

DC Motors

1 Objectives

The objectives of this laboratory exercise are to introduce the pulse-width-modulation method of controlling the speed of a DC motor and to explore the operation of an optical encoder. The relationship between PWM and RPM will be investigated for one particular motor.

2 Theoretical background

2.1 Pulse Width Modulation (PWM)

There are two methods of controlling the speed of a DC motor. With an analog controller, the magnitude of the input voltage is adjusted to change the speed. With a digital controller, a pulse-width-modulated signal is generally used.

Figure 1 shows a pulse-width-modulated signal. By adjusting the width of each pulse, the average power supplied to a motor can be controlled.

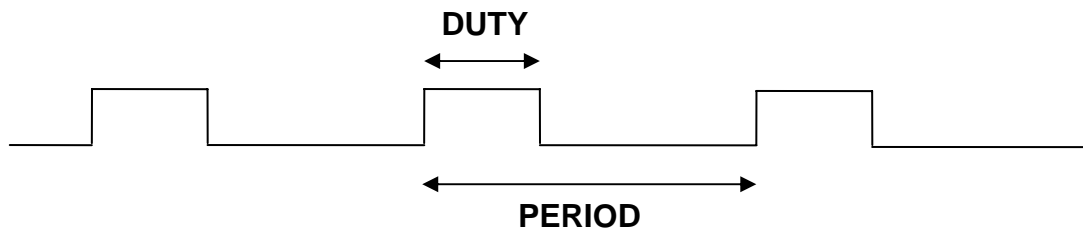


Figure 1: Pulse width modulation

The period of the PWM signal must be selected so that the motor runs smoothly. The period must be short enough that the motor does not slow down noticeably during the low part of the PWM signal. This will depend on the rotational inertia of the rotor and the loading applied to the motor.

The duty cycle of a PWM signal is defined as the percent of time that the signal is high.

$$\text{duty cycle} = \frac{\text{DUTY}}{\text{PERIOD}} \times 100\%$$

2.2 Drive Circuitry

The PWM signal produced by a microcontroller can control only the speed of a motor – to control the direction, additional circuitry is required. Furthermore, a microcontroller cannot provide enough power to drive a motor, so the drive circuit must also provide a higher voltage and current. Figure 2 shows one drive circuit that utilizes an L293D chip. When the PWM signal is high, the motor outputs 1Y and 2Y are enabled, meaning that they will respond to the control pins 1A and 2A. If ‘direction’ is set to 1, then output 1Y will be connected to the motor supply voltage and 2Y will be connected to ground. If ‘direction’ is set to 0, then output 2Y will be connected to the motor supply voltage and 1Y will be connected to ground.

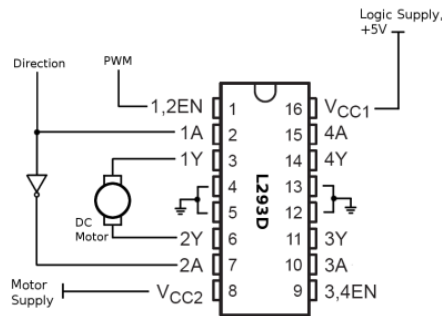


Figure 2: DC motor drive circuit

The MRK board has a drive circuit built-in that can supply 15 V and reverse the polarity to change the motor direction.

2.3 Optical Encoders

The basic optical encoder consists of a code wheel and an LED emitter-detector pair. As the code wheel rotates, light from the emitter LED is either blocked or allowed to pass through to the detector. The frequency of the resulting square wave is related to the speed of the motor.

With two emitter-detector pairs, the direction of rotation can be determined through quadrature decoding. The sensors are staggered so that the signals are 90 degrees out of phase, as shown in Figure 3. Note that on rising edges of Ch. A, Ch. B is low when rotating clockwise and it is high when rotating counter-clockwise. Finer decoding can be obtained by looking at both rising and falling edges or by looking at edges on both channels.

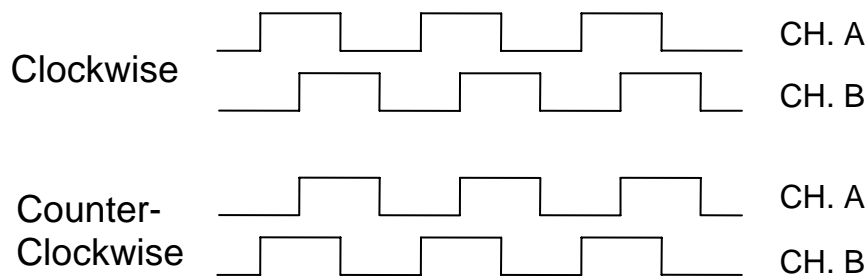
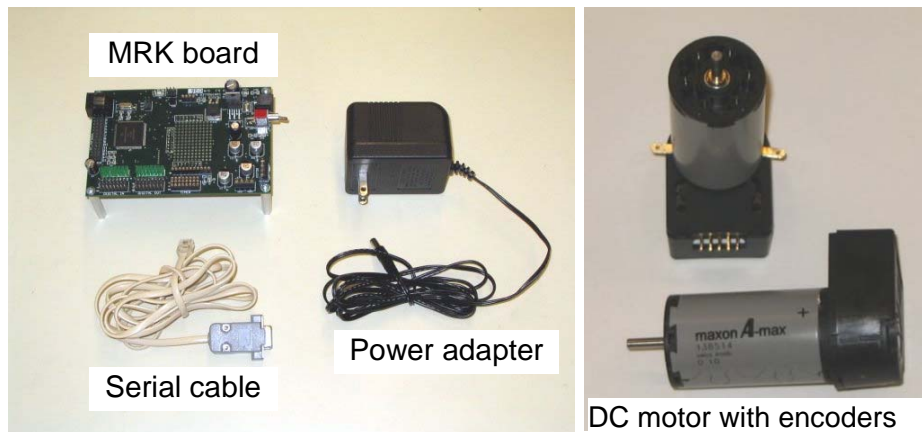


Figure 3: Two channel encoder.

3 Equipment



- MRK Controller Board
- Serial communication cable
- Power adapter
- DC motor with encoder
- Oscilloscope

4 Procedure

Exercise #1: In this exercise, you will display the PWM signal on an oscilloscope. Using the program given below, sketch the waveform for the following `pwm` values: 50, 100, and 255. Determine the frequency of the PWM signal and the duty cycle for each `pwm` value. The High Power switch on the MRK board must be enabled for the motor output ports to work.

While you are waiting for an oscilloscope, start working on the other exercises.

```
#include "MRK.h"

int pwm, print_flag;

void motor_ctrl(void) {
    int c;
    c = fgetchar(SCI0);
    if (c == '=') {                // + key (=) increases pwm
        pwm = pwm + 5;
        if (pwm > 255) pwm = 255;
        motor_out(LEFT, pwm);
        print_flag = 1;
    }
    else if (c == '-') {          // - key decreases pwm
        pwm = pwm - 5;
        if (pwm < -255) pwm = -255;
        motor_out(LEFT, pwm);
        print_flag = 1;
    }
}
```

```

}

int main() {
    setup_sci(SCI0, INTERRUPT, &motor_ctrl);
    pwm = 0;
    print_flag = 1;
    fprintf(SCI0, "Press '=' to increase pwm and '-' to decrease pwm.\n\r");
    while (1) {
        if (print_flag == 1) {
            fprintf(SCI0, "pwm = %d \r", pwm);
            print_flag = 0;
        }
    }
}

```

Exercise #2: In this exercise, you will use the timer subsystem and the motor's encoder to determine the motor's RPM. First, modify the program from Exercise #1 to calculate the time (number of timer clock ticks) between two rising edges on encoder Ch. A (timer port 0). You wrote a similar program in Lab 6 which calculated the frequency of a square wave. When this part of the program is working, add an algorithm to convert this value into revolutions per minute. The encoder you are using has 1024 pulses per revolution.

Test out your program. With $pwm = 50$, the motor speed should be in the range of 800 to 900 RPM.

Exercise #3: Use your program from Exercise #2 to determine the relationship between PWM and RPM. You should modify the program so that each time it prints, it prints to a new line. This will make it easier to average a few RPM values. Plot RPM vs. PWM, using enough data points for adequate representation. How linear is the relationship?

Exercise #4: The following program demonstrates one method of quadrature decoding. Run the program, and note that the output is accurate only at low speeds. What could cause the result to be wrong at higher speeds?

```

#include "MRK.h"

int pwm, print_flag, chB;

void motor_ctrl(void) {
    int c;
    c = fgetchar(SCI0);
    if (c == '=') { // + key (=) increases pwm
        pwm = pwm + 5;
        if (pwm > 255) pwm = 255;
        motor_out(LEFT, pwm);
        print_flag = 1;
    }
    else if (c == '-') { // - key decreases pwm
        pwm = pwm - 5;
        if (pwm < -255) pwm = -255;
    }
}

```

```

        motor_out(LEFT, pwm);
        print_flag = 1;
    }
}

void quad_decode(void) {
    if (timer_port(IN,1)==TRUE) chB = 1;
    else chB = 0;
    print_flag = 1;
}

int main() {
    setup_timer(ENABLE,0);
    setup_timer(INPUT_CAPTURE,0,RISING,&quad_decode);
    setup_timer(INPUT,1);

    setup_sci(SCI0,INTERRUPT,&motor_ctrl);
    pwm = 0;
    chB = 0;
    print_flag = 1;
    fprintf(SCI0,"Press '=' to increase pwm and '-' to decrease pwm.\n\r");
    while (1) {
        if (print_flag == 1) {
            fprintf(SCI0,"ChA rising edge, ChB = %d (pwm = %d) \r",
chB,pwm);
            print_flag = 0;
        }
    }
}

```

5 Report and analysis requirements

5.1 Theory

Describe the procedure enabling the digital control of a DC motor.

What is the major parameter affecting the motor speed?

Describe quadrature decoding.

5.2 Results and analysis

Answer all questions in previous sections of these laboratory instructions.

Present and discuss you results.

Is it possible to digitally control the motor by connecting the motor's input directly to the pins of the PWM block? Discuss your connection options.

For a DC motor without build-in encoders how would you determine (a) speed and (b) direction of rotation?

Discuss relationship between RPM and PWN. How close is your data to the linear relationship? Is it linear in a whole range of motor speeds?

*Last updated
November 9, 2006.*