Lab 5 – LCD Display

# Youssef Samwel

# [yo800238@ucf.edu](mailto:yo800238@ucf.edu)

# EEL4742C Embedded Systems

Prof. Dr. Zakhia Abichar - Section 00419

# 2/22/2024



# **Project Description**

In this lab, we will learn how to use the segmented LCD display.

# **2.0 Experiment Code**

#include <msp430fr6989.h>

#include <stdint.h>

#include <stdbool.h>

#include <math.h>

#define BUTTON1 BIT1

#define BUTTON2 BIT2

#define redLED BIT0   // Red LED at P1.0

#define greenLED BIT7 // Green LED at P9.7

void Initialize\_LCD();

void config\_ACLK\_to\_32KHz\_crystal();

static struct

{

    int8\_t mode;

    int8\_t interruptCounter;

    uint32\_t seconds;

    bool flashColon;

} ApplicationState;

// A, B, C, D, E, F, G, M

static uint8\_t patterns[] = {

    0b11111100, // 0, A, B, C, D, E, F

    0b01100000, // 1, B, C

    0b11011011, // 2, A, B, D, E, G, M

    0b11110011, // 3, A, B, C, D, G, M

    0b01100111, // 4, B, C, F, G, M

    0b10110111, // 5, A, C, D, F, G, M

    0b10111111, // 6, A, C, D, E, F, G, M

    0b11100000, // 7, A, B, C

    0b11111111, // 8, A, B, C, D, E, F, G, M

    0b11110111  // 9, A, B, C, D, F, G, M

};

static volatile uint8\_t \*regs[] = {&LCDM8, &LCDM15, &LCDM19, &LCDM4, &LCDM6, &LCDM10};

void LightRedLED(bool state)

{

    if (state)

        P1OUT |= redLED;

    else

        P1OUT &= ~redLED;

}

void LightGreenLED(bool state)

{

    if (state)

        P9OUT |= greenLED;

    else

        P9OUT &= ~greenLED;

}

void ToggleRedLED() { P1OUT ^= redLED; }

void ToggleGreenLED() { P9OUT ^= greenLED; }

void DisplayTime(uint32\_t totalSeconds, bool locked);

bool IsButton1Pressed()

{

    return (~P1IN & BUTTON1) ? true : false;

}

bool IsButton2Pressed()

{

    return (~P1IN & BUTTON2) ? true : false;

}

void ClearDisplay()

{

    LCDCMEMCTL = LCDCLRM; // Clear all segements

}

void lcd\_write\_uint16(uint16\_t number)

{

    ClearDisplay();

    // In-case the user wants to display 0

    if (number == 0)

    {

        \*regs[0] = patterns[0];

        return;

    }

    uint8\_t digitIdx = 0;

    while (number)

    {

        uint8\_t digit = number % 10;

        number /= 10;

        \*regs[digitIdx] = patterns[digit];

        digitIdx++;

    }

}

#pragma vector = TIMER0\_A0\_VECTOR

\_\_interrupt void ISR\_T0A0()

{

    ApplicationState.interruptCounter++;

    bool secondPassed = false;

    if (ApplicationState.interruptCounter > 30)

    {

        ApplicationState.interruptCounter = 1;

    }

    switch (ApplicationState.interruptCounter)

    {

    case 10:

    case 20:

    case 30:

        if (ApplicationState.mode < 3)

        {

            ApplicationState.seconds++;

            secondPassed = true;

        }

        break;

    }

    if (ApplicationState.mode == 1 && secondPassed)

    {

        lcd\_write\_uint16(ApplicationState.seconds);

    }

    else if (ApplicationState.mode == 2)

    {

        if (secondPassed)

            DisplayTime(ApplicationState.seconds, false);

        if (IsButton2Pressed())

        {

            TA0CCR0 = 50;

            ApplicationState.seconds += IsButton1Pressed() ? -1 : 1;

            if(ApplicationState.seconds == UINT32\_MAX) {

                ApplicationState.seconds = 86399;

            }

            DisplayTime(ApplicationState.seconds, false);

        }

        else

        {

            TA0CCR0 = 3276;

        }

    }

    else if (ApplicationState.mode == 3)

    {

        if (ApplicationState.interruptCounter >= 25 && IsButton1Pressed())

        {

            // clear time

            ApplicationState.seconds = 0;

        }

        // present locked mode

        DisplayTime(ApplicationState.seconds, true);

    }

    if(ApplicationState.seconds >= 86400) {

        ApplicationState.seconds = 0;

    }

}

#pragma vector = PORT1\_VECTOR

\_\_interrupt void ISR\_PORT1()

{

    static bool firstRelease = false;

    if (ApplicationState.mode == 1)

    {

        if (P1IFG & BUTTON1)

        {

            ApplicationState.seconds = 0;

        }

        if (P1IFG & BUTTON2)

        {

            ApplicationState.seconds += 1000;

        }

        lcd\_write\_uint16(ApplicationState.seconds);

    }

    else if (ApplicationState.mode == 2)

    {

        if (P1IFG & BUTTON1 && !IsButton2Pressed())

        {

            ApplicationState.mode = 3;

            ApplicationState.interruptCounter = 0;

            DisplayTime(ApplicationState.seconds, true);

            firstRelease = true;

            P1IES &= ~BUTTON1;

        }

    }

    else if (ApplicationState.mode == 3)

    {

        if (P1IFG & BUTTON1 && !firstRelease && !IsButton2Pressed())

        {

            // exit mode 3

            ApplicationState.mode = 2;

            ApplicationState.interruptCounter = 0;

            DisplayTime(ApplicationState.seconds, false);

            P1IES |= BUTTON1;

        }

        firstRelease = false;

    }

    \_\_delay\_cycles(1e5);

    P1IFG &= ~(BUTTON1 | BUTTON2);

}

void ConfigureOneTenthSecondTimer()

{

    TA0CCR0 = 3276;

    TA0CCTL0 |= CCIE;

    TA0CCTL0 &= ~CCIFG;

    TA0CTL = MC\_\_UP | ID\_\_1 | TASSEL\_\_ACLK | TACLR | CCIE;

    \_enable\_interrupts();

}

void ImplementingCounter5\_2()

{

    ConfigureOneTenthSecondTimer();

    ApplicationState.mode = 1;

    ApplicationState.seconds = 0;

    lcd\_write\_uint16(0);

}

void DisplayTime(uint32\_t totalSeconds, bool locked)

{

    ClearDisplay();

    int32\_t hours = 0;

    int32\_t minutes = 0;

    int32\_t seconds = 0;

    hours = totalSeconds / 3600;

    minutes = (totalSeconds - (hours \* 3600)) / 60;

    if (minutes < 0)

        minutes = 0;

    seconds = totalSeconds - (hours \* 3600 + minutes \* 60);

    if (seconds < 0)

        seconds = 0;

    uint8\_t lowerDigit = 0;

    uint8\_t upperDigit = 0;

    // display seconds

    lowerDigit = seconds % 10;

    upperDigit = (seconds / 10) % 10;

    \*regs[0] = patterns[lowerDigit];

    \*regs[1] = patterns[upperDigit];

    // display minutes

    lowerDigit = minutes % 10;

    upperDigit = (minutes / 10) % 10;

    \*regs[2] = patterns[lowerDigit];

    \*regs[3] = patterns[upperDigit];

    // display hours

    lowerDigit = hours % 10;

    upperDigit = (hours / 10) % 10;

    \*regs[4] = patterns[lowerDigit];

    \*regs[5] = patterns[upperDigit];

    LCDM3 = 0x8;

    LCDM7 = ApplicationState.flashColon ? 0x4 : 0x0;

    LCDM20 = 1;

    ApplicationState.flashColon = !ApplicationState.flashColon;

    if (locked)

    {

        LCDM7 = 0x4;

        LCDM3 = 0x1;

    }

}

void ApplicationUtilityChronometer\_5\_3()

{

    ConfigureOneTenthSecondTimer();

    ApplicationState.mode = 2;

    ApplicationState.seconds = 0;

    ApplicationState.flashColon = true;

    DisplayTime(0, false);

}

/\*\*

 \* main.c

 \*/

int main(void)

{

    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

    PM5CTL0 &= ~LOCKLPM5;     // Enable the GPIO pins

    P1DIR |= redLED;   // Direct pin as output

    P9DIR |= greenLED; // Direct pin as output

    LightRedLED(false);

    LightGreenLED(false);

    // Set input direction for buttons

    P1DIR &= ~(BUTTON1 | BUTTON2);

    // enable pull-up resistors to avoid false triggers caused by static fields

    P1REN |= BUTTON1 | BUTTON2;

    // pull high buttons

    P1OUT |= BUTTON1 | BUTTON2;

    // interrupt on falling edge

    P1IES = BUTTON1 | BUTTON2;

    // reset interrupt flags

    P1IFG &= (BUTTON1 | BUTTON2);

    // enable PORT1 interrupts

    P1IE |= BUTTON1 | BUTTON2;

    Initialize\_LCD();

    config\_ACLK\_to\_32KHz\_crystal();

    ApplicationState.interruptCounter = 0;

    ApplicationUtilityChronometer\_5\_3();

    \_low\_power\_mode\_3();

    return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 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;

}

// 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;

}

# **3.0 Student Q&A**

1. What is the maximum duration your chronometer application supports? Explain.

23 hours, 59 minutes and 59 seconds because that is the maximum amount of time in one day.

2. Explain whether this statement is true or false. If false, explain the correct operation. “An LCD segment works just like a colored LED. It’s turned on/off by writing digital high/low to it, respectively”.

No, this statement is incorrect. The LCD works by using a PWM signal to turn on the LCD segments, this is because if the segments are constantly on it will damage the display. This also reduces power consumption.

3. What is the name of the LCD controller that interfaces the LCD display of our board? Is the LCD controller located on the display module or in the microcontroller?

LCD controller A, LCD controller is inside the microcontroller.

4. 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 MUX is used for the LCD. The MUX reduces the number of PINs used by the LCD displays which free’s more pins for other uses.

# **4.0 Conclusion**

In summary, this lab focused on utilizing a segmented LCD display for the MSP430 MCU. The implemented code successfully demonstrated implementation with the MSP430FR6989 microcontroller. The practical application included a chronometer with features like time display, stopwatch, and a locked mode. The lab fostered both practical skills and theoretical understanding, providing a comprehensive insight into LCD interfacing in embedded systems.