# Laboratory 14

*Pulse-Width-Modulation Motor Speed Control with an Arduino Uno (modified from lab text by Alciatore)*

## Required Components:

- 1x Arduino microcontroller
- 3x 0.1 μF 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 1kΩ resistors
- 3x red LEDs
- 1x green LED
- 4x 330Ω resistors

## 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 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:

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

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.

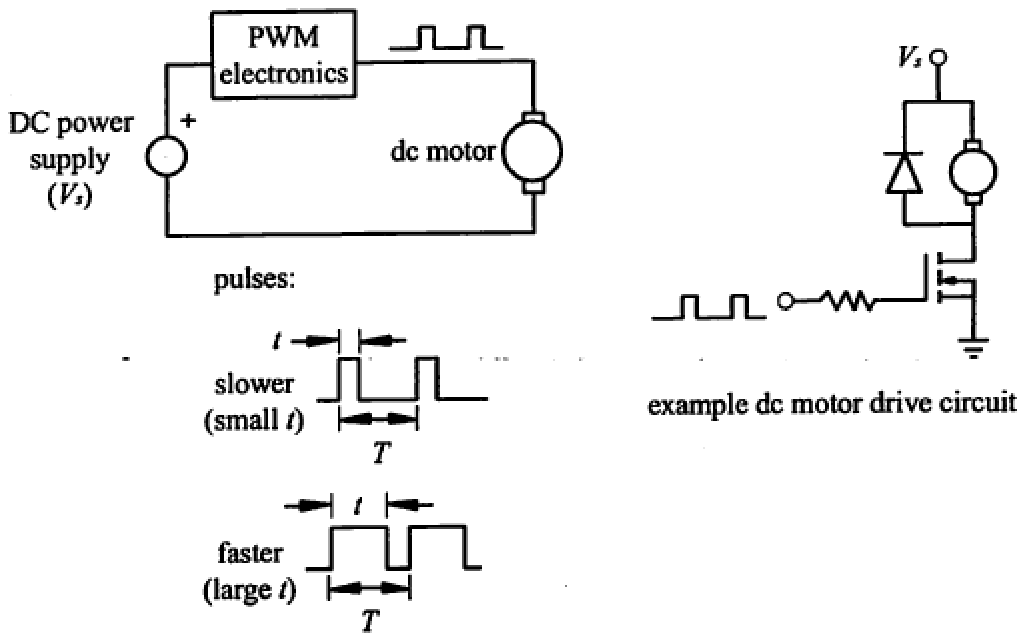


Figure 14.1 Pulse-width Modulation (PWM)

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.

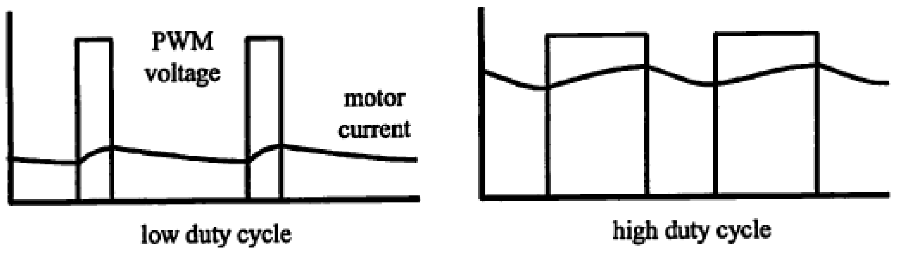


Figure 14.2 PWM voltage and motor current

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. 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.

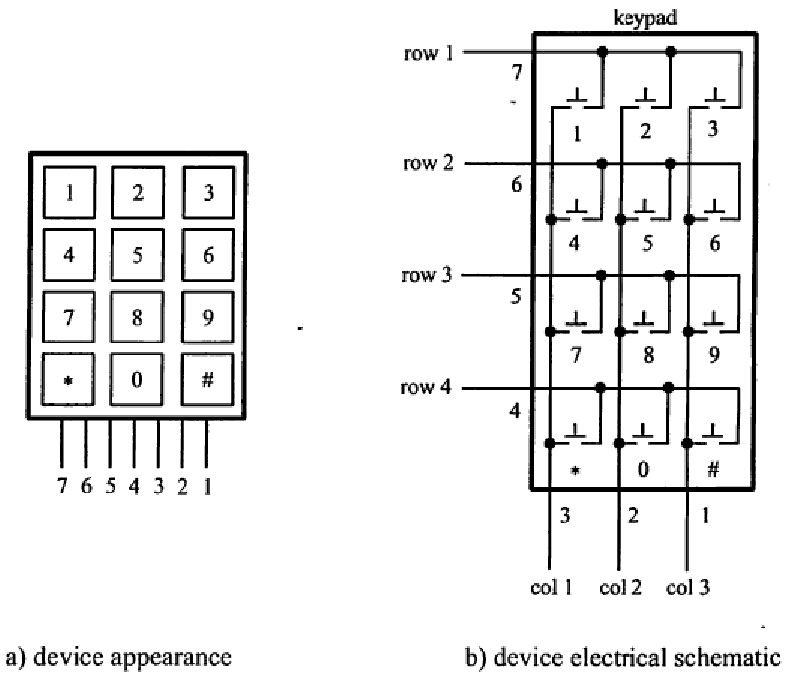


Figure 14.3 Standard 12-key numeric keypad

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 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 an Arduino 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:
   - 1 for the 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.
4. *Select an appropriate microcontroller.*
   For this problem, we will use the Arduino whose 12 lines of digital I/O provide and 6 lines of analog in are more than enough capability for our I/O requirements.
5. *Identify necessary interface circuits.*
   You will learn how to use a numeric keypad by poling 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 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 the Arduino IDE.
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 powered by a connection to 5V, this can be permanent (as show) or by connecting pin 3 and providing power only you wish to read the keyboard (to save power). The keypad is wired such that different resistors are in series with the 2k$\Omega$ pull up resistor (forming different voltage dividors) depending upon which button is held down (l0k$\Omega$ for the 1-key, 2k$\Omega$ for the 4-key, and 1k$\Omega$ for the *-key). The analog input pin A0 reads the voltage divided value. The LEDs are attached to the four lowest order bits of PORTB (pins 8, 9, 10, 11). 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 pin 2.

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

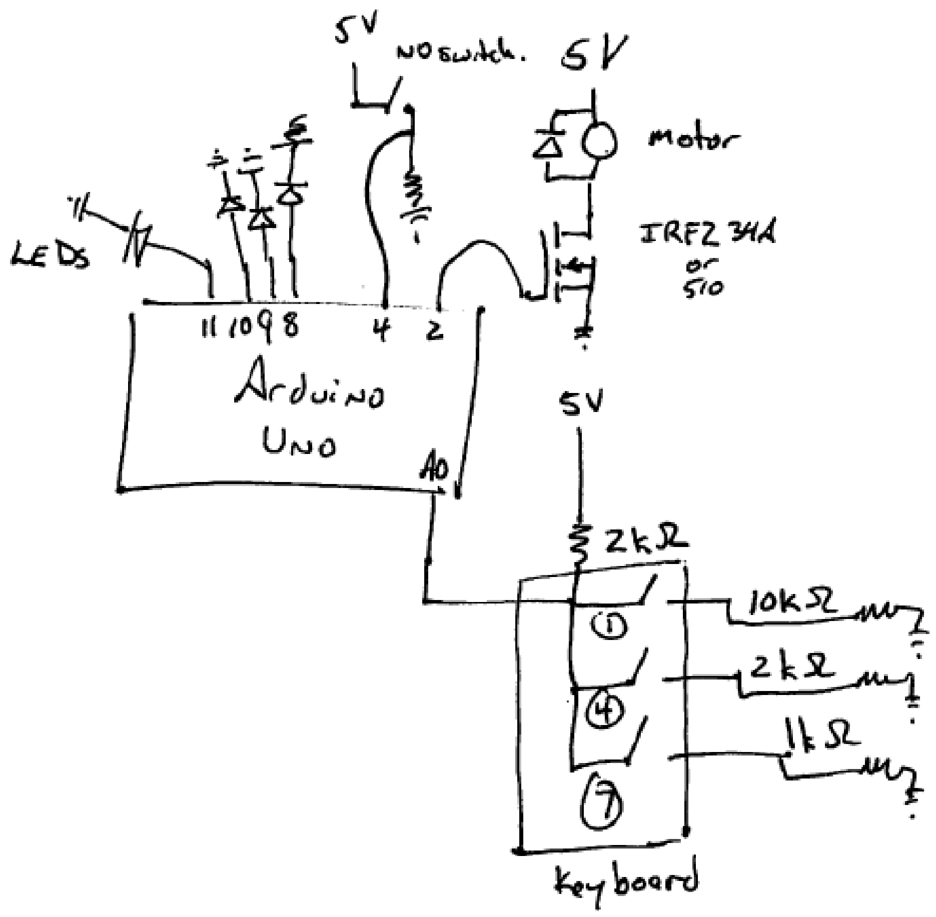


Figure 14.4 Complete schematic showing all components and connections. Note that LEDs are arranged from least significant bit (LSB) to most (MSB).

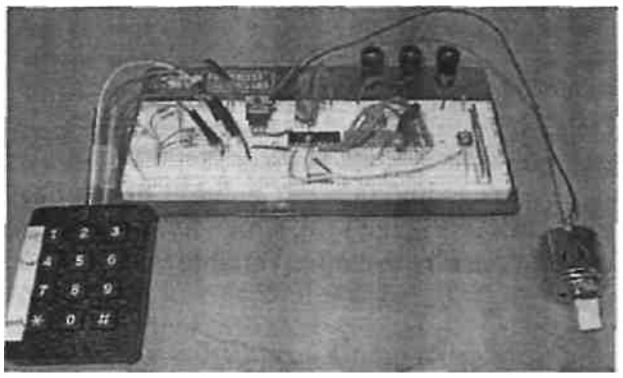


Figure 14.5 Photograph of the actual device.

8. *Draw a program flowchart.*

Figure 14.7 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.

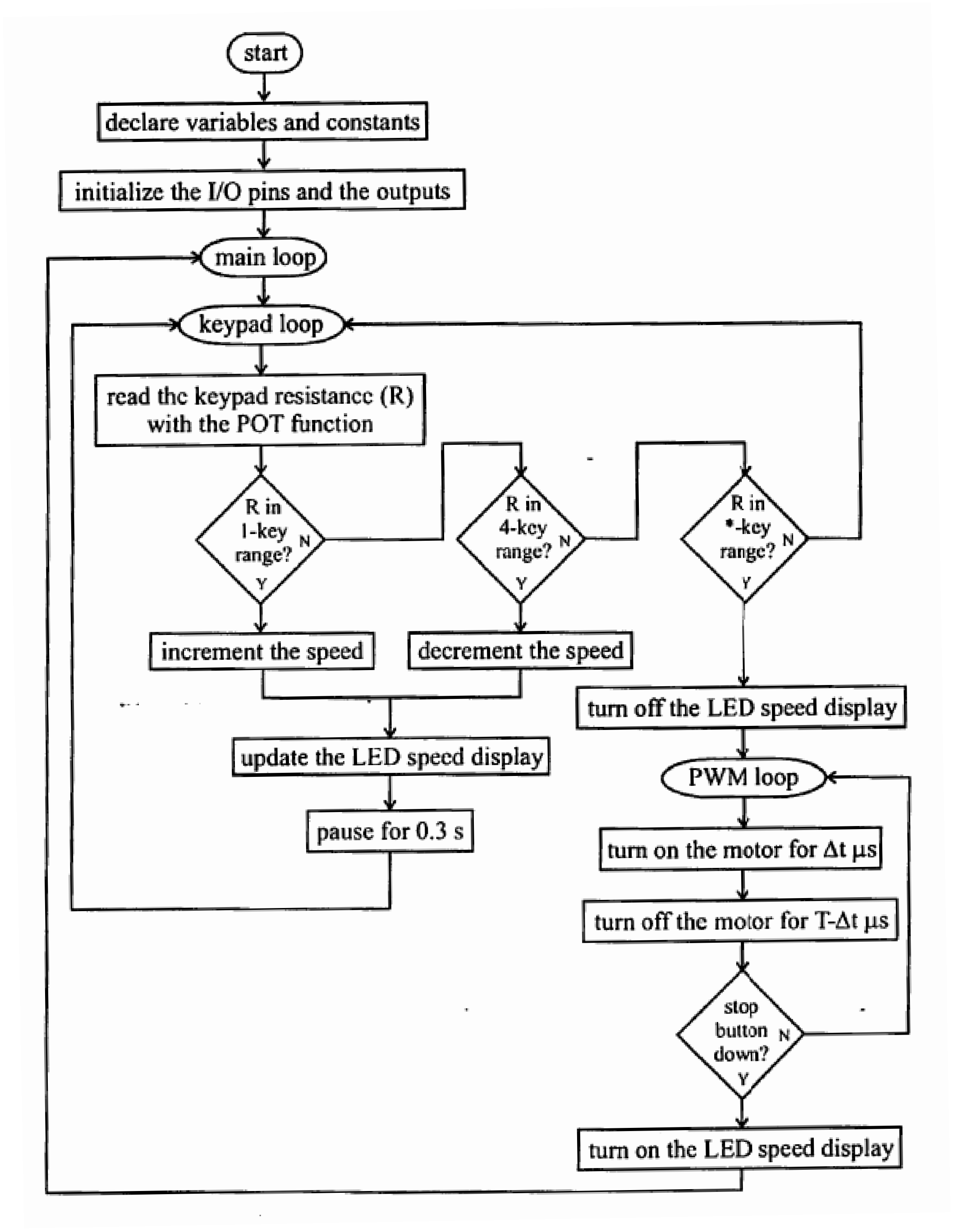
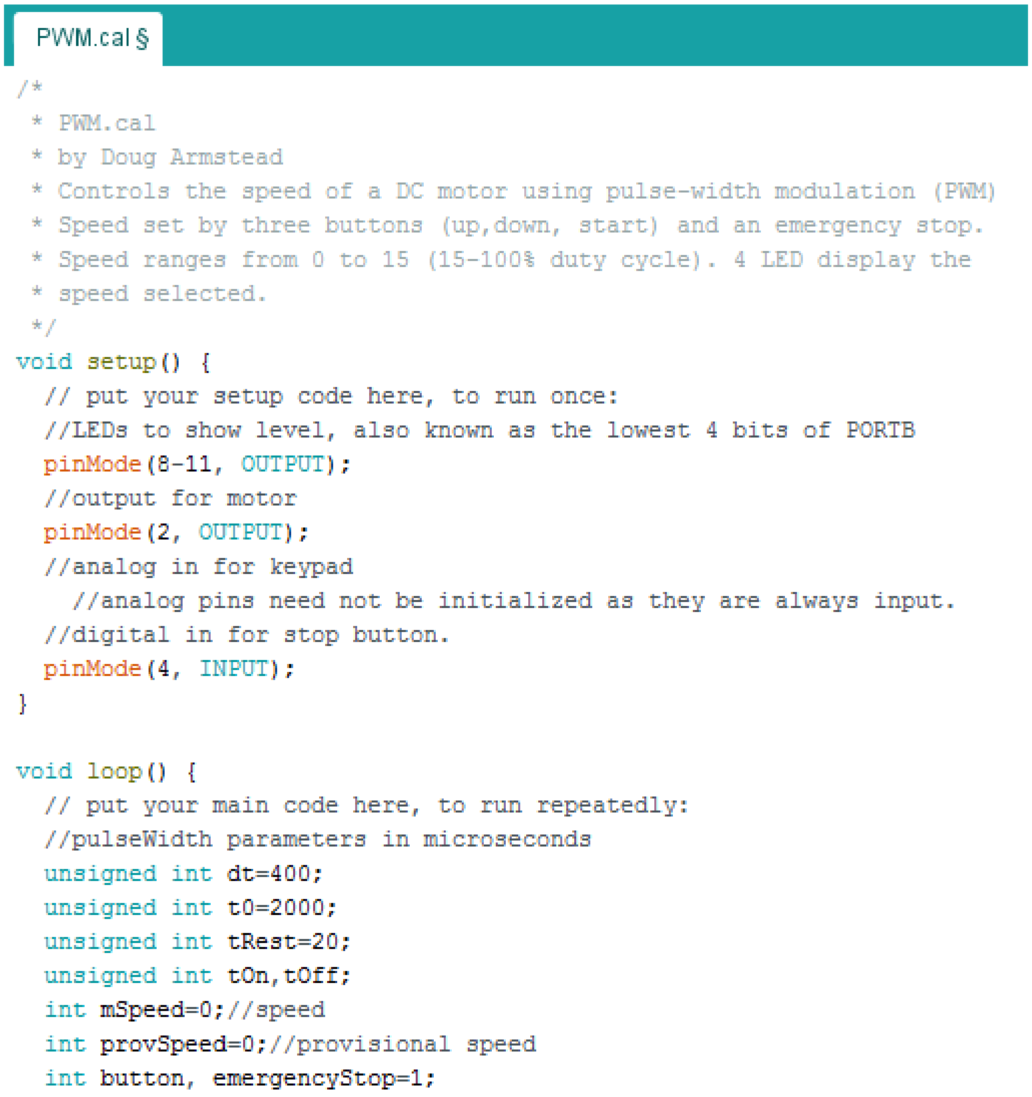

start
declare variables and constants
initialize the I/O pins and the outputs
main loop
keypad loop
read the keypad resistance (R) with the POT function
R in 1-key range?
N
Y
R in 4-key range?
N
Y
R in *-key range?
N
Y
increment the speed
decrement the speed
turn off the LED speed display
update the LED speed display
PWM loop
pause for 0.3 s
turn on the motor for Δt μs
turn off the motor for T-Δt μs
stop button down?
N
Y
turn on the LED speed display

Figure 14.6 One possible complete Program Flowchart

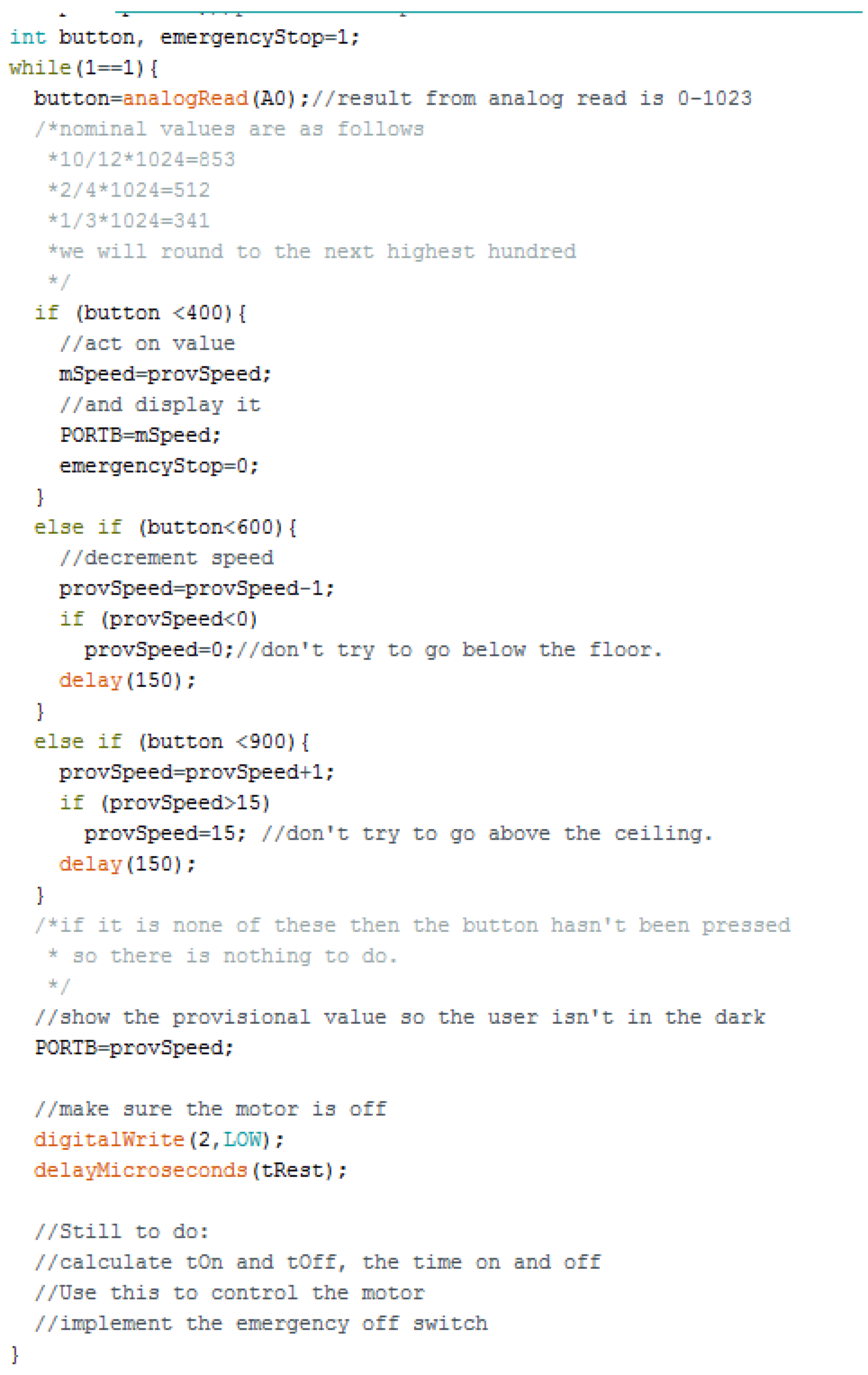
9. *Write the code.*
   The code ("PWM.cal") corresponding to the flowchart shown in Figure 14.7 using the hardware illustrated in Figure 14.4 follows. The code is comented throughout with remarks so it should be self-explanatory. Whenever you write programs, you should always include copious remarks 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.**



PWM.cal §

```
/*
 * PWM.cal
 * by Doug Armstead
 * Controls the speed of a DC motor using pulse-width modulation (PWM)
 * Speed set by three buttons (up,down, start) and an emergency stop.
 * Speed ranges from 0 to 15 (15-100% duty cycle). 4 LED display the
 * speed selected.
 */
void setup() {
  // put your setup code here, to run once:
  //LEDs to show level, also known as the lowest 4 bits of PORTB
  pinMode(8-11, OUTPUT);
  //output for motor
  pinMode(2, OUTPUT);
  //analog in for keypad
    //analog pins need not be initialized as they are always input.
  //digital in for stop button.
  pinMode(4, INPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  //pulseWidth parameters in microseconds
  unsigned int dt=400;
  unsigned int t0=2000;
  unsigned int tRest=20;
  unsigned int tOn,tOff;
  int mSpeed=0;//speed
  int provSpeed=0;//provisional speed
  int button, emergencyStop=1;
```



```
int button, emergencyStop=1;
while(1==1){
  button=analogRead(A0);//result from analog read is 0-1023
  /*nominal values are as follows
   *10/12*1024=853
   *2/4*1024=512
   *1/3*1024=341
   *we will round to the next highest hundred
   */
  if (button <400){
    //act on value
    mSpeed=provSpeed;
    //and display it
    PORTB=mSpeed;
    emergencyStop=0;
  }
  else if (button<600){
    //decrement speed
    provSpeed=provSpeed-1;
    if (provSpeed<0)
      provSpeed=0;//don't try to go below the floor.
    delay(150);
  }
  else if (button <900){
    provSpeed=provSpeed+1;
    if (provSpeed>15)
      provSpeed=15; //don't try to go above the ceiling.
    delay(150);
  }
  /*if it is none of these then the button hasn't been pressed
   * so there is nothing to do.
   */
  //show the provisional value so the user isn't in the dark
  PORTB=provSpeed;

  //make sure the motor is off
  digitalWrite(2,LOW);
  delayMicroseconds(tRest);

  //Still to do:
  //calculate tOn and tOff, the time on and off
  //Use this to control the motor
  //implement the emergency off switch
}
```

The variable "mSpeed" 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 have defined 3 constants, _T0 which corresponds to the minimum time (in µs) for the motor to be run in a duty cycle, _DT which is the amount that gets progressively added to the on time (in µs), and _TREST the amount that the motor is certainly off in a cycle (in ms). As a result the on-time for the motor is

$$t_{on} = T0 + DT * speed$$

(in µs) and

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

(in µs) where the 1000 is for the number of µs in a ms.

Note that the period is constant

$$T = T_{on} + T_{off} = T0 + TREST * 1000 + DT * 15$$

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 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)**. You can also 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. 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 the Arduino IDE (either down load it to your computer or use the online version) to create a project sketch called "PWM.cal" listed in Section 14.3. Save the file in a folder in your network file space.
3. Load the sketch onto the microcontroller.
4. Build the circuit shown in Figure 14.4 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. **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.

## LAB 14 QUESTIONS

Group: ____ Names: _________________________ __________________________

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

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

3. 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 lower or higher) change the motor response?

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)?