EEL4742C: Embedded Systems

Experiment Five:

Liquid Crystal Display

Ryan Koons 10/17/2020

Introduction: The main focus of this lab is to understand how to use the LCD and specifically display digits on the display. The first step in understanding is identifying the significance of digit shape. The digit shape is encoded by hex; in other words, each segment of a digit corresponds with a certain bit. A combination of bits (1-set, 0-off) will ensure the right shape is printed for each digit. A global character array will be used to store the digit shapes; these shapes can then be called by the LCD display num function. The other important part of the LCD is that each digit (and particular groups of segments) has its own LCDMx address; this address must be written to in order to print on the display. An LCD initialization function will be called (provided by TI). After that, another function will be able to parse through an integer and display each digit on the display from (right to left). Another useful skill is turning off excess digits. Later on in the lab, additional features for a stopwatch will be introduced—such as how to create a pause button as well as a reset button.

5.1 Printing Numbers on the LCD: Printing the LCD is very straight forward but does require a bit of workload and overhead prior to the actual printing taking place. The most important aspect of which is that the shapes of the "x-segment" display most be encoded as hexadecimal values and loaded into a char array entitled "LCD_Num". The Launchpad user's guide provides insight as far as the address of each digit as well as which individual bit corresponds with each segment for a particular digit. Then, it is the programmer's duty to correctly utilize the correct digits address and assign it an encoded hex. Value that will produce the correct digit shape on the display.

The main function defines an unsigned int 'n'—this will be the integer that is displayed on the LCD. After the typical initialization steps with the Watchdog timer, and LEDs, a function is called to initialize the LCD. This function is provided in msp430fr6989 example code and mainly configures the crystal, clock, multiplexer, and voltages for the LCD. The next two commented out segments signify how to clear all segments on the display as well as pass individual digits with set addressing to the LCD. The next step is to call the function that prints the integer to the LCD. There is some other code that follows in the main, but this just toggles the LEDs to ensure that they are working.

The function display_num_lcd receives an integer and parses through the digits of the 16-bit (max) integer and prints them on the LCD from right to left (incredibly quickly). The last part of the function then goes through any used digits and ensures that they are turned off (either from previous passed integers or flickers on the display, due to floating pins. This function defines an index for counting the number of digits on the LCD, and also a character pointer array that has the corresponding address for printing the 5 rightmost 7-segment digits on the LCD. A do while loop runs, because even if a 0 is passed we want to display the a right-most zero on the display. This loop uses modulo and division to take the right most digit, pass it to the function, and then cut it off from the integer. Each iteration of the loop also increases the pointer to move to the next digit on the display. The rest of the program iterates through

the remaining (unused) digits and ensures that all of their segments are turned off. NOTEL This program works with small and large values (0-->65535)

5.1 CCS Code: Print a 16-bit integer on the LCD

```
//Lab 5.1--Print 16 bit unsigned int  to LCD
//Goal: Design a program that toggles the R/G LEDs and prints up to a 5  digit
(16bit) number on the screen, no leading zeroes
#include <msp430.h>
#define redLED BIT0    //red LED at P1.0
#define greenLED BIT7  //green LED at P9.7

void Initialize_LCD();

void display_num_lcd(unsigned int n); //function that correctly displays a passed int

//array of digit shapes, defined globally---0 through 9
const unsigned char LCD_Num[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0,
0xFF, 0xE7};


int main(void)
{
    //int for display LCD function: 16 Bit unsigned int, modify this variable to
change LCD number
    volatile unsigned int n=65535;
      WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
      PM5CTL0 &= ~LOCKLPM5;       //enable GPIO Pins

      P1DIR |= redLED; //set as output
      P9DIR |= greenLED;
      P1OUT |= redLED;    //red ON
      P9OUT &= ~greenLED; //green OFF

      //initialize LCD_C module--prepare LCD controller within uC
      Initialize_LCD();

      //Line below can be used to clear all segments
      //LCDCMEMCTL= LCDCLRM;   //clear all segments

      //Display 430 on rightmost digits
      //LCDM19 = LCD_Num[4];  //display 4 at Leftmost (of 3 used) digits
      //LCDM15 = LCD_Num[3];
      //LCDM8 = LCD_Num[0];

      //call function to print int n on LCD
      display_num_lcd(n);

      //flash the red and green LEDs
      for(;;){
```

```
            for(n=0; n<= 50000; n++){} //delay loop
            P1OUT ^=redLED; //LEDs toggle
            P9OUT ^= greenLED;
        }

        return 0;
}

//function called after LCD init. prints int n to LCD
void display_num_lcd(unsigned int n)
{
    int  i=0;  //ptr index for parsing through
   unsigned char * ptr [6] = { &LCDM8, &LCDM15, &LCDM19 , &LCDM4, &LCDM6, &LCDM10};
//ptr for parsing
    int digit; //current digit to pass to display
//do, while, need at least one iteration through (0 case)
    do{
        digit = n%10; //grab rightmost digit
        *ptr[i] =LCD_Num[digit]; //print digit to display
        n=n/10; //cut off rightmost digit
        i++; //move to next ptr for addressing LCD
    } while (n>0);

    //turn all other digits off, i keeps incrementing thorugh the display digits
    while (i<8)
    {
        ptr[i] = 0;
        i++;
        }
}



// Initializes 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; // Enable LFXT

    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, Pre-divider = 16; 4-pin MUX
    LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;

// VLCD generated internally,
// V2-V4 generated internally, v5 to ground
// Set VLCD voltage to 2.60v
// Enable charge pump and select internal reference for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
    LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
    LCDCMEMCTL = LCDCLRM; // Clear LCD memory

    //Turn LCD on
    LCDCCTL0 |= LCDON;
return;
}
```

5.2 Implementing a Stopwatch: This program utilizes display_num_lcd from 5.1, and the convert-to-32kHz function from the timer lab to implement a stopwatch that counts from 1-65535 (1-second increments) and rolls back to one. Typical initialization took place at the beginning of the program, this included configure timer a with ACLK in up mode with a precise one second delay. It is important to note that display_num_lcd requires the same global array of digit shapes defined, just like in part 5.1. The next step is to call the 32kHz function, and to call the LCD initialization function also from 5.1. Integer n is also defined (starting at 1) which will serve as the counting variable that will also appear on the LCD.

Since the LEDs will each toggle every second to indicate the notion of 'a second' to the user, these should also be initialized at this step (one on and on off). Next, the infinite loop takes place, and the TAIFG is polled to create a particular event every second. Whenever this flag is raised, int n is passed to the LCD display function (same function from 5.1), the LEDs toggle, the flag is cleared (due to polling), and then the value of n is increased. This variable was utilized since it was unknown how to utilize the TAR variable. Additionally, the program will essentially set n back to 1 after it prints 65535 to the display (through a conditional statement within in the for loop). NOTE: Due to design, one second passes before the first counter "1" prints on the display; however, rollback will go directly from 65535 to 1 with no additional second (0 second) in between the two values. The timing of this program very closely resembles a smartphone's stopwatch.

5.2 CCS Code: Stopwatch, continuously counts up

```
//Lab 5.2: Timer---Up Mode, 0-65535
//After 65535, roll back to 0
#include <msp430.h>
#define redLED BIT0    //red LED at P1.0
#define greenLED BIT7  //green LED at P9.7

void Initialize_LCD();

void display_num_lcd(unsigned int n);

//Function Prototype for 32Khz ACLK
void convert_32khz();

//array of digit shapes, defined globally---0 through 9
```

```
const unsigned char LCD_Num[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0,
0xFF, 0xE7};


int main(void)
{
    //int for display LCD function: 16 Bit unsigned int
    volatile unsigned int n=1;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       //enable GPIO Pins

    P1DIR |= redLED; //set as output
    P9DIR |= greenLED;
    P1OUT &= ~redLED;    //red ON
    P9OUT &= ~greenLED; //green OFF

    //Set timer Period
             TA0CCR0 = (32768-1); // @32Khz, 1 second
             //Configure Timer_A: ACLK, Divide by 1, Up Mode, clear TAR
             TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
             TA0CTL &= ~TAIFG; //Clear flag at start


    //Initialize Timer
    //Configure ACLK to 32kHz
      convert_32khz();


    //initialize LCD_C module--prepare LCD controller within uC
    Initialize_LCD();


         P1OUT |= redLED;    //red ON
         P9OUT &= ~greenLED; //green OFF

    //flash the red and green LEDs
    for(;;){

        while((TA0CTL&TAIFG) == 0){} //delay loop
        TA0CTL &= ~TAIFG; //dont forget to clear flag!!!
        display_num_lcd(n);
        P1OUT ^=redLED;
        P9OUT ^= greenLED;
        if(n==65535)
        {
            n=0; //reset and then increment will make it 1 again
        }
         n++;

    }

    return 0;
}
```

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

    do{
        digit = n%10;
        *ptr[i] =LCD_Num[digit];
        n=n/10;
        i++;
    } while (n>0);

    while (i<8)
    {
        *ptr[i] = 0;
        i++;
        }
}

//Function for Converting ACLK to 32Khz
void convert_32khz()
{
    //By default, ACLK runs on  LFMODCLK at 5Mhz/128= 39 Khz

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

    //wait until oscillator fault flags are cleared
    CSCTL0 = CSKEY;  //unlock CS registers
    do {
        CSCTL5 &= ~LFXTOFFG;     //local fault flag
        SFRIFG1 &= ~OFIFG;       //global fault flag
    } while((CSCTL5 & LFXTOFFG) != 0 );

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



// Initializes 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; // Enable LFXT
```

```
    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, Pre-divider = 16; 4-pin MUX
    LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;

// VLCD generated internally,
// V2-V4 generated internally, v5 to ground
// Set VLCD voltage to 2.60v
// Enable charge pump and select internal reference for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
    LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
    LCDCMEMCTL = LCDCLRM; // Clear LCD memory

    //Turn LCD on
    LCDCCTL0 |= LCDON;
return;
}
```

5.3 Stopwatch with halt/resume and Reset Functionality: This code is very similar to 5.2 but has some additional functionality; therefore, only the new additions will be covered during this explanation. One addition being a global pause variable that is used (assuming it never reaches 65535) so that an even value (or 0) indicates a running state, whereas an odd value indicates a paused state. Modulo operation is used to determine whether the program is paused. Within the polling loop, everything is the same, including TAIFG that will toggle the LEDs and print the next number to the display. However, there is a conditional statement that checks for “running mode” of the stopwatch. So, if the stopwatch is running, the green LED is set to ON, the LCD display function is called, and n is incremented. Otherwise (halt mode) do nothing in the poll loop. The interrupt event is utilized for both buttons to implement the new functionality. Therefore, the ISR services both buttons and therefore handles the additional stopwatch functionality.

The ISR contains the majority of the additional functionality. The first section checks to see if button 1 has been pressed. Then it increments the pause variable, clears the flag, and if halted, it also turns the red LED on. IF button2 is pressed, the ISR resets n back to 1, and clears the flag. Also, if it is halt mode it displays the current value (1) to the LCD. The conditional statements in both the ISR and polling loop, in tandem with a pause variable allow for easy implementation of the halt/resume and reset functionality.

5.3 CCS Code: Stopwatch with additional functionality

```
//Lab 5.3: Timer---Up Mode, w/ enhanced functionality
//pause/resume and reset
#include <msp430.h>
#define redLED BIT0    //red LED at P1.0
#define greenLED BIT7  //green LED at P9.7
#define BUT1 BIT1     //Button 1 at P1.1
#define BUT2 BIT2     //BUtton 2 at P1.2

void Initialize_LCD();

void display_num_lcd(unsigned int n);

//Function Prototype for 32Khz ACLK
void convert_32khz();

//array of digit shapes, defined globally---0 through 9
const unsigned char LCD_Num[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0,
0xFF, 0xE7};

//global pause variable
int pause=0; //0-denotes running, 1 indicates pause
volatile unsigned int n=1;


int main(void)
{
    //int for display LCD function: 16 Bit unsigned int
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;       //enable GPIO Pins

    P1DIR |= redLED; //set as output
    P9DIR |= greenLED;
    P1OUT &= ~redLED;    //red ON
    P9OUT &= ~greenLED; //green OFF

    //Button Init.
    P1DIR &= ~(BUT1|BUT2);            //direct pins as inputs
        P1REN |= (BUT1 | BUT2);           //enable built-in resistor for P1.1 and
P1.2
        P1OUT |= (BUT1 | BUT2);           //set resistor as pull-up to Vcc for P1.1
and P1.2
        P1IE |= (BUT1|BUT2);   //1:enable interrupts
         P1IES |= (BUT1|BUT2);  //1: interrupt on falling edge
         P1IFG &= ~(BUT1|BUT2); //0: Clear interrupt flags

    //Set timer Period
             TA0CCR0 = (32768-1); // @32Khz, 1 second
             //Configure Timer_A: ACLK, Divide by 1, Up Mode, clear TAR
             TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
             TA0CTL &= ~TAIFG; //Clear flag at start


    //Initialize Timer
    //Configure ACLK to 32kHz
```

```
      convert_32khz();


    //initialize LCD_C module--prepare LCD controller within uC
    Initialize_LCD();

    _enable_interrupts(); //GIE set

    //flash the red and green LEDs
    for(;;){

        while((TA0CTL&TAIFG) == 0){} //delay loop
        TA0CTL &= ~TAIFG; //dont forget to clear flag!!!

        //if running mode, execute timer normally, else do nothing
        if((pause%2) == 0)//running mode
        {
            P1OUT &=~redLED;//red LED off
            P9OUT |= greenLED; //green LED on
        display_num_lcd(n);
        if(n==65535)
        {
          n=0; //reset and then increment will make it 1 again
        }
         n++;
        }

    }

    return 0;
}

//**********ISR********//
//*********************//
#pragma vector = PORT1_VECTOR  //Link the ISR to the Vector
__interrupt void P1_ISR()
{
    //BUT1 is pressed
   if( (P1IFG&BUT1) !=0 ) {
        pause++;//increment pause variable
        //if halt mode
        if( (pause%2) != 0)  //halt mode
        {
            P1OUT |= redLED; //red LED ON
            P9OUT &= ~greenLED;
        }
   P1IFG &= ~BUT1; //clear flag (shared)
   }

   //BUT2 is pressed (reset)
   if( (P1IFG&BUT2)!= 0 ) {
        n=1;//reset counter back to 1
        //if in halt mode display current value on display
        if( (pause%2) != 0)
        {
```

```
            display_num_lcd(n);
        }
    P1IFG &= ~BUT2; //clear flag (shared)
    }

}



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

    do{
        digit = n%10;
        *ptr[i] =LCD_Num[digit];
        n=n/10;
        i++;
    } while (n>0);

    while (i<8)
    {
        *ptr[i] = 0;
        i++;
        }
}

//Function for Converting ACLK to 32Khz
void convert_32khz()
{
    //By default, ACLK runs on  LFMODCLK at 5Mhz/128= 39 Khz

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

    //wait until oscillator fault flags are cleared
    CSCTL0 = CSKEY;  //unlock CS registers
    do {
        CSCTL5 &= ~LFXTOFFG;     //local fault flag
        SFRIFG1 &= ~OFIFG;       //global fault flag
    } while((CSCTL5 & LFXTOFFG) != 0 );

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



// Initializes 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; // Enable LFXT

    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, Pre-divider = 16; 4-pin MUX
    LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;

// VLCD generated internally,
// V2-V4 generated internally, v5 to ground
// Set VLCD voltage to 2.60v
// Enable charge pump and select internal reference for it
    LCDCVCTL = VLCD_1 | VLCDREF_0 | LCDCPEN;
    LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled
    LCDCMEMCTL = LCDCLRM; // Clear LCD memory

    //Turn LCD on
    LCDCCTL0 |= LCDON;
return;
}
```

Student Q&A:

1) That is not true, it is more complicated than just setting a one and a zero. The voltage is in fact always alternating, since a constant voltage would burn out the segment. Each segment lies on a particular COM line and is compared with the alternating voltage designated by its corresponding COM. And to keep the segment off it will match it to achieve zero bias; whereas, to turn it on you mirror the COM to achieve full bias. Pins left floating will achieve half bias (half on).
2) In the MSP430FR6989, the LCD Controller is entitled "LCD_C" and is found as a module in the microcontroller. This is done to take advantage of the excess silicon in the microcontroller; it also allows for a cheap LCD on the board.
3) The LCD_C has 2 to 8 MUX, however the LCD Init function reveals that a 4-way mux is being utilized. The chip's datasheet shows that the LCD has up to 320 segments. But, here the LCD (according to the lab manual) utilizes 108 pins on the LCD. With 4-way Mux., this results in (108/4)+4= 31 pins on the microcontroller.

Conclusion: After this lab it is now clear how to rudimentarily use the LCD on the MSP430. It is important to note that this lab focused on a traditional 7-segment implementation; however, this particular display has a lot more segments that are capable of outputting all sorts of other shapes. Nonetheless, this lab most importantly taught how to print a 16-bit integer to the display. This undoubtedly will be used in future labs.

In order to use the display, an example LCD initialization function is often used to set up the multiplexing and LCD clock. An important step in using the LCD, is to ensure that the shapes are encoded as hexadecimal values in a global char array. The LCD_display_num function will serve as a great tool in future labs; this function is highly important as it can appropriately display an 16-bit integer on the LCD. This lab also provided insight as to how to implement both a pause and reset feature.

One way to extract more information from this lab would be to refer back to the launchpad users guide and try to print other digit shapes on the LCD. For example, the hexadecimal values for the entire English alphabet could be implemented. Also, digits other than the right-most five could be utilized by varying the addressing.