

# Laboratory 14

---

*Pulse-Width-Modulation Motor Speed Control with a PIC (modified from lab text by Alciatore)*

## Required Components:

- 1x PIC 16F88 18P-DIP microcontroller
- 3x 0.1  $\mu$ F capacitors
- 1x 12-button numeric keypad
- 1x NO pushbutton switch
- 1x Radio Shack 1.5-3 V DC motor (RS part number: 273-223) or equivalent
- 1x IRFZ34N power MOSFET (4V transistion)
- 1x flyback diode (e.g., the IN4001 power diode)
- 4x 1k $\Omega$  resistors
- 3x red LEDs
- 1x green LED
- 4x 330 $\Omega$  resistors

## Required Special Equipment and Software:

- MPLab X, microchip technology's IDE
- XC8, opensource compiler for PICs
- PICkit 2 software
- CanaKit USB PIC Programmer

## Objective

The objective of this laboratory exercise is to design and build hardware and software to implement pulse-width modulation (PWM) speed control for a small permanent-magnet dc motor. You will also learn how to interface a microcontroller to a numeric keypad and how to provide a numerical display using a set of LEDs.

## Introduction

### Pulse Width Modulation

**Pulse width modulation (PWM)** offers a very simple way to control the speed of a dc motor and is one of the building blocks of a DC->AC inverter. Figure 14.1 illustrates the principles of operation of PWM

## Laboratory 14 PWM motor

control. A dc voltage is rapidly switched at a fixed frequency  $f$  between two values ("ON" and "OFF"). A pulse of duration  $t$  occurs during a fixed period  $T$ , where  $T = \frac{1}{f}$ . The resulting asymmetric waveform has a **duty cycle** defined as the ratio between the ON time and the period of the waveform, usually specified as a percentage:

$$\text{duty cycle} = \frac{t}{T} \cdot 100\% \quad (14.1)$$

As the duty cycle is changed (by varying the pulse width  $t$ ), the average current through the motor will change, causing changes in speed and torque at the output. It is primarily the duty cycle, and not the value of the power supply voltage, that is used to control the speed of the motor.

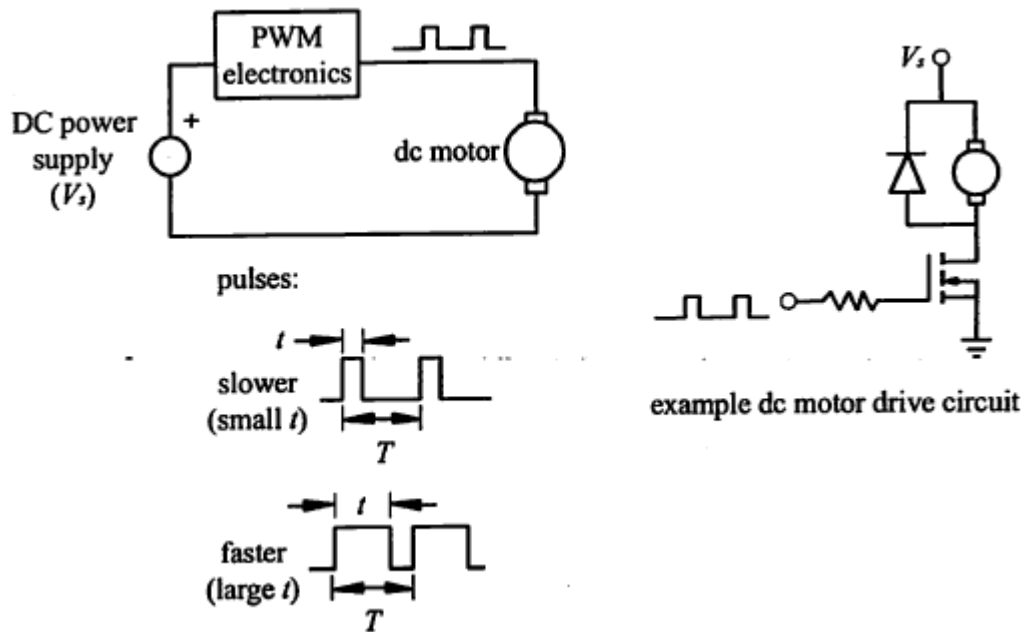


Figure 14.1 Pulse-width Modulation (PWM)

With a PWM motor controller, the motor armature voltage switches rapidly, and the current through the motor is affected by the motor inductance and resistance. For a fast switching speed (i.e., large  $f$ ), the resulting current through the motor will have only a small fluctuation around an average value, as illustrated in Figure 14.2. As the duty cycle gets larger, the average current gets larger and the motor speed increases.

## Laboratory 14 PWM motor

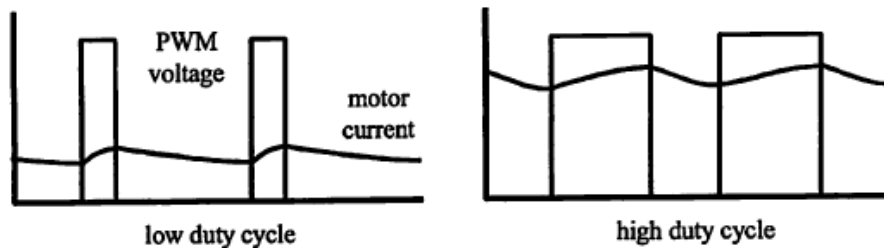
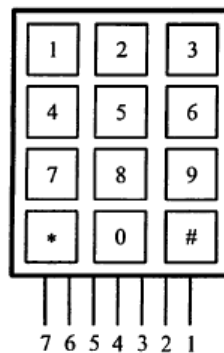


Figure 14.2 PWM voltage and motor current

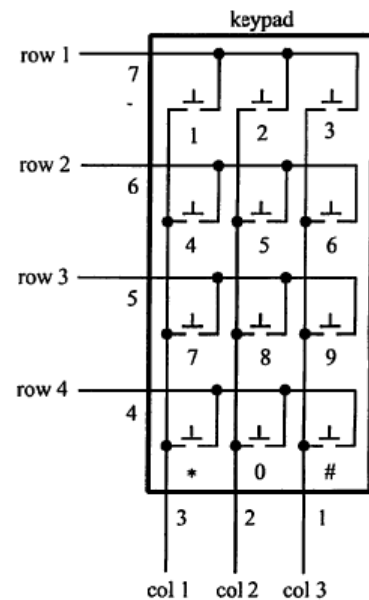
The type of PWM control described here is called "open loop" because there is no sensor feedback for speed. This results in a simple and inexpensive design, but it is not possible to achieve accurate speed control without feedback. For precision applications (e.g., industrial robotics, DC->DC power converters and DC->AC power inverters), either a speed sensor (for motors) or a voltage sensor (for power converters and inverters) is required to provide feedback to the electronics or software in order to adjust the PWM signal in real-time to maintain the desired result (i.e., speed or voltage). For more information about precision motor control see Section 10.5.3 in the textbook. Power converters and inverters (among other things) are covered in some depth in Power Electronics II (PHY541).

## Numeric Keypad Interface

Figure 14.3 illustrates the appearance and electrical schematic for a common 12-key **numeric keypad**; although, the pin numbering isn't always consistent from one manufacturer to another. When interfaced to a microcontroller, a keypad allows a user to input numeric data. A keypad can also be used simply as a set of general-purpose normally-open (NO) pushbutton switches. The standard method to interface a keypad to a microcontroller is to attach the four row pins to inputs of the microcontroller and attach the three column pins to outputs of the microcontroller.



a) device appearance



b) device electrical schematic

Figure 14.3 Standard 12-key numeric keypad

By polling the states of the row inputs while individually changing the states on the column outputs, you can determine which button is pressed. See Section 7.7.1 in the textbook for more information.

NOTE: If the pin-out of the keypad you are using is unknown, you can do a series of continuity tests (with different buttons held down) to easily determine the pin-out corresponding to Figure 14.3b.

## Hardware and Software Design

The hardware and software required for this exercise will be designed using the microcontroller-based design procedure presented in Section 7.9 of the textbook. Each step is presented below.

1. *Define the problem.*

Use a PIC16F88 microcontroller to design a pulse-width modulation speed controller for a small permanent magnet dc motor. The user should be able to change the speed via three buttons of a standard 12-key numeric keypad. One button (the 1-key) should increase the speed setting, a second button (the 4-key) should decrease the speed setting, and the third button (the \*-key) should start the motor at the selected speed. The speed setting should be displayed graphically via a set of 4 LEDs. The speed setting should vary from "slow" to "fast" according to a scaled number ranging from 0 to 15 so the full range can be depicted on the LED display. The motor should run at a constant speed until the motion is interrupted by the user with the press of a pushbutton switch.

2. *Draw a functional diagram.*

This is left as an exercise for you. Please include it on a separate sheet of paper with your summary sheet and questions at the end of the Lab. See Section 7.9 in the textbook for guidance.

3. *Identify I/O requirements.*

All inputs and outputs for this problem are digital and they are as follows:

inputs:

- 3 buttons on the numeric keypad increase and decrease the speed and to start the motion.
- 1 pushbutton switch to interrupt the constant speed motor motion.

outputs:

- 4 LEDs to indicate a relative speed setting from "slow" (0) to "fast" (15) as a binary number.
- 1 pulse-width modulation (on-off) signal for the motor.
- 1 to power a key-pad column.

4. *Select an appropriate microcontroller.*

For this problem, we will use the PIC16F88 whose 16 lines of digital I/O provide more than enough capability for our I/O requirements.

5. *Identify necessary interface circuits.*

You will learn how to use a numeric keypad by polling columns and searching rows for pressed buttons.

The motor speed will be controlled with a pulse-width modulation signal. We will use a power MOSFET to switch current to the motor. The gate of the MOSFET will be connected directly to a digital output pin on the PIC. The motor is placed on the drain side of the MOSFET with a diode

## Laboratory 14 PWM motor

for flyback protection. A MOSFET is easier to use than BJT because it does not require a base (gate) resistor, and you need not be concerned with base current and voltage biasing.

The LEDs will be connected directly to four digital outputs through current-limiting resistors to ground. When the output goes high, the LED will turn on.

### 6. *Decide on a programming language.*

For this laboratory exercise, we will use MPLab X with C for PICs.

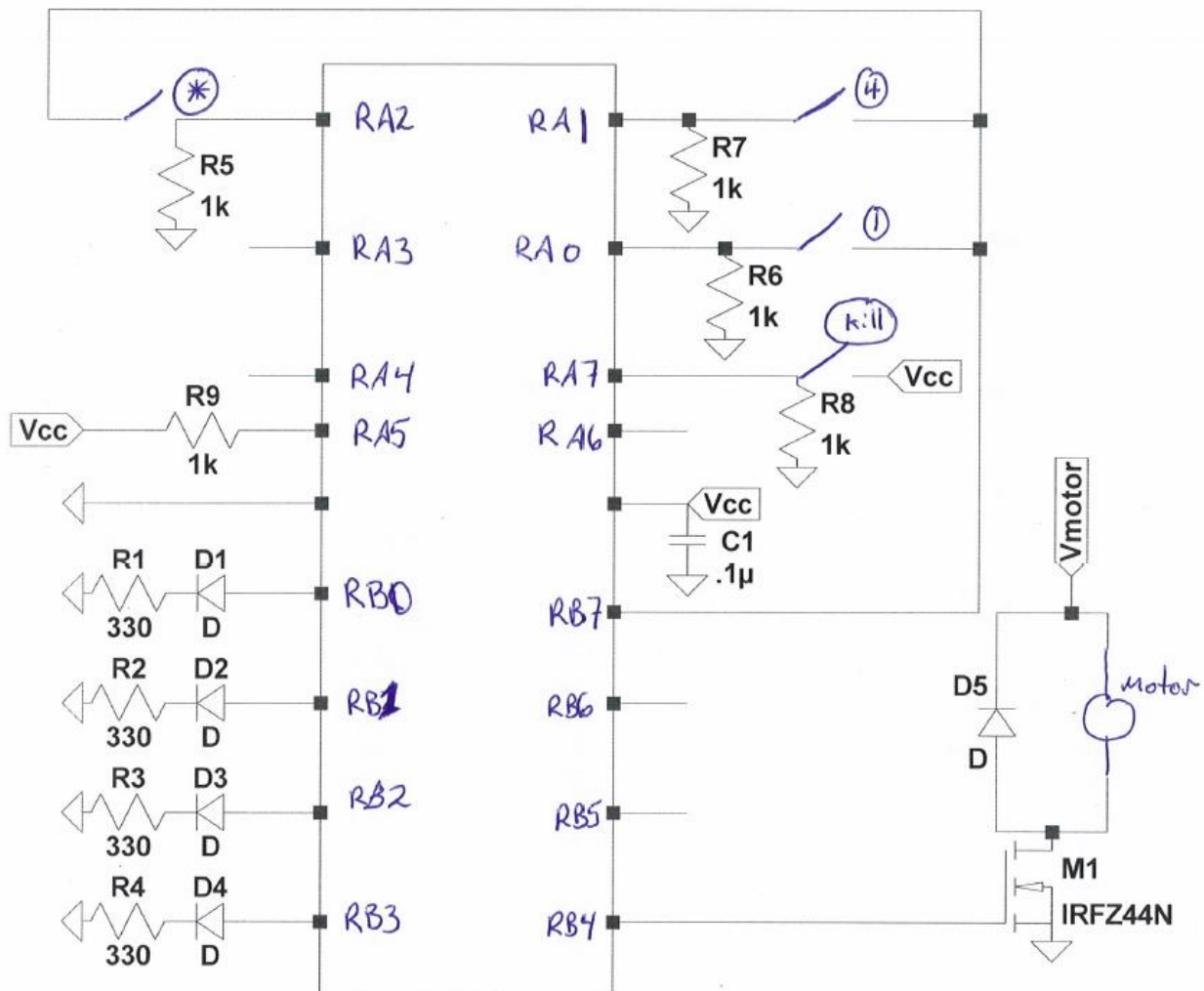


Figure 14.4 Complete schematic showing all components and connections. Note that LEDs are arranged from least significant bit (LSB) to most (MSB).

### 7. *Draw the schematic.*

Figure 14.4 shows the complete schematic showing all components and connections. Figure 14.5 shows a photograph of a completed design. The keypad is attached to bits 0-2 of PORTA and bit 7 of PORTB. The stop button is attached to bit 7 (of PORTA). The LEDs are attached to the four lowest order bits of PORTB. This allows the speed setting (0 to 15) to be output to PORTB directly (e.g., PORTB = speed). The result is a binary number display of the current speed where the green LED represents the least significant bit. The motor PWM signal is on bit 4 of PORTB.

**NOTE - We are using only one column of the keypad.**

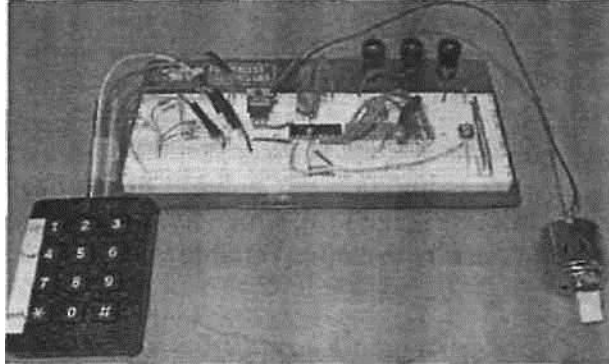


Figure 14.5 Photograph of the actual device.

8. *Draw a program flowchart.*

Figure 14.6 shows the complete flowchart for this problem with all required logic and looping. Note that the LED display is active only during the keypad loop while the user is adjusting the speed. The keypad is polled using the Pot command and the speed display is updated approximately three times a second. The motor runs continuously in the PWM loop until the stop button is pressed. At that point the user can adjust the speed again.

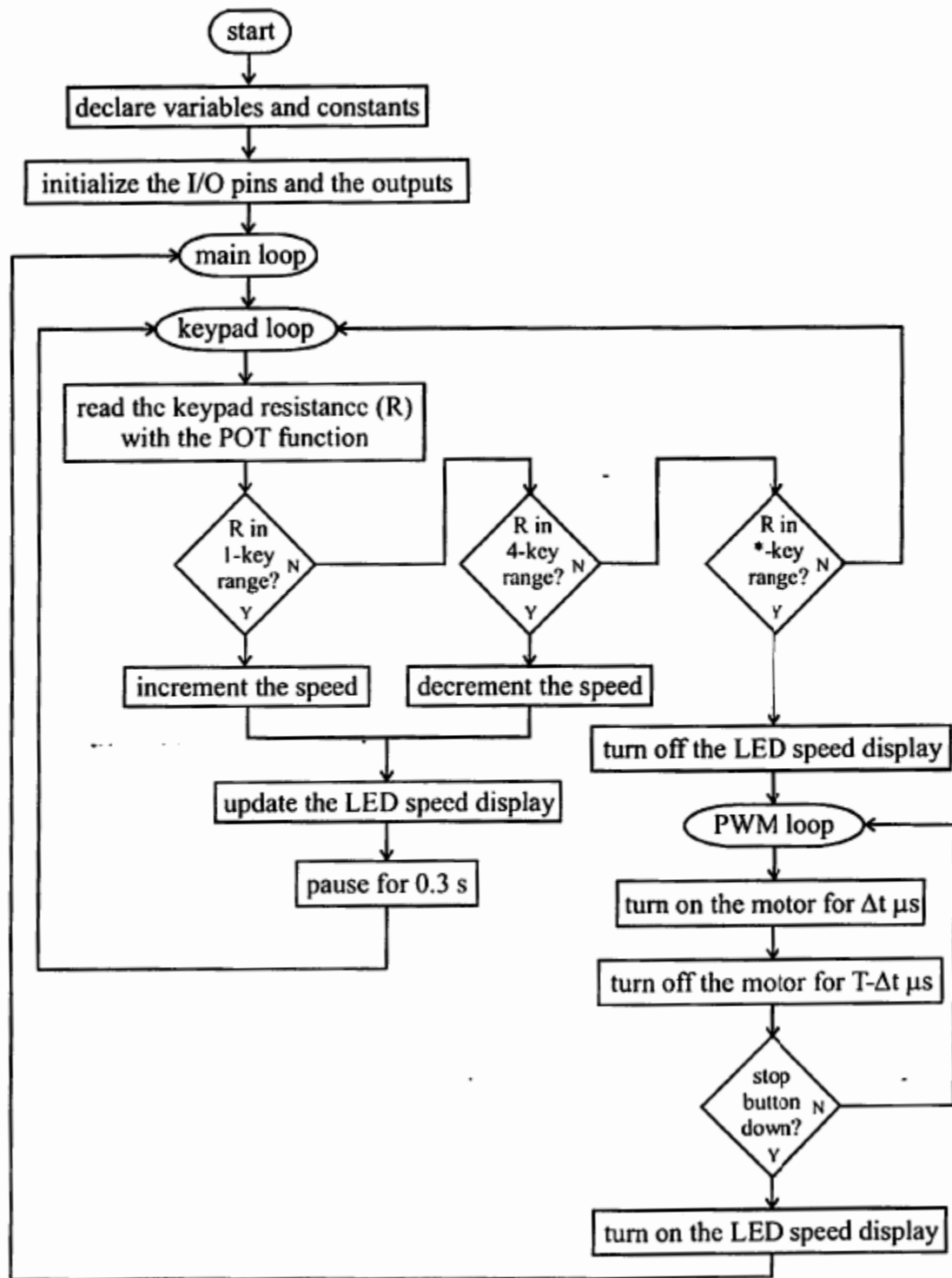


Figure 14.6 One possible complete Program Flowchart

## 9. Write the code.

The code ("PWM.c") corresponding to the flowchart shown in Figure 14.6 using the hardware illustrated in Figure 14.4 follows. The code is commented throughout with remarks so it should be self-explanatory. Whenever you write programs, you should always include copious remarks so you and others (e.g., co-workers and bosses) can later interpret what you have done. **Please recreate this before coming to Lab so you will have more time to successfully complete the Lab in the allotted time.**

```

/*
 * File:    PWM.c
 * Author: douglas.armstead
 * Controls the speed of a DC motor using pulse-width modulation
 * (PWM). The speed is adjusted via user input with three buttons
 * (increase, decrease, and enter) on a numeric keypad. The relative
 * speed is stored as a number that ranges from 0 (corresponding to
 * 15% duty cycle) to 15 (corresponding to 35% duty cycle). The provisional
 * value of the speed is displayed graphically with a set of
 * 4 LEDs that show the bits of the equivalent binary number.
 *
 * Created on May 4, 2018, 1:38 AM
 */

#include <stdio.h>
#include <stdlib.h>
#define _XTAL_FREQ 1000000 //Set the clock speed in the software
#include <xc.h>
#include "cortlandStd.h"
//These parameters are for use with the delay functions
#define DT 400 //delay increment dt in us
#define TO 2000 //minimum delay To in us
#define TREST 22 //minimum off time in us

/*
 *
 */
//speed is global
int speed=0, setSpeed=0;

```



## Laboratory 14 PWM motor

```
void getSpeed(void) {
    //read the input
    //activate the first (and only) column
    PORTBbits.RB7=1;
    __delay_us(10);
    //Check row 1, 2, and 4
    if (PORTAbits.RA0==1) {
        speed+=1;
        if (speed==16) {
            speed=15;
        }
        __delay_ms(150);
    }
    if (PORTAbits.RA1==1) {
        speed-=1;
        if (speed==-1) {
            speed=0;
        }
        __delay_ms(150);
    }
    if (PORTAbits.RA2==1) {
        setSpeed=speed;
        //flash to acknowledge that the speed update.
        PORTB=15;
        __delay_ms(150);
    }
    //display the desired speed
    PORTB=speed;
    __delay_us(10);
    PORTBbits.RB7=0;
    //done reading input
    return;
}
```

## Laboratory 14 PWM motor

```
void runMotor(void) {
    int i;
    __delay_ms (TREST);
    //Kill button is positive logic and on RA7.
    if (PORTAbits.RA7==0) {
        PORTBbits.RB4=1;
    }
    __delay_us (TO);
    for (i=0;i<16;i++){
        if (i>=setSpeed) {
            PORTBbits.RB4=0;
        }
        __delay_us (DT);
    }
    return;
}

int main(int argc, char** argv) {
    OSCCONbits.IRCF=0b100; //Set chip clock to 1MHz
    ANSEL=0;
    //PortB is all output:
    //RB0-3 are LEDs,
    //RB4 is the motor,
    //RB7 is for pad column.
    TRISB=0b000000;
    //PortA is all input:
    //RA0-2 are pad rows
    //RA7 is the kill switch.
    TRISA=0b10000111;
    while (1==1) {
        getSpeed();
        runMotor();
    }

    return (EXIT_SUCCESS);
}
```

The variable "speed" stores a relative measure of the motor speed as an integer that varies from 0 (slow) to 15 (fast). A speed of 0 corresponds to a duty cycle of 15% and a speed of 15 corresponds to a duty cycle of 35%. These duty cycle percentages were determined experimentally to produce a good range of motor speeds using a 5 V supply. **(Note - the motor is rated at only 1.5 to 3 V so high duty cycles would result in excessive average voltage, which could damage the motor.)**

One challenge is how to deal with the variable amount of time to have the motor on and off (according to the speed). The delay functions require constants, the language's way of ensuring that the delay doesn't get extended because of calculations happening in the delay function input. To deal with this I

## Laboratory 14 PWM motor

have defined 3 constants,  $T_0$  which corresponds to the minimum time (in  $\mu\text{s}$ ) for the motor to be run in a duty cycle,  $DT$  which is the amount that gets progressively added to the on time (in  $\mu\text{s}$ ), and  $T_{REST}$  the amount that the motor is certainly off in a cycle (in ms). As a result the on-time for the motor is

$$t_{on} = T_0 + DT * speed$$

(in  $\mu\text{s}$ ) and

$$t_{off} = T_{REST} * 1000 + DT * (15 - speed)$$

(in  $\mu\text{s}$ ) where the 1000 is for the number of  $\mu\text{s}$  in a ms.

Note that the period is constant

$$T = T_{on} + T_{off} = T_0 + T_{REST} * 1000 + DT * 15$$

### 10. Build and test the system.

That is your job using the procedure in Section 14.5.

## Troubleshooting and Design Improvement

There are several changes you can make to the circuit to improve the design's robustness. **You will definitely want to explore some of these recommendations if you have trouble getting your circuit to function properly.** Because the motor is being switched on and off, and because the currents in the motor are being switched by the internal commutator, spikes and noise can occur on the 5V and ground lines. Also, the Lab power supply voltage might be affected by current spikes (e.g., the voltage can drop suddenly, causing the PIC to reset). To help minimize these effects, you can add a 1  $\mu\text{F}$  or larger capacitor across the 5V and ground line inputs to your breadboard to help stabilize the voltage there. You might also **try increasing the capacitance between  $V_{dd}$  and ground on the PIC (i.e., replace the 0.1  $\mu\text{F}$  with 1  $\mu\text{F}$  or more).** You can also add capacitance (e.g., 0.1-1.0  $\mu\text{F}$ ) across the tabs of the motor to help filter out spikes and noise from the commutation. Also, make sure the wires attached to the motor are soldered to the motor tabs to ensure solid and reliable connections.

Another alternative is to use separate power sources for the PIC circuit (a Lab power supply) and the motor (e.g., a second channel of the Lab power supply, or a 9V battery with a 5V voltage regulator). This will help limit voltage fluctuations in the PIC circuit when the motor turns on and runs.

**For other advice and recommendations, see Section 15.5 in Lab 15.**

## Procedure / Summary Sheet

1. Complete and attach a detailed functional diagram, using Sections 1.3 and 7.9 in the textbook for guidance. Submit this on a separate sheet of paper.
2. Use an ASCII editor (e.g., Windows Notepad or MS Word - Text Only) to create the program "PWM.c" listed in Section 14.3. Save the file in a folder in your network file space.
3. Follow the procedure in Section. 12.4 of Lab 12 to load your program in a PIC microcontroller that you can insert into your circuit.
4. Build the circuit shown in Figure 14.4 and insert the PIC programmed with "PWM.c" You should omit the motor driver circuit at first so and make sure that the keypad and LED displays work. Only then should you insert the motor driver circuit.
5. **See the Trouble Shooting Section if your circuit is assembled correctly but does not work properly.** Show me your functioning circuit so I can verify it is working.



## Laboratory 14 PWM motor

4. What would happen if other keys (besides the 1-key, 4-key, and \*-key) are pressed down during the keypad loop?

What would happen if two of the three valid keys are pressed and held down at once (e.g., the 1-key and the \*-key)?