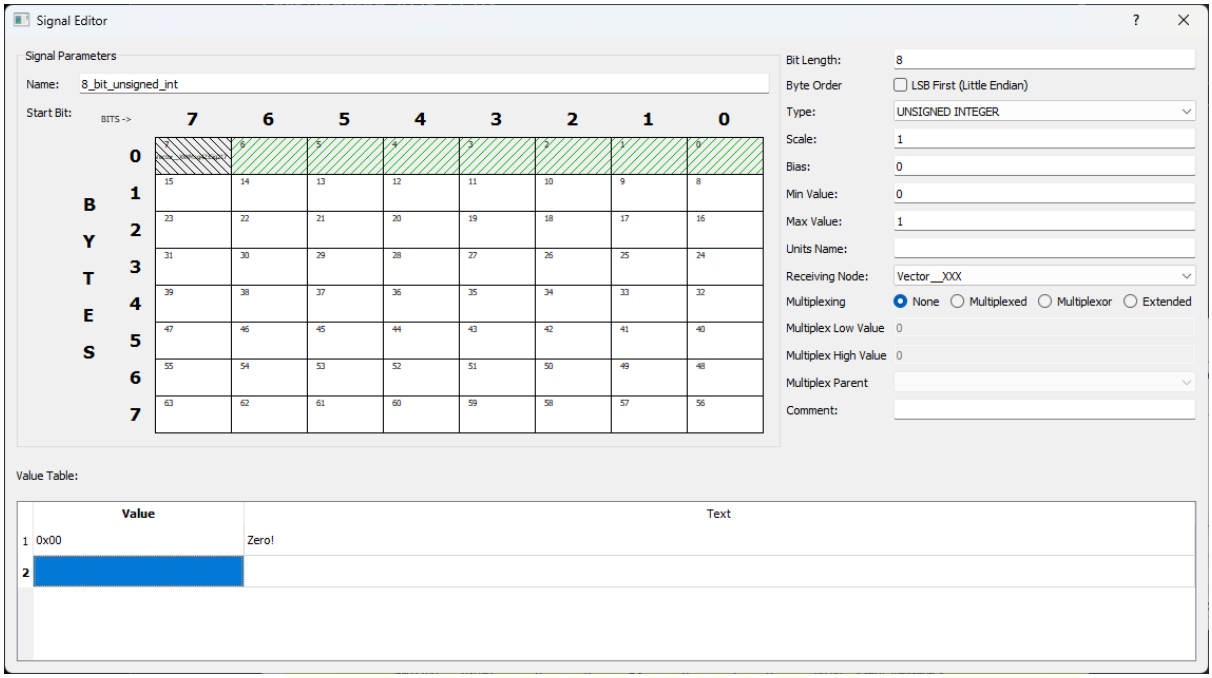
4. Create a new DBC file and edit it
   ([Create new DBC])



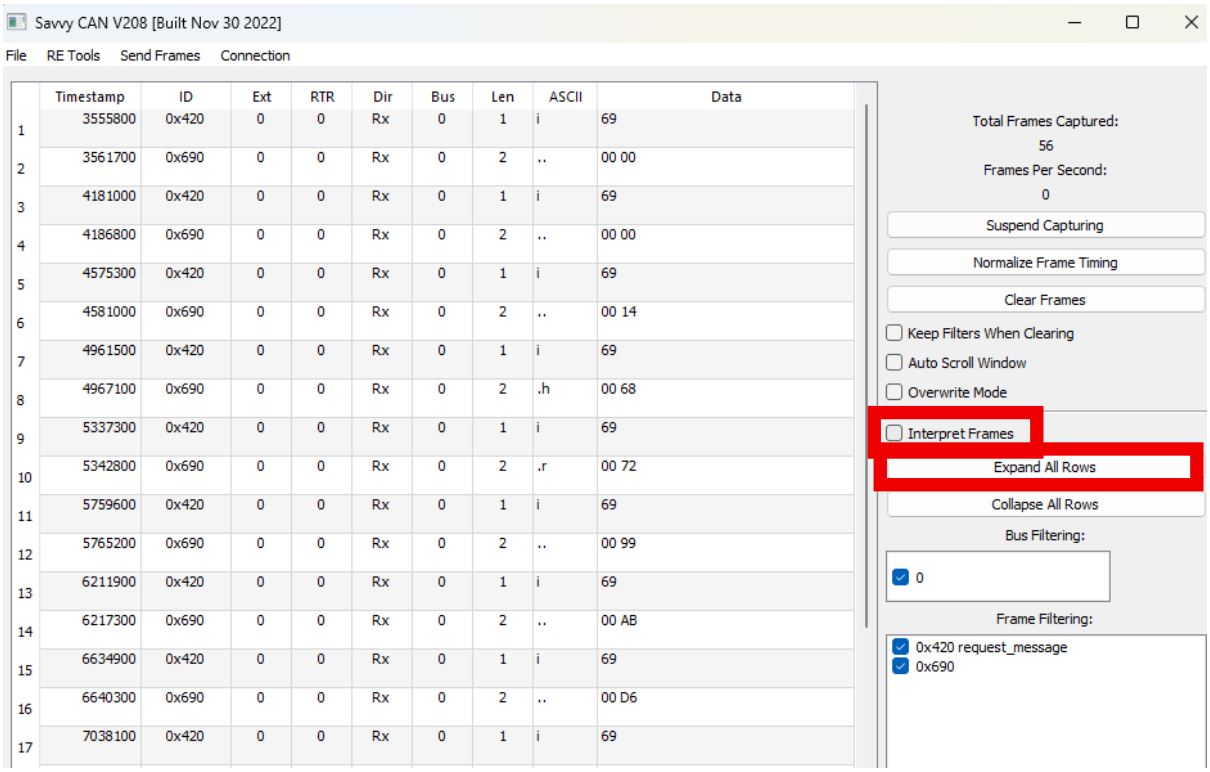
5. Open and edit the DBC file
   ([UNNAMED FILE])

   Set the content.

   1. [New Message]
   2. Double click the message, edit the following:
      a. Frame_ID
      b. name (anything you like, just an alias)
      c. Text/Background Colour
      d. (Optional) Comment
   3. Selecting this message, press [New Signal]
   4. Double click the new signal, set the following:
      a. Name
      b. Bit Length
      c. Byte Order (if LSB first)
      d. Type
      e. Position of the signal bits (set in the table on the left)
      f. Specific messages for values (e.g. state messages)



To view the results, close the editors and ***press save in the DBC file manager***. Your results will be ***lost*** if you forget to save! Go back to the table of the CAN trace, enable [Interpret Frames] on the right, then click [Expand All Rows] and check your results.



Tasks

Make a .dbc file that can decode the given trace file from a correctly written Q1. View the sample to see what it's supposed to to look like.

The data is as follows.

CAN ID	Data length	Data
0x420	1 byte	0x69 – request message; Others – don't care (i.e. no need handle);
0x690	2 byte	Unsigned integer of analog reading

Submit the .dbc file. Name it as q2.dbc and submit it.

Sample

	Timestamp	ID	Ext	RTR	Dir	Bus	Len	ASCII	Data
1	3555800	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
2	3561700	0x690	0	0	Rx	0	2	..	00 00 <sent_message> analog_reading: 0
3	4181000	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
4	4186800	0x690	0	0	Rx	0	2	..	00 00 <sent_message> analog_reading: 0
5	4575300	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
6	4581000	0x690	0	0	Rx	0	2	..	00 14 <sent_message> analog_reading: 20
7	4961500	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
8	4967100	0x690	0	0	Rx	0	2	.h	00 68 <sent_message> analog_reading: 104
9	5337300	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
10	5342800	0x690	0	0	Rx	0	2	.r	00 72 <sent_message> analog_reading: 114
11	5759600	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
12	5765200	0x690	0	0	Rx	0	2	..	00 99 <sent_message> analog_reading: 153
13	6211900	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
14	6217300	0x690	0	0	Rx	0	2	..	00 AB <sent_message> analog_reading: 171
15	6634900	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
16	6640300	0x690	0	0	Rx	0	2	..	00 D6 <sent_message> analog_reading: 214
17	7038100	0x420	0	0	Rx	0	1	i	69 <request_message> request_byte: Request_sent
18	7043500	0x690	0	0	Rx	0	2	..	00 F1 <sent_message> analog_reading: 241

Appendix

Given files

- Q1.cpp
  - C++ file for Q1.
- hw3q2.trc
  - CAN trace file for Q2.

Files to submit

- Q1.cpp
  - C++ file for Q1.
- Q2.dbc
  - DBC file to interpret the given trace.