Lab 5

NOTICE: The labels on Port E are wrong. The pin labelled PE2 is actually PE4.

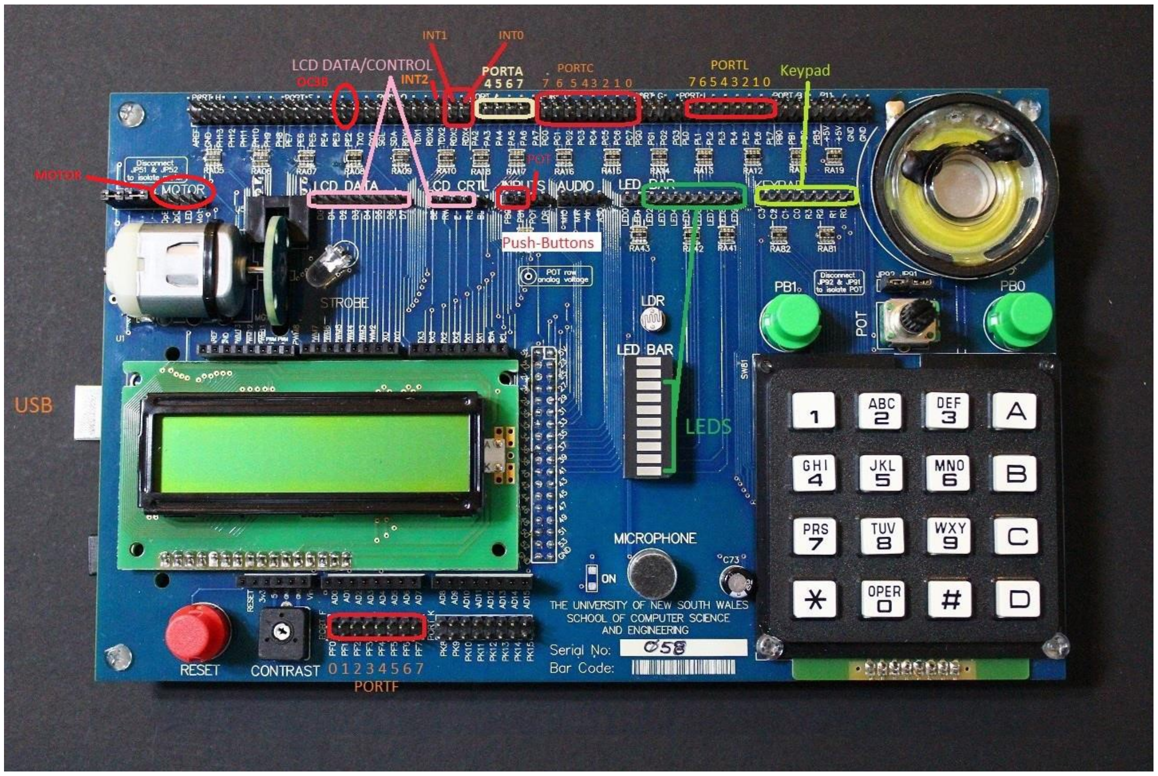
1. Objective

In this lab, you will learn how to:

- Use emitter-detector to measure the speed of the motor.
- Control the speed of a DC motor by using Pulse Width Modulation (PWM).

2. Tasks

There are **two** compulsory tasks. It may take several hours to read the relevant material before you can start. So you need to finish reading all the relevant material and work out most of the code at home. The following board picture helps you find the relevant pins for this lab.



3. Task A: Motor Speed Detection (5 marks)

The Figure 1 shows the circuit for motor speed detection. The emitter is enabled when OpE is high. To enable the emitter, you need to connect OpE to a port pin and configure this pin as output. Notice that there are four holes on the wheel. When the infrared light from the emitter goes through a hole, the detector pin (OpO) will be low. Otherwise, OpO will be high. To read OpO, you need to connect it to a port pin and configure this pin as input.

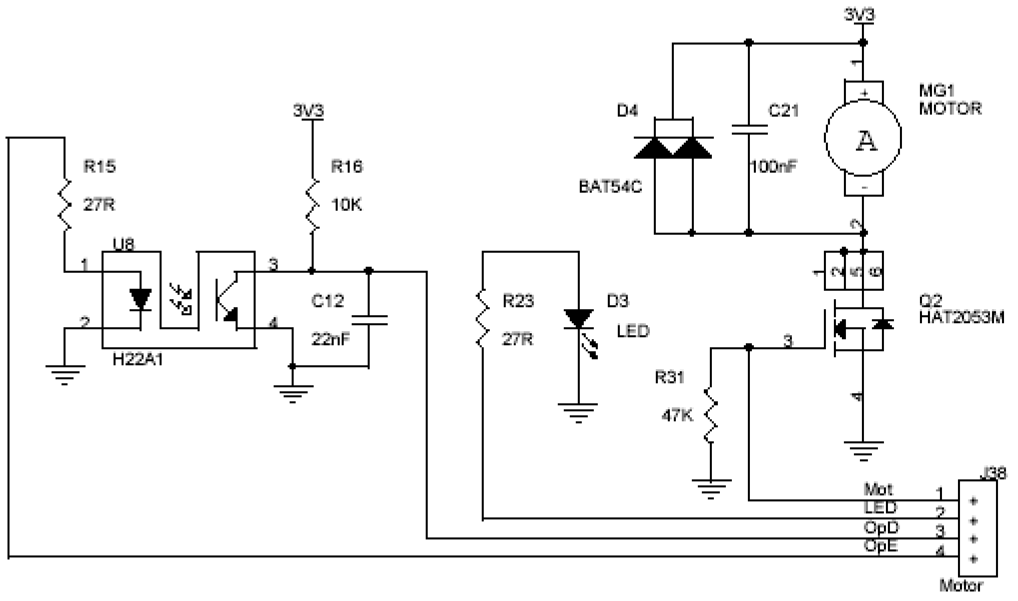


Figure 1: The schematic of the DC motor and speed detection circuit.

Now wire up the POT to the MOT using the patch cables provided, and power up the AVR microcontroller board. As you turn the POT, the current flowing to the motor will change. As a result, the motor spins at different speeds. To calculate the motor speed, you need to find the number of state changes of OpO within a fixed amount of time.

Hints for Motor Speed Detection

To detect the motor speed, you need to count the number of revolutions the motor spins in one second.

The detector can be read by using an external interrupt. The detector will output a 1 when there is no hole, so you can use the falling edge of the detector to trigger an interrupt to count a hole.

Board set-up: Connect the Mot pin to the potentiometer (labelled POT, next to PB1) and use it to control the motor's speed. Connect OpE to one of the +5V pins to enable the emitter, and the OpO pin to the external interrupt INT2 (labelled TDX2) for counting holes.

Write a program to calculate the motor's speed in revolutions per second and display it on the LCD. You should update the display at every 100ms.

Assemble your program using AVR Studio, download the program to the AVR microcontroller board and show your working program to a lab tutor.

Comments: You need to refer to the previous labs for the timer interrupt, external interrupt and LCD.

4. Task B: Motor Speed Control (5 marks)

Analog signals

An analog signal has a continuously varying value, with infinite resolution in both time and magnitude. A nine-volt battery is an example of an analog device, in that its output voltage is not precisely 9V, changes over time, and can take any real-numbered value. Similarly, the amount of current drawn from a battery is not limited to a finite set of possible values. Analog signals are distinguishable from digital signals because the latter always take values only from a finite set of predetermined possibilities, such as the set 0V, 5V.

Analog voltages and currents can be used to control things directly, like the motor. In a simple DC motor, a POT is connected to a variable resistor. As you turn the POT, the resistance goes up or down. As that happens, the current flowing through the resistor increases or decreases. This changes the amount of current driving the motor, thus increasing or decreasing the speed. An analog circuit is one, like the motor, whose output is linearly proportional to its input. As intuitive and simple as analog control may seem, it is not always economically attractive or otherwise practical. For one thing, analog circuits tend to drift over time and can, therefore, be very difficult to tune. Precision analog circuits, which solve that problem, can be very large, heavy (just think of older home stereo equipment), and expensive. Analog circuits can also get very hot; the power dissipated is proportional to the voltage across the active elements multiplied by the current through them. Analog circuitry can also be sensitive to noise. Because of its infinite resolution, any perturbation or noise on an analog signal necessarily changes the current value.

Digital signals

By controlling analog circuits digitally, system costs and power consumption can be drastically reduced. What's more, many microcontrollers and already include on-chip PWM controllers, making implementation easy.

PWM is a way of digitally encoding analog signal levels. Through the use of high-resolution counters, the duty cycle of a square wave is modulated to encode a specific analog signal level. The PWM signal is still digital signal because, at any given instant of time, the full DC supply is either fully on or fully off. The voltage or current source is supplied to the analog load by means of a repeating series of on and off pulses. The on time is the time during which the DC power is supplied and the off time is the period during which that the power is not supplied. Given a sufficient bandwidth, any analog value can be encoded with PWM.

Figure 2 shows three different PWM signals with duty cycles of 10%, 50% and 90%, respectively. These three PWM outputs encode three different analog signal values, at 10%, 50%, and 90% of the full strength. If, for example, the supply is 5V and the duty cycle is 10%, a 0.5V analog signal results.

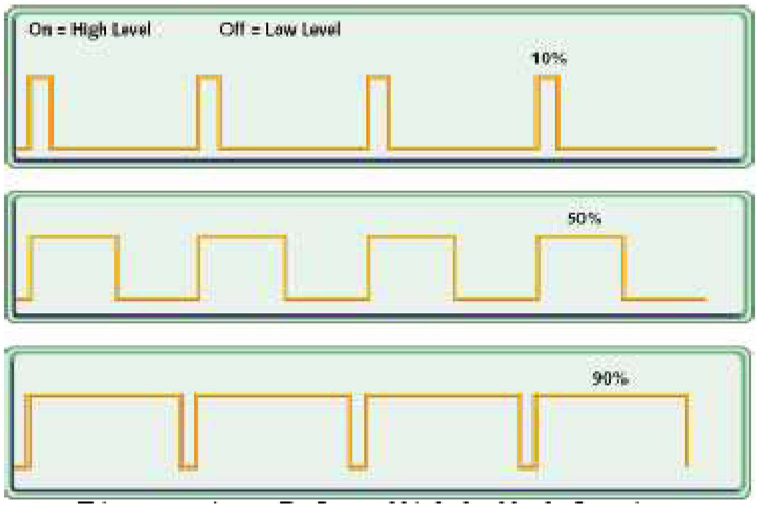


Figure 2: Pulse Width Modulation

PWM is a common technique for speed control. A good analogy is bicycle riding. You peddle (exert energy) and then coast (relax) using your momentum to carry you forward. As you slow down (due to wind resistance, friction, road shape) you peddle to speed up and then coast again. The duty cycle is the ratio of peddling time to the total time (peddle+coast time). A 100% duty cycle means you are peddling all the time, and 50% only half the time.

PWM for motor speed control works in a very similar way. Instead of peddling, your motor is given a fixed voltage value (say +5 V) and starts spinning. The voltage is then removed and the motor "coasts". By continuing this voltage on-off duty cycle, motor speed is controlled. To control the speed of a DC motor, we need a variable voltage DC power source. However, if you take a 5V motor and switch on the power, the motor will start to speed up, motors do not respond immediately so it will take a small time to reach full speed. If we switch the power off, the motor will start to slow down. If we switch the power on the off quickly enough, the motor will run at some speed part way between zero and full speed. This is exactly what a PWM controller does; it switches the motor on in series of pulses. To control the motor speed it varies the width of the pulses – hence Pulse Width Modulation.

PWM waves can be generated on the ATmega2560 through a timer. For this task, you need to use OC3B of Timer 3 to control the motor's speed. Before continuing, you need to read 17. 16-bit Timer/Counter (Timer/Counter 1, 3, 4, and 5), ATmega2560 Data Sheet, pages 136-168. **Understand how to set up Timer 3 so that the PWM wave on OC3B has a predefined duty cycle**. An example about using Timer 5 will be shown.

Board set-up: Connect Mot to OC3B (labelled PE2) so that the PWM wave output on OC3B (Timer 3) can control the motor's speed.

Write an assembly program to control the motor's speed. You program should work as follows:

1. Initially, the motor's speed is 0 rps (revolutions per second).
2. When the push-button PB0 is pressed, the motor's speed is increased by 20 rps. The maximum speed is 100 rps.
3. When the push-button PB1 is pressed, the motor's speed is decreased by 20 rps.

Assemble your program using AVR Studio, download the program to the AVR microcontroller board and show your working program to a lab tutor.

Notice: On some boards, a PWM signal to the motor causes the board to reset, or freeze. To fix this problem, connect the Mot pin to the POT pin, then remove the right-most isolation jumper above the potentiometer and connect your PWM pin (PE2) to the rightmost jumper pin. Turning the potentiometer will then introduce a resistance in series with the motor, and it should be possible to find a position where the motor spins but does not crash the board. You may not be able to reach 100 rps with this configuration. Do not lose the PE2 jumper.

Key points:
1. Refer to Table 7.2 on page 148 of ATmega2560 Data Sheet for the operations modes. You may use any one of the modes 1, 2 and 3 (phase correct mode), or any one of the modes 5, 6 and 7 (fast mode).
2. Since you are required to use OC3B to control the motor's speed, you need to understand how the value in OCR3B determines the duty cycle in a particular PWM mode.

An example: use the OC5A pin of Timer 5 as PWM output

Timer 5 set-up:
1. Set OC5A as output.
2. Choose a specific prescalar value to enable Timer 5 (refer to Table 7-6 on page 161 of ATmega2560 Data Sheet). Otherwise, Timer 5 will stop.
3. Set the Timer 5 operation mode to the 8 bit phase correct PWM mode (refer to Table 7.2 on page 148 of ATmega2560 Data Sheet).
4. Make OC5A override the normal port functionality of the I/O pin PL3 (refer 17.11.4 TCCR5A – Timer/Counter 5 Control Register A on page 158 of ATmega2560 Data Sheet).

Refer to Pin Configurations on page 2 of ATmega2560 Data Sheet for the PWM output pins. As you can see, those pins are connected to port pins. For example, OC3B is connected to the pin PE4 of Port E, and OC5A is connected to PL3. **In order to make a PWM pin as output, you need to configure the corresponding port pin as output, and override the normal port functionality of the corresponding port pin.**

The sample code is as follows:

```
.include "m2560def.inc"
.def temp = r16
ldi temp, 0b00001000
sts DDRL, temp  ; set PL3 (OC5A) as output.
ldi temp, 0x4A  ; this value and the operation mode determine the PWM duty cycle
sts OCR5AL, temp
clr temp
sts OCR5AH, temp
ldi temp, (1 << CS50)  ; CS50=1: no prescaling
sts TCCR5B, temp
```

```
ldi temp, (1<< WGM50)|(1<<COM5A1)
; WGM50=1: phase correct PWM, 8 bits
; COM5A1=1: make OC5A override the normal port functionality of the I/O pin PL3
sts TCCR5A, temp
halt: rjmp halt
```

4. Deadline

The deadline of this lab is Week 10.