EEL 4742C: Embedded Systems

Christopher Badolato
10/14/2019
EEL4742-0011
LAB 5 LCD display

Introduction

In this lab we will be printing to the LCD display. Different segments of the LCD will be programmed to oscillate continuously to display the intended data. First, we display any value from 0 to 65535 (value must be hard coded) on the LCD using a displayLCD function called within main. Then, we create a timer that counts up by 1 each second using the timer_A interrupt and our display LCD function created in part 5.1. Finally, we use this timer to create a stopwatch using the buttons S1 and S2. Button s1will pause or continue the timer, while button s2 will reset the timer to all zeros.

In this part of the experiment we are introducting a timer interrupt that will display a timer on the LCD with 1 second periods. On each timer intterupt we will be incrementing a global varible n that will be displayed on the LCD each second starting at 0. This code is very similar to the code for 5.1 but instead we initialize the timer_A to up_mode and initalize TARR0 to 32768 to ensure 1 second intervals between LCD and n incrememnts.

Part 5.1: ...

First, we display any value from 0 to 65535 (value must be hard coded) on the LCD. We can fill these segments by using the array below which represents the digits 0 – 9 in hex. Each hex value corresponds to the 8-bit value that will turn on the specific segments of the LCD we would like.

```
(const unsigned char LCD_Num[10] = {0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF,
0xE4, 0xFF, 0xF7}; )
```

To fill in the correct value sent as an integer to our displayLCD function we must first, grab each digit by using the mod operation by 10 to 1000000 to grab the corresponding decimal values.

If our value is 65535 then,

```
n0 = (n % 100000)/10000;
n1 = (n % 10000)/1000;
n2 = (n % 1000)/100;
n3 = (n % 100)/10;
n4 = (n % 10);
```

n0	6
n1	5
n2	5

n3	3
n4	5

We can then store these the digits into the corresponding LCD variable directly with our LCD_Num array. As shown below.

```
    LCDM6 = LCD_Num[n0]; //stores value of n0 in BIT4 of LCD
    LCDM4 = LCD_Num[n1]; //stores value of n1 in BIT3 of LCD
Given by instructor:
    LCDM19 = LCD_Num[n2]; //stores value of n2 in BIT2 of LCD
    LCDM15 = LCD_Num[n3]; //stores value of n3 in BIT1 of LCD
    LCDM8 = LCD_Num[n4]; //stores value of n4 in BIT0 of LCD
```

```
//Christopher Badolato
//10/14/2019
//LAB 5.1
//EEL 4742L-0011

#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, 0xE4,
0xFF, 0xF7};

int main(void) {
    volatile unsigned int n;

    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    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();

        // The line below can be used to clear all the segments
        //LCDCMEMCTL = LCDCLRM; // Clears all the segments
        // Display 430 on the rightmost three digits
    n = 65535;
    display_num_lcd(n);
```

```
        // Flash the red and green LEDs
    for(;;) {
        for(n=0; n<=50000; n++) {} // Delay loop
            P1OUT ^= redLED;
            P9OUT ^= greenLED;
    }

}
    //*************************************************************
    // 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;
}

void display_num_lcd(unsigned int n){

    unsigned int n0 = 0, n1 = 0, n2 = 0, n3 = 0, n4 = 0;
        //Get each digit of the decimal value
    n0 = (n % 100000)/10000;
    n1 = (n % 10000)/1000;
    n2 = (n % 1000)/100;
    n3 = (n % 100)/10;
    n4 = (n % 10);
```

```
        //Flash the corresponding LCD lines.
    LCDM6 = LCD_Num[n0];
    LCDM4 = LCD_Num[n1];
    LCDM19 = LCD_Num[n2];
    LCDM15 = LCD_Num[n3];
    LCDM8 = LCD_Num[n4];
}
```

Part 5.2: ...

In this part of the experiment we are introducting a timer interrupt that will display a incrementing timer on the LCD with 1 second periods. On each timer intterupt we will be incrementing a global varible n that will be displayed on the LCD each second starting at 0. This code is very similar to the code for 5.1 but we include an initialized timer_A to up_mode and set TARR0 to (32768-1) to ensure 1 second intervals between LCD and n incrememnts. The n value will then be displayed on the LCD using our previously use displayLCD function. To keep our timer value (n) which is limited to 16-bits (65535), if our value for n exceeds 65535 we automatically reset it to zero rolling back the display. Because we are uising Timer_A in the program we can use the low power mode 3.

```
//Christopher Badolato
//10/14/2019
//LAB 5.2
//EEL 4742L-0011

#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);
void config_ACLK_to_32KHz_crystal();

// 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, 0xE4,
0xFF, 0xF7};
volatile unsigned int n = 0;

int main(void) {

    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    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
        //Configure ACLK to the 32 KHz crystal
```

```
    Initialize_LCD();
    config_ACLK_to_32KHz_crystal();

        // Configure Channel 0 for up mode with interrupt
    TA0CCR0 = (32768-1); // Fill to get 1 second @ 32 KHz
    TA0CCTL0 |= CCIE; // Enable Channel 0 CCIE bit
    TA0CCTL0 &= ~CCIFG; // Clear Channel 0 CCIFG b
        //initialize clock and LCD
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    display_num_lcd(n);
        // Infinite loop... the code waits here between interrupts
    _low_power_mode_3();

}
//******* Writing the ISR *******
#pragma vector = TIMER0_A0_VECTOR // Link the ISR to the vector

__interrupt void T0A0_ISR() {

        // Toggle both LEDs
    P9OUT ^= greenLED;
    P1OUT ^= redLED;
        //Cap our timer at 65535.
    if(n == 65536){
        n = 0;
    }
    n++;
    display_num_lcd(n);

        // Clear the TAIFG flag
    TA0CCTL0 &= ~CCIFG;
}


    //*********************************************************
    // 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;
}

void display_num_lcd(unsigned int n){

    unsigned int n0 = 0, n1 = 0, n2 = 0, n3 = 0, n4 = 0;
        //Get each digit of the decimal value
    n4 = (n % 100000)/10000;
    n3 = (n % 10000)/1000;
    n2 = (n % 1000)/100;
    n1 = (n % 100)/10;
    n0 = (n % 10);
        //Flash the corresponding LCD lines.
    LCDM6 = LCD_Num[n4];
    LCDM4 = LCD_Num[n3];
    LCDM19 = LCD_Num[n2];
    LCDM15 = LCD_Num[n1];
    LCDM8 = LCD_Num[n0];
}

//*********************************
// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal(){
        // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
        // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
        // Wait until the oscillator fault flags remain 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;
}
```

Part 5.3.

Part 5.3 we are implementing a set reset button, as well as a clear display button to the LCD and timer to create a stopwatch. To do this we first must initialize the buttons and clock and initialize the LCD to 000000. Once we encounter an interrupt, which in this case will be once every .1 seconds (3277-1) if button S1 is pressed a global variable s1Pushed will be toggled. This value will represent whether the n value (Value to be displayed) is incremented or paused. The timer will be continuously running, but our value for n will be paused or un-paused and displayed on the LCD as requested. If s1Pushed is 0 (default), n, the display and timer will be incrementing. If s1Pushed is 1 the n, and the display will be paused but the timer_A will still continuously run.

```
    if(s1Pushed == 0){
        n++;
    }
    if(s1Pushed == 1){
        n = n;
    }
```

To add a reset to zero we simply check if s2 is pressed then reset n to zero.

```
    if((P1IN & BIT2) == 0){
        n = 0;
    }
```

```
//Christopher Badolato
//10/14/2019
//LAB 5.3
//EEL 4742L-0011

#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);
void config_ACLK_to_32KHz_crystal();

const unsigned char LCD_Num[10] = {0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE4,
0xFF, 0xF7};
volatile unsigned int n = 0;
volatile int s1Pushed = 0;


int main(void) {

    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
    P1DIR |= redLED; // Pins as output
    P9DIR |= greenLED;

    P1OUT &= ~redLED; // Red off
    P9OUT |= greenLED; // Green on
```

```
        // Configure button s1
    P1DIR &= ~BIT1; // Direct pin as input
    P1REN |= BIT1; // Enable built-in resistor
    P1OUT |= BIT1; // Set resistor as pull-up

        // Configure button s2
    P1DIR &= ~BIT2; // Direct pin as input
    P1REN |= BIT2; // Enable built-in resistor
    P1OUT |= BIT2; // Set resistor as pull-up

        // Initializes the LCD_C module
        //Configure ACLK to the 32 KHz crystal
    Initialize_LCD();
    config_ACLK_to_32KHz_crystal();

        // Configure Channel 0 for up mode with interrupt
    TA0CCR0 = (3277-1); // Fill to get .1 second @ 32 KHz
    TA0CCTL0 |= CCIE; // Enable Channel 0 CCIE bit
    TA0CCTL0 &= ~CCIFG; // Clear Channel 0 CCIFG b
        //config clock and display 0 on LCD
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    display_num_lcd(n);
        //Low power mode 3 to shut timer off between use
    _low_power_mode_3();

}

//******* Writing the ISR *******
#pragma vector = TIMER0_A0_VECTOR // Link the ISR to the vector

__interrupt void T0A0_ISR() {
        //if s1 is pushed, toggle LED's Toggle continuous mode flag.
    if((P1IN & BIT1) == 0){
        s1Pushed ^= BIT1;
        P9OUT ^= greenLED;
        P1OUT ^= redLED;
    }
        //if s1 is pressed and continuous flag is 0 we are in incrememnting the
timer.
    if(s1Pushed == 0){
        n++;
    }
        //if s1 is pressed and continuous flag is 1, the timer is paused.
    if(s1Pushed == 1){
        n = n;
    }
        //if s2 is pressed reset the LCD to 0
    if((P1IN & BIT2) == 0){
        n = 0;
    }
        //if timer passes 65535 then we reset to 0.
    if(n == 65536){
        n = 0;
    }
```

```
        //display n value on LCD
    display_num_lcd(n);
        // Clear the TAIFG flag
    TA0CCTL0 &= ~CCIFG;
}


    //*************************************************************
    // 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;
}

void display_num_lcd(unsigned int n){

    unsigned int n0 = 0, n1 = 0, n2 = 0, n3 = 0, n4 = 0;
    //Get each digit of the decimal value
    n0 = (n % 100000)/10000;
    n1 = (n % 10000)/1000;
    n2 = (n % 1000)/100;
    n3 = (n % 100)/10;
    n4 = (n % 10);
        //Flash the corresponding LCD lines.
    LCDM6 = LCD_Num[n0];
```

```
    LCDM4 = LCD_Num[n1];
    LCDM19 = LCD_Num[n2];
    LCDM15 = LCD_Num[n3];
    LCDM8 = LCD_Num[n4];
}

//*********************************
// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal(){
        // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
        // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
        // Wait until the oscillator fault flags remain 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;
}
```

Student Q&A

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 off, just like the colored LEDs”. Explain whether it’

False, the segments are not turned on continuously, they will burnout if so. They are multiplexed to oscillate fast enough that our eyes cannot notice. The register variables such as `LCDM6` LCD control `LCDCCTL0 = LCDDIV__1 | LCDPRE__16 | LCD4MUX | LCDLP;` take care of this multiplexing.

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?

ADKOM model FH-1138P, has 108 segments
The Module is located in the 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 multiplexing
One pin from the microcontroller controls four segments of the display.

Conclusion

In this Lab we learn how to configure and control the LCD display to show at most a 16-bit binary value. First, we displayed a hardcoded value on display. Next, using the timer, we increment this value on the display starting from zero reaching a maximum of 65535 (Unsinged 16-bit int max value) then rolling back to zero. Finally, we used the timer created in the previous version to implement a stopwatch. Using buttons s1 to set and reset the count and button s2 to reset the timer to in either incrementing or paused modes.