Laboratory 11

Pulse-Width-Modulation Motor Speed Control with a PIC

Required Components:

- 1 PIC16F88 18P-DIP microcontroller
- 3 0.1 μF capacitors
- 1 12-button numeric keypad
- 1 NO pushbutton switch
- 1 Radio Shack 1.5-3V DC motor (RS part number: 273-223) or equivalent
- 1 IRF620 power MOSFET
- 1 flyback diode (e.g., the 1N4001 power diode)
- 3 1kΩ resistors
- 1 2kΩ resistor (or 2 1kΩ resistors)
- 1 3kΩ resistor (or 3 1kΩ resistors)
- 3 red LEDs
- 1 green LED
- 4 330Ω resistors or a 330Ω 8-resistor DIP

Required Special Equipment and Software:

- Mecanique's Microcode Studio integrated development environment software
- MicroEngineering Labs' PicBasic Pro compiler
- MicroEngineering Labs' U2 USB Programmer

11.1 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.

11.2 Introduction

Pulse Width Modulation

Pulse width modulation (PWM) offers a very simple way to control the speed of a dc motor. Figure 11.1 illustrates the principles of operation of PWM 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

$$T = \frac{1}{f} \tag{11.1}$$

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:

$$\text{duty cycle} = \frac{t}{T}100\% \tag{11.2}$$

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.

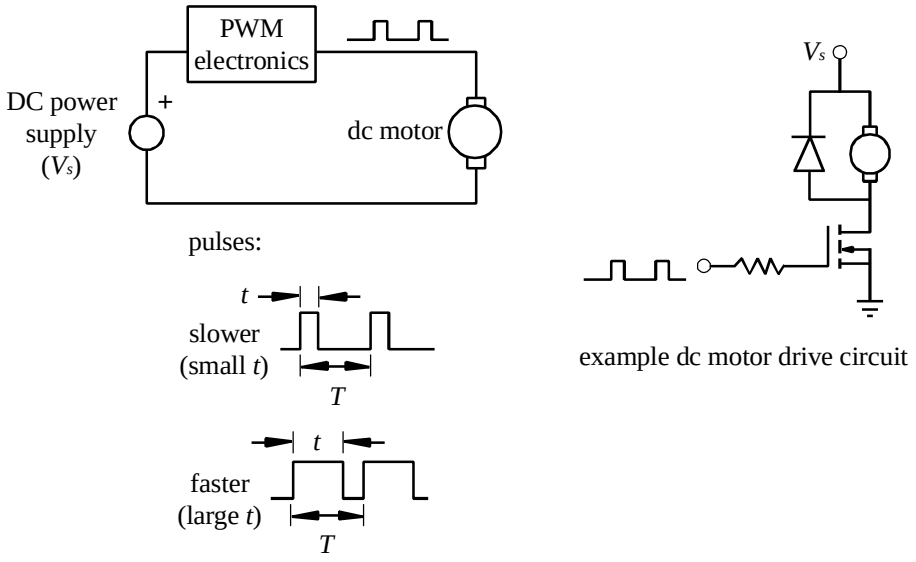


Figure 11.1 Pulse-width Modulation (PWM)

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 11.2. As the duty cycle gets larger, the average current gets larger and the motor speed increases.

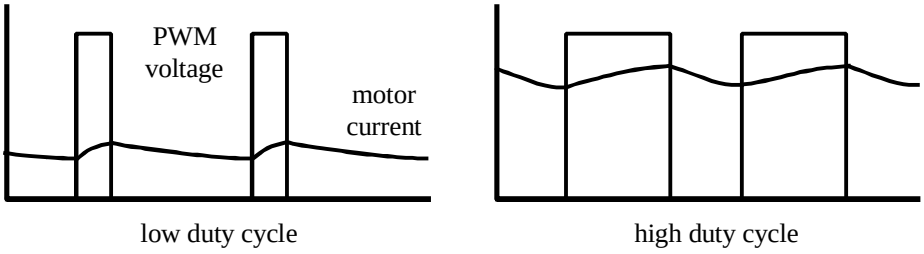
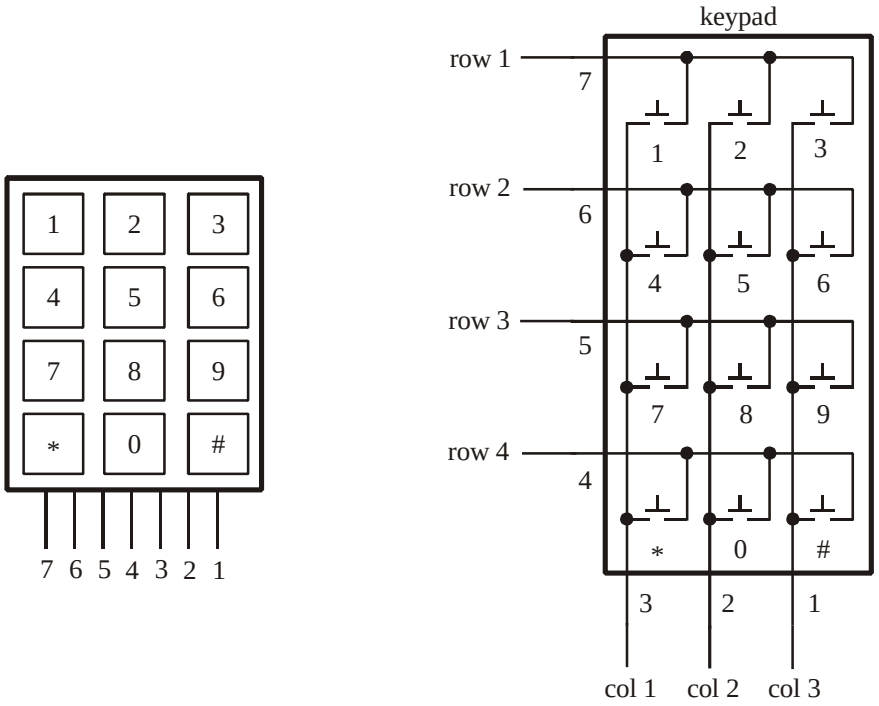


Figure 11.2 PWM voltage and motor current

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), a speed sensor (e.g., a tachometer) is required to provide feedback to the electronics or software in order to adjust the PWM signal in real-time to maintain the desired speed. See Section 10.5.3 in the textbook for more information.

Numeric Keypad Interface

Figure 11.3 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. 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. An alternative method to interface the keypad, if you do not have the luxury of seven spare I/O lines, is to wire the keypad through a set of resistors in series with a capacitor to ground. This allows you to use the PicBasic Pro "Pot" command to determine which button is pressed by reading the effective resistance of the keypad through a single pin of the microcontroller. The circuit presented in the next section uses this method.



a) device appearance b) device electrical schematic

Figure 11.3 Standard 12-key numeric keypad

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 11.3b.

NOTE: Keypads sometimes include an 8th pin, but it is not used in the wiring of the buttons.

11.3 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 PIC16F84 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 to 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.

(4) *Select an appropriate microcontroller*.

For this problem, we will use the PIC16F84 whose 13 lines of digital I/O provide more than enough capability for our I/O requirements.

(5) *Identify necessary interface circuits*.

To help you learn how to use a numeric keypad in the most efficient way, we will show you how to connect the rows and columns of the keypad through a network of resistors in series with a capacitor through a single pin on the PIC. With the help of the PICBasic Pro command "Pot," we can determine which button is pressed based on the time constant of the resulting RC network. The resistance will change based on which button is pressed. Only a single digital input is required to implement this method.

The motor speed will be controlled with a pulse-width modulation signal. We will use a power MOSFET to switch current to the motor. Figure 11.4 shows the pin-out diagram for the MOSFET. The gate (G), drain (D) and source (S) are analogous to the BJT base (B), collector (C) and emitter (E), respectively. 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 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.

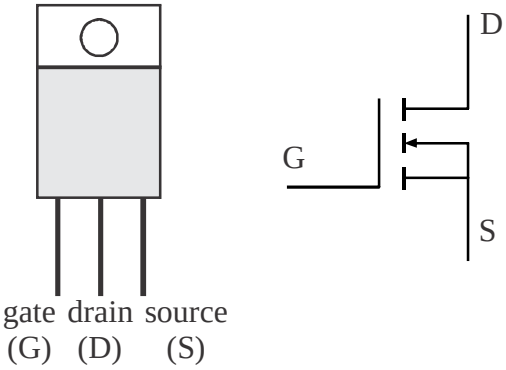


Figure 11.4 MOSFET pin-out and schematic symbol

(6) *Decide on a programming language*.

For this laboratory exercise, we will use PicBasic Pro.

(7) *Draw the detailed wiring diagram*.

Figure 11.5 shows the complete wiring diagram showing all components and connections. Figure 11.6 shows a photograph of a completed design.

The keypad is attached to PORTA.2 and the stop button is attached to PORTA.3. The keypad is wired such that different resistors are in series with a fixed capacitor depending upon which button is held down (1kΩ for the 1-key, 2kΩ for the 4-key, and 3kΩ for the *-key). 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 LSB. The motor PWM signal is on PORTA.1.

NOTE - Since we are using only one column of the keypad, the alternative RC circuit wiring shown in Figure 11.7, which uses only 1k resistors, is a good option (e.g., if 2k and 3k resistors are not available).

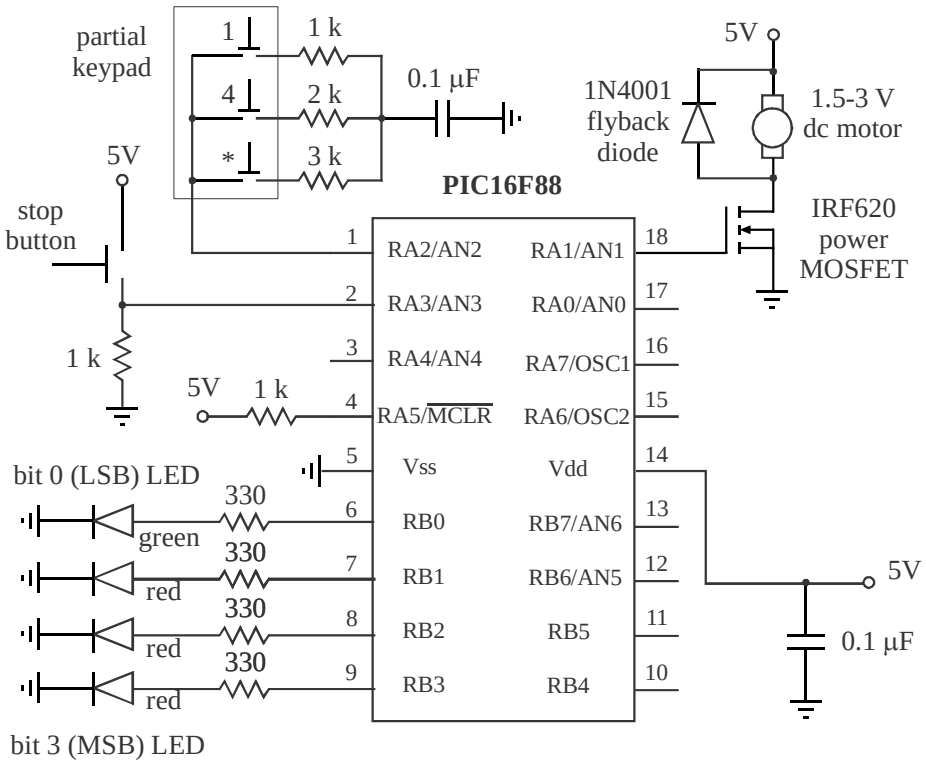


Figure 11.5 Complete wiring diagram showing all components and connections

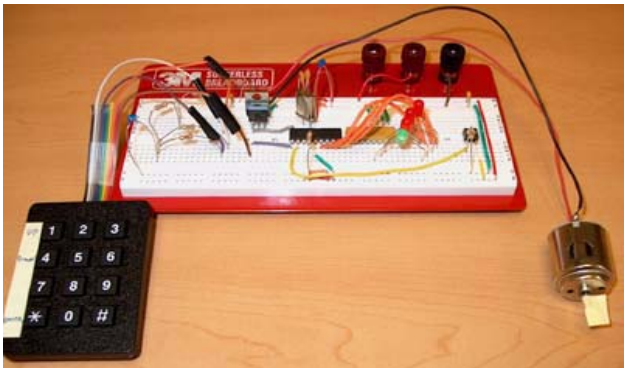


Figure 11.6 Photograph of the actual design

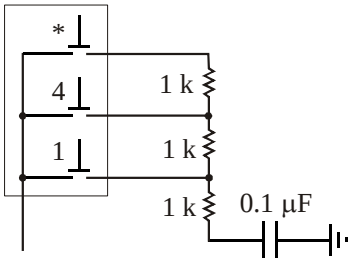


Figure 11.7 Alternative wiring for single-column keypad RC circuit

(8) *Draw a program flowchart.*

Figure 11.8 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. Each keypad button results in a different resistance value that can vary over a small range. The motor runs continuously in the PWM loop until the stop button is pressed. At that point the user can adjust the speed again.

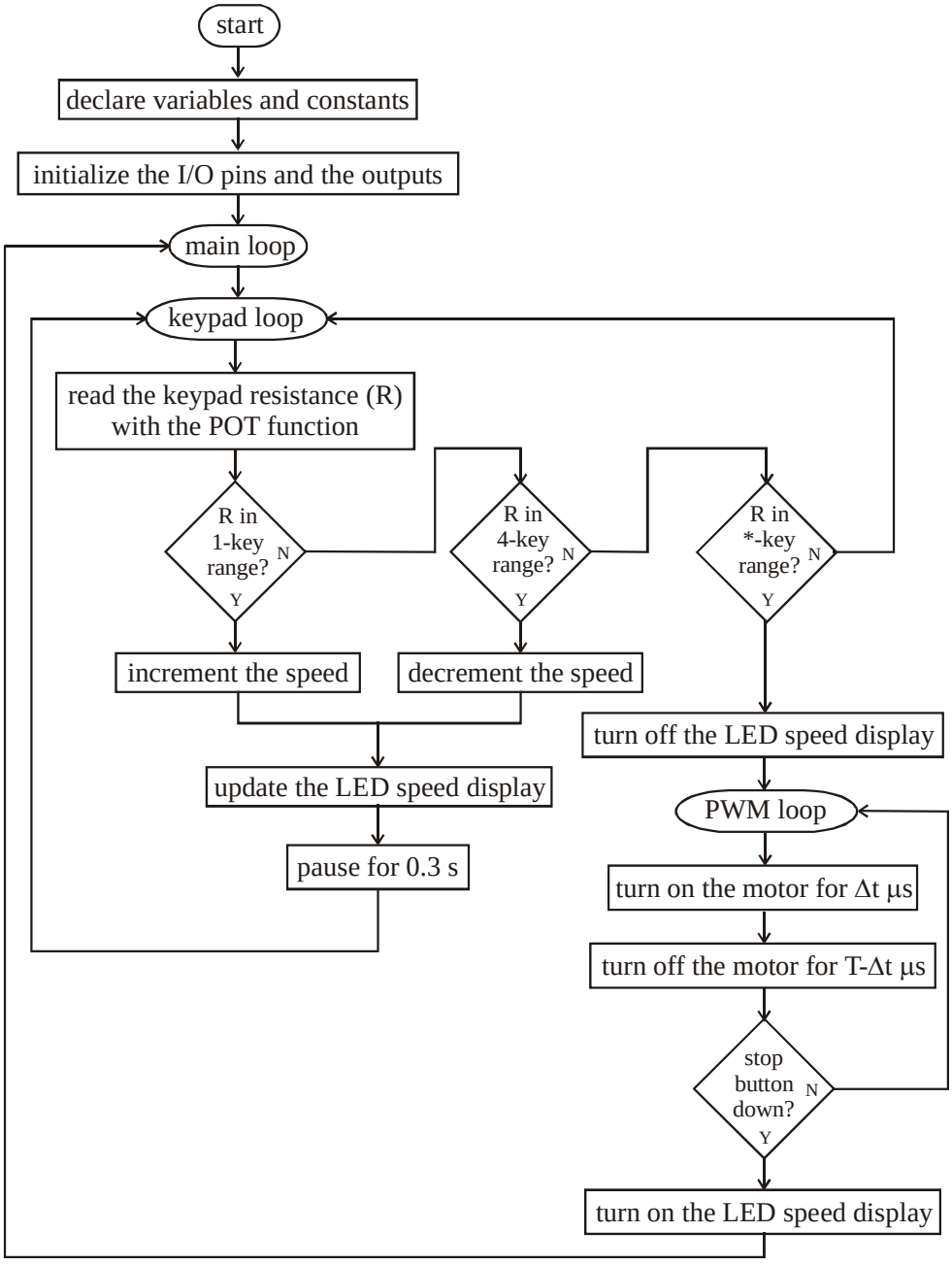


Figure 11.8 Complete Program Flowchart

(9) *Write the code.*

The PicBasic Pro code ("PWM.bas") corresponding to the flowchart shown in Figure 11.8 using the hardware illustrated in Figure 11.5 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 create this (PWM.bas) and the later file (PWM.cal) before coming to Lab so you will have more time to successfully complete the Lab in the allotted time.**

NOTE: Be sure to follow the procedure in Section 11.5 and run PWM_cal.bas (shown later) first before loading and running PWM.bas.

```
' PWM.bas
'
' 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 current value of the speed is displayed graphically
'  with a set of 4 LEDs that show the bits of the equivalent binary number.

' Identify and set the internal oscillator clock speed (required for the PIC16F88)
DEFINE OSC 8
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1

'Turn off the A/D converter (required for the PIC16F88)
ANSEL = 0

' Define pin assignments, variables, and constants
led0        Var  PORTB.0    ' LSB (bit 0) green LED
led1        Var  PORTB.1    ' bit 1 red LED
led2        Var  PORTB.2    ' bit 2 red LED
led3        Var  PORTB.3    ' MSB (bit 3) red LED

motor       Var  PORTA.1    ' PWM output pin to motor MOSFET gate
change      Var  PORTA.3    ' button causing the motor to stop for
                            '   speed adjustment
speed       Var  BYTE       ' User-input speed
MAX_SPEED   Con  15         ' Maximum relative speed
T           Var  WORD       ' pulse period in milliseconds
t_on        Var  WORD       ' pulse width (high state)
T_t         Var  WORD       ' pulse down (low state) time: (T - t)

pot_pin     Var  PORTA.2    ' keypad pin for POT command
SCALE       Con  255        ' Pot statement scale factor
pot_val     Var  BYTE       ' value returned by POT command
```

```
' Initialize the I/O pins
TRISA = %11101              ' designate PORTA pins as inputs and output (RA1)
TRISB = %00000000           ' designate PORTB pins as outputs
PORTB = 0
Low motor                   ' make sure the motor remains off initially
' Initialize the speed display information
T = 30000                   ' pulse period in microseconds
speed = 7                   ' select a medium speed to begin (the middle of the 0 to 15 range)
PORTB = speed               ' display the speed as a binary number on the 4 LEDs

' Main Loop
myloop:
        ' Endless speed change loop (until Exit with *-key)
        Do While (1)   ' 1:true
                ' Read the keypad resistance
                POT pot_pin, SCALE, pot_val

                ' Check for the 1-key to increase the speed
                If (pot_val > 30) && (pot_val < 95) && (speed < MAX_SPEED) Then
                        speed = speed + 1
                        PORTB = speed
                        Pause 300
                ' Check for the 4-key to decrease the speed
                ElseIf (pot_val > 95) && (pot_val < 160) && (speed > 0) Then
                        speed = speed – 1
                        PORTB = speed
                        Pause 300
                ' Check for the *-key to start motor motion
                ElseIf (pot_val > 160) Then
                        Exit      ' break out of the endless loop
                Endif
        Loop

        ' Turn off the LEDs
        PORTB = 0

        ' Initialize the pulse information
        t_on = T/5 / MAX_SPEED * speed + T/20*3          ' duty cycle range = 15% to 35%
        T_t = T – t_on

        ' Run the PWM until the user presses the stop button
        Do While (change == 0)
                High motor
                Pauseus t_on
                Low motor
                Pauseus T_t
        Loop
```

```
        ' Turn the LED speed display back on
        PORTB = speed
Goto myloop
' End of the program (never reached)
End
```

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 the 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 not so obvious challenge in the program is how the variable "t" is calculated. Because PicBasic Pro stores variables and does arithmetic with limited size integers, you have to be careful with truncation and overflow effects when performing calculations. For example, the equation:

$$t\_on = T/5 / MAX\_SPEED * speed + T/20*3 \qquad (11.3)$$

would not work properly if it were written as:

$$t\_on = speed / MAX\_SPEED * T/5 + T/20*3 \qquad (11.4)$$

or as:

$$t\_on = T/5 * speed / MAX\_SPEED + T/20*3 \qquad (11.5)$$

The variable speed can vary from 0 to 15, so from Equation 11.3 where MAX_SPEED is 15, t can vary from 3/20 T (15% of T) to 7/20 T (1/5 T + 3/20 T = 35% of T). Note that parentheses are not required to have the calculations in the equation execute in the correct order because, as with all programming languages, PicBasic Pro gives higher precedence to multiplication and division (which occur from left to right), than with addition and subtraction. Therefore, to PicBasic Pro, Equation 11.3 looks like:

$$t\_on = (((T/5) / MAX\_SPEED) * speed) \quad + \quad ((T/20) * 3) \qquad (11.6)$$

There is a problem with Equation 11.4 due to integer arithmetic **truncation**. Because "speed" varies from 0 to 15 and MAX_SPEED is 15, for all values of speed except 15 (0 through 14), the integer fraction "speed/MAX_SPEED" will be truncated to 0 (because the result of the division is less than 1) before the remaining calculations are executed. Equation 11.5 will not work as desired because, for high speed values the product "(T/5)*speed" will exceed the largest value that can be stored with a 16-bit WORD variable ($2^{16} - 1 = 65{,}535$). This is called **overflow**. For all values of "speed" greater than 10, the product "(T/5)*speed" will result in overflow, throwing off the remaining calculations. In Equation 11.3, the order of calculations is chosen carefully so no truncation or overflow occurs.

The If statements in the While loop check to determine the range within which the Pot command variable "pot_val" falls. This allows the program to determine which button on the keypad is pressed. A separate calibration program is used to determine the appropriate values for the range limits. This program ("PWM_cal.bas" below) uses the same hardware as for the program above ("PWM.bas"), but here the LEDs are being used to graphically display the value returned by the Pot command. The three red LEDs blink individually and sequentially to indicate the number

of 100s, 10s, and 1s in the "pot_val" number. The green LED is flashed as a signal between each red LED's digit value display. If you had a liquid crystal display (LCD) in your design, it would be a simple matter to display the decimal number on the LCD for easy viewing. However, to use an LCD with the Pic Basic Pro command "Lcdout" requires 7 I/O pins, and many project designs will not have enough spare pins to drive the display. If you only have one or a few output pins available, blinking LEDs offer an alternative method to graphically display the values of numbers within your running program. In "PWM_cal.bas," since we have four LEDs, we used three different LEDs to indicate the different decimal places for the number. If you didn't have multiple LEDs in your design or if you only had one pin to spare, you could achieve the same result by blinking a single LED with pauses between each digit number display.

Through testing with the "PWM_cal.bas" program, using a "Pot" command scale value of 255, we found the following values for the three keys: 65 for the 1-key, 128 for the 4-key, and 189 for the *-key. That is why the following pot_val ranges where used in the "PWM.bas" program: 30 to 95 for the 1-key, 95 to 160 for the 4-key, and above 160 for the *-key. The nominal values (65, 128, and 189) fall in the middle of these ranges allowing for small random fluctuations due to temperature and connection resistance changes. Refer to the PicBasic Pro manual for details on how to select an appropriate value for the "Pot" command scale value. The value 255 is appropriate for the resistance and capacitance values we selected.

' **PWM_cal.bas**

```
' Displays the Pot values for the keypad buttons by blinking the upper three red LEDs.  Each
' LED is blinked individually to indicate the number of 100s, 10s, and 1s in the
' Pot value number.  The green LED is flashed once between each blinking red LED display.

' Identify and set the internal oscillator clock speed (required for the PIC16F88)
DEFINE OSC 8
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1

' Turn off the A/D converter (required for the PIC16F88)
ANSEL = 0

' Define variables, pin assignments, and constants
led0        Var   PORTB.0     ' LSB (bit 0) LED
led1        Var   PORTB.1     ' bit 1 LED
led2        Var   PORTB.2     ' bit 2 LED
led3        Var   PORTB.3     ' MSB (bit 3) LED
motor       Var   PORTA.1     ' PWM output pin to motor MOSFET gate
pot_pin     Var   PORTA.2     ' keypad pin for POT command
SCALE       Con   255         ' Pot statement scale factor
pot_val     Var   BYTE        ' value returned by POT command
i           Var   BYTE        ' loop variable
digs        Var   BYTE        ' digit number for each decimal place

' Initialize the I/O pins
TRISA = %11101              ' designate PORTA pins as inputs and output (RA1)
```

```
TRISB = %00000000          ' designate PORTB pins as outputs
PORTB = 0
Low motor                  ' make sure the motor remains off

' User speed change loop
enter:
        POT pot_pin, SCALE, pot_val

        ' Flash the LSB green LED and blink each of the upper 3 red LEDs to indicate the number of
        '   100s, 10s, and 1s in pot_val
        PORTB = 0

        High led0
        Pause 500
        Low led0
        Pause 100
        digs = pot_val / 100
        For i = 1 To digs
                High led3
                Pause 300
                Low led3
                Pause 300
        Next i

        pot_val = pot_val - digs*100
        High led0
        Pause 500
        Low led0
        Pause 100
        digs = pot_val / 10
        For i = 1 To digs
                High led2
                Pause 300
                Low led2
                Pause 300
        Next i

        digs = pot_val - digs*10
        High led0
        Pause 500
        Low led0
        Pause 100
        For i = 1 To digs
                High led1
                Pause 300
                Low led1
                Pause 300
        Next i
        Goto enter
```

```
' End of program (never reached)
End
```

(10) *Build and test the system*.

That is your job using the procedure in Section 11.5.

11.4 Troubleshooting and Design Improvements

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.**

If your PIC doesn't seem to be running properly (e.g., it resets when the motor start button is pressed), it might be because the Lab power supply voltage can be affected by current spikes (e.g., the voltage can drop suddenly, causing the PIC to reset). Because the motor is being switched on and off abruptly, 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. To help minimize these effects, you can **add capacitance (e.g., 0.1-1.0 μF) across the tabs of the motor** to help filter out spikes and noise from the commutation. You can also **add a 1 μF 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 μF with 1 μF or more). The TA can provide capacitors for testing.** Also, make sure the wires attached to the motor are soldered to the motor tabs to ensure solid and reliable connections. The motor wires should also be twisted together to limit potential electromagnetic interference (EMI) caused by the wire currents. You should also be careful to **limit ground loops** in your wiring, and **keep all wires as short as possible** (e.g., buy cutting and stripping wires to length) to minimize EMI. You can also build the circuit on a **separate breadboard that has a metal backing**, which adds capacitance to all connect points and helps reduce EMI.

Another alternative is to use separate power sources for the PIC circuit (e.g., the function generator) and the motor (e.g., a Lab power supply, 4 AA batteries in series for 6V, or a 9V battery with a 5V voltage regulator and 1 μF capacitor). This will help limit voltage fluctuations in the PIC circuit when the motor turns on and runs. Using a battery or AC adapter to power the whole system (the PIC circuit and the motor) is another alternative. In this case, a capacitor (e.g., 1μF or more) is required across the power and ground lines to help keep the output voltage stable. The TA will demonstrate the battery-power alternative.

If the motor has a difficult time starting at slow speed with the low-duty-cycle PWM signal, it can help to turn the motor on briefly (e.g., 0.5 s) with a non-PWM constant voltage to help get the motor starting, before starting the PWM signal. An alternative is to just give the motor a nudge manually by turning the shaft in the rotation direction.

If you can't get the Pot command stuff to work properly, an alternative is to wire up the buttons to separate inputs (with pull-up or pull-down resistors) to read them directly as digital inputs instead. The TA will demonstrate this alternative.

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

11.5 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_cal.bas" listed in Section 11.3. Save the file in a folder in your network file space.

(3) Follow the procedure in Section 9.4 of Lab 9 to store your program in a PIC microcontroller that you can insert into your circuit.

(4) Build the circuit shown in Figure 11.5 and insert the PIC programmed with "PWM_cal." You can omit the motor driver circuit for now because it is not used in the calibration program.

(5) Report the nominal Pot values displayed for your program for each of the active keypad buttons. Be sure to hold each button down long enough (for 2 green LED blinks) to start the red LED sequence.

pot_val for the 1-key: 100s: _____ 10s: _____ 1s: _____ value: _____

pot_val for the 4-key: 100s: _____ 10s: _____ 1s: _____ value: _____

pot_val for the *-key: 100s: _____ 10s: _____ 1s: _____ value: _____

(6) Repeat Steps 2 and 3 for the "PWM.bas" program, replacing the "PWM_cal" program on your PIC. **Modify the "pot_val" ranges in the PWM.bas "speed change loop" *If* statements, if necessary based on the values you found in Step 5.** Add the motor driver circuit to your board if you haven't done so already. Insert the reprogrammed PIC into your circuit.

(7) See Section 11.4 if your circuit is assembled correctly but does not work properly. One thing worth checking is whether or not the motor PWM signal is working as expected. To do this, disconnect the transistor and look at the PIC output signal on the oscilloscope as the speed is changed.

(8) Show your functioning circuit to your TA so he or she can verify it is working.

LAB 11 QUESTIONS

Group: ____ Names: _____ _____

_____ _____

(1) Did your circuit work the first time, without modifications? If not, what things did you try from Section 11.4? Which things worked, and why do you think they worked?

(2) Explain in detail how you think the Pot command works.

(3) In the PWM.bas program, we used 30,000 microseconds for the PWM period. What frequency f (in Hz) does this correspond to?

(4) How would the motor respond to a very low (close to 0%) duty cycle PWM signal?

How would changing the PWM signal frequency f (i.e., making it much lower or much higher) change the motor response?

(5) 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)?

(6) In PicBasic Pro, to what values would the following expressions evaluate? Hint: PicBasic Pro uses integer division and performs one operation at a time.

a) 2 / 3 * 4

b) 2 * 4 / 3