ECE 298 – Powered Wheelchair Functionality Tests

ECE 298 – W2021

Jakub Mroczek

Part 1 – Pin Mapping

Table 1: MCU pin use

MCU Pin	Pin Mode	Functional Description
PA5	TIM2_CH1	Outputs a PWM signal used to time motor duty cycle and LED blinking
PA6	GPIO_Input	Gets Q1 value from MR (right motor) encoder
PA7	GPIO_EXTI7	Gets IDX value from MR, calling an interrupt when high to calculate RPM for MR
PA8	GPIO_Input	Gets Q2 value from MR (right motor) encoder
PA10	GPIO_Output	Sends CSB value to ADC for the joysick controlling right/left chair rotation
PA11	GPIO_Input	Takes in input from ADC for the joysick controlling right/left chair rotation, effectively getting a value proportional to the voltage at the potentiometer
PA12	GPIO_Output	Sends CLK value to ADC for the joystick controlling right/left chair rotation
PB1	GPIO_Output	Sends PWM signal to IN1 of MR, controlling the forward (clockwise) rotation speed
PB2	GPIO_Output	Sends PWM signal to IN2 of MR, controlling the backward (counter clockwise) rotation speed
PB5	GPIO_Output	Sends CSB value to ADC for the battery level measurement
PB6	GPIO_Input	Takes in input from ADC for the battery level, effectively getting a value proportional to the voltage at the potentiometer
PB7	GPIO_Output	Sends CLK value to ADC for the battery level measurement
PB12	GPIO_Output	Sends CSB value to ADC for the joystick controlling forward/backward wheel movement
PB13	GPIO_Input	Takes in input from ADC for the joystick controlling forward/backward wheel movement, effectively getting a value proportional to the voltage at the potentiometer
PB14	GPIO_Output	Sends CLK value to ADC for the joystick controlling forward/backward wheel movement
PB15	GPIO_Input	Takes in whether the on/off switch is set or not
PC1	GPIO_Input	Gets Q1 value from ML (left motor) encoder
PC2	GPIO_EXTI2	Gets IDX value from ML, calling an interrupt when high to calculate RPM for ML
PC3	GPIO_Input	Gets Q2 value from ML (left motor) encoder
PC4	GPIO_Output	Sends PWM signal to IN1 of ML, controlling the forward (clockwise) rotation speed
PC5	GPIO_Output	Sends PWM signal to IN2 of ML, controlling the backward (counter clockwise) rotation speed
PC6	USART6_TX	Tranmits character data to LCD screen asynchronous serial communication
PC7	USART6_RX	Receives data back from LCD screen for asynchronous serial communication
PC9	GPIO_Output	Controls red LED state

PC10	GPIO_Output	Controls orange LED state
PC11	GPIO_Output	Controls yellow LED state
PC12	GPIO_Output	Controls green LED state

Part 2 – MCU Resources

Table 2: MCU resource use

MCU Resource	Functional Description
USART6	Communicates with the LCD screen module
TIM2	Generate PWM signal for DC motor driver FETs, as well as signal used for red LED blinking
GPIO	Communicates will all device pins (except LCD)
NVIC	Manages and controls interrupts and exceptions when triggered, including prioritizing the interrupts

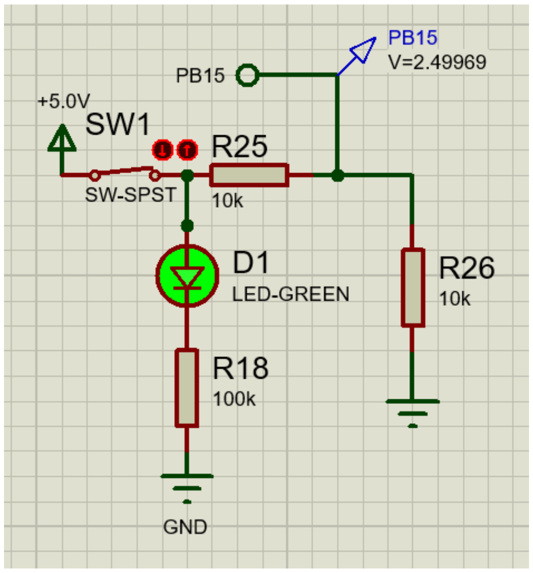
Part 3 – Schematics and Test Cases

Test 1 – Forward/Backward Run Mode

In this test, we will determine if the prototype can be controlled to move forward and backward during run mode. We will also determine if the motors encoder can properly transmit its data to the MCU, allowing the RPM to read on an LCD display.

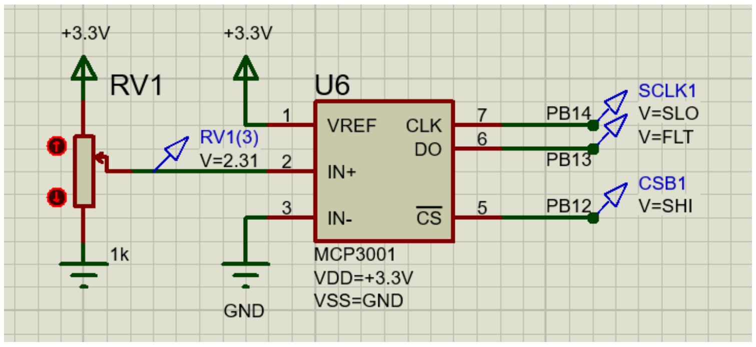
Mode Switch

During this test the mode will be set to run. This means the ON/OFF switch will be pushed down, or ON. Additionally, a green LED should be on when system is in run mode. The following screenshot of the simulation demonstrates the green LED lighting up when in run mode:



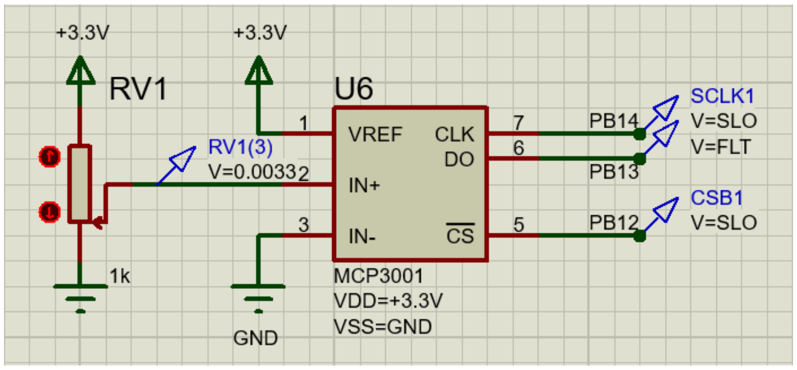
Joystick Forward/Backward

For this test the joystick component controlling forward/backward movement will be first set in its partially forward state. On the potentiometer, it will look like the following:



Where the joystick is pressed forward, but not all the way. This will let us see if the motor can operate at a RPM lower than its maximum, which in turn will tell us the duty cycle of the PWM signal has been set correctly and not constantly high. We do not have to worry about the voltage range coming into the MCU pin, as it is limited by the potentiometer and ADC to be between 0V to 3.3V, which are safe values. Additionally, a value over 1.65V means we want forward movement, and in this case we have a voltage of 2.31V.

We will then press the joystick backward fully, letting us know if the motors can operate backwards and if they can reach a higher RPM magnitude. The circuit will look like the following:



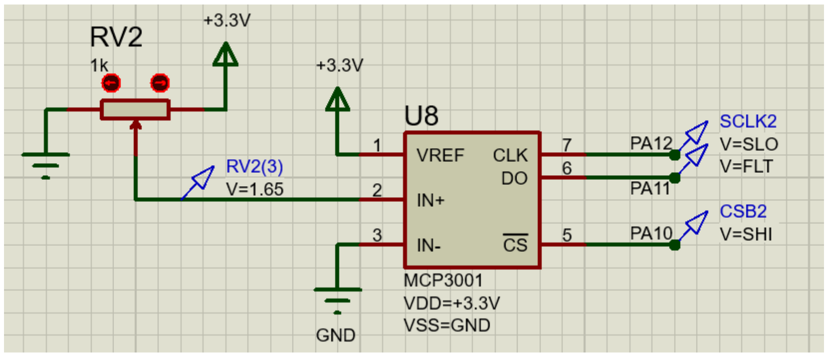
The approximately 0V coming into the ADC tells us this is the furthest the joystick can be pushed back.

In addition to these two states, we will then also set the potentiometer to its middle state, with a voltage of 1.65V being inputted into the MCU. This will test if the motors can be stopped to achieve braking.

Joystick Right/Left

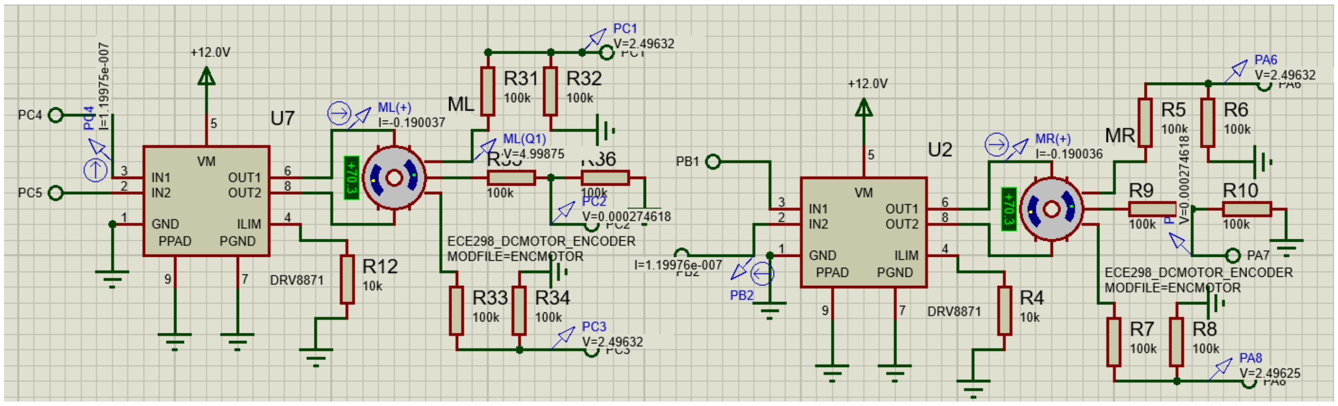
In this case, the right/left component of the joystick will be let to rest in the middle. Please note the joystick we have chosen as our input device is a 4 way joystick. This means the stick can be only positioned in one of four ways, so turning and moving forward/backward has to be two separate motions. In the future we would like to add software support for allowing the user to have both wheels move while turning, giving them the ability to turn as they move forward/backward.

The following is the setting used during this test for the potentiometer controlling left/right turning:



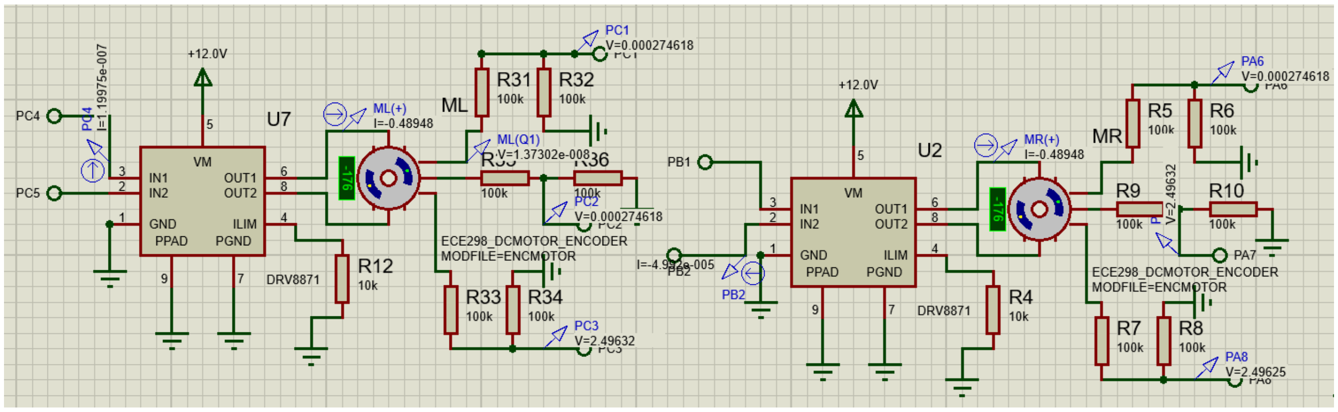
DC Motors

For the first stage of this test the joystick is set to enable partial forward movement. We are expecting both wheels to have a RPM less than half of maximum (where maximum is about 170 RPM as we will see later). This is simply because the potentiometer controlling forward/backward movement is set to less than half of its maximum forward state (precisely, it's set to two fifths of max). The following is a screenshot of the simulation running after a few seconds of ramp up to achieve its steady state:



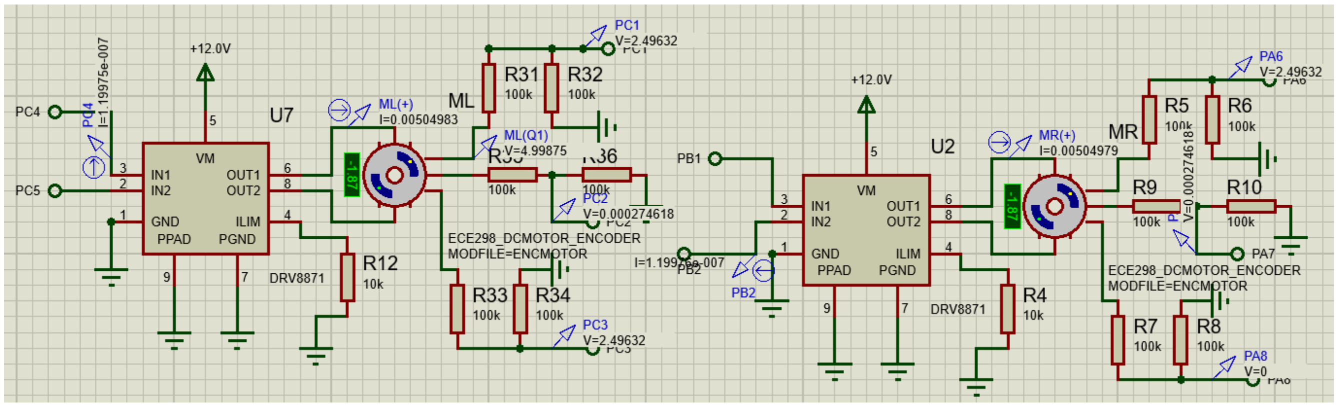
As we can see, both motors are operating at an RPM of approximately +70. This is less than half of the maximum as predicted, telling us that the duty cycle of the PWM signal controlling the motors is set correctly. If there was no PWN signal (and the motor was simply given 'high' into PC4 and PB1) the RPM would be close to maximum. The other states of the forward/backward joystick were also checked and resulted in a RPM proportional to how far they were set. As we can also see, the current flowing into the MCU is very low (essentially 0) so it meets the MCU's input current specifications.

Next, we are going to check how the motors respond to the joystick being set fully backwards. We should see a RPM with a higher magnitude than before, and in the negative direction. The following is a screenshot a few seconds after adjusting the joystick to be in the fully backward state:



As we can see, the RPM is -176. This previous value of +70 has 0. 398 times the magnitude, or two fifths. This tells us the duty cycle of the PWM has been set correctly depending on how far the joystick pushed. It also tells us that we are able to achieve backwards motion, establishing that we have accurate control of a full range of motion from about 175 RPM to -175 RPM. Again, there is negligible going into the MCU.

To show that we can brake and achieve a RPM of 0, we will set the joystick to its middle state and observe the effects. The following screenshot was taken just a couple seconds after swtiching the potentiometer from maxium backward to its middle state of 'stop':

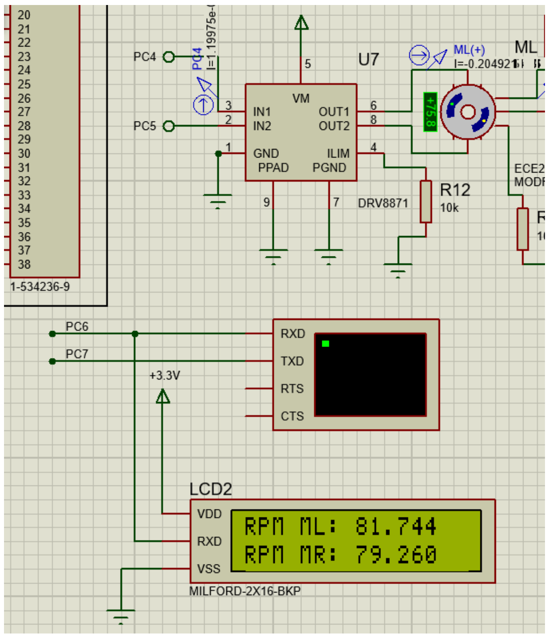


The motor RPM is now approximately 0, indicating the chair has stopped. To demonstate the ramp down, we can meausre the time it takes to reach 0 RPM after adjusting the joystick. In our case, it took a little over two seconds to achieve this, meeting the criteria of ramping down the motors while also doing it quick enough to keep the user safe.

LCD and Encoder

To show both the LCD and encoder worked during our simulations we will compare the RPM we get from the motors built in counter to the values obtained from the LCD. If they are similar this will show the IDX and Q1/Q2 pulses are measured correctly, and the resultant RPM was calculated correctly as well.

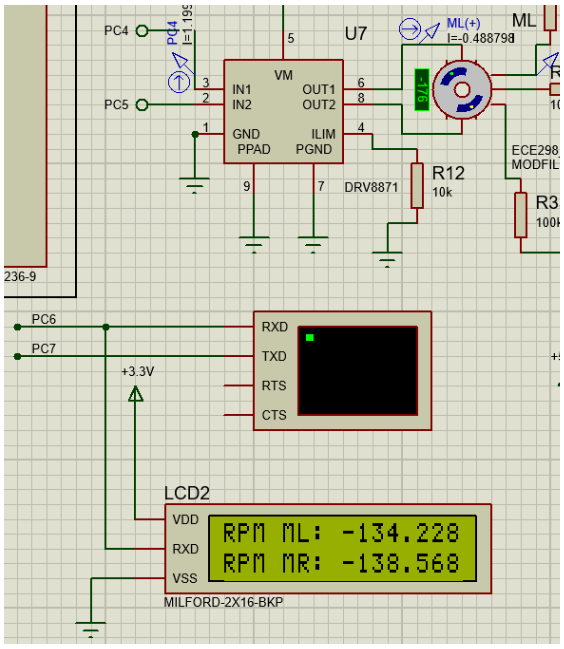
First, we have the case when the joystick was partially forward. The RPM read from the motors built in counter and the RPM calcuted are shown below:



```
RPM ML: 81.744
RPM MR: 79.260
```

The motors read a value of about +76 RPM, while we calcuated a value of about +80 RPM. This is approximately 5% error, which is more than appropriate for our case. While this a good result, our calculated values are not always this close to the actual. Due to the latency of when our calculations are posted and the motors variable RPM at a sigle state, the calculated RPM does not always match with the one the motor shows. The variable motor RPM at a sigle state (usually ± 20 RPM) also negatively affects us as less consistent IDX pulses means less consistent calcuations. Additionally, one IDX signal has an interupt on a rising edge while the other has an interupt on a falling edge. We did this to try and make sure these two interupts did not happen at the same time, but it also makes one motors reading lag behind the other.

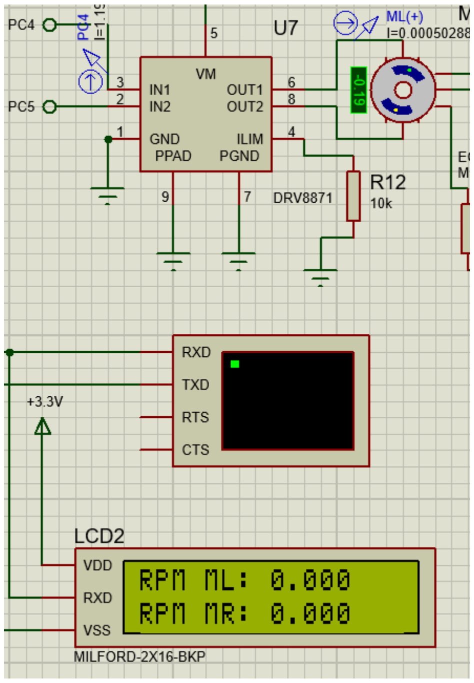
The following is a screenshot of the RPM values for a fully backward joystick:



```
RPM ML: -134.228
RPM MR: -138.568
```

We correctly display a negative RPM, which meets the projects criteria. We also have an error of about 23% here, which is worse than before but still fairly good for our purposes.

Finally, the RPM when the joystick is set to its middle 'stop' state:



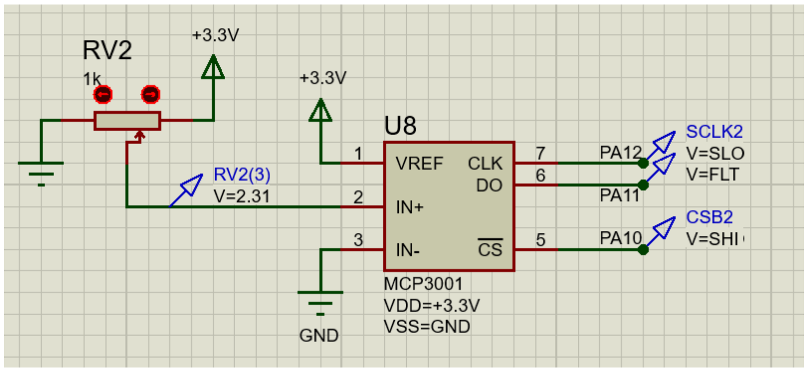
The accurate reading of 0 RPM and the previous readings demonstrate through testing the accuracy and robustness of our LCD and encoder, meeting the LCD criteria in the process.

Test 2 – Turn Left/Right Mode Run

This test will only look at what happens if the joystick is pushed in the left or right direction. We will still simulate the rest of the components, but their measurements will not be included as it is redundant with Test 1.

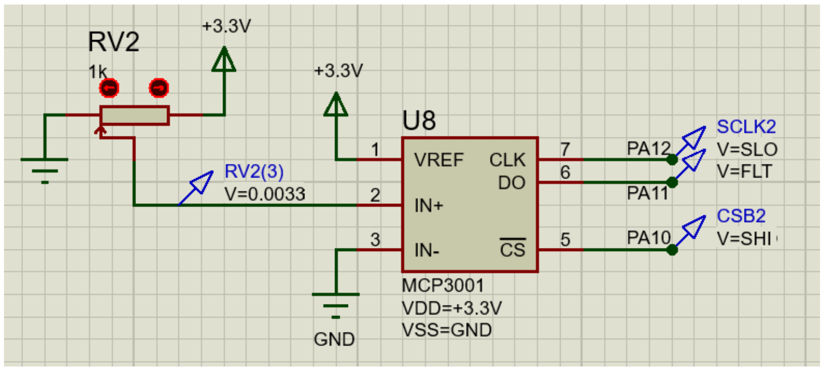
Joystick Right/Left

For this test the joystick component controlling right/left movement will be first set in its partially right state. On the potentiometer, it will look like the following:



Where the joystick is pressed right, but not all the way. As before this will let us see if the motor can operate at a RPM lower than its maximum, which in turn will tell us the duty cycle of the PWM signal has been set correctly and not constantly high. We do not have to worry about the voltage range coming into the MCU pin, as it is limited by the potentiometer and ADC to be between 0V to 3.3V, which are safe values. Additionally, a value over 1.65V means we want to turn right, and in this case we have a voltage of 2.31V.

We will then press the joystick left fully, letting us know if the chair can turn left and if the wheels can reach a higher RPM. The circuit will look like the following:



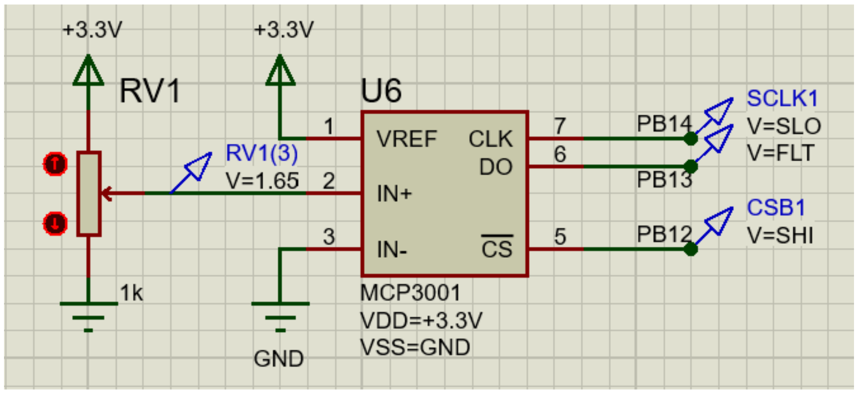
The approximately 0V coming into the ADC tells us this is the furthest the joystick can be pushed left.

In addition to these two states, we will then also set the potentiometer to its middle state, with a voltage of 1.65V being inputted into the MCU. This will test if the motors can be stopped to achieve braking.

Joystick Forward/Back

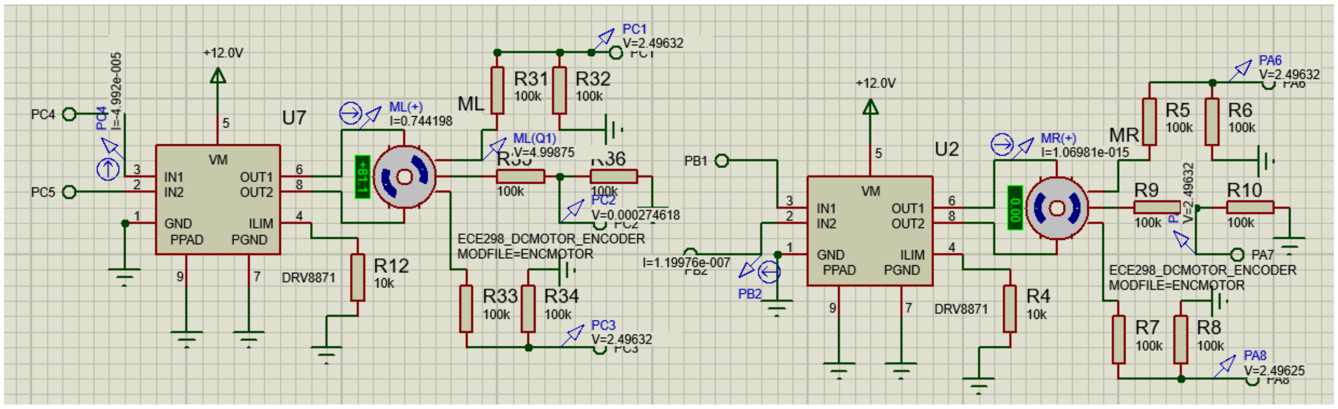
In this case, the forward/back component of the joystick will be let to rest in the middle. Please note the joystick we have chosen as our input device is a 4-way joystick.

The following is the setting used during this test for the potentiometer controlling forward/backward movement:



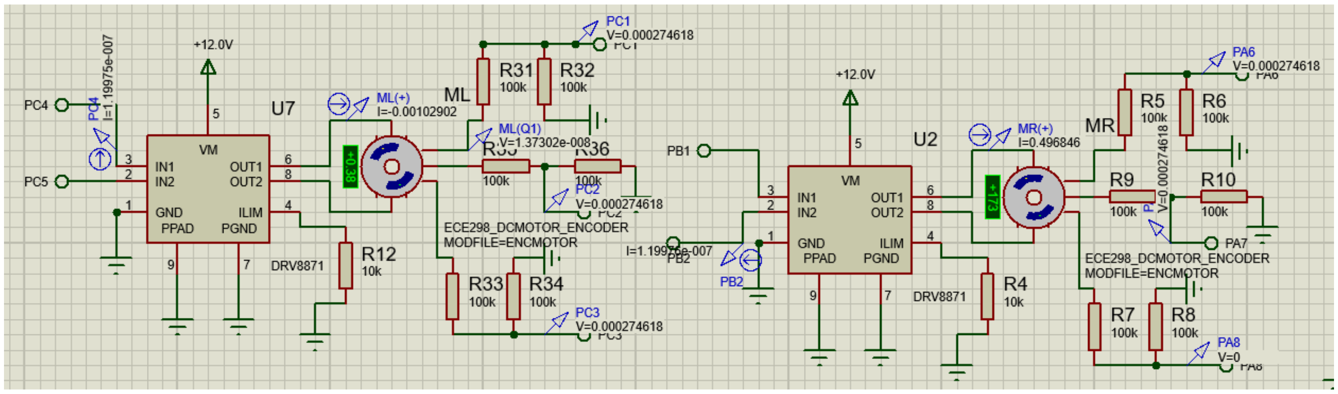
DC Motors

For the first stage of this test the joystick is set to enable slow right turning. We are expecting on the left wheel to turn, at a value of approximately 40% of RPM max (RPM max is about 170). The following is a screenshot of the simulation running after a few seconds of ramp up to achieve its steady state:



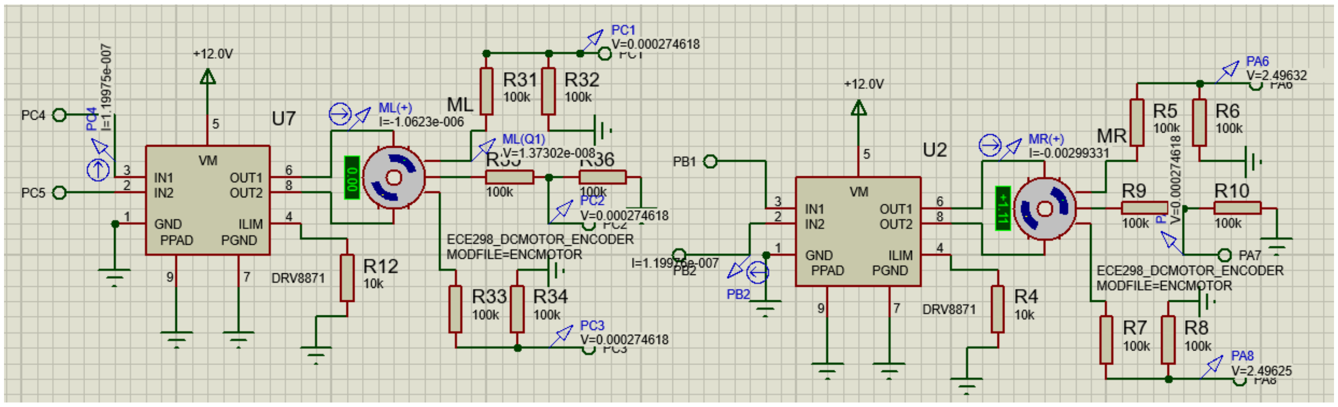
As we can see, only the left wheel is turning at +81 RPM, which proportional to the value set on the left/right potentiometer. It has about a 20% error compared to the value we were expecting, but like it was explained earlier this kind of variance is expected due to the motor RPM fluctuations. This is a slow-medium right turn.

Next, we are going to check how the motors respond to the joystick being set fully left. We should see a RPM of about +175 only in the right wheel. The following is a screenshot a few seconds after adjusting the joystick to be in the fully left state:



As we can see, the right motor RPM is +173 and the left motor RPM is about 0. This is a fast left turn, the fastest our wheelchair supports.

To show that we can brake and achieve a RPM of 0, we will set the joystick to its middle state (voltage into ADC = 1.65V) and observe the effects. The following screenshot was taken just a couple seconds after swtiching the potentiometer from maxium left to its middle state of 'stop':



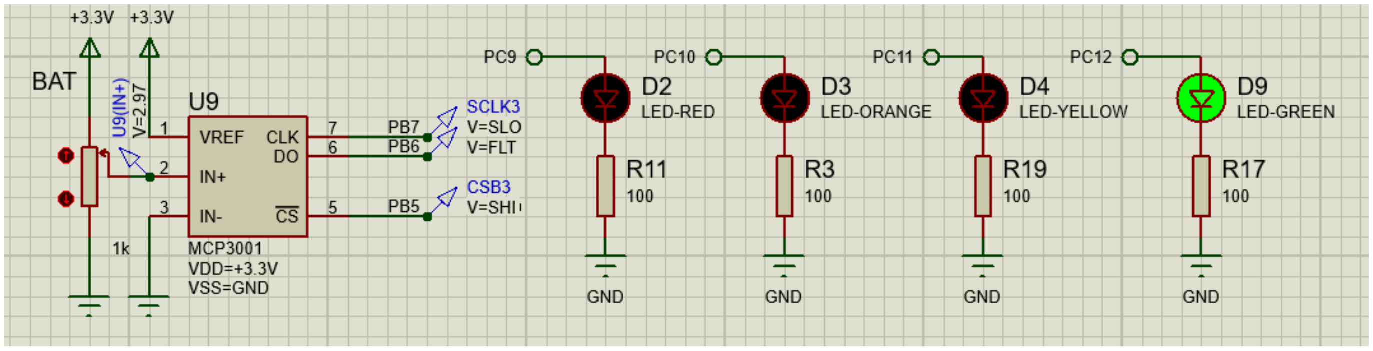
Both motors have an RPM of about 0, indicating the chair has stopped. From here the joystick can be moved in any direction to aheive forward/backward movement or another right/left turn.

Test 3 – Battery Level Indicator

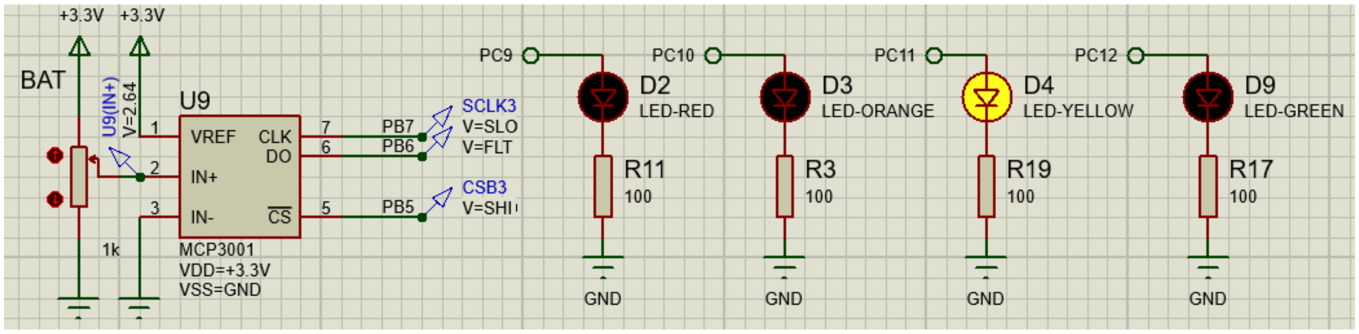
Sine we have established the other components working above, this test solely focuses on the battery level indicator. The indicator is made up of 4 LED's of varying colours, as well as a potentiometer to simulate falling battery levels and an ADC to convert this falling voltage into readable data to use within our program.

To test this system, we will cycle through the battery level stages and show the resultant LED outputs.

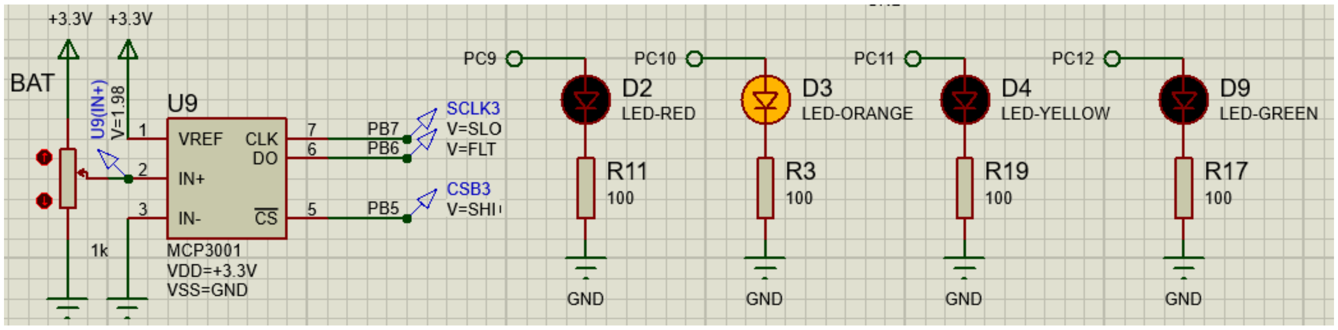
First, we tested a battery level of over 90% (potentiometer pulled down 1 notch):



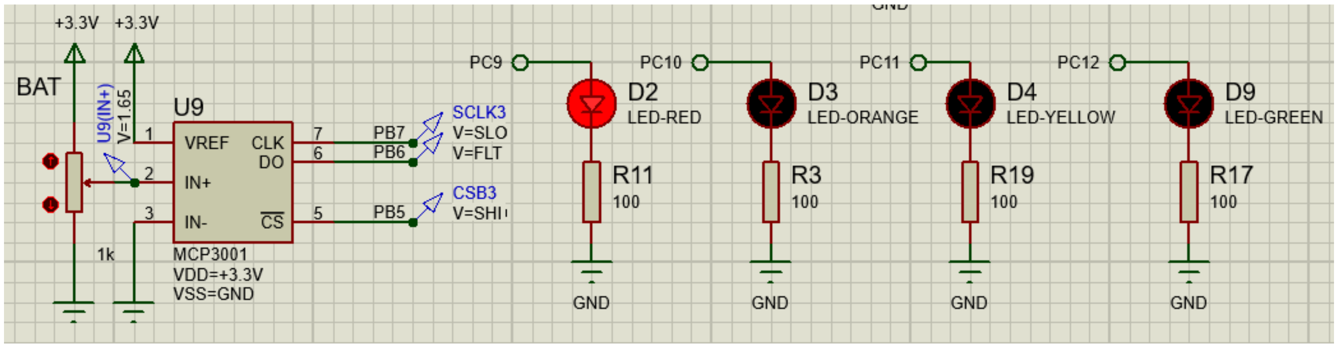
Then a battery level of between 80% to 90% (potentiometer pulled down 2 notches):

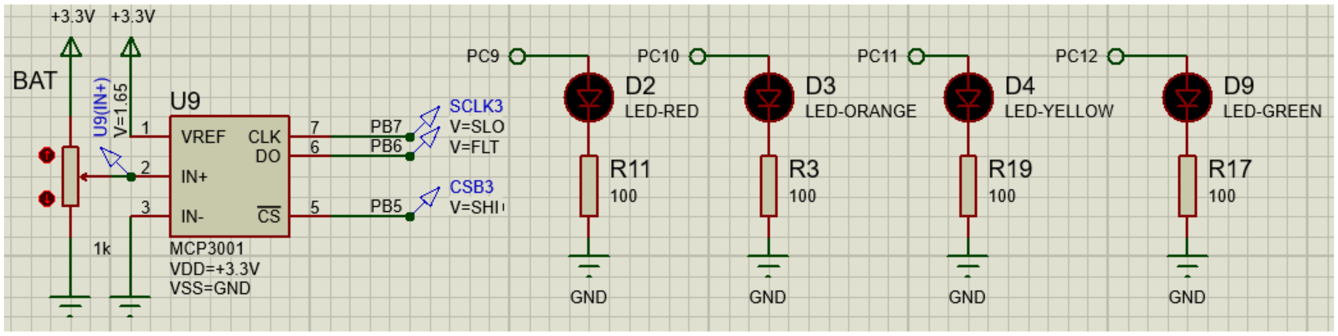


A battery level of between 60% to just under 80% (potentiometer pulled down 4 notches):



And finally, a battery level of less than 60% (potentiometer pulled down 5 notches):



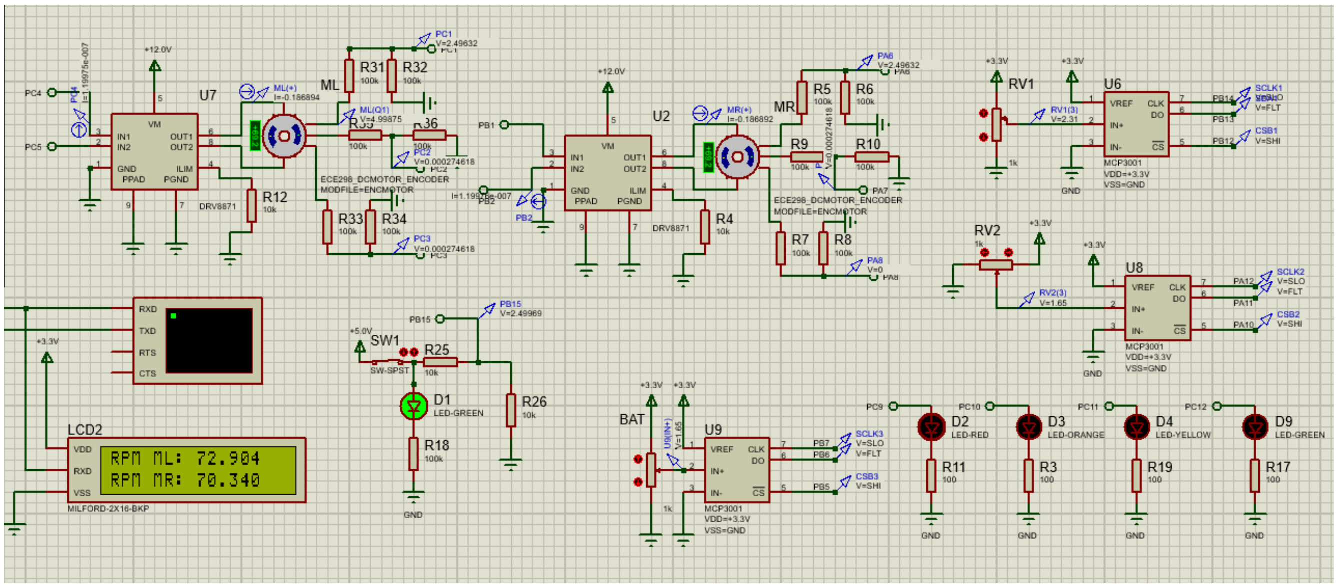


As it is shown, the LED colour indicating battery level criteria has been met. The red LED also flashes when it is on, as shown in the screenshots above.

Test 4 – Locked Mode

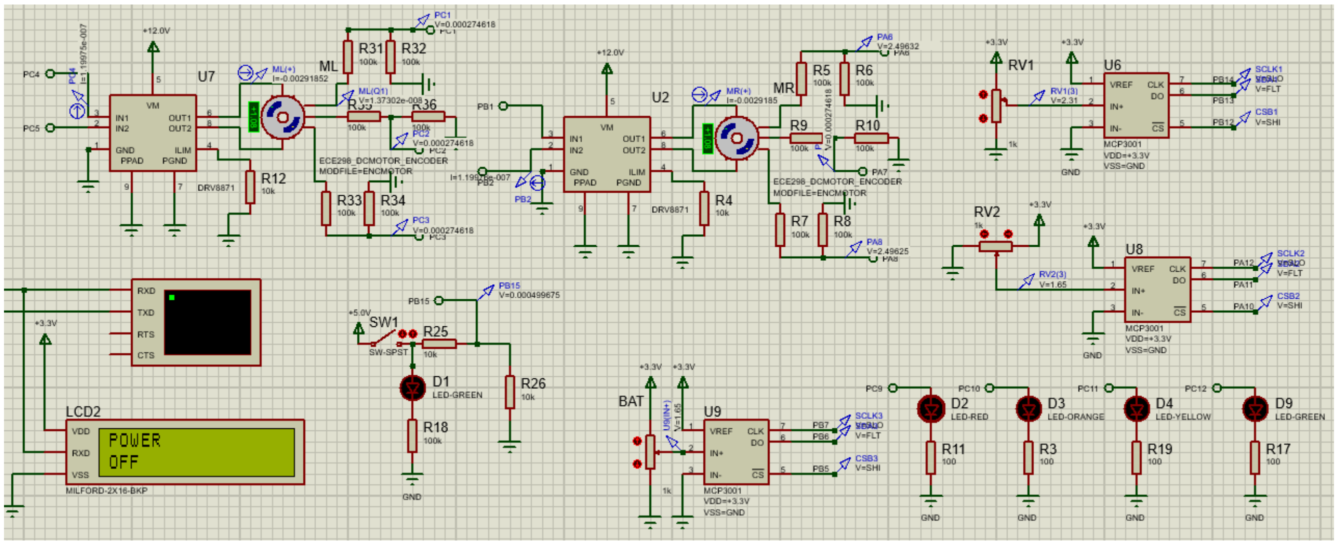
When the mode switch is open, a few things are expected to happen. The motors must turn off, the LCD must turn off, and all LED’s must turn off. To test whether this feature works or not, we will first have the system on in run mode. We will then flip the switch to locked mode and observe the results. Finally, we will press the switch back down, setting the system to on mode and observing if the system will start up again.

This system in its initial run mode looks as follows:



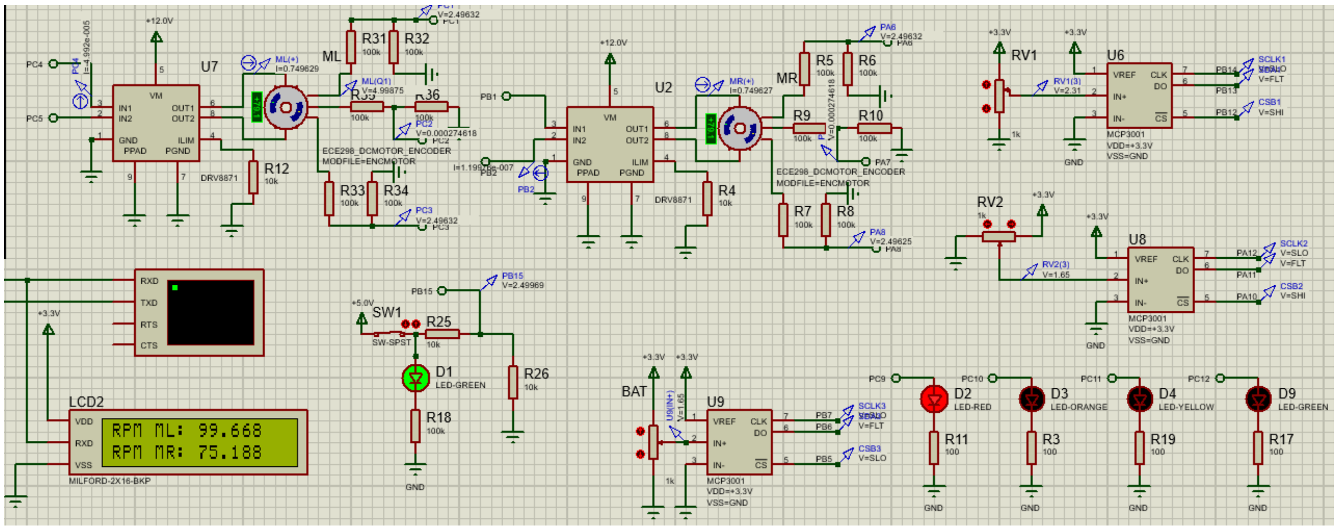
Its behaviour is as expected. Note that the battery level is less than 60% and the red LED happens to be in its off cycle.

Now the switch is set to locked:



All function has stopped as intended.

We will now set the switch back to run:



Everything is working correctly again, indicating the switch worked and it is able to cycle between run and locked.