EEL 4742C: Embedded Systems

Name: Hamzah Ullah

Lab 5: The LCD Display

Introduction:

This lab we are introduced to the LCD and how to program it, as well as the segment designation or each part of the LCD. The table given in the lab shows us which registers and which bits coordinate which each segment of the LCD and location on it respectively. They show us a quick guide on how each letter/number should look and a table which is extremely helpful and allow us to create more unique numbers and letters as well as patterns the more we experiment with it.

Part 1: Printing Numbers on the LCD Display

In the lab, we are given a task to determine the hexadecimal equivalent of each number depending on which bits are activated and which aren't. After that, we are given a simple number to input by locating the correct register and directly outputting it by assigning the register to the LCD_Num() method we are given. Finally, we are asked to create a function to be able to enter any 16 number and output a number on the LCD. After several iterations of my code, I have finally figured out how to use pointers to point to the address of each part of the array, remembering to dereference it when trying to access it within the array. Since doing that, I was able to successfully and efficiently write the code. All the sample codes from the lab manual outputs correctly.

Creating the function, and outputting via direct register, and via function(New and improved!)

```
#include <msp430fr6989.h>
#define redLED BIT0 //red at p1.0
#define greenLED BIT7   //green at p9.7
void Initialize_LCD();
void display_num_lcd(unsigned int n);

//The array has the shapes of the digits (0 to 9)
//Complete this array...
const unsigned char LCD_Num[10] = {0xFC, 0x60, 0xDB,0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xF7 };
int main(void)
{
        WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins

        P1DIR |= redLED;    //pins as output
        P9DIR |= greenLED;
        P1OUT |= redLED;    //red on
        P9OUT &= ~greenLED; //green off

        //initializes the LCD_C module
        Initialize_LCD();
        display_num_lcd(65535);
        //The line below can be used to clear all the segments
        //LCDCMEMCTL = LCDCLRM;     //clears all segments
        // Display 430 on the rightmost 3 digits
//      LCDM8 = LCD_Num[0];
//      LCDM15 = LCD_Num[3];
//      LCDM19 = LCD_Num[4];
        int n;
        //Flash the red and green LEDs
        for(;;){
            for(n=0; n<=50000; n++){} //delay loop
            P1OUT ^= redLED;
            P9OUT ^= greenLED;
        }
        return 0;
}
//****************************************************
//initialize the LCD_C module
// *** Source: Function obtained from MSP430FR6989's sample code**
void Initialize_LCD(){
    PJSEL0 = BIT4 | BIT5;      //for LFXT

    //Initialize LCD segments 0-21; 26-43
    LCDCPCTL0 = 0xFFFF;
    LCDCPCTL1 = 0xFC3F;
    LCDCPCTL2 = 0x0FFF;

    //Configure LFXT 32khz crystal
    CSCTL0_H = CSKEY >> 8;  //Unlock CS registers
    CSCTL4 &= ~LFXTOFF;
    do{
        CSCTL5 &= ~LFXTOFFG;    //Clear LFXT fault flag
        SFRIFG1 &= ~OFIFG;
    } while(SFRIFG1 & OFIFG);   //test oscillator fault flag
    CSCTL0_H = 0;               //Lock CS registers

    //initialize LCD_C
    //ACLK, DIVIDER=1, PREDIVIDER = 16; 4pin MUX
    LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;

    //VLCD generated internally,
    //V2-V4 generated internally, V5 to ground
    //Set VLCD to 2.6v
    //Enable charge pump and select internal references for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;

    LCDCCPCTL = LCDCPCLKSYNC; //clock sync enabled

    LCDCMEMCTL = LCDCLRM;    //CLear lcd memory

    //Turn LCD on
    LCDCCTL0 |= LCDON;

    return;
    }
void display_num_lcd(unsigned int n)
{
    unsigned char *ptr[6] = {&LCDM8,&LCDM15,&LCDM19,&LCDM4,&LCDM6,&LCDM10};
    int i = 0;
    int digit;
// do the first iteration of the number on lcd
        do{
           digit = n%10;
           *ptr[i] = LCD_Num[digit];
           n = n/10;
                   i++;
           }
//while the digit doesn't equal zero
        while(n!=0);
//clears zero on screen while not in use
        while(i<=6)
        {
            *ptr[i] = 0;
            i++;
        }
}
```

Part 2: Implementing a Stop Watch

This part of the lab, we use the clock to increment up and display each period (TA0CCR0) on the LCD screen. To do this, we simply create a register and increment up, and passing that register into the function created in the previous lab. Using up mode, we can manipulate the data to whichever time period needed for a quicker count at lower TA0CCR0 numbers and a faster count at higher. After timing the code, it was pretty accurate close to what was predicted.

Code for incrementing up TACCR0 and setting a limit, currently at 10000-1

```
#include <msp430fr6989.h>
#define redLED BIT0 //red at p1.0
#define greenLED BIT7   //green at p9.7
void Initialize_LCD();
void display_num_lcd(unsigned int n);

//The array has the shapes of the digits (0 to 9)
//Complete this array...
const unsigned char LCD_Num[10] = {0xFC, 0x60, 0xDB,0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xF7 };

/**
```

```c
 * main.c
 */
int main(void)
{
        WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins

        P1DIR |= redLED;    //pins as output
        P9DIR |= greenLED;
        P1OUT |= redLED;    //red on
        P9OUT &= ~greenLED; //green off

        //ACLK 32khz
        config_ACLK_to_32KHz_crystal();
        //up mode limit
        TA0CCR0 = (10000-1);
        //increment through upmode cycles
        int i=0;
        //TA0CTL config
        TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
        //clear flag
        TA0CTL &= ~TAIFG;
        //initializes the LCD_C module
        Initialize_LCD();

        //The line below can be used to clear all the segments
        //LCDCMEMCTL = LCDCLRM;     //clears all segments

        int n;
        //Flash the red and green LEDs
        for(;;){
           while((TA0CTL & TAIFG) == 0){}
           display_num_lcd(i++);
           TA0CTL &= ~TAIFG;
        }
}
//****************************************************
//initialize the LCD_C module
// *** Source: Function obtained from MSP430FR6989's sample code**
void Initialize_LCD(){
    PJSEL0 = BIT4 | BIT5;      //for LFXT

    //Initialize LCD segments 0-21; 26-43
    LCDCPCTL0 = 0xFFFF;
    LCDCPCTL1 = 0xFC3F;
    LCDCPCTL2 = 0x0FFF;

    //Configure LFXT 32khz crystak
    CSCTL0_H = CSKEY >> 8;  //Unlock CS registers
    CSCTL4 &= ~LFXTOFF;
    do{
        CSCTL5 &= ~LFXTOFFG;    //Clear LFXT fault flag
        SFRIFG1 &= ~OFIFG;
    } while(SFRIFG1 & OFIFG);   //test oscillator fault flag
    CSCTL0_H = 0;               //Lock CS registers

    //initialize LCD_C
    //ACLK, DIVIDER=1, PREDIVIDER = 16; 4pin MUX
    LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;

    //VLCD generated internally,
    //V2-V4 generated internally, V5 to ground
    //Set VLCD to 2.6v
    //Enable charge pump and select internal references for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;

    LCDCCPCTL = LCDCPCLKSYNC; //clock sync enabled

    LCDCMEMCTL = LCDCLRM;    //CLear lcd memory

    //Turn LCD on
    LCDCCTL0 |= LCDON;
    return;
    }
void display_num_lcd(unsigned int n)
{
    unsigned char *ptr[6] = {&LCDM8,&LCDM15,&LCDM19,&LCDM4,&LCDM6,&LCDM10};
    int i = 0;
    int digit;
// do the first iteration of the number on lcd
        do{
           digit = n%10;
           *ptr[i] = LCD_Num[digit];
           n = n/10;
                   i++;
           }
//while the digit doesn't equal zero
        while(n!=0);
//clears zero on screen while not in use
        while(i<=6)
        {
            *ptr[i] = 0;
            i++;
        }
}
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

        //Reroute pins to LFXIN/LFXOUT functionality
        PJSEL1 &= ~BIT4;
        PJSEL0 |= BIT4;

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //Local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

        CSCTL0_H = 0; //lock CS registers
        return;
    }
```

Part 3: Stopwatch with halt/resume and reset functions

In this part of the lab we use the timers to create a halt/resume and reset function using the buttons as a way to manipulate it. One tricky part about this part of the lab is to read each individual Button click as unique with its own function for each one. I used a status function to indicate which button clicks were complete and which were yet to be executed, to keep track of when button 1 halted and when it resumed. Using a bunch of if while statements I was able to complete the assigned task. One problem I had with the code was the debouncing still exist when resuming the count. But the reset works fine. Below is the code:

Using either interrupt or within the for loop, we have the clock increment up, than stop on But1 and turn on the redLED, and switch on the greenLED which was already on. Resume counting on second BUT1 count, where BUT2 is reset:

```
#include <msp430fr6989.h>
#define redLED BIT0 //red at p1.0
#define greenLED BIT7   //green at p9.7
#define BUT1 BIT1
#define BUT2 BIT2

void Initialize_LCD();
void display_num_lcd(unsigned int n);
```

```
//The array has the shapes of the digits (0 to 9)
//Complete this array...
const unsigned char LCD_Num[10] = {0xFC, 0x60, 0xDB,0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xF7 };

/**
 * main.c
 */
int main(void)
{
        WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins

        P1DIR |= redLED;    //pins as output
        P9DIR |= greenLED;
        P1OUT &= ~redLED;    //red on
        P9OUT |= greenLED; //green off

        P1DIR &= ~(BUT1|BUT2);
        P1REN |= (BUT1|BUT2);
        P1OUT |= (BUT1|BUT2);

        //ACLK 32khz
        config_ACLK_to_32KHz_crystal();
        //up mode limit
        TA0CCR0 = (7500-1);
        //increment through upmode cycles
        int i=0;
        //TA0CTL config
        TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
        //clear flag
        TA0CTL &= ~TAIFG;
        //initializes the LCD_C module
        Initialize_LCD();

        //The line below can be used to clear all the segments
        //LCDCMEMCTL = LCDCLRM;     //clears all segments

        int n;
        int status = 0;
        //Flash the red and green LEDs
        for(;;){
           while((TA0CTL & TAIFG) == 0){}
           //First BUT1 Press to change status and set LEDs
           while((P1IN & BUT1)==0){
               P9OUT &=~greenLED;
               P1OUT |=redLED;
               //Checks LED status before status change to reduce debouncing
               if(redLED!=0)
               status = 1;
          }
           //Checks Status
           if(status == 1)
           {
               //LCD Display stays on if 1
               while(status == 1)
               {
               display_num_lcd(i);
                   //Reset to zero BUT2
                   if((P1IN & BUT2)== 0)
                   {
                         i=0;
                   }
                   //Set Status to 0 on second press
                   if((P1IN & BUT1) ==0)
                    status = 0;
               }
           }
           //reset to zero
           if((P1IN & BUT2)== 0){
               i=0;
           }
           // @ zero status counts
           if(status == 0)
```

```
            {
            display_num_lcd(i++);
            P9OUT |= greenLED;
            P1OUT &= ~redLED;
            }
            TA0CTL &= ~TAIFG;
            }
}
//****************************************************
//initialize the LCD_C module
// *** Source: Function obtained from MSP430FR6989's sample code**
void Initialize_LCD(){
    PJSEL0 = BIT4 | BIT5;      //for LFXT

    //Initialize LCD segments 0-21; 26-43
    LCDCPCTL0 = 0xFFFF;
    LCDCPCTL1 = 0xFC3F;
    LCDCPCTL2 = 0x0FFF;

    //Configure LFXT 32khz crystak
    CSCTL0_H = CSKEY >> 8;  //Unlock CS registers
    CSCTL4 &= ~LFXTOFF;
    do{
        CSCTL5 &= ~LFXTOFFG;    //Clear LFXT fault flag
        SFRIFG1 &= ~OFIFG;
    } while(SFRIFG1 & OFIFG);   //test oscillator fault flag
    CSCTL0_H = 0;               //Lock CS registers

    //initialize LCD_C
    //ACLK, DIVIDER=1, PREDIVIDER = 16; 4pin MUX
    LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;

    //VLCD generated internally,
    //V2-V4 generated internally, V5 to ground
    //Set VLCD to 2.6v
    //Enable charge pump and select internal references for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;

    LCDCCPCTL = LCDCPCLKSYNC; //clock sync enabled

    LCDCMEMCTL = LCDCLRM;    //CLear lcd memory

    //Turn LCD on
    LCDCCTL0 |= LCDON;

    return;
    }
void display_num_lcd(unsigned int n)
{
    unsigned char *ptr[6] = {&LCDM8,&LCDM15,&LCDM19,&LCDM4,&LCDM6,&LCDM10};
    int i = 0;
    int digit;

        do{
           digit = n%10;
           *ptr[i] = LCD_Num[digit];
           n = n/10;
                   i++;
           }
        while(n!=0);
        while(i<=6)
        {
            *ptr[i] = 0;
            i++;
        }
}
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

        //Reroute pins to LFXIN/LFXOUT functionality
        PJSEL1 &= ~BIT4;
        PJSEL0 |= BIT4;
        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;     //Local fault flag
            SFRIFG1 &= ~OFIFG;       //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

        CSCTL0_H = 0; //Lock CS registers
        return;
    }
```

QUESTIONS:

1. **Explain whether this statement is true or false. If False, explain the correct operation. "an LCD segment is given 1 to turn it on and 0 to turn it of, just like the colored LEDs". Explain whether it'**
   This is false to a certain degree. An LCD segment does need to initialized to 1 to turn on, but the register for the LCD contains 8 bit. This means that each bit of a register controls a certain segment of the corresponding register on the LCD panel. We mask it by assigning the values in hexadecimal which turn 'on' the LCD segment. It also allows us to control multiple LCD segments with a single register. On top of that, the segment is required to oscillate its value to avoid burning the LCD out.

2. **What is the name of the LCD controller the Launchpad uses to interface the LCD display? Is the LCD controller located on the display module or in the microcontroller?**
   The name of the controller that interfaces with the display is the LCD_C module and is located in the MCU or microcontroller.

3. **In what multiplexing configuration is the LCD module wired (2-way , 4-way, etc)? What does this mean regarding the number of pins used at the microcontroller?**
   4-way multiplex. It means that one of the pins on the microcontroller controls 4 of the segments on the LCD. The larger the mux, the fewer the pins, but also the less ability to control the contrast because the pin is dividing so much of its output that the range is limited as opposed to a 4 or 2 way multiplexed LCD module.