COMP9032 Experiment 4

Sept, 2015

1. Objectives

In this lab, you will gain more practical experience on microcontroller interfacing. Specifically, your tasks in this lab are

- studying Dot Matrix LCD and programming LCD to display output data, and
- studying the DC motor and programming to detect and control the motor speed.

2. Preparation

Before coming to the laboratory, you should:

- read through the document available at www.cse.unsw.edu.au/~cs9032/references/Documents/LCD_Manual.pdf for general description of Dot Matrix LCD,
- read the Atmega2560 data sheet on how to use Timer 0 to generate Phase Correct PWM signals,
- read through the task description of this experiment, trying to understand what you will be doing, and
- write your programs at home in order to finish the experiment on time.

3. Introduction to the DMC LCD and the DC Motor

3.1 DMC LCD

The AVR Microcontroller Board comes with a 2 × 16 character Liquid Crystal Display (LCD) module. This module can be controlled via a ATmega2560 port.

In order to read/write from/to the LCD, here is the list of things you must perform:

- Initializing LCD
- Checking the busy flag of LCD
- Determining which register (instruction register or data register) in the LCD controller to write to or read from
- Read/write data from/to the LCD

The pin descriptions, timing constraints and detail instruction specifications can be found in the LCD User's Manual.

3.2 DC Motor

The motor on the AVR Microcontroller Board is DC (Direct Current) voltage driven. It takes the input electrical energy and converts the energy into rotating motion.

The speed of the motor is measured in revolutions (or rounds) per second (rps).

To determine the motor speed, we can use the shaft encoder. The encoder uses the infra red emitter and detector that are placed fairly close to each other (about half an inch or less). When there is a line of light between them, the detector is “on”’ and produces a voltage (normally around 1 to 3 volts). When the line of light is blocked the detector doesn’t pick up any infra red light so it is “off”, producing 0 volts.

Now, carefully examine the motor and shaft encoder on the AVR Microcontroller Board. Can you identify the emitter and the detector? The emitter is active high and the detector is active low (namely, it will go low when it can see the light).

Power up the AVR Microcontroller board and connect the pin named as POT to the MOT pin on the lab board. As you turn the POT (namely, change the DC input value), the speed of the motor changes accordingly.

As you may have noticed already, there are 4 holes on the wheel that is fixed on the motor, the combination of the motor and shaft encoder enables you to measure the speed of the motor by counting the number of the holes the shaft encoder detects per unit time and the motor speed is the count value divided by 4.

4. Tasks

4.1 Task 1 (Due week 10)

Write an assembly program that receives a character typed in from the keypad and prints it on the LCD. When the first line is full, the display goes to the next line. When the two lines are all full, the display is cleared and ready to display a new set of characters.

Assemble your program using AVR Studio, and run it on the AVR Microcontroller Board. Demonstrate your working program to the laboratory tutor.

4.2 Task 2 (Due week 11)

Write an AVR assembly language program that measures the speed of the motor by counting the number of holes that has been detected using the shaft encoder, and displays the motor speed of on the LCD.

Note: There are two ways to measure the speed: counting holes in a given period of time or measuring time for a complete revolution of the wheel (or several revolutions to average things out).

The detector will normally output 1 when there is no hole, so you can use the falling edge of the detector to trigger an (external) interrupt to count a hole.

1. Method 1: Timing can be done with Timer 0 or any of the other timers on the chip. Count Overflow interrupts to measure a period of one second while counting the number of holes that pass by using an external interrupt. Every second you can update your speed on the LCD.

2. Method 2: Set up the timer to time over an appropriate period. Wait for the first external interrupt to indicate the first hole and record the timer value. After four more external interrupts you know that 1 revolution has occurred. Find the difference between the start and end times (you may also need to count the number of timer overflows) and you can calculate the period of rotation for the wheel. From this value you can work out how many revolutions could occur in a second and display it.

The first method is fairly simple but will only update every second. The second method is a little more tricky, but gives you much higher accuracy and a much faster refresh rate (in fact you will probably have to limit this rate if you want to read the number off the LCD in time).

Assemble your program using AVR Studio, and run it on the AVR Microcontroller Board. Demonstrate your working program to the laboratory tutor.

4.3 Task 3 (due week 11)

Write an assembly program for DC motor speed control. You program should work as follows:

- When it is started, the motor spins at the speed of 20 rps (rounds per second).
- Every time when key 1 on the keypad is pressed, the motor speed is increased by 10 rps. What is the maximal speed you can achieve?
- Every time when key 2 on the keypad is pressed, the motor speed is decreased by 10 rps. What is the minimum speed you can achieve?
- When key 3 is pressed, the motor stops.
- When key 4 is pressed, the motor starts spinning at the speed of 20 rps.

Assemble your program using AVR Studio, and run it on the AVR Microcontroller Board. Demonstrate your working program to the laboratory tutor.

Note: Task 1 is worth 5 marks, and Tasks 2 and 3, each worth 8 marks. All your programs should be well commented. Up to 1 mark will be deducted for each program without proper and sufficient comments.

Appendix: Pulse Width Modulation

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 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 1 shows three different PWM signals. The top plot shows a PWM output at a 10% duty cycle. That is, the signal is on for 10% of the period and off the other 90%. The blow two plots show PWM outputs at 50% and 90% duty cycles, 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 will be produced.

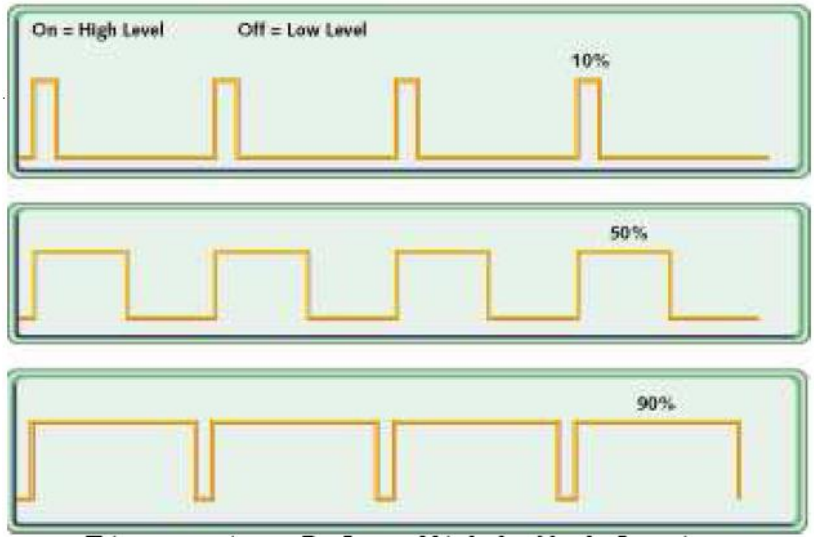

Figure 1 PWM signals

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, and 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. PWM can be controlled on the Atmega2560 through the use of a timer (e.g. Timer5) and its Phase Correct PWM mode. Basically, you control the timer to vary the duty cycle of the signal on the output pin that varies the average voltage on the pin that controls the motor speed.