COMPSYS301[1]

Hardware Task 2 - Driving a DC Motor

Task 2 – Motor Control

This task will require you to –

- Use a PWM and Quadrature Decoder Cypress library components.
- You will add additonal components to modify the default signal to suit the motor driver IC (MC33926)
- You will generate PWM signals to control the a DC motor's speed and direction
- You will determine the position and speed of the motor
- You will write C code to determine the motor's speed vs voltage characteristics.

The Goal

The main purpose is to determine the motor's speed vs voltage characteristics. You will determine the speed of the motor as a result of applying different voltages to the motor terminal.

- To achieve this, you will need to generate a variable voltage. This can be easily done with a PWM generator on the PSoC. However, the motor will draw more current than the PSOC can provide and hence the MC33926 is used as a high current source. Thus, you need to undertsand how the MC33926 operates and configure the PWM generator's output to suit the MC33926's requirements[2]. You will automate the experiment by automatically adjusting the PWM signal periodically.
- Secondly, The motors you will use have a shaft position sensor – a quadrature encoder[3] - integrated into the assembly. The Quadrature Decoder component interprets the sensor output and returns *a change in angular position*. It does not return an *absolute* position. Since we are only interested in speed, the absolute position of the shaft is not important. If you can determine the change in position over a known period of time, then speed can be calculated.
- The motors can be driven in 'Signed-Magnitude' or 'locked anti-phase' mode.

Shown below in Figure 1 is a copy of the schematic used in the Cypress project in "`psoc_code_base`". This project can be obtained from CANVAS but should only be used as reference. You would build your own schematic by including the elements that you think you require. The changes I imagine are -

1. Only one PWM and QuadDecoder are necessary
2. The solution is for 'locked anti-phase' mode. You may choose to try out the 'Signed-Magnitude'
3. A timer with an interrupt to calculate frequency

[1] Version 1.0
[2] Table 1 and Table 5
[3] See lecture slide 28

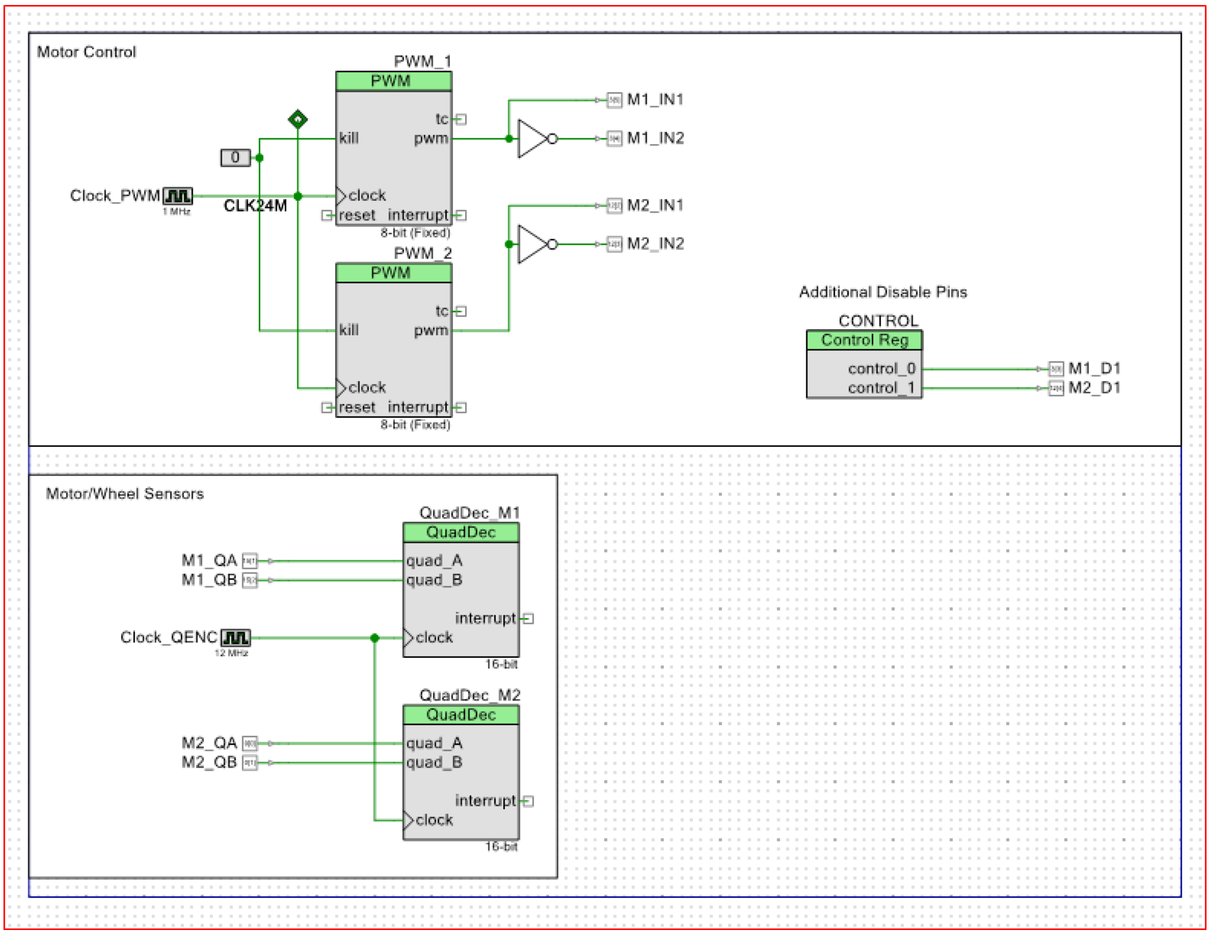

Figure 1 – Actual Schematic used for the robot

Step 0

Familairize your self with the gear: the robot schematic, the motor + sensor specifications, the relevant Cypress libary components and the MC33926.

Step 1 - PWM

In a blank schematic, build a PWM generator with supporting C code (e.g. `Start()`). Assign a suitable pin and probe the output using a scope. The signal will have a measureable duty cycle and a frequency.

1. The duty cycle can be easily changed in 'C' using `WriteCompare()`. Vary the duty cycle and observe the impact.
2. The frequency of the PWM is related to the clock driving the PWM block and the `Period` value. The `Period` value should be based on the desired resolution[4] of the output voltage. If you need to an adjustable frequency, you need to vary the frequency of the clock that drives the PWM block. This is just a little bit more involved but is not important at this stage. Nevertheless, it is something to keep this

[4] The smallest change you to see in the average voltage. If you are uncertain, use 100. This gives you a resolution of 1%.

at the back of your mind. Typically one decides the operating frequency and you do not have to change it in a stable design.

3. Measure the freqency and duty cycle and compare with expectations.
4. Modify the signal as required by the MC33926[5]. Verify the signals on the scope.

Step 3 – Driving the motor

You can now drive one of the motors on the robot[6]. The motor is assembled into the robot and its structure may not be clearly notable. A typical motor is shown below Figure 2.

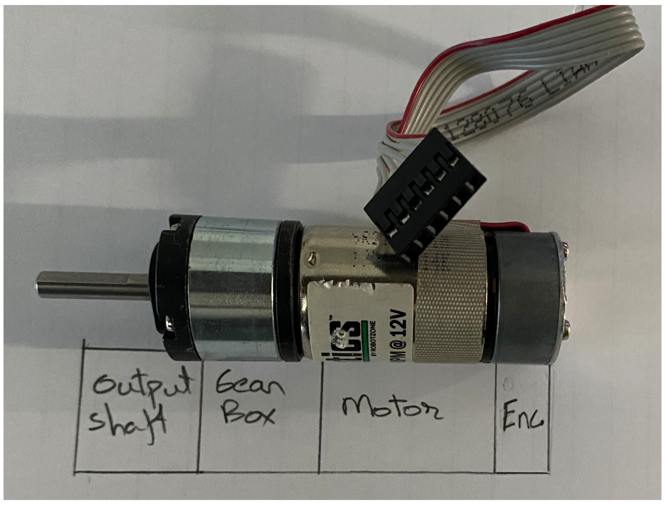

Figure 2 - DC Motor

One end of the motor shaft is connected to 19:1 gear box and the other end to the quadrature encoder. Thus the encoder senses (changes to) **motor** shaft position and not the **output** shaft position. See the motor datasheet on CANVAS.

Note : one rotation of the output shaft will require 19 rotations of the motor shaft. The encoder produce 3 pulses (on channel A and B) per rotation and hence one rotation of the output shaft will produce 19*3 pulses.

The interface PCB, Figure 3, on the robot should house the PSoC (into J3/J4). Ensure the orientation is correct. Note -

- J18/J13 exposes the connections to the MC33926 that is placed on the daughter board (not visible). The connections to the motor will be internally routed through J2/J1.
- J18/J13 also exposes the motor's quadrature encoder outputs.

1. Without making any connections to J18/J13 ensure that your Cypress schematic - PWM + mods is producing signals suitable for the IN1/IN2 pins of the MC33926. The duty cycle should be 50%. Once you have verified this, power down the robot

[5] Note that the MC33926 also requires additional En/Dis signals. For this exercise, these can be hardwired to VCC or GND in accordance to the datasheet. In a practical application, you could use these to power down the PWM power chip when the robot is 'sleep' mode.
[6] If a Robot is not at your station, ask the TA for one. We also provide a platform on to which the robot should be placed. The platform will raise the wheels of the robot for obvious reasons.

2. Complete the connections to J18/J13. Recall that the En/D1/D2 pins are enable/disable pins and can be hard-wired to VCC or GND as required.
3. Perhaps you should consider getting the TA to verify your connections before you power up the robot.
4. Power up the robot. The motor should be spinning. If its not, check the En/Dis signals. If other debugging steps fail call on the TA for assistance
5. With the motor running, probe the ENCD pins on J18/J13. You should see two square waves with a phase difference of 90 degrees.

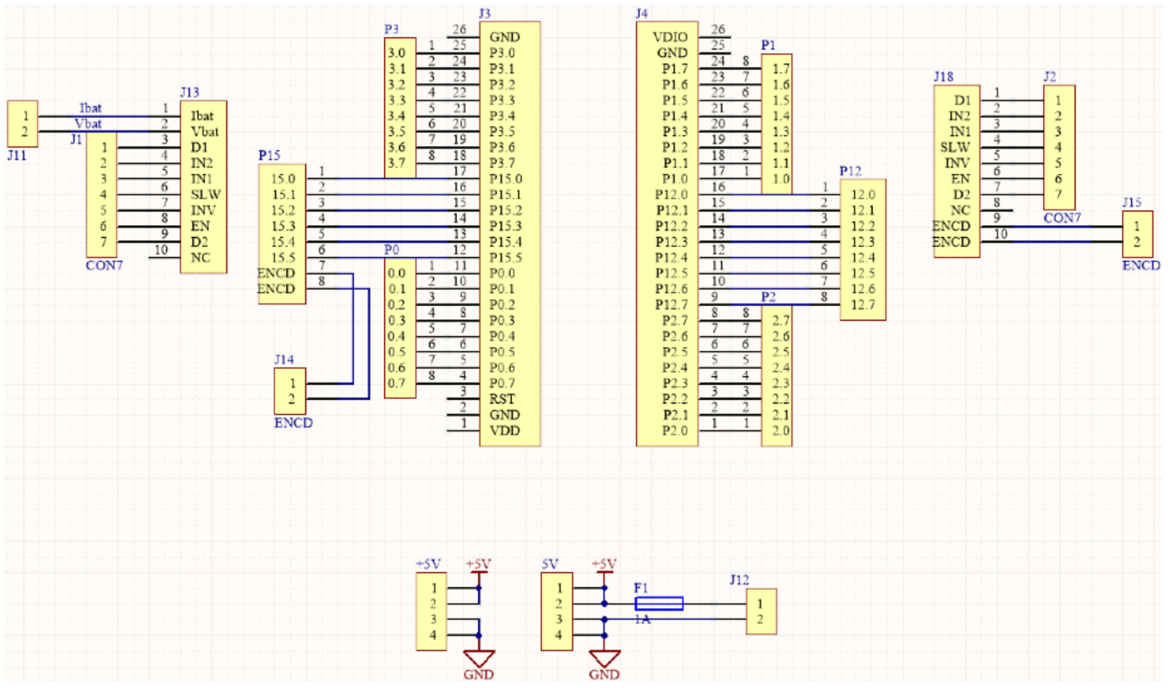

Figure 3 Interface PCB

The frequency of the square waves will need to be measured to establish the speed of the output shaft. This will be obtained in Step 4 below.

Step 4 – Rotor position and speed sensing

The QuadDecoder block measures a change in rotor position by counting transitions on Channels A/B. It maintains a running sum of the transitions. The current count can be obtained from `GetCounter()`. You can reset the count to zero or any value using `SetCounter()`.

- If $c(i)$and $c(j)$ are the count values at time i and j, the the shaft speed is given by $s = \frac{c(j)-c(i)}{j-i}$ counts/sec. This can be converted in to rads/sec or RPM
- You will need to know 'when' you execute `GetCounter()`. It is best to arrange for a timer to raise an interrupt at regular intervals and hence $(j - i)$ is a predefined constant. Hence speed can be calculated in the Timer ISR
- A higher dynamic range will require a longer integrating time but if you require a faster speed update you will have a lower resolution.

- The shaft position or speed is a binary value and hence you will need to use a USBUART and putty[7].

1. Determine the maximum frequency of the quadrature encoder signals from the motor. You can do this experimentally by applying a duty cycle of 100%. Compare the max speed with the motor datasheet.
2. In your Cypress Schematic add the QuadDecoder, a Timer and the USBUART. Include the USBUART functionality (C code) from previous projects. Check that the USBUART and PUTTY are working OK
3. Determine the required timer value. The binary resolution of your calculated speed will establish the minimum Timer rate[8]. The measure of maximum frequency measured above will give you the maximum calculated speed. Write the timer ISR to calculate the speed. If you are uncertain, try 1sec. Check that the timer is working using using a scope (the exact value of time is important)
4. The ENCD signals can be applied directly to the Cypress QuadDecoder library block. From the component's Configuration panel, choose a resolution of 16bits (or 32bits), 4x mode, do not use the Index mode and enable Glitch Filtering. Add two digital input pins (ENCD to PSoC). Add additonal code to send position and speed values to PUTTY.
5. Vary the PWM duty cycle and note the impact on scope and PUTTY measurements.

Step 5 – Ploting the Speed vs voltage profile

Finally we require you to determine the motor's speed as the applied voltage changes from -V to +V in arbitary increments (say 0.5V). You can assume that the battery voltage is constant. It will be good to measure the battery voltage under these conditions: motor not running, motor running at full speed and finally voltage after the sweep has been completed.

1. Enhance your C code to perform this sweep automatically. You must send appropriate values via PUTTY so that you can plot the speed vs voltage profile in Matlab.
2. Comment on the charateristics: the non-linearities and symmtery for positive and negative voltages
3. If time permits, obtain the same charateristics for the second motor

Questions

1. Why is PWM used to drive some high power loads (motors, heaters) and not others – like audiophillic loudspeakers[9]?
2. What factor affects the maximum frequency of the PWM signal in this exercise?

[7] Or perhaps a DAC with a scope! The numeric representation via PUTTY is required.
[8] The sensor update rate is very closely related to the sampling rate of the control loop. Usually they should equal.
[9] Look up Class D amplifiers. A purist will usully balk at the suggestion of using Class D for audio!

3. What are the advantages and disadvantages of the Signed-Magnitude and the Locked Antiphase method?
4. Can the trade-off between speed update rate and resolution be oversome?

Conclusions

On completion, you will have -

- Driven a brushed DC motor using software
- Measure the speed of the motor
- Detemine the speed vs voltage charateristics

- … and Matlab is still pretty cool