ECE 3720

Microcomputer Interfacing Laboratory

Section 8

Timothy Driscoll

Date Performed: October / 31 / 2018

Lab 8

ABSTRACT:

A lab designed to show how the PIC32MX150F128D microcontroller’s output compare peripheral could be used to generate a PWM signal to drive a motor.

**INTRODUCTION:**

In this lab, we created a new project that showed how the PIC32MX150F128D microcontroller’s output compare could be used to generate PWM signals. Some microcontrollers have dedicated PWM peripherals but the PIC32 does not. In order to circumnavigate this problem, the output compare peripheral was used to generate PWM signals. The objectives of this lab were to set up the microcontroller’s output compare module into PWM mode. This PWM signal was then used to drive a motor using a motor driver IC (L293DNE). The external interrupt (INT0) was then used to modify the duty cycle. The duty cycle determines the percent of time the signal is high when compared to the total time to complete a cycle. The duty cycle will change from 0% to 100% and wrap around back to 0% after reaching 100%. This change in the duty cycle will effectively increase the speed at which the motor is operating at.

**EXPERIMENTAL PROCEDURES:**

In this lab, we started by unpacking the microcontroller and attaching the chip kit in the proper orientation to ensure that the pins wouldn’t have a flipped result. Then we could start wiring the circuit to complete the project. This circuit is represented by figure 1 below. The circuit includes the three necessary external hardware components, the motor driver IC, the motor and the push button. When the circuit was designed in lab the DIO on the NI-ELVIS board was used to represent the button and simplify the circuit. When a 1 was written to the given DIO pin it represented a push of the button. The button (DIO) was wired to pin 37 (register B7) specifically because INT0 is the only hard-mapped external interrupt on the microcontroller and it is connected to pin 37. Pin 13 (register B13) was then wired to the input of the motor driver. Register B13 was used because it represented a pin that an output compare from the microcontroller could be mapped to using PPSOutput. Pin 13 was the output that we used to represent the PWM signal. Output four from the motor driver was then wired to the motor which corresponded to where we inputted the PWM signal (Input 4). It was also critical to wire power to the Enable 3, 4 pin on the motor driver as well as include a diode in the implementation of the motor as seen below. After the completion of the circuit wiring, we created a new project using the MPLAB X IDE software. After creating the project, we added a new C type source file where we wrote the code in order to have the microcontroller function properly. The code can be viewed below in the figures and tables section. The flow of this code starts with the delay function that was used in previous labs. In this lab the delay function is implemented in the ISR for this lab to create a brief pause allowing us to see a transition in duty cycles. The next block of code was the interrupt service routine for the INT0 external interrupt. This block of code tells the microcontroller what to do when the INT0 interrupt has occurred. The interrupt service routine was designed to change the duty cycle of the PWM signal and effectively change the speed of the motor. Now looking at the main function INTEnableSystemMultiVectoredInt() had to be included to tell the microcontroller to handle interrupts. TRISxbits was then used to initialize INT0 as an input and the PWM signal as an output. INT0 was then properly initialized and the PPSOutput function was used to map Output Compare 5 to register B13. The output compare was then properly initialized, and the timer was setup with the desired period. There was nothing needed in the while loop for this lab, but it was included to have the program continuously run while the microcontroller had power.

**RESULTS and DISCUSSION:**

In this lab, we were successfully able to get the desired output from the project and achieve all the desired goals that were described in the slides. This involved having the motor speed increase each time a 1 was written to the DIO using the NI-ELVIS. The speed was designed to increase before wrapping back to the starting speed. The different duty cycles that were run through were 0%, 25%, 50%, 75%, 100% and then the next button push would wrap back around to 0%. For this lab to work there were three main things that we needed to implement. These main things included setting up an external interrupt, setting up an output compare to generate a PWM signal and lastly implementing the motor driver IC. Since there is no hard mapped output compare on the microcontroller we had to use the PPSOutput function. The PPSOutput function excepted three different arguments which included the group, function and pin. The pin and function were easily identified to be OC5 and B13 respectively. Where OC5 was the desired output compare and B13 was the register corresponding to the pin we wanted the interrupt to be mapped to. The last parameter that needed to be imputed was the group, this value could be found by looking at far right side of table 11-1 in the PIC32 Datasheet and determining which group OC5 corresponded to (1-4). After mapping the output compare peripheral all the parameters could be set for both INT0 and OC5. Starting with INT0, the external interrupt was triggered on a falling edge. In order to set this up we used the INTCON register and set it to 1. There were three other registers that had to be setup in order to properly initialize the interrupt. These registers included the IEC, IFS and IPC. The IEC register was set to a one in order to enable the interrupt. The IFS register was set to a zero to indicate that the interrupt has not yet occurred. Lastly the IPC was initialized to a one to give this interrupt the highest priority, this was done because we only had one external interrupt. Next, to initialize the output compare OC5 we had to set a couple of different bits in the OC5CON register. We started by setting the ON bit to 0 in order to disable the output compare while we set it up. The next bit that was set was the OC32 bit, it was set to a 0 because our comparisons were made to the 16 bit timer source. The next bits that were set were the OCM bits. The OCM bits were set to 6 (110) in order to turn on the PWM mode but disable the fault pin. The OCTSEL bit was then set to a 0 or 1 in order to tell the microcontroller what the clock source for the output compare was. I set OCTSEL to a 1 meaning that timer3 was the clock source. Lastly after completing this setup the ON bit could be set to 1 in order to enable the output compare. After initializing the output compare the timer period had to be set and the duty cycle was initialized before the code ran into the infinite while loop. Using the T3CON register the ON bit of timer 3 was set to 0 to disable the timer prior to setup. The period of timer three was then set using the PR3 register and the duty cycle was initialized to 0 using the OC5RS register. The T3CON register’s ON bit could then be set to 1, which enabled the timer and completed the setup. After the external interrupt was triggered it sets a flag which tells the microcontroller that an interrupt has occurred. The microcontroller then saves the main state and looks up to the ISR corresponding to the specific interrupt. In the ISR the there was simple if else statement to set the duty cycle to 0 if it had already reached 100% otherwise the duty cycle was increased by 25. The delay function was then called, and the interrupt flag was cleared. The delay was needed because once the duty cycle hit 50% it would automatically step through 75% and 100%. The delay was used so we could see when the duty cycle was increasing. Lastly, in order to implement the motor driver IC there were a few key points that needed to be accounted for which included suppling a logic high to the needed enable pin, grounding all for grounds and using the matching input an output pins. In this lab the only issues that I ran into were small coding errors that included things like forgetting to add a semicolon and accidently setting the period for timer 2 and not the desired timer 3.

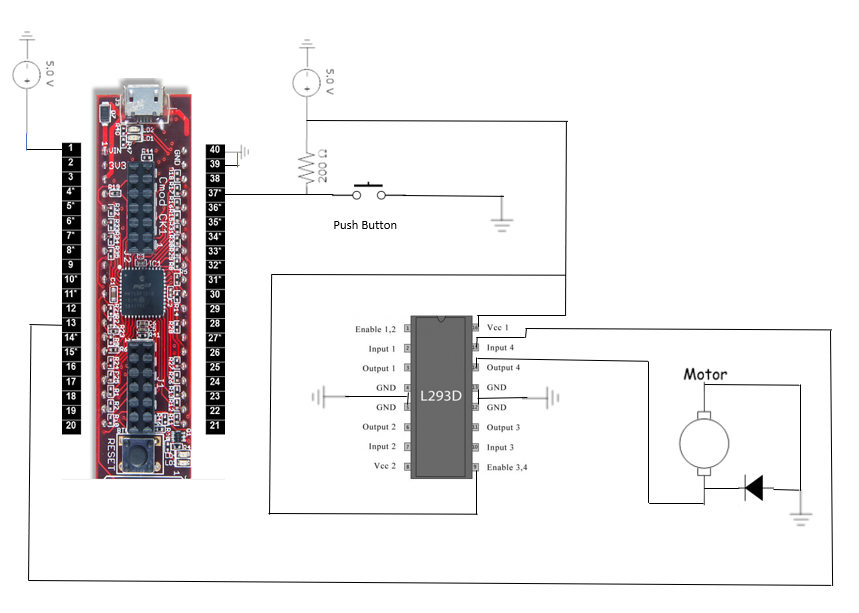
**CONCLUSSION:**

We have come to the conclusion that in this lab there were some key procedural points that needed to be learned in order to achieve all the presented goals. To start we largest thing we needed to learn was how we can implement a PWM signal without a dedicated PWM peripheral. Since the PIC32 doesn’t include a dedicated PWM peripheral the output compare peripheral can be set up to produce a PWM signal. The keys to setting up the output compare correctly to implement PWM is described in the section above. The next thing that we needed to learn was how to properly implement the motor driver IC, which required an understanding of the hardware and the functionality of the IC. The key points in setting up the IC can also be seen in the section above. When adding these new ideas to some of the ideas learned in previous weeks, we were able to achieve the desired results. For the success of this lab it was critical that we were able to set up external interrupts, use the PPSInput/ PPSOutput functions, setup timers and write interrupt service routines. By working through the lab, I was able to determine all the necessary procedural points to understand and achieve the desired outputs for the lab.

**REFERENCES:**

Clemson University’s ECE 372 Lab 8 PowerPoint.

**FIGURES AND TABLES:**

****

**Figure 1: Wiring for lab 8 (Pin connections described in experimental procedures)**

**CODE:**

#include <plib.h>  
  
delay(){  
    int i, j;  
    for(i = 0; i < 500; i++)  
        for(j = 0; j < 500; j++);  
}  
  
//Interrupt Function based off of button push (DIO) to increase speed  
void \_\_ISR(3) lab8\_func(void){  
    //check to see if wrapping needs to occur  
    if(OC5RS == 100)  
    {  
        OC5RS = 0;  
    }  
    else  
    {  
        OC5RS = OC5RS + 25;  
    }  
    delay();  
    // Clear the flags  
    IFS0bits.INT0IF = 0;  
}  
  
main()  
{  
    INTEnableSystemMultiVectoredInt();  
     
    TRISB = 0x00; //Set all pins to output.  
    //DIO (button) is an input  
    TRISBbits.TRISB7 =1;  
    //set pwm signal to output  
    TRISBbits.TRISB13 = 0; //Step is not needed because of initial TRIS  
    ANSELB = 0x00;  
     
    //Initialize the interrupt  
    INTCONbits.INT0EP  = 1;  
    IFS0bits.INT0IF = 0;  
    IPC0bits.INT0IP = 1;  
    IEC0bits.INT0IE = 1;  
     
    //Set up the output compare  
    PPSOutput(3,RPB13,OC5);  
     
    OC5CONbits.ON = 0; //Turn off to setup  
    OC5CONbits.OC32 = 0;  
    OC5CONbits.OCM = 6;  
    OC5CONbits.OCTSEL = 1; //Using timer 3  
    OC5CONbits.ON = 1;  
     
    T3CONbits.ON = 0;  
    PR3 = 100;  
    OC5RS = 0;  
    T3CONbits.ON = 1;  
     
    while(1){}                     //Run Continuously  
   
}