EMK 310   
Group 1   
Lab book

# Team members:

## Armandt Fourie – 17028002

## Franco van Tonder – 17007594

## Henko Borstlap – 15010563

## Albertus Richter – 17020353



# Project Proposal

In EMK 310 the students are instructed to design a Microcontroller based Autonomous Robotic Vehicle (MARV) from first principles. The MARV will have to complete a lap on the final race day. On the final race day there will be a race track with four different colors printed on it. The MARV is to follow one of the color tracks on the race track in the fastest possible time, after which it must stop 100mm from the end of the track.

Group 1 (the team) will design and build the MARV in three separate stages. The EMK 310 assessments will take place in three practical sessions. In these sessions the students hill have to demonstrate each practicals required subsystems.

The aim of the first practical is to communicate with the MARV. This communication will allow the group select the operation mode of the MARV and calibrate the color sensors to race on a specific track. The first practical is focused on the MARV’s communication, data transfer and testing protocol. The communication system will implement serial communication with the aid of RS232 and I2C protocols. Data will then be transferred between the PIC18F45K22 and a serial EEPROM. The data will be written on the EEPROM and read when required. Lastly the group will have to implement a unit test protocol that allows the group to test and demonstrate the communication, data transfer and storage.

The second practical…

In the second practical the team now works toward to develop the colour sensor subsystem and integration with practical one. The colour sensor subsystem dictates the navigation of the MARV along the predefined colour track. To achieve accurate outputs from the sensors the team also implements a calibration strategy. Then to allow smooth on/off control for the MARV the team will build a touch start sensor using a metal disc. Lastly is the integration of practical 1 work.

The third practical…

On race day...

# Design

In order for the MARV to function the group has to design the entire system. The MARV system consists of various subsystems. The microcontroller (PIC18F45K22) is the main engine of the MARV and through it all the subsystems are integrated and communicate with one another.

The design is divided into subsystems namely:

* Power supply system
* Sensing system
* Start/stop system
* Serial communication system
* Data storage system
* Motor system
* Chassis subsystem

## Specifications:

### General specifications:

* The MARV should use either a PIC18F45K22 or a PIC18F45K20. We will use the former.
* The MARV should only use the motors that were given, we therefore have four to our disposal.
* The MARV should not exceed the dimensions 10cm x 18cm x 12cm (WxLxH) except for the “chimney.”
* The MARV should survive a drop from a height of 50cm.
* The MARV should use either one or two rechargeable 9V batteries.

### Technical requirements:

* Must be able to reliably distinguish between the colour white, and any of the following: green, blue, red and black in order to determine whether the correct line is being followed. It must do this with 95% accuracy.
* The MARV must complete the race track in less than three minutes [1]. On a 5m track, this means it should have a minimum average speed of 0.03m/s.
* The MARV must be able to detect when a line curves and turn in the correct direction within 100ms.
* The MARV must continue to function regardless of ambient light – the MARV must be able to follow a line even if lighting conditions are changed during runtime.
* The MARV should stop within 10cm of the end of the track automatically [1].  
  It should take less than one minute to calibrate the MARV [cite prac guidelines].
* The MARV must survive a drop from a height of 50cm without breaking [1].

# Hardware concept design

### Power Supply Subsystem:

This subsystem must provide power to all the components of the MARV. It will receive power from either one 9V battery and must convert this to a 5V DC output. The PIC will be powered by this 5V supply, for example. This subsystem must also facilitate charging of the batteries: using either a separate PIC16 or the PIC18F45K22 as a power supply, the system must be able to recharge the 9V battery.

Testing:

1. Using a multimeter or an oscilloscope, it should be confirmed that this subsystem outputs a 5V DC signal.
2. Using a multimeter or an oscilloscope, it must be confirmed that this subsystem supplies power (in the form of a DC voltage source) to the 9V battery until a predefined threshold for charge current is passed, at which point it stops supplying power to the charging subsystem.

### Sensor Subsystem:

This subsystem will accept receive power from the power supply subsystem. This means that this subsystem will be powered by a 5V DC power supply. This subsystem will be used to detect the colour/presence of a line beneath the MARV using an array of phototransistors with accompanying LEDs. This subsystem therefore accepts DC voltage and outputs an amplified signal that can be processed to determine the colour detected by each sensor.

Testing:

1. The LEDs of this subsystem must light up.

2. This subsystem must output a voltage depending on the colour track beneath each phototransistor. I.e. this subsystem must output distinct voltages for different colours.

Note that this subsystem will not be demonstrated or tested for practical 1.

### Start/Stop Button:

The start/stop button is responsible for starting and stopping the MARV. While in race mode, the MARV will initially wait for instructions given through the serial communications subsystem. If the start/stop button is pressed while the MARV is in static race mode, the MARV must transition into the moving race mode, where it rides along the lines of the racetrack. If the button is pressed while the MARV is in moving race mode, the MARV will stop and transition into static race mode.

Testing:

1. If the button is pressed while the MARV is in static race mode, the MARV must start moving.

2. If the button is pressed while the MARV is in moving race mode, the MARV must stop moving.

Note that this subsystem will not be demonstrated or tested for practical 1.

### Serial Communications Subsystem:

This system provides the terminal-based interface between the team and the MARV. This is used to transition between states and to set the welcome message. This is facilitated by a USB-to-serial bridge. The input to the bridge is connected to a PC, while the output is connected to the PIC microcontroller. This system must receive strings of arbitrary size and transmit them to the PIC. It must also be able to transmit string of arbitrary length from the PIC to the PC. Part of this subsystem will be implemented in firmware: depending on the input received, the MARV’s firmware must change between states that have their own test criteria. To test whether the system is functional, two LEDs will be used. One will be lit up to indicate that serial communication *to* the PIC was successful and the other will be lit up to indicate that serial communication ­*from* the PIC was successful.

Testing:

1. After a message is transmitted from the PC connected to the USB-to-serial bridge, the *­serial receive* LED lights up.

2. After a message is received from the USB-to-serial bridge on the PC, the *serial send* LED lights up.

### Data Storage Subsystem:

This subsystem will be used to store string to an EEPROM module. This module will be powered by the Power Supply Subsystem. This system is also mainly implemented using firmware. The functionality of this subsystem will be confirmed using two LEDs much like the Serial Communications subsystem. After a string is saved to EEPROM, one LED will be lit up and the other LED will be lit after a string is received from EEPROM and transmitted via the Serial Communications subsystem to the team member on a PC. The team member will be able to read the string (which is decided by the team) and can check the LED to confirm that a string was transmitted.

Testing:

1. After a string is transmitted from the EEPROM connected to the PIC, the *I2C receive* LED lights up.

2. After a string is received from the PIC to the Serial Communications subsystem, the *I2C send* LED lights up.

3. Following user prompts, the string (welcome message) is received from the PIC and displayed in the terminal.

### Motor Subsystem:

This subsystem is used to propel the MARV along the track. This subsystem receives its power from the Power Supply subsystem and will be controlled by the PIC according to the firmware. This system entails four electric motors that will be controlled in groups of two, based on the side of the MARV that they’re on. In order to move forward, the MARV will make all motors turn in the positive direction. In order to turn right, the left motors must turn in a positive direction while the right motors turn in a negative direction. The opposite is true for turning left. For testing and development purposes, a small routine will be created that drives the motors in such a way to demonstrate all necessary mobility.

Testing:

1. All four motors can be made to turn in the positive direction to move the MARV forward.
2. All four motors can be made to turn in the negative direction to move the MARV backward.
3. Two motors on any side can be made to turn in the positive direction while the opposite side is made to turn in the negative direction, to turn the MARV either left or right.

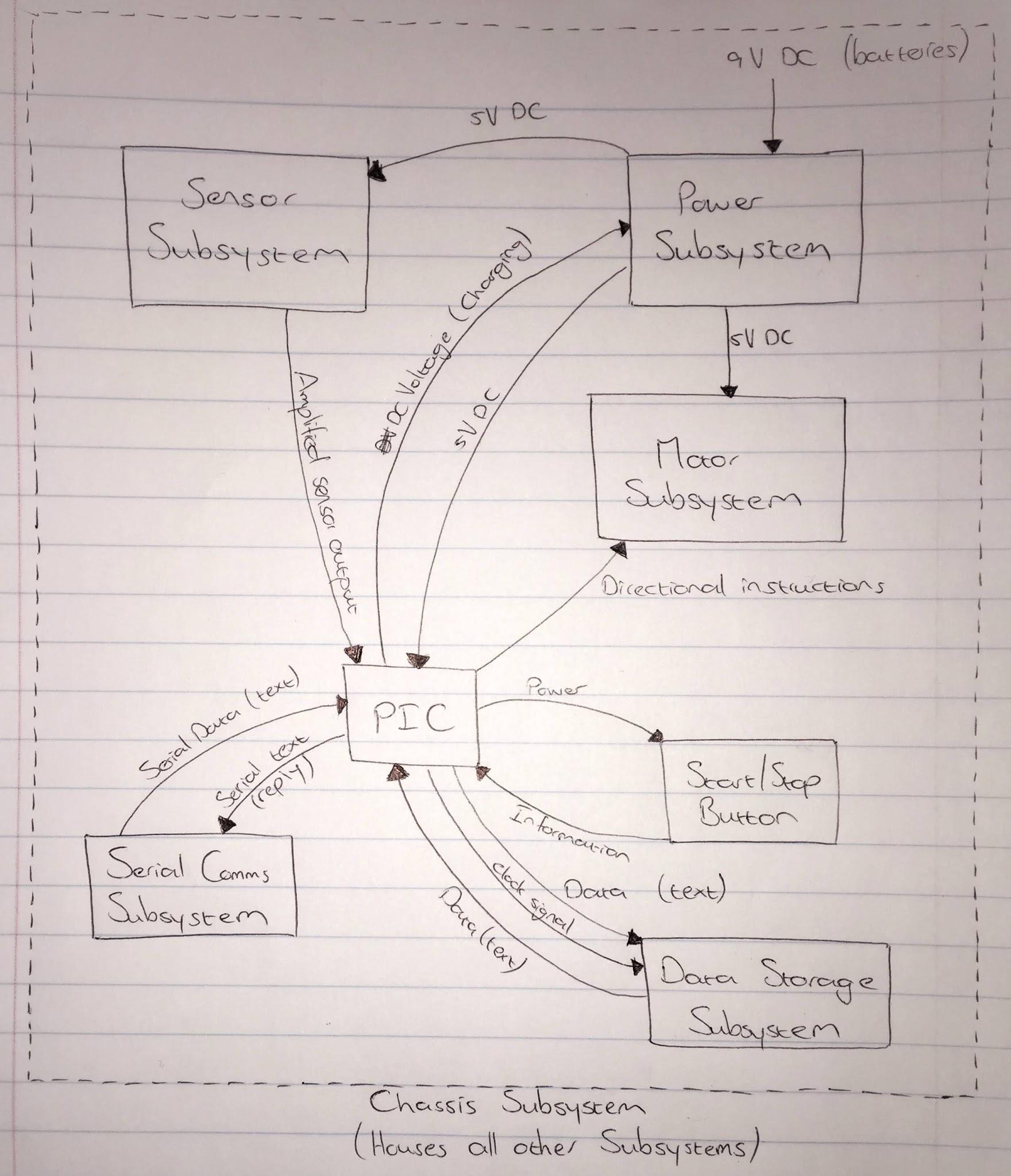
### Chassis Subsystem:

The chassis of the MARV must facilitate the connection and integration of all the other subsystems in a body that is smaller than the specified maximum dimensions of 10cm x 18cm x 12cm [1]. The chassis must be light enough for the MARV’s motors to propel the MARV at the minimum speed of 0.03m/s. The chassis must also be durable enough to survive a drop from a height of 50cm onto a hard surface.

Testing:

1. The chassis must have dimensions smaller than 10cm x 18cm x 12cm.
2. The chassis must weigh less than a maximum weight that will be determined later.
3. The chassis must not break or allow any subsystems on it to break when dropped onto a hard surface from a height of 50cm.

### Hardware block diagram:



**Figure 1: Block diagram for hardware subsystems**

# Hardware detail design

### Power Supply, Motor and Start/Stop Button Subsystems:

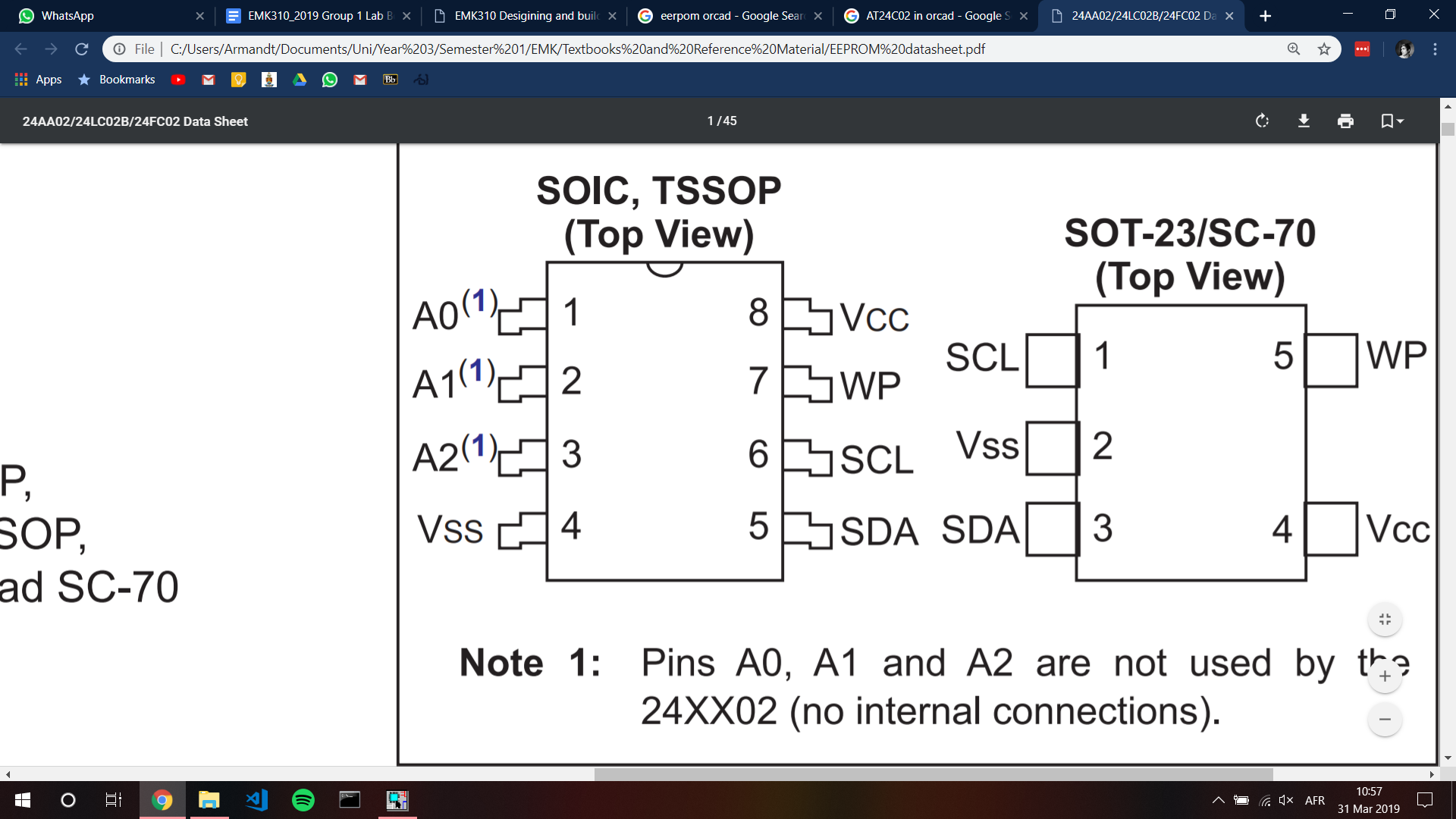
These subsystems are not required to be operational for the first practical and have therefore not yet been designed in their entirety. Once this has been done, this section will be completed for each of the subsystems.

### Serial Communications Subsystem:

This subsystem did not require any calculations for physical implementation. The USB-to-serial converter is connected to the PC via USB port on one end, and the relevant pins are connected to the appropriate ports on the PIC (TX pin on the converter is connected to PORTC 7, while RX pin is connected to PORTC 6).

### Data Storage Subsystem:

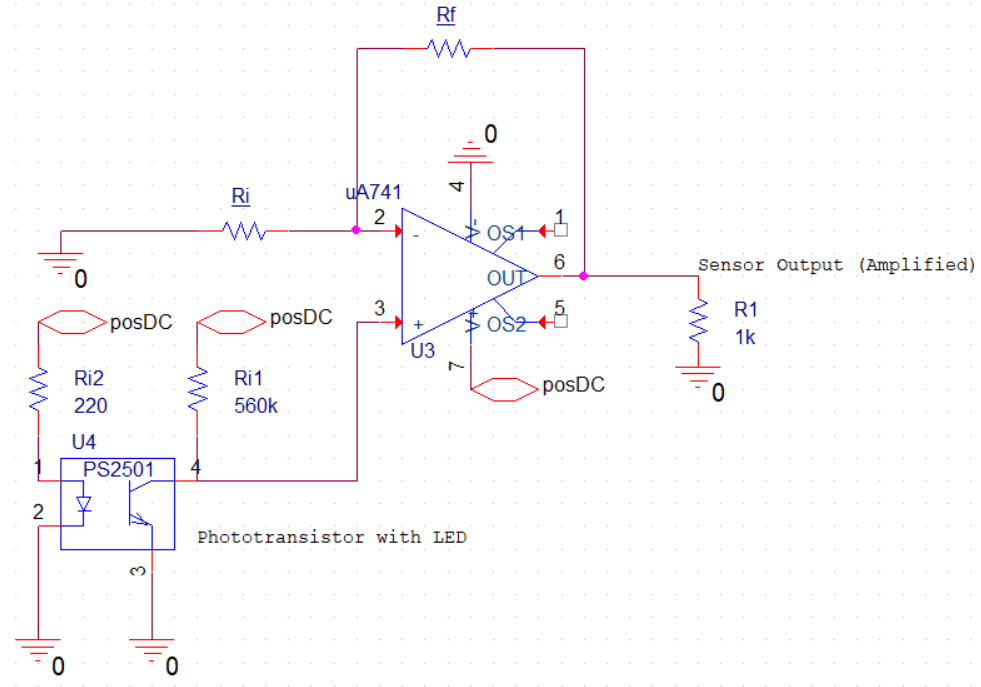
In order to save power, an frequency of 1MHz was chosen for the internal oscillator of the PIC. As a result, the only feasible frequency that could be used for the data storage subsystem was 100kHz. As specified by the EEPROM module datasheet, the pull-up resistor connected to the data transfer pin has a resistance of 10kΩ [3]. The clock input pin (SCL) to the EEPROM was connected to the PORTC3 pin and the data transfer pin (SDA) was connected to the PORTC4 pin. The power source pin (VCC) was connected to a 5V pin on the Microchip curiosity board. Pins A0 to A2 and Vss were connected to the ground pin of the curiosity board. The write protection pin (WP) was also connected to ground to enable writing to the device.



**Figure 2: 24LC02B EEPROM module pinout**

### Sensor subsystem:

The sensor subsystem was designed in the first practical for the module ENE310. The following figure depicts the circuit diagram for one sensor-LED pair. This team’s MARV will implement five of these pairs in order to sense the presence and relative position of lines beneath it.



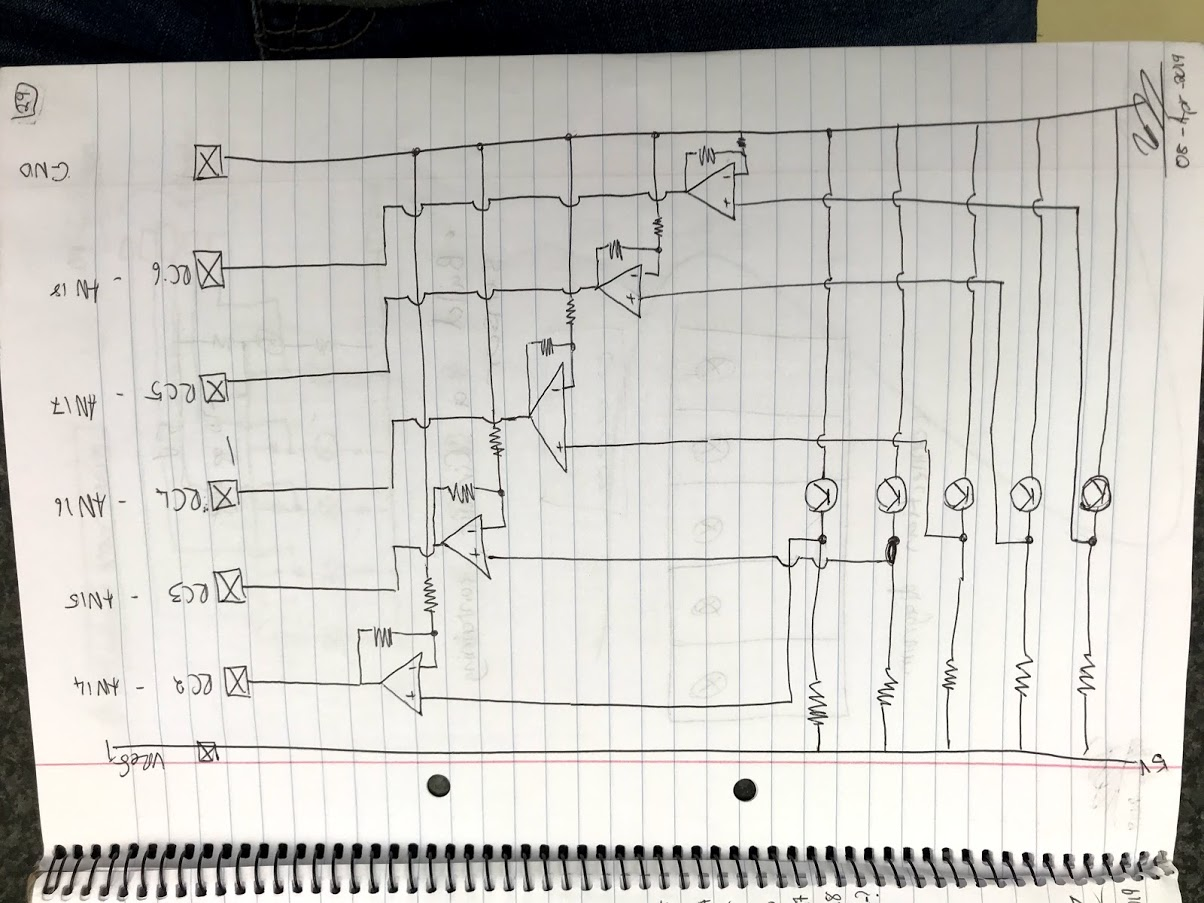
**Figure 3: Circuit diagram from Sensor Subsystem sensor-LED pair.**

The phototransistor and LED can be seen in the lower left corner of Figure 1. The output from the sensor is fed into a non-inverting amplifier to apply a gain factor to the output. The necessary current limiting resistance to the phototransistor was calculated as 560kΩ and for the arrays of five LEDs as 56Ω [2]. The values of Rf and Ri are arbitrary as long as they have roughly equal resistances, resulting in a gain factor of at least 20 V/V. It was decided that Ri would have a resistance value of 1kΩ while Rf would have a variable resistance of between 19kΩ and 22kΩ [2]. The output of each of these sensor-LED pairs would be fed into the ADC pins of the PIC to facilitate sensing. The ports labeled “posDC” represent 5V DC power supplies.

For the purposes of practical 1, the hardware implementation of the sensor subsystem was not implemented.

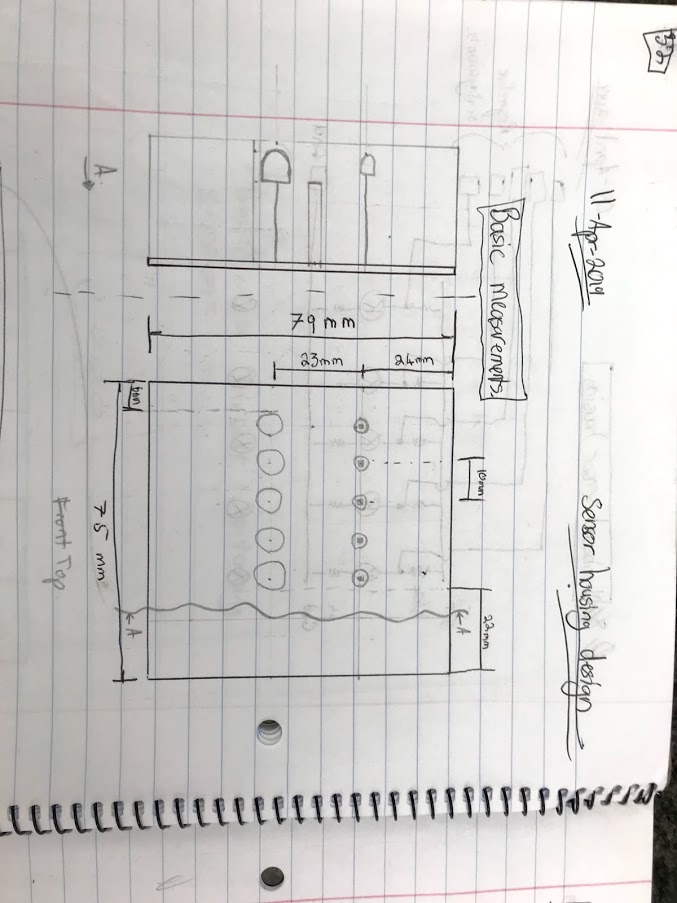
Practical two required the team to design and build the previously mentioned sensor array. The design is split into three sections to integrate in the end to form a whole colour sensor subsystem. The sections are the housing, the phototransistor sensors together with the LEDs and the amplification circuit.

First the team decided to use 5 sensors as previously mentioned. This should make the navigation of the MARV more controllable. The circuit is then designed around the fact that the subsystem requires 5 sensors and 5 LEDs. Below is the design of the colour sensor system circuit. The circuit is built on a veroboard, because of the robustness of the soldered connections.

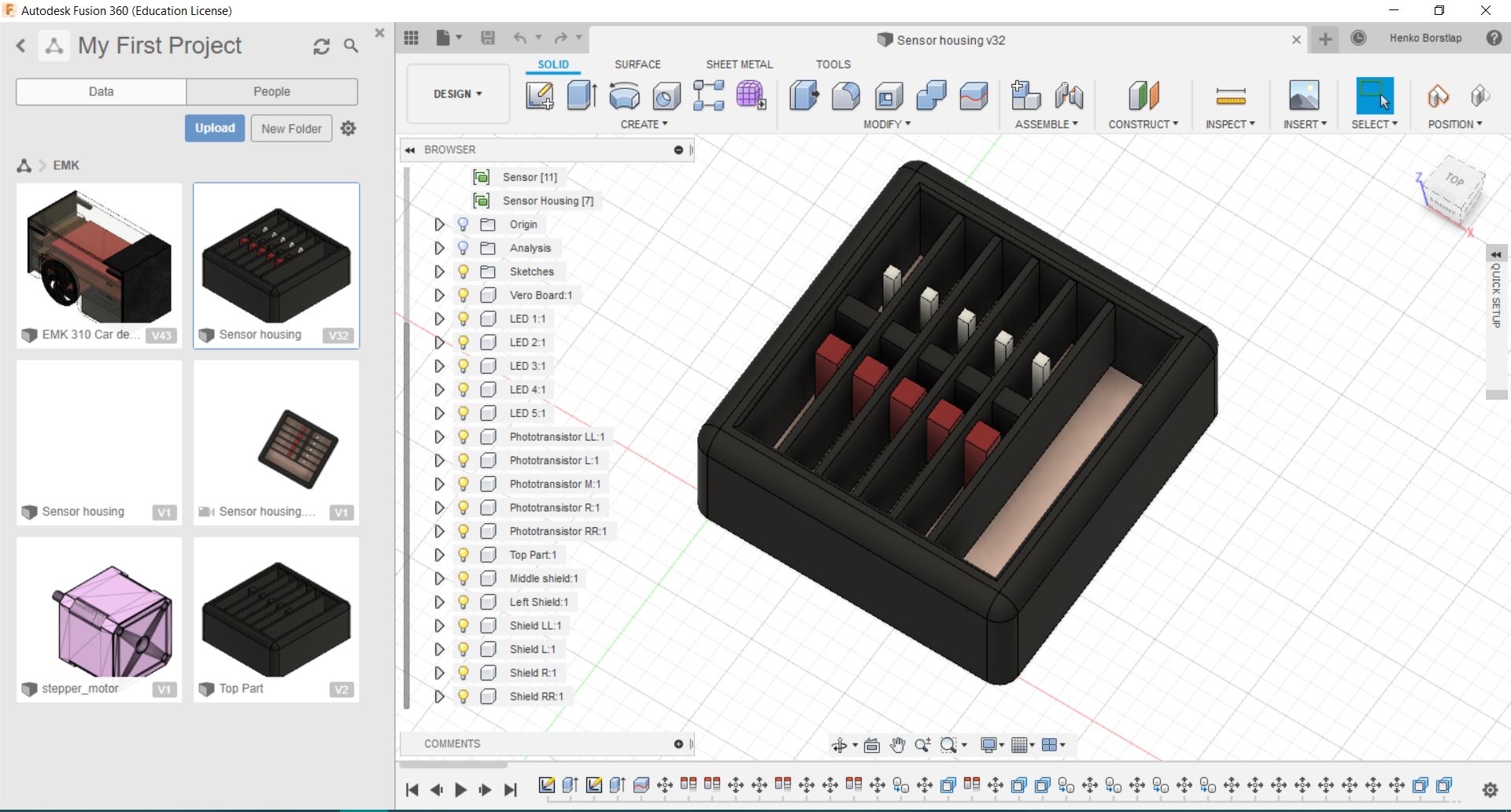


**Figure 4: Circuit diagram for LED and phototransistor arrays.**

The last design objective for this subsystem is to shield out ambient lighting effectively. The group used the existing LED’s and sensors that was already on the veroboard and designed the housing around that. The first step was to model what we have, then to design a housing that can be printed and used in the practical.



**Figure 5: Sketch of veroboard circuit.**

****

**Figure 6: 3D design of sensor housing and existing veroboard.**

### Touch start subsystem:

The MARV requires a person to touch it and let it enter race mode as this happens. The decision was made to use a capacitive touch sensor. It is fast reacting and easy to implement.

The metal disc should have a potential difference between ground and the RC2 pin. The PIC should read the voltage difference every set time to allow the voltage to return back, after it has been touched. The TS (Touch Start) should be able to output a voltage change of at least 19.61mV.

## Hardware results

For the first practical the only hardware subsystems that had to be operational were the Serial Communications subsystem, the Data Storage subsystem and the integrated debugging system. The debugging system was confirmed to be functional because the relevant LEDs were lit up as expected.

### Serial Communications Subsystem:

The *serial receive LED* and well as the *serial transmit LED* were lit following serial communications with the PIC, indicating that communication was successful. It was also seen that the PIC was sending text to the terminal on the

### Data Storage subsystem:

The *I2C send* as well as the *I2C receive* LEDs were lit following the MSG command and upon startup of the PIC (welcome message), indicating that both sending and receiving data using the I2C was implemented successfully.

### Colour sensor subsystem

The colour sensor subsystem consists of two stages:

Stage one is sensing a voltage level from the phototransistors. Stage two is amplifying the voltages to ensure a distinct voltage level for each colour.

Below is a table of the amplified voltage level colour sensor outputs.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Colour** | **LL [V]** | **L [V]** | **M [V]** | **R [V]** | **RR [V]** |
| **Black** | 5 | 5 | 5 | 5 | 5 |
| **Red** | 4.6 | 4.7 | 4.7 | 4.6 | 4.3 |
| **Blue** | 3.85 | 3.85 | 3.8 | 3.0 | 3.4 |
| **Green** | 2.8 | 1.85 | 1.75 | 1.5 | 1.15 |
| **White** | 2.5 | 1.7 | 1.55 | 1.3 | 1.14 |



**Figure 7: Voltage levels of 5 voltage sensors reading BLACK.**

### Touch start subsystem

The TS switches the MARV from idle race mode to dynamic race mode.

### System integration

Remove the EEPROM from the system to store the start-up message. Rather store and read the startup message from the onboard EEPROM memory.

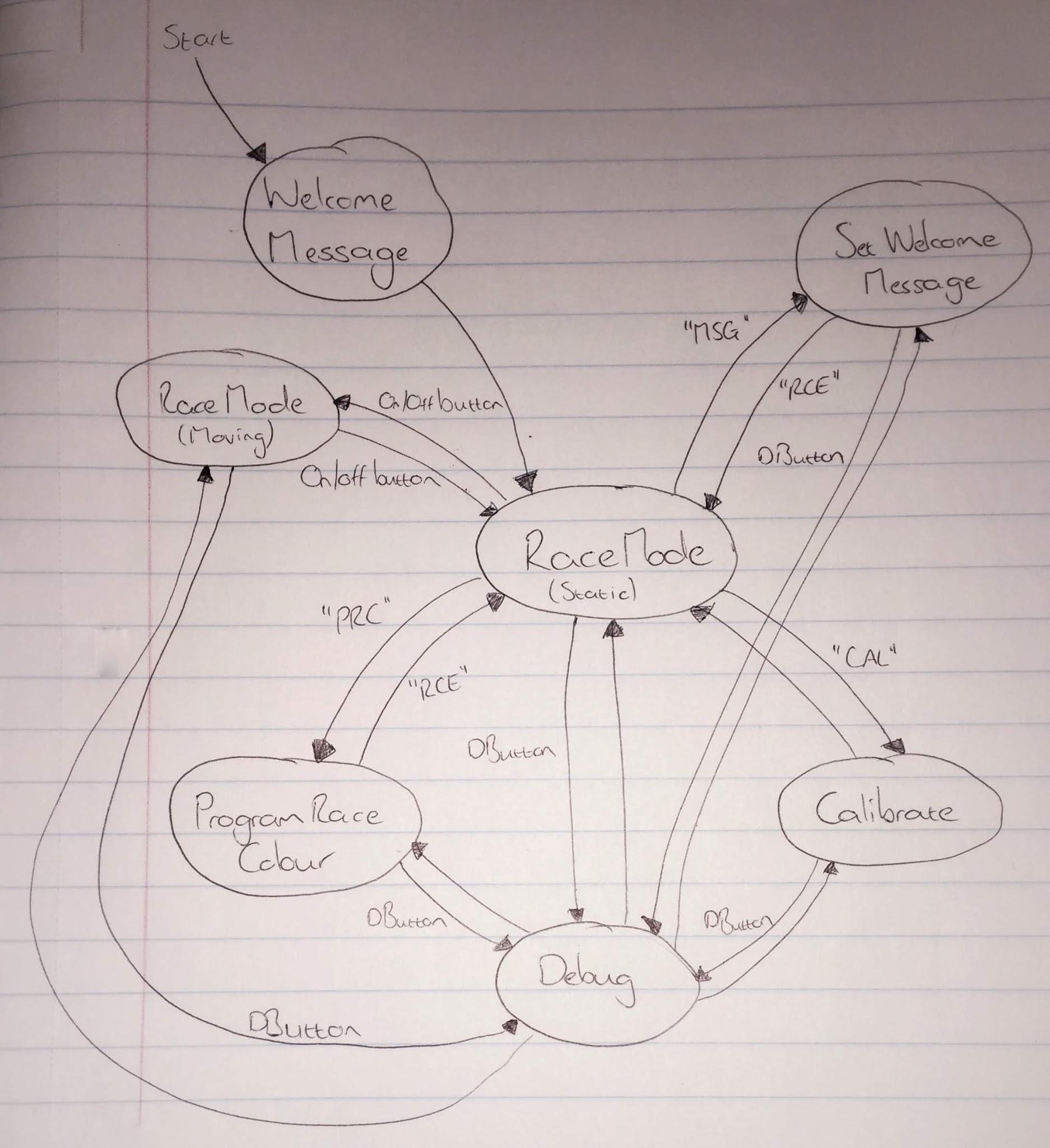
Integrate the serial communication from practical one.

### 

# Firmware design

In order to test whether each state/mode has been successfully implemented, the outputs of the state will be checked. If the state delivers the correct outputs as indicated in this section, the state is deemed functional. For example, the “Welcome Message” state is deemed functional if it lights the LEDs indicating successful serial communications and successful read from data memory as well as transmitting the welcome message to the team member’s PC.

## Firmware concept design



**Figure 4: Firmware state diagram**

Figure 4 shows the state diagram for the MARV’s firmware. It shows that the MARV will initialize and send a welcome message and then go to race mode. From here it changes state depending on the input from the team member. At any time, the state can be changed to Debug Mode by pressing the DButton (debug button). After the debug subroutine completes, the MARV transitions back to the prior state.

Program race colour will wait until the user enter a valid character to race. When it receives one, it will store this value on the PIC. When the team member enters the “RCE” command, it will transition back to the race mode.

Calibrate mode will cycle through the five colours present on the race track, each time measuring the output from the sensor array and storing its value in a register. After each successful calibration, an LED is lit. During calibration, a character corresponding to the colour being calibrated is displayed on the SSD. When done, calibration mode returns automatically back to race mode.

Set welcome message will wait for the team member to enter a message followed by a $ (the delimiting character). When received, the MARV will store this value on EEPROM memory. The team member can enter the welcome message as many times as they like. When the MARV receives the “RCE” command, it will transition back to race mode.

The moving race mode will employ the navigation subroutine detailed in the section below. This is the mode in which the MARV races on the track. The MARV will follow the line, driving forward and turning left and right as necessary to stay on the line. When the MARV reaches a black line, it will stop within ten centimetres of that line. The moving race mode is entered when the MARV was in static race mode and the start/stop button is touched. If the button is touched while the MARV is moving, the MARV will transition back to static race mode and stop.

## Firmware Detailed Design and Results

### Data storage Subsystem

I2C Initialization

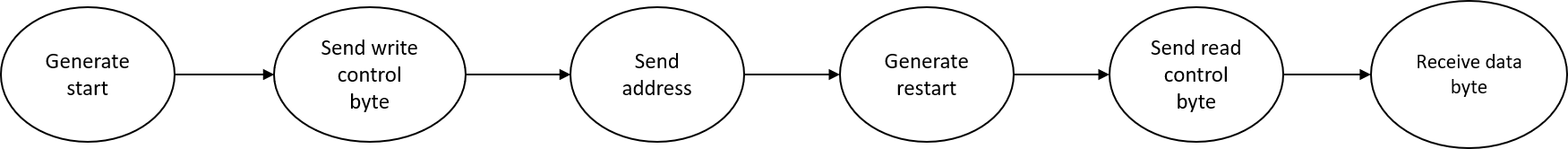
I2C master mode is selected by setting SSPxEN and SSPxM3 of SSPxCON1. The I2C clock is set to 100kHz for a pic oscillator of 4MHz by storing the BRG value 0x09 in the SSPxADD register. The SSPxSTAT, SMP bit is set to disable slew rate control for a standard speed of 100kHz

I2C WRITE



The I2C write sequence is initiated by generating a start condition. This is done by setting the SEN bit of SSPxCON2. After the SEN bit is cleared by hardware, the write control byte must be transmitted to the EEPROM. The write control byte for the 24LC02B is 0b1010xxx0. Data transmission to the EEPROM is initiated by storing data in the SSPxBUF. The address one wants to write to is transmitted next. The 24LC02B has one block of 256 Bytes. The Data to be written is transmitted next and a stop is issued by setting the PEN bit of SSPxCON2. The 24LC02B does not acknowledge during a write cycle. To determine when the write cycle is complete the SSPxCON2, ACKSTAT bit is polled after the stop condition.

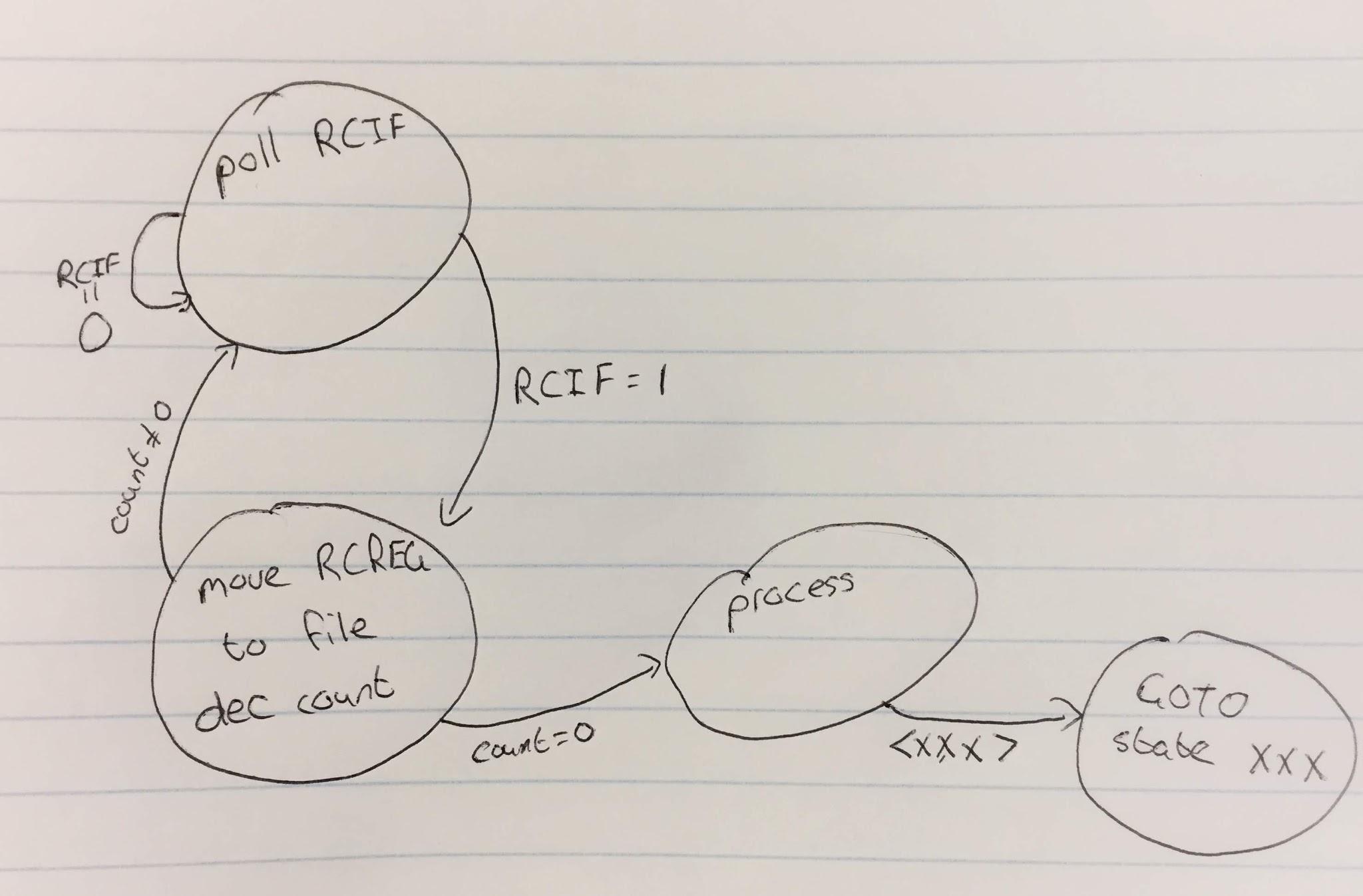
I2C READ



The I2C read sequence is initiated by generating the start condition; done by setting the SEN bit of SSPxCON2. After the SEN bit is cleared by hardware, the write control byte must be transmitted to the EEPROM. The write control byte for the 24LC02B is 0b1010xxx0. Data transmission to the EEPROM is initiated by storing data in the SSPxBUF. The address one wants to read from is transmitted next. The write sequence is interrupted by generating a restart condition. This keeps the 24LC02B internal address pointer at the address you transmitted before the restart. The read control byte is transmitted to the EEPROM. The read control byte for the 24LC02B is 0b1010xxx1. The data is read by setting the SSPxCON2, RCEN bit and moving the received data from the SSPxBUF. The Read sequence is terminated with a stop condition.

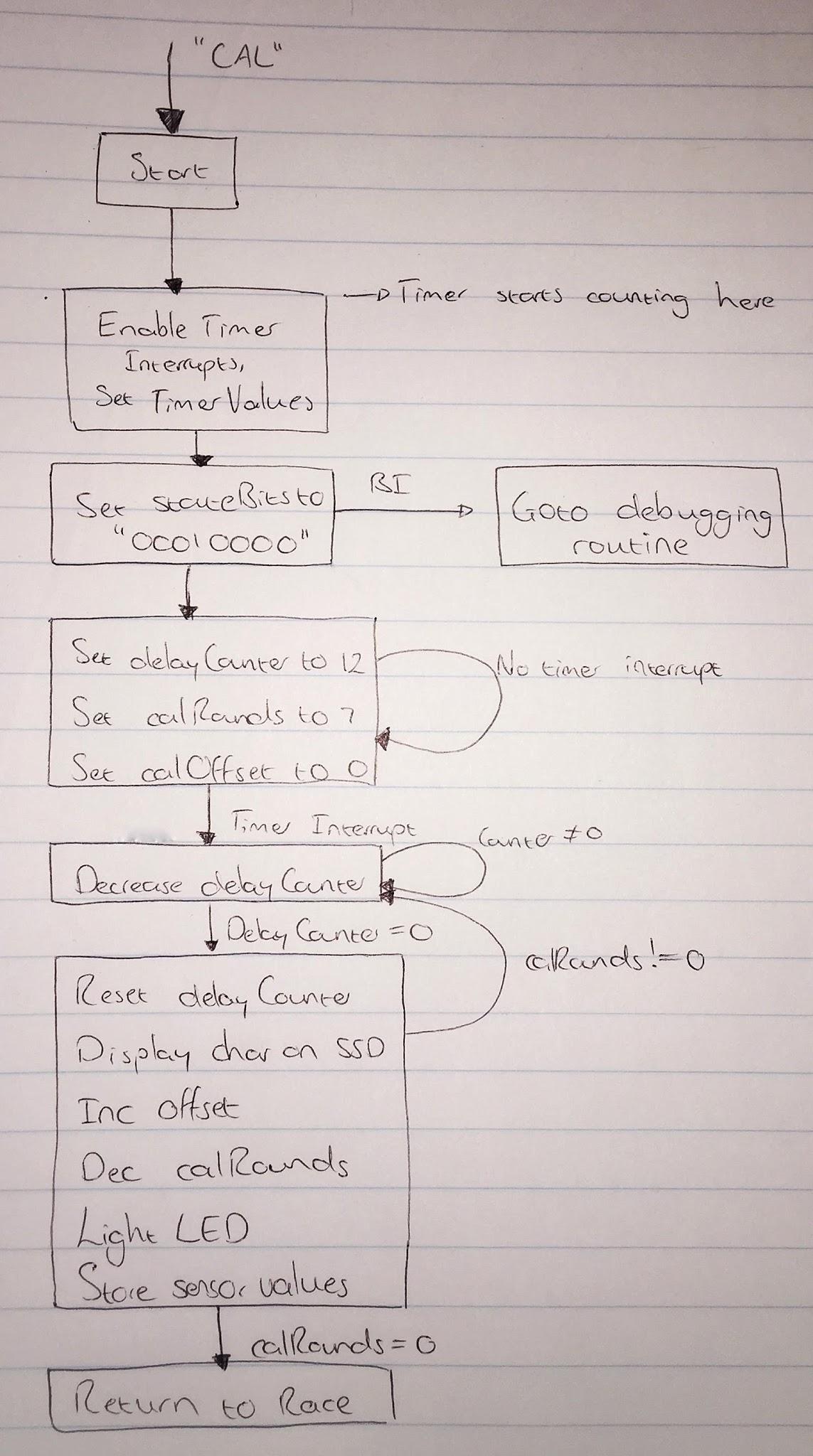
In subsequent discussion, reading and writing to storage is done as discussed above.

### Serial Communications:



**Figure 5: Serial communications flowchart**

### Calibration Subroutine:



**Figure 5: Flow diagram for the calibration subsystem**

The purpose of the calibration subroutine is to store the values received from the sensor subsystem into

The calibration subsystem is entered when the MARV receives the “CAL” message from the user through the serial communications system.Once initiated, this state sets the relevant bit to enable Timer 2 of the PIC. It sets the pre and post scalers to 16x each and the rollover register is loaded with the value of 245, and sets the delayCounter register to 12. It also sets the stateBits register to “0001000” so that the debugging ISR can dump the correct port to the output LEDs. Following this step, the PIC will wait in the same position until the timer causes in interrupt. At this point the state will decrement the value of the delayCounter register. If the register is equal to 0, three seconds have passed and the state will display the first character on the SSD for the first colour, red. Each time the delayCounter reaches zero, the calRounds register is also decremented and the calOffset register is incremented by two, which in turn changes the character that is displayed with each execution of the code. When calRounds reaches zero, the subroutine finishes, disables timer 2 and returns to the subroutine that called it.

Outputs:

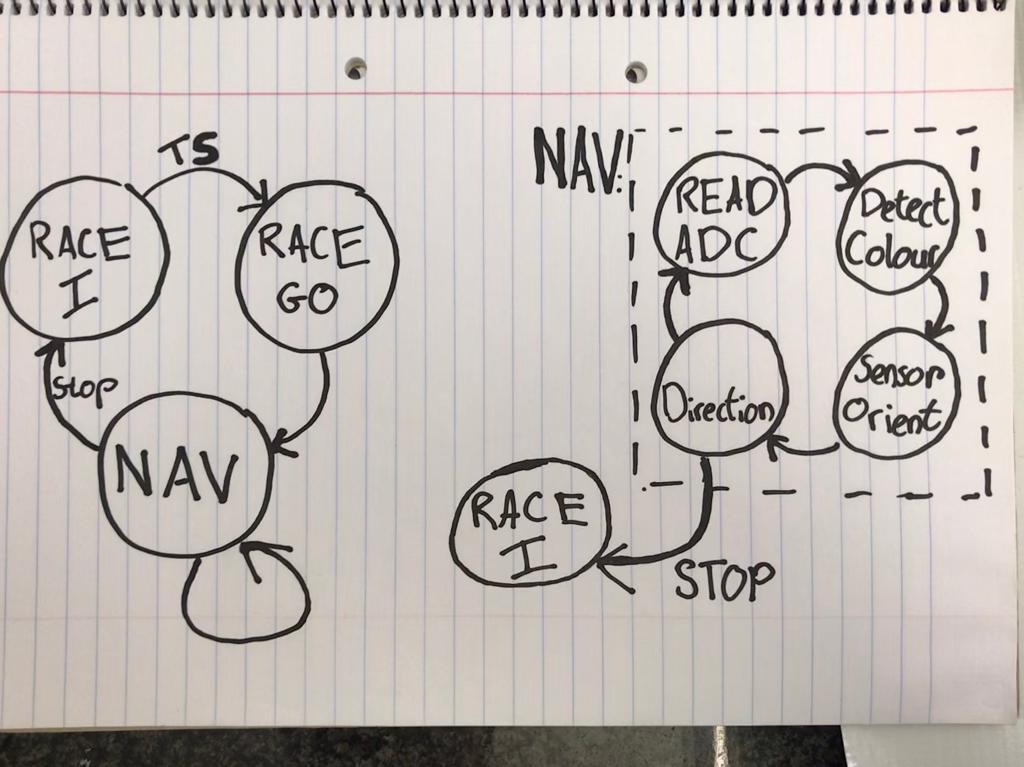
* LED confirmation for calibration success
* SSD character indicating colour being calibrated
* Measurement and storage of sensor values for five colours within 1 minute. I.e maximum 12s per colour to calibrate.

### Navigation Subroutine

The navigation subroutine will use the inputs from the sensor subsystem to facilitate navigation of the MARV. This routine must attempt to keep the desired colour line in the middle of the sensor array by turning in the correct direction based on the inputs. This subroutine is responsible for moving the MARV forward as well.

The navigation subroutine is accessed from the idle RACE mode by using the Touch Start subsystem. The subroutine periodically reads data from the sensor subsystem for each sensor. After the voltage readings are read, they are compared to the calibrated range of voltages for each track colour. At this stage a bit is assigned to each sensor indicating what colour it is currently on. After assigning colours, the orientation of the the entire sensor relative to the track is determined. This is done by determining which sensors are on race colour. After the orientation is established, the direction that the MARV needs to move is decided. The different orientations of the five sensors and corresponding direction decisions shown in the following tables as done for the Nibble. The navigation subroutine when the stop race condition is met.

The navigation subroutine is accessed from the idle RACE mode by using the Touch Start subsystem. The subroutine periodically reads data from the sensor subsystem for each sensor. After the voltage readings are read, they are compared to the calibrated range of voltages for each track colour. At this stage a bit is assigned to each sensor indicating what colour it is currently on. After assigning colours, the orientation of the the entire sensor relative to the track is determined. This is done by determining which sensors are on race colour. After the orientation is established, the direction that the MARV needs to move is decided. The different orientations of the five sensors and corresponding direction decisions shown in the following tables as done for the Nibble. The navigation subroutine when the stop race condition is met.



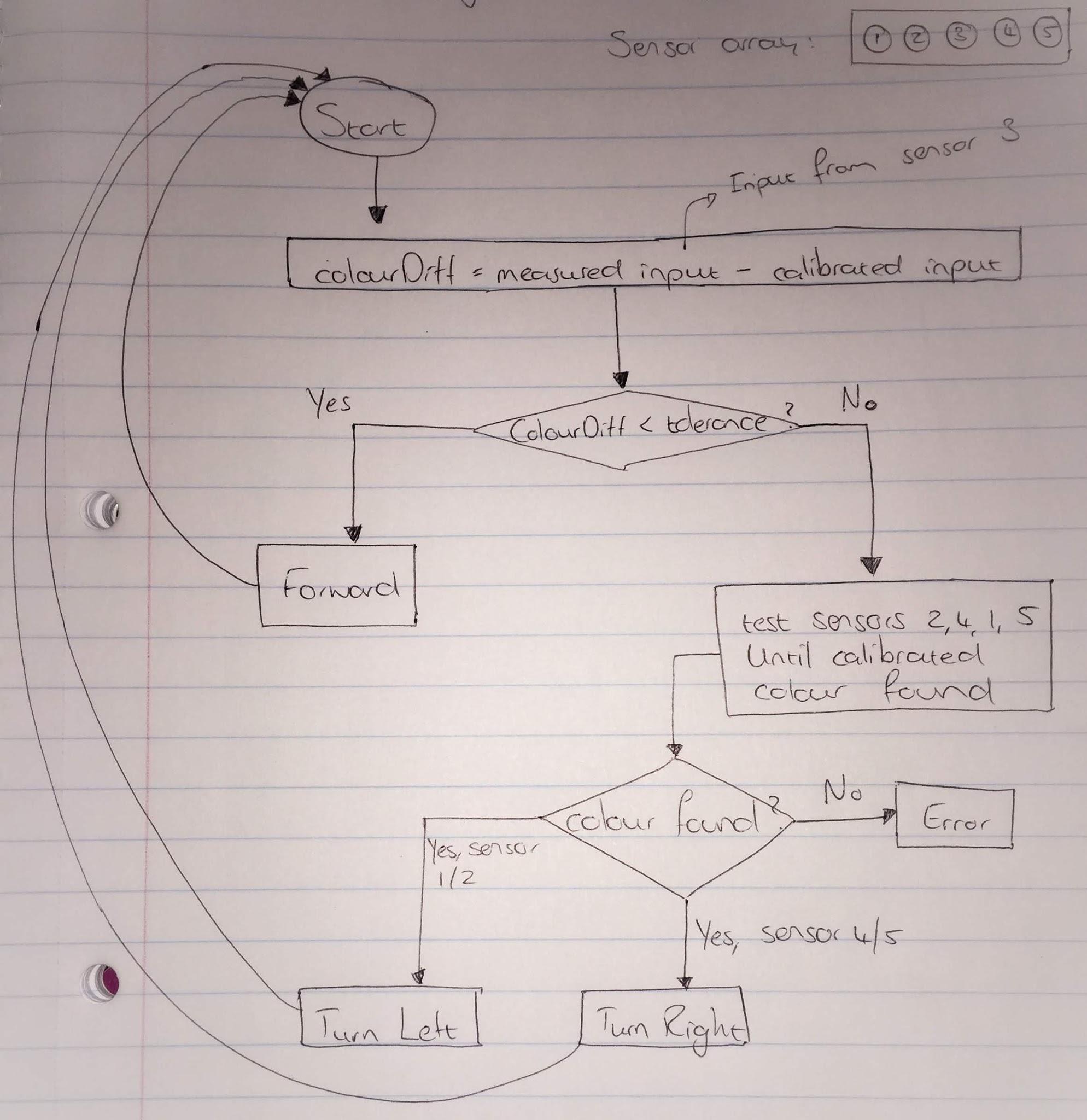
**State transitions for navigation subsystem**

|  |  |  |  |
| --- | --- | --- | --- |
| **Orientation**  (1 = on race colour, 0 = off race colour) | **Direction**  (On-off control) | **Left Motor**  **Control** | **Right Motor Control** |
| 00000 | Sweep | TBD | TBD |
| 10000 | Left | OFF | ON |
| 01000 | Left | OFF | ON |
| 00100 | Straight | ON | ON |
| 00010 | Right | ON | OFF |
| 00001 | Right | ON | OFF |

**Direction decision and motor control for On-off control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Orientation**  (1 = on race colour, 0 = off race colour) | **Direction**  (Proportional control) | **Left Motor**  **Control** | **Right Motor Control** |
| 00000 | Sweep | TBD | TBD |
| 10000 | Sharp left | Slow | Fast |
| 01000 | Left | Medium | Fast |
| 00100 | Straight | Fast | Fast |
| 00010 | Right | Fast | Medium |
| 00001 | Sharp right | Fast | Slow |

**Direction decision and motor control for Proportional control**



**Figure 6: Flow diagram for the navigation subroutine**

### Motor Control Firmware

PWM is done using the CCP modules on the PIC. The ECCP modules allow you to switch between half-bridge and full bridge PWM. However, there is also a standard mode that I’ll be using since we only need a plain old square wave. This is only supported by CCP4. Regardless of which module you choose, you get up to 10 bits of resolution, which is controlled by the registers:

* PR2
* T2CON
* CCPRxL
* CCPcCON

The following steps should be taken when configuring the CCP module for standard PWM operation:

1. Disable the CCPx pin output driver by setting the associated TRIS bit.
2. Select the 8-bit TimerX resource, (Timer2, Timer4 or Timer6) to be used for PWM generation by setting the CxTSEL bits in the CCPTMRSx register.(1)
3. Load the PRx register for the selected TimerX with the PWM period value.
4. Configure the CCP module for the PWM mode by loading the CCPxCON register with the appropriate values.
5. Load the CCPRxL register and the DCxB bits of the CCPxCON register, with the PWM duty cycle value.
6. Configure and start the 8-bit TimerX resource:
   * Clear the TMRxIF interrupt flag bit of the PIR2 or PIR4 register. See Note 1 below.
   * Configure the TxCKPS bits of the TxCON register with the Timer prescale value.
   * Enable the Timer by setting the TMRxON bit of the TxCON register.
7. Enable PWM output pin:
   * Wait until the Timer overflows and the TMRxIF bit of the PIR2 or PIR4 register is set. See Note 1 below.
   * Enable the CCPx pin output driver b

Set CCP module in PWM mode deur Bit 3-0 in CCP4CON te set na 11xx.

The period is found using the following formula:

The value to be held in the PR2 register is found using the following equation:

Using the equation above, the maximum FPWM is achieved using:

FPWM, max is equal to **500kHz**. Similarly, the minimum FPWM is found using:

FPWM, min is equal to **244Hz**. It is unclear to me exactly what effect this will have on the operation of the motor. Nevertheless, using a FPWM value of 5kHz leads to using a value of **199 for PR2** when using a timer prescaler of 1.

Using this value in PR2, we can calculate the valued that need to be loaded into the CCPR4L and DC4B registers.

|  |  |
| --- | --- |
| Duty Cycle | CCPR4L:DC4B Value |
| 100% | 800 |
| 75% | 600 |
| 50% | 400 |
| 25% | 200 |

### Debugging Functionality

This firmware “entity” will in reality be implemented in pieces in more than one firmware state. The purpose of this “subroutine” is to provide feedback to the team regarding functionality of other subroutines. For the serial communications and data storage subsystems (and subroutines) LEDs must be lit following successful reception and transmission of data via serial and I2C. These LEDs are simply turned on and off following relevant events in the respective subroutines.

The more “tangible” component of this subroutine is implemented as a single subroutine in the code. This component must dump the contents of a register to an output port, lighting up a collection of LEDs. This subroutine is triggered by an external interrupt from a pushbutton. When the pushbutton is pressed, the PIC enters the debugging subroutine. Once in, the subroutine branches depending on the state the PIC was in before the interrupt. This is determined by the contents of the stateBits register. In each branch, the contents of the relevant register is moved to the working register and then to PORTA, where the LEDs are connected. There will then be a delay, the duration of which is modular. Following the delay, the PORTA is reset to the value it held before the interrupt and the code returns to wherever it was when the interrupt occurred. The port dumping segment of this subroutine is discussed later.

### Welcome message

This is executed when the PIC starts up. It reads the welcome message from the EEProm memory module and transmits it via serial to the team member’s PC. The MARV does not display anything on the SSD at this stage.

Outputs:

* Welcome message sent through serial
* Light *serial send LED* PORTA5
* Light *data read LED* PORTA7

Race mode:

While in race mode, the MARV must stay in place and await commands from the team member’s PC. Depending on the message received, the state must be changed. While in this mode, the MARV will display a 2 on the SSD. When the MARV enters this state, it must send “MARV races X” where X indicates the colour line it will race on, or L for maze racing. Before the Start/Stop button is pressed, the MARV will be stationary and will display the 2 on the SSD. After the button has been pressed, it will begin moving and will display the character representing the colour it is racing on the SSD.

Outputs:

* “MARV races X” message through serial
* Character output on SSD - 2 for stationary mode
* Character output on SSD - colour code in moving mode
* Light *serial receive LED* PORTA4
* Light *serial send LED* PORTA5

### Program Colour Subroutine

This subroutine is entered by sending “PRC” to the MARV while it is in race mode. Once in this mode, the MARV must send “What shall MARV race?” through serial. This is read from EEPROM to serial. The user will then enter a character indicating the colour, or alternatively L for maze racing. While in this mode, the SSD must display 1. When the user has sent the colour to the MARV, it will set the racingColour register accordingly. Following this, the MARV transitions back to race mode.

Outputs:

* “What shall MARV race?” message through serial
* Light *serial receive LED* PORTA4
* Character 1 on SSD

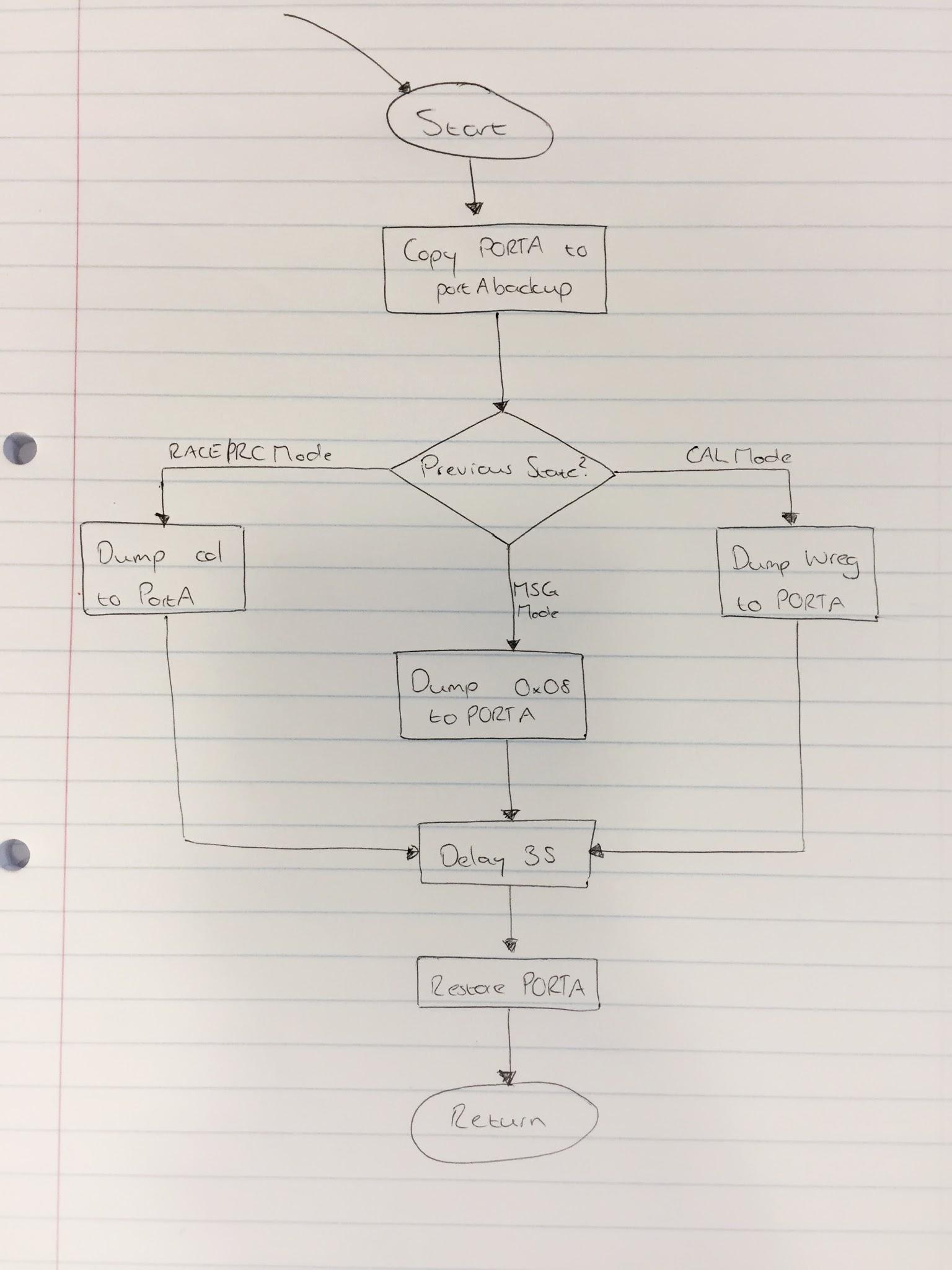
### Message Mode

This mode is used to customize the message that the MARV transmits upon startup. From race mode, when the characters “MSG” are received from the team member, the MARV will display the character 0 on the SSD. The user can then send a new string to the MARV, which it will store on the EEPROM module so that it can be read later. This mode is exited when the team member issues the “RCE” command.

Outputs:

* Character 0 on SSD.
* Light *data write LED* PORTA6
* Light *serial receive LED* PORTA4

### Debug Mode



**Figure 7: Flow chart for Debug state**

This mode is included to make debugging the MARV easier. At any point during operation, when the “debug button” (DButton) is pressed, the MARV must stop what it is doing and dump a register to some LEDs. Following this, the MARV must return to its previous state and continue doing what it was doing there. The register that gets dumped to the LEDs depends on the state it was in before. For this reason, each state in the firmware of the MARV sets one bit of a register called stateBits. The debug routine uses this encoding to determine which port gets dumped. After dumping the register, there is a delay before the MARV returns to its previous state.

Outputs:

* Race mode: the col register gets dumped to LEDs
* Cal mode: the wreg register gets dumped to LEDs
* Program Colour mode: col register gets dumped to LEDs
* Message mode: 0x08 gets dumped to the LEDs (first character of the message)

### Error Handling

If at any point during execution of the firmware the MARV receives invalid input, it will send the string “ERROR” to the team member via serial and will then go to race mode.

# Administrative Matters

## GANTT Chart

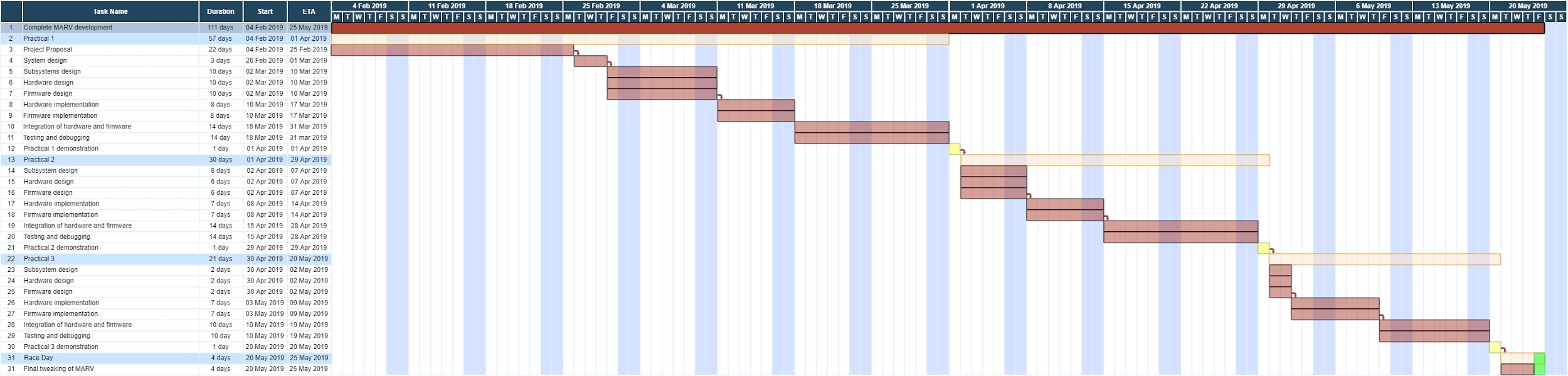


Fig1: Link to higher resolution chart - <http://bit.ly/Group1Gantt>

## Budget:

Current expenses are as follows:

|  |  |  |
| --- | --- | --- |
| **Item** | **Quantity** | **Cost** |
| USB to serial converter | 1 | R25.00 |
| 9V Rechargeable battery | 1 | R160.00 |
| Phototransistors | 20 | R30.00 |
| LN347 Op Amps | 10 | R80.00 |
| **Total:** | | R295.00 |

This means that we have R205 left for the remainder of the project. We do not expect to spend any money on materials for the chassis of the MARV, with the possible exception of a small sum for 3D printing the sensor array.

## References:

1. T. Hanekom, *“Development of a microcontroller-based autonomous robotic vehicle (MARV)”*, University of Pretoria, 18 Feb. 2019.
2. A. Richter, F. M. van Tonder, H. Borstlap, W. A. Fourie, *“ENE310: Practical Report 1 - Sensor Subsystem”*, University of Pretoria, 31 March 2019.
3. *2K I²C Serial EEPROM,* Microchip Technology Inc., “24AA02/ 24LC02B/24FC02”, Jan. 2007, Revised Nov. 2018
4. *This was once revealed to me in a dream.*