DC Motor Speed using PWM and ADC control

Performance Check

_____ (5) Check off 1 (IO and debugger)

_____ (45) Check off 2 (flashing LED)

_____ (60) Check off 3 (Speed profile)

_____ (70) Check off 4 (all code working together, ADC adjust DC)

Objectives:

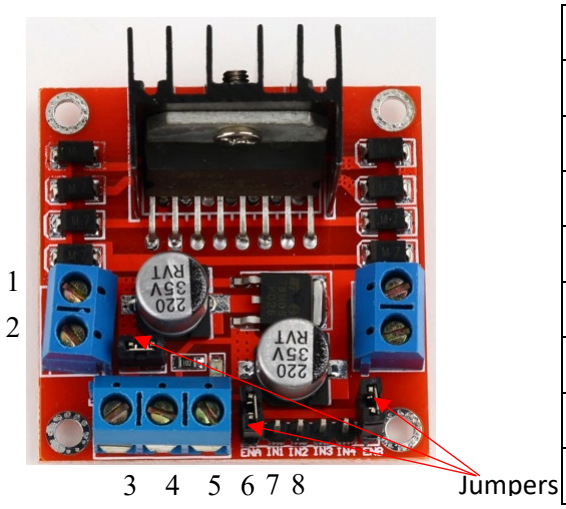
1. To write your own milli-second delay function
2. To write code using 16 bit timer to control a DC motor's speed with a specific speed profile.
3. To write an ADC code that utilizes ADC value at a specific channel to control the DC motor's speed.

Introduction

In your lab kit, there is an H-bridge IC that controls a DC motor's speed and direction. Instruction for connecting to the IC should be included with the kit, or you can find instruction by a search online. To control the DC motor, you need to also have additional power supply, which also comes in your kit. For those using the EEinaBox kit, you may use the power supply in the lab to supply power to your H-bridge.

If you have L298N similar to the picture below, see the suggested hardware connections. Please note it is highly recommended that you solder two wires to the DC motor before the lab. Soldering Iron is available in the project lab.

1. Connect the L298N and the DC motor according to the diagram below:



PIN	Hardware connection
1	DC Motor
2	DC Motor
3	External Power Supply 6.4 V
4	Ground (both controller and external power)
5	Open
6	PWM output from controller
7	Direction from controller
8	Direction from controller

Figure 1: L298N H-Bridge Motor Driver Hardware connections

a. Connect PIN 7 to Ground and PIN8 to 5V.
b. Remove the jumper in PIN 6, this is where you connect the PWM output to control the speed.

This is one of the links that provide a thorough description of this Motor Driver Module and its interface with Arduino.

If you have the L293D IC, you may review the wiring instructions from the ELEGOO user manual starting at p200. https://m.media-amazon.com/images/I/D1oC-c3G5TS.pdf . This link also provides some useful information on how to connect the chip to the motor. https://lastminuteengineers.com/l293d-dc-motor-arduino-tutorial/

This lab involves developing a simple speed control algorithm for the ATmega2560 in which the speed of a DC motor is controlled over a specified speed profile. A Speed profile defines how the speed of the motor must change with time. In this lab, the speed profile code should be written so that it can be changed easily. Typically, DC servomotors' speed is managed using closed-loop control. The motor shaft's actual speed is measured, and a signal proportional to the speed fed back into the control algorithm to be compared to the set (desired) speed. Any difference is then adjusted to a minimum (zero) by the action of the control algorithm. In this lab, we use open-loop control. We rely on the accuracy of the relationship between the voltage applied to the motor and the actual motor speed, i.e., there is no speed information fed back – hence open (no feedback) loop.

Suggestion

Write a small section of code, then test and debug before the next section. You will make a millisecond delay function, which will be used for delays in other functions, including the speed profile function. This is to replace the _delay_ms() function that we used before this lab. Thus, make sure this function is working properly before attempting on other functions. There should be two secondary modules (Timer and ADC). Initialization of IO for the timers should reside in the Timer module. The initialization of ADC inputs should reside in the ADC module.

There is less instruction for labs. Instruction will include mostly requirements for check off. Thus, it is imperative to plan ahead, consider what to do in each module, and what function you may have. Identify all the IO needed and then write it down. Do not attempt to write any code until you have the IO completed. Then you may use comments to write down the steps you need to complete specific tasks.

IO connections

Read the specifications below, then complete the IO connections table below:

Inputs Hardware	**Controller In**	**Outputs**	**Controller Out**
Start profile Push Button		Test LED / mode LED	
Stop PWM output		DC motor	Timer1 OC1A, PORT___.___
Potentiometer		Internal 1ms Delay	Timer 0
Potentiometer change PWM mode switch			

Specifications (what you need to check off)

1. Connect your input and outputs. Create a new project with the debugger (reference lab 3) and initial set up of the IO the project. Run the debugger to verify IO connections and set up are correct.

Instructor Check off 1

a. Show completed IO Table.
b. Demonstrate pushbuttons work with the Debugger

2. An LED flashing every 500 ms. 500ms ON, 500ms OFF. There should be a function called delay_ms (500) in the code, where 500 is the delay time passed to this function to

produce the delay. If a one is passed to it, it will delay for 1ms. The function should use the polling method. This function must be your own creation. Use of _delay_ms() standard delay is prohibited. Use Timer0 for this task. Once the timer is initialized, use the Debugger to verify the register values of the timer.

Instructor Check off 2

a. Use the Debugger to verify the Timer 0 register values
b. LED flashing on and off every 500ms
c. Verify code for 1ms by student explaining it

3. When a push button is pressed, Timer1 OC1A produces a PWM signal at **3900Hz** to control the DC-motor (determine the timer mode (must use either 9 bit or 10 bit, exclude 8 bit) and the prescaler you like to use). The code for the speed profile should be adaptable. The speed profile starts at a X%, then ramp up to a Y% within a specified time (Z). An example of a speed profile can be: 1) the motor starts with a 10% (X) duty cycle; it slowly goes up to 90% (Y) in 8 seconds (Z), using eight equal distance steps. In your code, all these data should be easily changed using variables instead of formula. **During check off, a different profile may be given.** After the speed profile, to stop the motor, the stop PWM output push button is pressed, all the PWM output is removed, and the output pin OC1A is returned to a normal IO pin. It is best to test the operation using the oscilloscope before you connect to the dc motor.

   Below are some hints for this exercise

   a. Write a function that changes the duty cycle only without the speed profile. Verify the frequency and duty cycle before continuing with the rest of the code.
   b. At home, it can be difficult to verify the operation. One way to test this is to connect to a LED with a current limiting resistor. As the duty cycle increase, you should see the LED gets brighter. If you connect the DC-motor, you should hear a slight humming sound even though the motor does not move. From testing, most of the DC motor from our kits start to move around 55% to 60%. Please note, you need to have an external power supply to provide enough current through the H-bridge to move the DC motor. Thus, using the LED as a quick test was suggested.
   c. Your PWM signal (OC1A) controls the enable pins. Reference beginning of this lab on how to connect to the DC motor with the H-bridge.

d. Set up a separate function to perform the speed profile. This function has the following function prototype:
    i. void ramp_up_delay_n_steps (uint8_t start, uint8_t end, uint16_t ms_time, uint8_t num_steps)
    ii. Where:start = initial % duty cycle
    end = final % duty cycle
    ms_time = duration of ramping
    num_steps = number of increments used to increase or decrease the duty cycle.

    Part of the pseudo-code:
        a. calculate change in duty cycle / step = duty_cycle_change
        b. convert the duty-cycle/step
        c. calculate time for each step = step_time
        d. Loop for number of steps
        e. make sure OCR1A number is within the BOTTOM and TOP number
        f. Change OCR1A
        g. Delay step_time
        h. return
e. To turn off PWM output, change the compare output mode to Normal port operation.
f. When changing the status register values, make sure not to erase what you have set at initialization.
g. The simulation doesn't work well in this case, but you may need to make sure the calculations are correct and that the preset values are still there. Comment out the delays, leaving the code where you set the values to ensure proper operations of the timer.
h. Don't write the complete code at one time. Test out the PWM function first, then leave that alone. Test out the 1ms timer, then leave it alone. Call those functions that you know works.
i. Use the debugger to make sure your PWM initialization function works correctly.
j. The 500ms delay can cause the start pushbutton to be not read. Hold the pushbutton down until the flashing LED no longer flash. This means it is in the PWM ramp function. This flashing LED now acts as your diagnostic lamp,

indicating the portion of code you are running. Also, show you when your pushbutton is read.

Instructor Check off 3
a. Pushbutton start speed profile. Instructor announcement the starting duty cycle, the ending duty cycle (> 70) number of steps (<10), and total time during check off. Review using oscilloscope b. Connection to motor, same profile demonstrated using the dc motor c. Press the PWM stop pushbutton to stop the motor from running d. Explain the code

4. For the next check off, use ADC to change the duty cycle of the PWM signal. Since we have a delay_ms(500) that can interfere with the operation of the ADC. Therefore, use a switch (not pushbuttons) to enter into the PWM adjust mode. When the switches are activated, the corresponding ADC channel changes the duty cycle. When the switch is not activated, the 500ms LED flashes. This allows you to notice the mode of operation.
    a. The ADC conversion should use 10-bit conversion. You may choose to either poll the ADIF flag or the ADSC flag for the conversion. A potentiometer is connected to the channel that you choose (reference IO table at the beginning of this lab). You may use a 10kΩ potentiometer; the voltage at ADC0 will control the duty cycle from OC1A. When the voltage is at 0V, the duty cycle is at 0%; when the voltage is maximum, the duty cycle is 100%.

    Hint:

    b. Prepare an ADC module where you set up the ADC initialization for only the ADC pins and a function to convert ADC values at different channels. Use debugger to verify the initialization operates correctly.
    c. The potentiometers should be connected between 5V and ground, with the wiper arm connected to the ADC channel in your io table.
    d. Initialize the ADC ports, be sure to **disable** the pull-up resistors.
    e. A function to read ADC values using polling. This function must start an ADC conversion and then wait until the **ADIF** flag is Set, reset the ADIF flag, and return the 10bit unsigned integer to change the duty cycle. (conversion is needed in your code)
    f. Read ADC value from the channel using single-ended input with range 0 to 5V, and 10-bit resolution (Pre-scaler = 128).

g. Test one item at a time. You may use multiple LEDs to test your ADC functionality before using it to control the duty cycle.

5. All the functions combined should work properly; this is the last check off. This encourages you to work on each code separately, then combine them at the end. Make sure to allow time to troubleshoot issues at this stage.

Instructor Check off 4
a. Turn on ADC adjust PWM mode, use the potentiometer to change the duty cycle. Can monitor the duty cycle from the oscilloscope, an LED, or from a dc motor. b. Turn off the ADC adjust PWM mode, Test LED / Mode LED flashes every 500ms