Mark Mersnik

Jasdeep Sihota

Lab 4: Encoder and Motor Speed Control

Achieved Results

There were two parts to this lab. In part 1 of the lab, we implemented an external interrupt using the GPIO pins on the board. The interrupt was triggered with a rotary optical encoder and during the interrupt service routine, a counter was incremented or decremented depending on whether the encoder turned clockwise or counterclockwise. Part 1 of the lab was demoed to the lab instructor, so it will not be explained in detail in this lab report. This lab report will explain part 2 of the lab in detail as it was completed, but not demoed. Part 2 of this lab experiment combined motor control from part 2 of lab 3 and the interrupt controlled encoder from part 1 of this lab. There were five steps to part 2. In part 2 of this lab, an external and a timer based interrupt were used to calculate the speed of the motor and a POT was used to control the motor speed. The speed was displayed on the 7 segment display after it was calculated. Step 1 was to find a lowpass digital filter with a sampling frequency of 200Hz and a bandwidth of 1Hz using Matlab. This filter was implemented in the interrupt service routine for the timer based interrupt. Step 2 was to incorporate the 7-segment display and the digital filter equation so that the speed of the motor would be displayed on the 7-segment display. This was done by having both the digital filter and the 7-segment display as two separate routines. After the digital filter was used, the variable holding the speed was returned to the routine which displayed the value on the 7-segment display. Step 3 was to incorporate the motor speed control software from lab 3 as a

third routine so that the motor speed could be controlled by the potentiometer. The fourth step to the lab was to implement the encoder from part 1 of this lab and the quadrature signal decoding logic to display the speed on the seven segment display. As soon as the speed was calculated, it was displayed, without using the digital filter. The last step to part 2 was to send the speed calculated from the quadrature signal decoding logic through the digital filter, and then display the final speed on the 7-segment display.

Diagram

Figure 1 below shows a high level diagram of the software for task 2 of this lab. The program stayed in a while loop where the motor speed was displayed on the 7 segment display and the motor speed could be controlled by the POT until an interrupt was triggered. When the external interrupt was triggered by observing a change in the motor speed, the quadrature signal decoding logic was used to calculate the speed. This calculated speed was then displayed on the 7-segment display. This means the software exited the ISR of the external interrupt and returned the variable holding the speed to the while loop where the speed is displayed on the 7-segment display. Then, when the timer based interrupt expired, the motor speed calculated during the ISR of the external interrupt, was sent to the ISR of the timer based interrupt, where a digital filter was implemented. When this ISR was completed, the variable holding the speed was returned to the while loop where the speed was displayed on the 7-segment display.

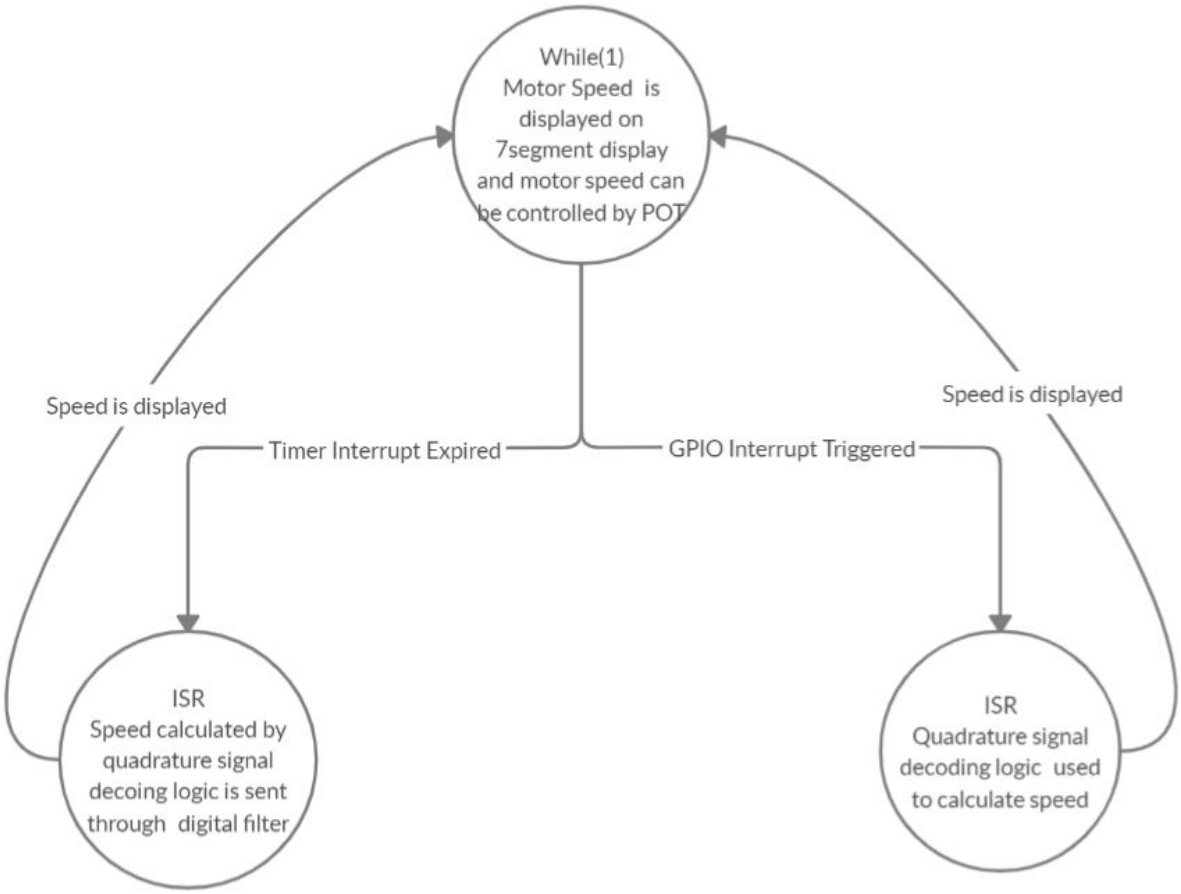

Figure 1: Part 2 High Level Software Diagram

Software Design

Part 1:

In part 1 of the lab, two external interrupts using GPIO pins were intilizated, one for phase A and one for phase B. Figure 2 below shows the two interrupt service routines associated with these interrupts.

```
void PORT3_IRQHandler(void)
{
    //check phase B logic state. if it is high, then decrease counter variable by 1
    //else if it is low, then increase counter variable by 1
    if(P5->IN & 0x08) {
        count--;
    }
    else {
        count++;
    }
    P3->IFG &= ~0x40;
}

void PORT5_IRQHandler(void)
{
    // check phase A logic state. If Phase A pin is High, increase the counter by 1.
    //else if it is low, then decrease counter variable by 1
    if(P3->IN & 0x40) {
        count++;
    }
    else {
        count--;
    }
    P5->IFG &= ~0x08;
}
```

Figure 2: Interrupt service routines for part 1

The PORT3_IRQHandler(), checked the phase B logic. If the phase B logic was high, the counter variable was decreased by 1, signifying a movement in the counterclockwise direction. If the phase B logic was low, the counter variable was increased by 1, signifying a movement in the clockwise direction. The PORT5_IRQHandler(), checked the phase A logic. If the phase A logic was high, the counter variable was increased by 1. If the phase B logic was low, the counter variable was decreased by 1.

Part 2:

Identifying Digital Filter

The first step to part 2 of the lab was to identify the lowpass filter needed. The requirements of the filter were that the sampling frequency is 200Hz and the bandwidth is 1Hz. The coefficients

form this transfer function were implemented in software to create the filter. Figure 3 below shows the Matlab code that was used to find the transfer function.

```
>> Hc= tf([pi/2], [(pi/2), 1])

Hc =

     1.571
  -----------
  1.571 s + 1

Continuous-time transfer function.

>> Hd=c2d(Hc,0.005)

Hd =

   0.004992
  ----------
  z - 0.9968

Sample time: 0.005 seconds
Discrete-time transfer function.
```

Figure 3: Obtaining coefficients for the filter in Matlab

The first step was to enter the continuous time transfer function in matlab based on the filter requirements and then use the "bode" function to plot the bode plot. After this, the continuous-time transfer function was converted into a discrete time transfer function using the "c2d" command. The resulting discrete time function was simplified into equation 1, which is shown below. This equation was implemented in the software.

$$Y[0] = (0.9968f * y[1]) + (0.004992f * x[1]) \quad \text{(Eq 1)}$$

Figure 4 below shows a bode plot of the low pass transfer function.

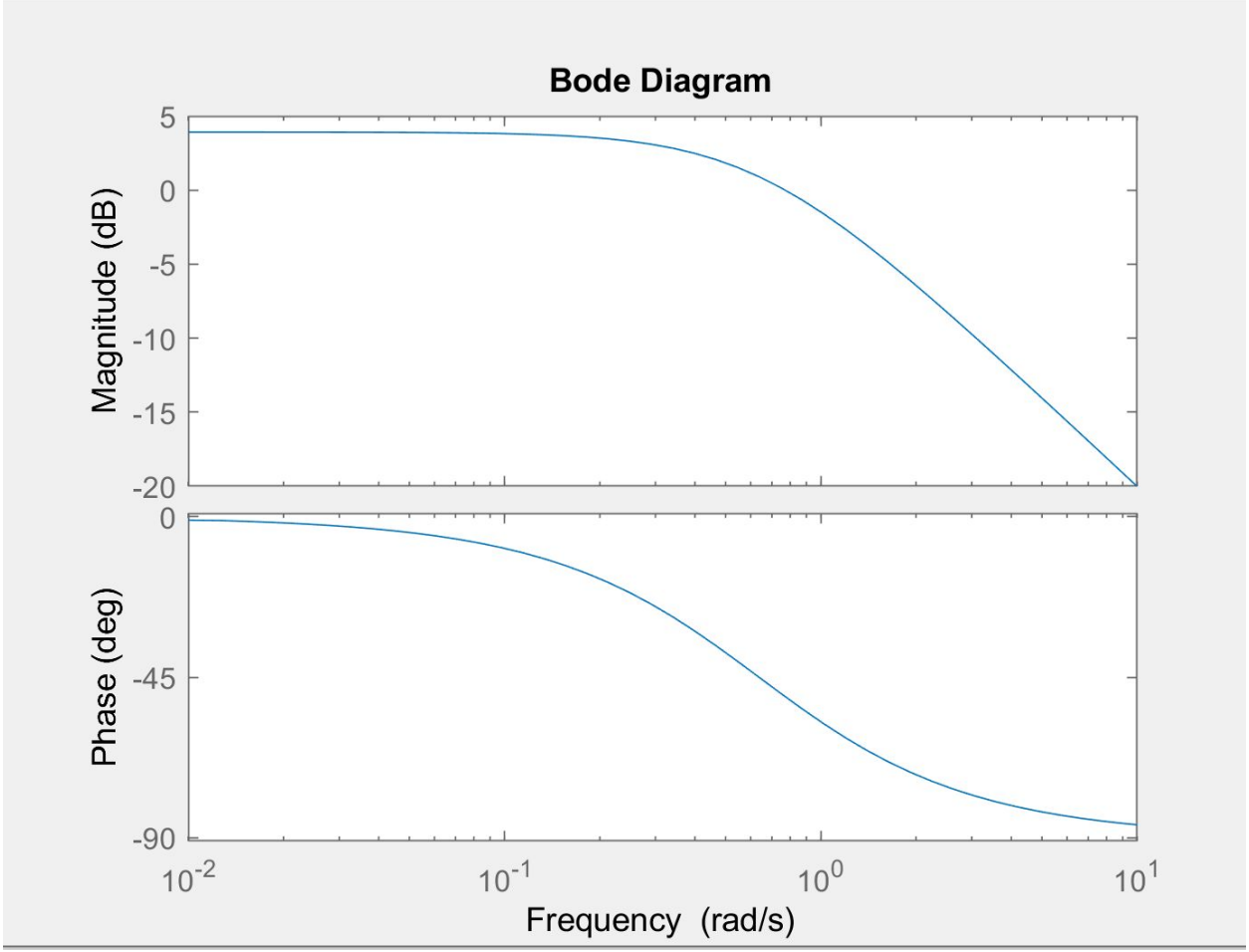

Figure 4: Bode Plot of continuous Time Transfer Function from Matlab

We then created a simulink model to plot the bode plot of the filter with a square wave input signal with a frequency of 1Hz. Figure 5 below shows an image of the simulink model.

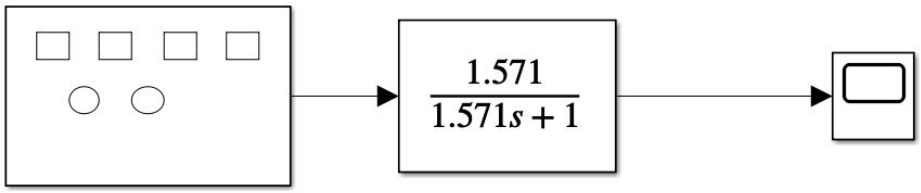

Figure 5: Simulink model of transfer function with square waveform input

The bode plot simulation showed a filtered version of a square wave shape. This can be seen below in Figure 6.

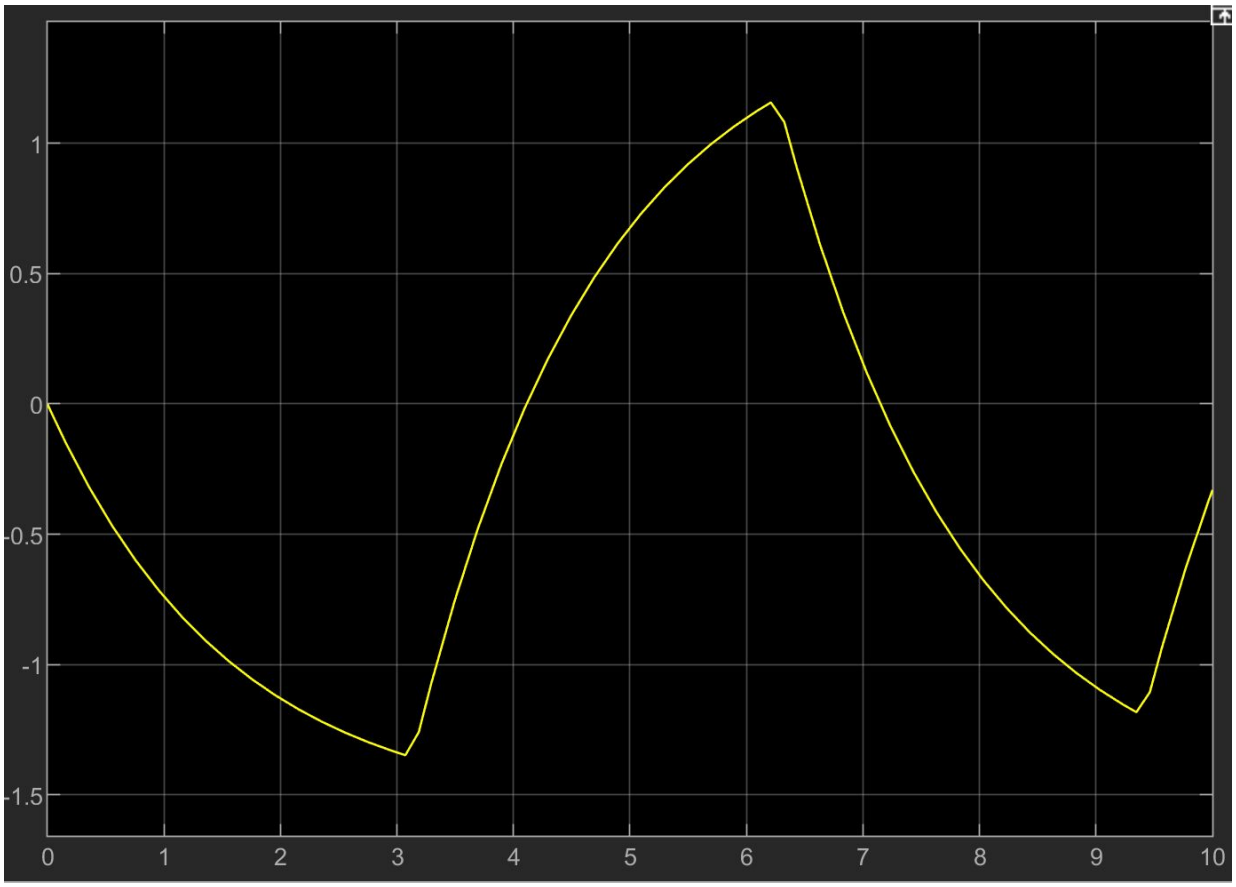

Figure 6: Simulink Bode Plot result with square waveform input

The remainder of this section will explain segments of the code used in part 2 of the lab. A few pieces of software were used from earlier labs. For example, this lab implemented the motor control function from Lab 3 in the while loop. This function was used so that the POT could control the speed of the motor. The encoder function was also used from part 1 of this lab to calculate the speed.

The first section of code, shown in figure 7, dealt with the initialization of our interrupts. The first function, TimerA2_Init() initializes our Timer A2 interrupt that deals with calculating and displaying our RPM. The interrupt performs its tasks at a frequency of 150Hz. This frequency was obtained by utilizing the SMCLK frequency which is set at 1.5MHz just like in part 2 of lab 3. The second function, Interrupt_Init(), initialized our Port 3.6 and 5.3 interrupts for each phase of the optical encoder. Both interrupts were set to rising edge detection.

```
//Timer_A register on page 797
void TimerA2_Init(){
    TIMER_A2->CTL = 0x02C0;  //sets clock to SMCLK and divide input by /8
    TIMER_A2->CCTL[0] = 0x0010;
    TIMER_A2->CCR[0] = 0xffff;   // compare match value
    TIMER_A2->EX0 = 0x0007;    // configure for input clock divider /8 making 150Hz
    NVIC->IP[3] = (NVIC->IP[3]&0xFFFFFF00)|0x00000060; // priority 2
    NVIC->ISER[0] = 0x00001000;   // enable interrupt 12 in NVIC
    TIMER_A2->CTL |= 0x0014;      // reset and start Timer A in up mode
}

//2 pins, p5.3 (phase B) and p3.6 (phase A)
//the code assumes encoder count increase when turning CW and decrease when turning CCW
//count is stored in a counter variable

void Interrupt_Init() {
    P3->SEL0 &= ~0x40;  //Function group set to 0 for IO
    P3->SEL1 &= ~0x40;  //Function group set to 0 for IO
    P3->DIR &= ~0x40;   //P3.6 set to input
    P3->IES &= ~0x40;   //Interrupt Edge Selector set to 0 for rising edge
    P3->IFG &= ~0x40;   //Interrupt Flag Register cleared (if its 1 then interrupt was triggered, has to be
    P3->IE |= 0x40;     //Interrupt Enabled registers set to 1 (enabled)
    NVIC->IP[9] = (NVIC->IP[9]&0x00FFFFFF)|0x40000000; //set to priority 3
    NVIC->ISER[1] = 0x00000020; //page 116 in the manual. Sets IRQ 37 enabling Port3_IRQ

    P5->SEL0 &= ~0x08;  //Function group set to 0 for IO
    P5->SEL1 &= ~0x08;  //Function group set to 0 for IO
    P5->DIR &= ~0x08;   //P5.3 set to input
    P5->IES &= ~0x08;   //Interrupt Edge Selector set to 0 for rising edge
    P5->IFG &= ~0x08;   //Interrupt Flag Register cleared (if its 1 then interrupt was triggered)
    P5->IE |= 0x08;     //Interrupt Enabled registers set to 1 (enabled)

    //NVIC registers on page 115
    NVIC->IP[9] = (NVIC->IP[9]&0x00FFFFFF)|0x40000000; //set to priority 3
    NVIC->ISER[1] = 0x00000080; //page 116 in the manual. Sets IRQ 39 enabling Port5_IRQ
}
```

Figure 7: Interrupt service routines for part 1.

Once we had all of our interrupts initialized, we set up the IRQHandlers for each of the interrupts. These are the functions that are carried out when the interrupts are called. For the port 3 and port 5 interrupts we performed the same task as explained in part 1. It simply increments or decrements count by about 800 for every completed rotation, depending on the direction of rotation. As for the Timer A2 IRQHandler, it deals with performing the calculations for converting count into RPM and then calls parse_rpm() which preps the value for the 7-segment display.

```
void PORT3_IRQHandler(void)
{
    //check phase B logic state. if it is high, then decrease counter variable by 1
    //else if it is low, then increase counter variable by 1
    if(P5->IN & 0x08) {
        count--;
    }
    else {
        count++;
    }
    P3->IFG &= ~0x40;
}

void PORT5_IRQHandler(void)
{
    // check phase A logic state. If Phase A pin is High, increase the counter by 1.
    //else if it is low, then decrease counter variable by 1
    if(P3->IN & 0x40) {
        count++;
    }
    else {
        count--;
    }
    P5->IFG &= ~0x08;
}

void TA2_0_IRQHandler(void) {
    //start_motor();
    rpm = (count * 187.5 * 60)/800;
    y[1] = (0.9968)*y[0] + (0.004992)*rpm;
    y[0] = y[1];
    count = 0;
    parse_rpm();
    TIMER_A2->CCTL[0] &= ~(BIT0); //clear the int flag
}
```

Figure 8: The IRQHandler functions for each of our 3 interrupts.

In figure 9, we show the parse_rpm() function. This function simply preps our RPM values for each of the four 7-segment displays. In addition to this, it checks whether our value is positive or negative. If negative, it lights up the red LED connected to P5.0.

```
void parse_rpm(){

    if(rpm < 0) {
        P5->OUT |= 0x01;
    }
    else {
        P5->OUT &= ~0x01;
    }

    ones = rpm%10;
    rpm = rpm/10;
    tens = rpm%10;
    rpm = rpm/10;
    hundreds = rpm%10;
    rpm = rpm/10;
    thousands = rpm%10;

}
```

Figure 9: The parse_rpm() function which preps the value for the 7-segment display.

The last function to discuss is our main() function, shown in figure 10. This function initializes our clock, configures our ADC, sets up the ports for our display and LEDs, and runs our interrupt initialization functions. In addition to this, it contains an infinite while loop that constantly calls our start_motor() function from lab 3 and our display function which outputs our values to the 7-segment display.

```
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    initialize_clock();
    configure_ADC();

    P8->DIR = 0xff;
    P4->DIR = 0xff;

    P8->OUT |= (BIT5 | BIT4 | BIT3| BIT2);
    P4->OUT = 0xff;
    P8->OUT &= ~BIT5;

    P7->DIR = 0xff;
    P10->DIR = 0xff;

    P5->DIR |= 0x01;    //sets P5.0 LED to output


    //New Lab 4 program to initialize interrupts
    Interrupt_Init();
    TimerA2_Init();

    while(1) {
        start_motor();
        display();
        //aitTime(1000);
    }

}
```

Figure 10: The main function of our program.

Post lab questions:

1. What is the purpose and importance of interrupts?

   The purpose of an interrupt is to allow a software developer the ability to write a routine that is ran when a specific event occurs. Interrupts can be given priority and triggered internally or externally. One important feature of interrupts is that when the interrupt is triggered, the program immediately stops the tasks it is completing and jumps to the interrupt service routine and does not leave this routine until the interrupt flag is cleared. This means if a system needs to respond to a specific event very quickly and prioritize it, an interrupt would be ideal.

2. Where can you see devices or machines utilize interrupts in our daily lives and how is it used?

   Interrupts are used in various devices and machines. It is difficult to specify examples without understanding the software and hardware of the device because there are alternatives to interrupts that could be implemented, but an example could be in a home security system. The system could have multiple timed tasks running for maintenance and then a “home invasion” event programmed as an interrupt. This would ensure the home invasion event is prioritized and the system would immediately stop the maintenance tasks and begin completing the interrupt service routine, which may be calling 911. Interrupts are used to prioritize a specific event where the routine associated with that event should be completed immediately. Interrupts are useful in systems that

spend most of the time polling for an event to occur. A few other examples of these machines and devices are: washing machines, keyboards, or to implement Amber Alerts.

Conclusion

In conclusion, the lab was a success. We were able to demo part 1 of the lab to the course instructor and although we could not demo part 2 of the lab, it is explained in detail in this lab report.