EGEC 451 Lab Report 2

Dominic Jacobo

1.Code from final program of the Lab.

LCD.c:

```
#include <stdint.h>
#include "Timer0A.h"
#include "SSI2.h"
#include "LCD.h"
#include "tm4c123gh6pm.h"
#include <stdio.h>

void DisableInterrupts(void);    // Disable interrupts
void EnableInterrupts(void);     // Enable interrupts
uint32_t StartCritical (void);   // previous I bit, disable interrupts
void EndCritical( uint32_t sr ); // restore I bit to previous value
void WaitForInterrupt(void);     // low power mode

// Macros
#define RS 1    // BIT0 mask for reg select
#define EN 2    // BIT1 mask for E

/*************** Private Functions ***************/

// LCD's SPI chip select is at PC6 (mask of 0x40 for SSI2_Write)
void LCD_nibble_write( uint8_t data, uint8_t control) {
  data &= 0xF0;       // clear lower nibble for control
  control &= 0x0F;    // clear upper nibble for data
  SSI2_write( (data | control), 0x40 );       // RS = 0, R/W = 0
```

```
  SSI2_write( (data | control | EN), 0x40 );  // pulse E
  //delayMs(0);
  SSI2_write( data, 0x40 );
  return;
}

/*************** Public Functions ***************/

// Clear the LCD
// Inputs: none
// Outputs: none
void LCD_Clear(void) {
  LCD_command(0x01);  // Clear Display
  // not necessary //LCD_command(0x80);  // Move cursor back to 1st position
}

// initialize SSI2 CS for LCD, then initialize LCD controller
// assumes Timer0A and SSI2 have already been initialized
void LCD_init(void) {
  SYSCTL_RCGCGPIO_R |= 0x04;   // enable clock to GPIOC

  // PORTC 6 for SSI2 chip select
  GPIO_PORTC_AMSEL_R &= ~0x40;      // disable analog
  GPIO_PORTC_DATA_R |= 0x40;        // set PORTC6 idle high
  GPIO_PORTC_DIR_R |= 0x40;         // set PORTC6 as output for CS
  GPIO_PORTC_DEN_R |= 0x40;         // set PORTC6 as digital pins

  Timer0A_Wait1ms(20);        // LCD controller reset sequence
```

```
  LCD_nibble_write(0x30, 0);
  Timer0A_Wait1ms(5);
  LCD_nibble_write(0x30, 0);
  Timer0A_Wait1ms(1);
  LCD_nibble_write(0x30, 0);
  Timer0A_Wait1ms(1);

  LCD_nibble_write(0x20, 0);  // use 4-bit data mode
  Timer0A_Wait1ms(1);
  LCD_command(0x28);          // set 4-bit data, 2-line, 5x7 font
  LCD_command(0x06);          // move cursor right
  LCD_command(0x01);          // clear screen, move cursor to home
  LCD_command(0x0F);          // turn on display, cursor blinking

  return;
}

// send a command to the LCD
void LCD_command( uint8_t command ) {
  uint32_t intStatus = StartCritical();
  LCD_nibble_write(command & 0xF0, 0);    // upper nibble first
  LCD_nibble_write(command << 4, 0);      // then lower nibble
  EndCritical( intStatus );

  if (command < 4)
    Timer0A_Wait1ms(2);        // command 1 and 2 needs up to 1.64ms
  else
    Timer0A_Wait1ms(1);        // all others 40 us
```

```
  return;
}

// send data (a character) to the LCD
void LCD_data( uint8_t data ) {
  uint32_t intStatus = StartCritical();
  LCD_nibble_write(data & 0xF0, RS);      // upper nibble first
  LCD_nibble_write(data << 4, RS);        // then lower nibble
  EndCritical( intStatus );

  Timer0A_Wait1ms(1);

  return;
}

//------------LCD_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void LCD_OutString( uint8_t *ptr ) {
  return;
}

//----------------------LCD_OutUDec----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
```

```
// Variable format 1-10 digits with no space before or after
void LCD_OutUDec( uint32_t n ) {
  // This function uses recursion to convert decimal number
  //   of unspecified length as an ASCII string
  if(n < 10){
      uint8_t decimal_to_char = (n % 10) + '0';
      LCD_data(decimal_to_char);
      return;
  }
  LCD_OutUDec(n / 10);
  LCD_OutUDec(n % 10);
}

//-----------------------LCD_OutUHex-----------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void LCD_OutUHex( uint32_t number ) {
// This function uses recursion to convert the number of
// unspecified length as an ASCII string
  return;
}

// -----------------------LCD_OutUFix----------------------
// Output characters to LCD display in fixed-point format
// unsigned decimal, resolution 0.1, range 000.0 to 999.9
// Inputs:  an unsigned 32-bit number
```

```
// Outputs: none
// E.g., 0,    then output "0.0"
//       3,    then output "0.3"
//       89,   then output "8.9"
//       123,  then output "12.3"
//       9999, then output "999.9"
//     > 9999, then output "*.***"
void LCD_OutUFix( uint32_t number ) {
  if(number < 10){// maybe condition should be if number < 10
     //send decimal point to LCD_data
     LCD_data('.');
     //send last number to LCD_data using mod 10 I think would work
     uint8_t decimal_to_char = (number % 10) + '0';
     LCD_data(decimal_to_char);
     return;
  }
  else if(number > 9999){//This should never happen but just incase
     //Just send -.--- or *.*** to the LCD. I think this is what he said to do
     //To implement this I should be able to  just hard code this and return
     LCD_data('*');
     LCD_data('.');
     LCD_data('*');
     LCD_data('*');
     LCD_data('*');
     return;
  }
  else{
  LCD_OutUDec(number / 10);
```

```
  LCD_OutUFix(number % 10);
  }
}
```

Main.c:

```
#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "PLL.h"
#include "Timer0A.h"
#include "Timer2A.h"
#include "SSI2.h"
#include "LCD.h"

#define FREQUENCY 8000000.0f
#define SEC_PER_MIN 60
#define GEARBOX_RATIO 120
#define PULSE_PER_ROTATION 8

void delayMs(int n);

int main(void) {
    //Varible to hold value of Rpm calculation
    uint32_t RPM = 0;
    //Function calls for various setups
    PLL_Init(Bus8MHz);
    Timer0A_Init(8000000);
    Timer2A_Init();
    SSI2_init();
    LCD_init();
```

```
int pw = 0;

SYSCTL_RCGCPWM_R |= 0x02;        // enable clock to PWM1
SYSCTL_RCGCGPIO_R |= 0x20;       // enable clock to GPIOF
SYSCTL_RCGCGPIO_R |= 0x02;       // enable clock to GPIOB

//delayMs(1);                    // PWM1 seems to take a while to start
Timer0A_Wait1ms(1);

SYSCTL_RCC_R &= ~0x00100000;     // use system clock for PWM
PWM1_INVERT_R  |= 0x80;          // positive pulse
PWM1_3_CTL_R = 0;                // disable PWM1_3 during configuration
PWM1_3_GENB_R = 0x0000080C;      // output high when load and low when match
PWM1_3_LOAD_R = 3999;            // 4 kHz
PWM1_3_CTL_R = 1;                // enable PWM1_3
PWM1_ENABLE_R |= 0x80;           // enable PWM1

GPIO_PORTF_DIR_R |= 0x08;             // set PORTF 3 pins as output (LED) pin
GPIO_PORTF_DEN_R |= 0x08;             // set PORTF 3 pins as digital pins
GPIO_PORTF_AFSEL_R |= 0x08;           // enable alternate function
GPIO_PORTF_PCTL_R &= ~0x0000F000;     // clear PORTF 3 alternate function
GPIO_PORTF_PCTL_R |= 0x00005000;      // set PORTF 3 alternate funtion to PWM

GPIO_PORTB_DEN_R |= 0x0C;             // PORTB 3 as digital pins
GPIO_PORTB_DIR_R |= 0x0C;             // set PORTB 3 as output
GPIO_PORTB_DATA_R |= 0x08;            // enable PORTB 3

while(1) {
```

```
// set direction
GPIO_PORTB_DATA_R &= ~0x04;
GPIO_PORTB_DATA_R |= 0x08;

// ramp up speed
for (pw = 100; pw < 3999; pw += 20) {//Max PW 3999 (Which is MAX speed) Start from 100
    PWM1_3_CMPB_R = pw;
    Timer0A_Wait1ms(100);
    LCD_Clear();
    RPM = (SEC_PER_MIN*FREQUENCY)/(PULSE_PER_ROTATION*GEARBOX_RATIO*CC_Difference)*10;
    LCD_OutUFix(RPM);
}
// ramp down speed
for (pw = 3940; pw >100; pw -= 20) {//Min PW 100(Which is pretty much min speed) Start from 3900
    PWM1_3_CMPB_R = pw;
    Timer0A_Wait1ms(100);
    LCD_Clear();
    RPM = (SEC_PER_MIN*FREQUENCY)/(PULSE_PER_ROTATION*GEARBOX_RATIO*CC_Difference)*10;
    LCD_OutUFix(RPM);
}

// reverse direction
GPIO_PORTB_DATA_R &= ~0x08;
GPIO_PORTB_DATA_R |= 0x04;
```

```
        // ramp up speed
        for (pw = 100; pw < 3999; pw += 20) {//Max PW 3999 (Which is MAX speed) Start
from 100
            PWM1_3_CMPB_R = pw;
            Timer0A_Wait1ms(100);
            LCD_Clear();
            RPM =
(SEC_PER_MIN*FREQUENCY)/(PULSE_PER_ROTATION*GEARBOX_RATIO*CC_Di
fference)*10;
            LCD_OutUFix(RPM);
        }
        // ramp down speed
        for (pw = 3940; pw >100; pw -= 20) {//Min PW 100(Which is pretty much min speed)
Start from 3900
            PWM1_3_CMPB_R = pw;
            Timer0A_Wait1ms(100);
            LCD_Clear();
            RPM =
(SEC_PER_MIN*FREQUENCY)/(PULSE_PER_ROTATION*GEARBOX_RATIO*CC_Di
fference)*10;
            LCD_OutUFix(RPM);
        }
    }
}

/* delay n milliseconds (50 MHz CPU clock) */
void delayMs(int n) {
    int i, j;
    for(i = 0 ; i< n; i++)
        for(j = 0; j < 6265; j++)
```

```
        {}  /* do nothing for 1 ms */
}
```

Timer2A.c:

```
#include <stdint.h>
#include <stdlib.h>
#include "tm4c123gh6pm.h"
#include "Timer2A.h"

// Using PB0 for input capture (T2CCP0)
void Timer2A_Init(){
  SYSCTL_RCGCTIMER_R |= 0x00000004; // Activate Timer2
  SYSCTL_RCGCGPIO_R |= 0x00000002;  // Activate Port B
  GPIO_PORTB_DEN_R |= 0x01;   // Enable digital I/O on PB0
  GPIO_PORTB_AFSEL_R |= 0x01; // Enable alternate function on PB0
  GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R & 0xFFFFFFF0) | 0x00000007; // Enable T2CCP0
  TIMER2_CTL_R &= ~TIMER_CTL_TAEN; // Disable Timer2A during setup
  TIMER2_CFG_R = TIMER_CFG_16_BIT; // Configure for 16-bit timer mode
  TIMER2_TAMR_R = TIMER_TAMR_TACMR | TIMER_TAMR_TAMR_CAP; // Configure for capture mode
  TIMER2_CTL_R &= ~(TIMER_CTL_TAEVENT_POS | 0xC ); // Configure for rising-edge event
  TIMER2_TAILR_R = 0x0000FFFF;     // Start value
  TIMER2_IMR_R |= TIMER_IMR_CAEIM; // Enable capture match interrupt
  TIMER2_ICR_R = TIMER_ICR_CAECINT; // Clear Timer2A capture match flag
  TIMER2_CTL_R |= TIMER_CTL_TAEN;  // Enable Timer2A
  NVIC_PRI5_R = (NVIC_PRI5_R & 0x00FFFFFF) | 0x40000000; // Timer2A = Priority 2
  NVIC_EN0_R = 0x00800000; // Enable interrupt 23 in NVIC
  EnableInterrupts();
```

```
return;
}


void Timer2A_Handler(){
  TIMER2_ICR_R = TIMER_ICR_CAECINT; // Acknowledge Timer2A capture
  //TODO: Calculate the period or pulse length of the DC motor's encoder here
  Current_CC_Count = TIMER2_TAR_R;
  CC_Difference = abs(Last_CC_Count - Current_CC_Count);
  Last_CC_Count = Current_CC_Count;


  return;
}
```

2. Explain how the DC motor code controls the speed and direction of the DC motor and what I/O pins are used to control the motor.

-First the DC motor code uses the I/O pins PF3, and PB2 and PB3. PF3 is responsible for the PWM signal that controls terminals M3 and M4 on the Edubase board. In the codes main while loop, for loops are used to ramp up and down the value of the PWM, with the motor supply wires being connected to M3 and M4, the value of PWN accounts for how much voltage is being given to the motor. Higher PWM leading to a higher voltage and lower PWM leading to a lower voltage supplied. PB2 and PB3 are responsible for controlling the direction that the DC motor spins in. When PB2 is set to 0 and PB3 is set to 1 the motor shaft will spin counter clockwise and will spin clockwise when PB2 is set to 1 and PB3 is set to 0.

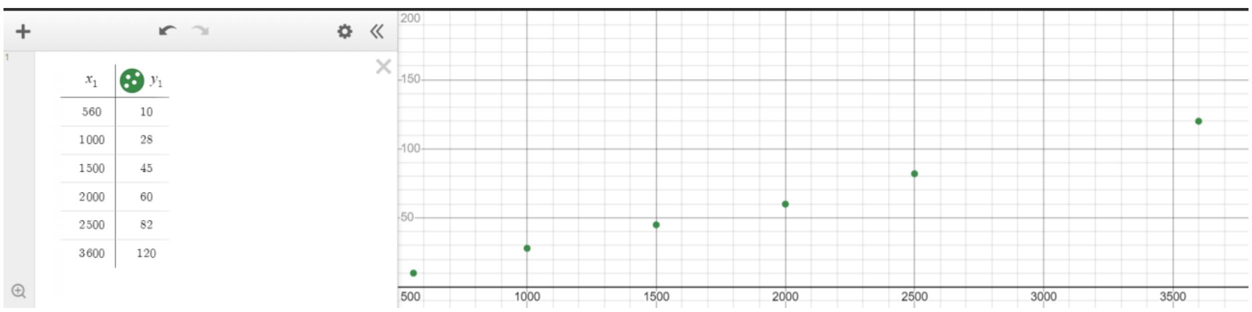
3. Explain your code for the Timer2A_Handler used for input capture. How are you able to calculate the period or pulse length of the DC motor's encoder signal by using this interrupt handler?

- We are using the TimerA system as an input capture from the motor's encoder. The Timer2A_Handler is triggered everything the motor's encoder detects a magnet. So, we calculate the period of the encoder by using global variables that the hander uses to store the timer's count down value each time the hander is called. This is important because this gives us the number of clock cycles before each interrupt. So, we use 3 variables to calculate the period, one is used for holding the new value of the timer's count down, another for holding the old value, and one more for holding the difference (subtraction) between the two which give us our period.

4.How did you calculate the rpm's of the DC motor from the encoder output? Please be specific and provide any equations used.

-To calculate the rpm I used the following equation: (Seconds_per_minute*frequency)/(pulse_per_round*gearbox_ratio*period_of_encoder). I came up with this equation with the help of my classmate Mohammad. We found a general equation for calculating RPM which stated that to calculate the RPM of a motor with no load we needed to use the following equation: RPM = (Frequency * 60)/(number of poles). This equation did not seem to calculate the correct value so we asked the instructor and we were told to take into account the pulse per round and gearbox ratio into account in our calculation. With this information we were able to come up with the following equation.

5.Graph of PWM value vs the motor rpm.



In the table above we can see the plot for PWM (x-axis) and RPM (y-axis). The graph shows a linear increase in RPM as the PWN increases.

6.Estimatation of PWM value required for a specific speed of the motor in rpm

-For my estimated PWM value I decided to pick 3000 because it is an obvious data point missing in my PWM vs RPM plot. I believed that this PWM would give the RPM value 105 because when looking at my graph it seems roughly with every increase of 500 PWM comes an increase of 20 RPM. The actual value that I got when testing was 100 rpm. Using the percentage error equation: Percentage Error = ((Estimated Number- Actual number)/Actual number) * 100. Do this equation gave me a percentage error of 5 percent.

7.Resources/Students whom I discussed this assignment.

- I did not use any resources. Mohammad and I discussed how to solve the RPM equation.