University of Texas at El Paso

Electrical and Computer Engineering Department

EE 3176 – Laboratory for Microprocessors I

Spring 2023

LAB 04

LCD Displays, SysTick Timer, and Port Interrupts

Goals:

- Learn how to interface a **16x2 LCD Display** with the MSP432.
- Build a simple 3 button stopwatch with: Start, Stop and Reset; stopwatch should measure minutes and seconds.
- Use port input interrupts.

Bonus:

Have you timer display milliseconds as well. +10
Add an LCD screen saver. 1. Clear the display; 2. select a random position in the LCD; 3. display each letter of your first name in the selected position, 4. hold each letter for about one second before displaying the next one, 5. after displaying your whole name go to step #1. +10

Pre Lab Questions:

- What is an interrupt?
- In which register could you check if an interrupt flag raised?
- Is it necesary to create an infinite loop when working with ISR?

Lab Guide

Keeping track of time is an important function in embedded design. You can perform precise timing tasks using peripherals of the MSP432. For this assignment, use the SysTick timer for keeping track of time in your design. Refer to Lab 3 for information on using the SysTick timer.

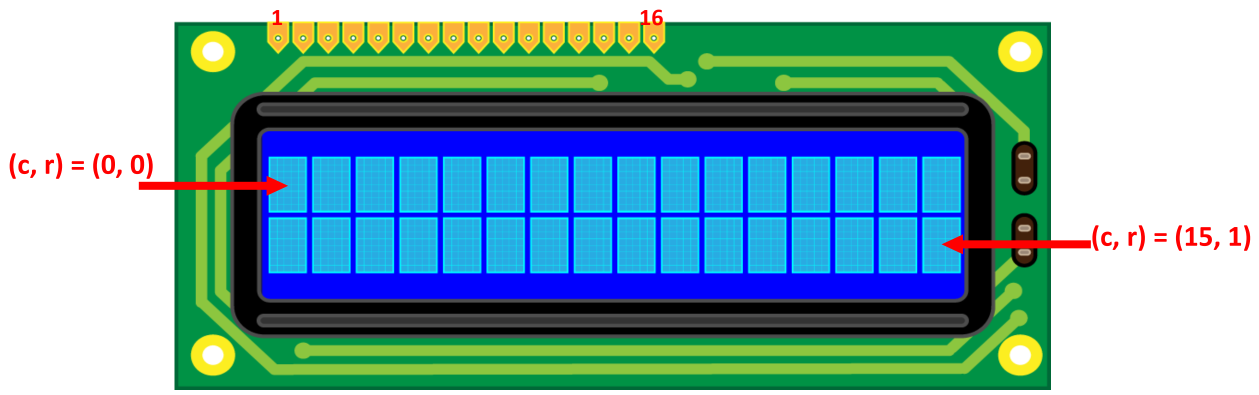
LCD Display

Embedded systems can provide information to users using a display device. In this laboratory you will learn how to use a liquid crystal display (LCD) device with a 16x2 character screen. The device has two rows and 16 columns of locations that can display a character. A driver library is available to use in your software to interface with the LCD screen. The most common LCD driver functions are the following:

```
lcdInit();                  // Initialize LCD Display
lcdClear();                 // Clear the LCD Display of
lcdSetText(string, c, r); // Print a string/character starting at column c and row r.
lcdSetInt(integer, c, r); // Print an integer starting at column c and row r.
delay_ms(x);                // Delay for x milliseconds.
```

Notes:

- Make sure that the **lcdLib_432.h** and **lcdLib_432.c** source files are in the same folder as your **main.c** source file in your **Code Composer Project**.
- Some **LCD Displays** only work with 5V, so your **LCD** Display might be dim.
- Clear the **LCD Display** before sending new information to display. Otherwise you will overwrite the current display contents and possibly mix data. For this Lab, you will be updating the display, so you can overwrite only some characters without clearing the whole display.
- You can connect the LCD VCC and backlight anode to a driver circuit to be controlled by a **Port Pin** if you want to be able to turn them on or off through software. However, doing so means that you will have to reinitialize the LCD Display in software every time it is powered on.



Pin Connections Table:

MICRO	LCD	
GND, 10k Potentiometer Output ($V_{\mp}$)	1	GND
VCC, 10k Potentiometer Output ($V_{\pm}$)	2	VCC
10k Potentiometer Output (V_o)	3	Contrast
P4.5	4	RS (Register Select): 0 – Command, 1 - Data
GND	5	R/W (Read/Write): 0 – Write, 1 - Read
P4.4	6	Clock Enable
Not Connected	7	Data 0
Not Connected	8	Data 1
Not Connected	9	Data 2
Not Connected	10	Data 3
P4.0	11	Data 4
P4.1	12	Data 5
P4.2	13	Data 6
P4.3	14	Data 7
100Ω Resistor to VCC	15	Backlight Anode (+)
GND	16	Backlight Cathode (-)

*Table 1 – Note that your LCD Display might have pins starting at 0; in that case subtract the above LCD pin numbers by 1. **Micro Pins** are labeled in **Red**; **LCD Pins** are labeled in **Blue**; **external connections in Black.***

Using the SysTick Timer and Port Interrupts

System Design

- Connect the LCD to your MSP432 Launchpad as indicated in the Pin Connection Table above.
- Run the Sample LCD program below to test your connections.
- To develop your stopwatch for this lab, use the Lab 3 SysTick timer interrupt service routine (ISR) to keep track of time.
- Keep timing information in global variables. For example, use **tics, secs,** and **mins** to keep track of how many times the t**imer ISR** was entered, elapsed seconds, and elapsed minutes, respectively.
- Set a SysTick timer cycle count **x** to enter the ISR and update the time-tracking variables. Make **x** an even divider of 3MHz. Using the fact that the DCO nominal frequency is 3MHz, use **tics** to compute seconds **(secs)** and **secs** to compute minutes **(min)**.

  tics = times the timer **ISR** was entered after **x** clock cycles

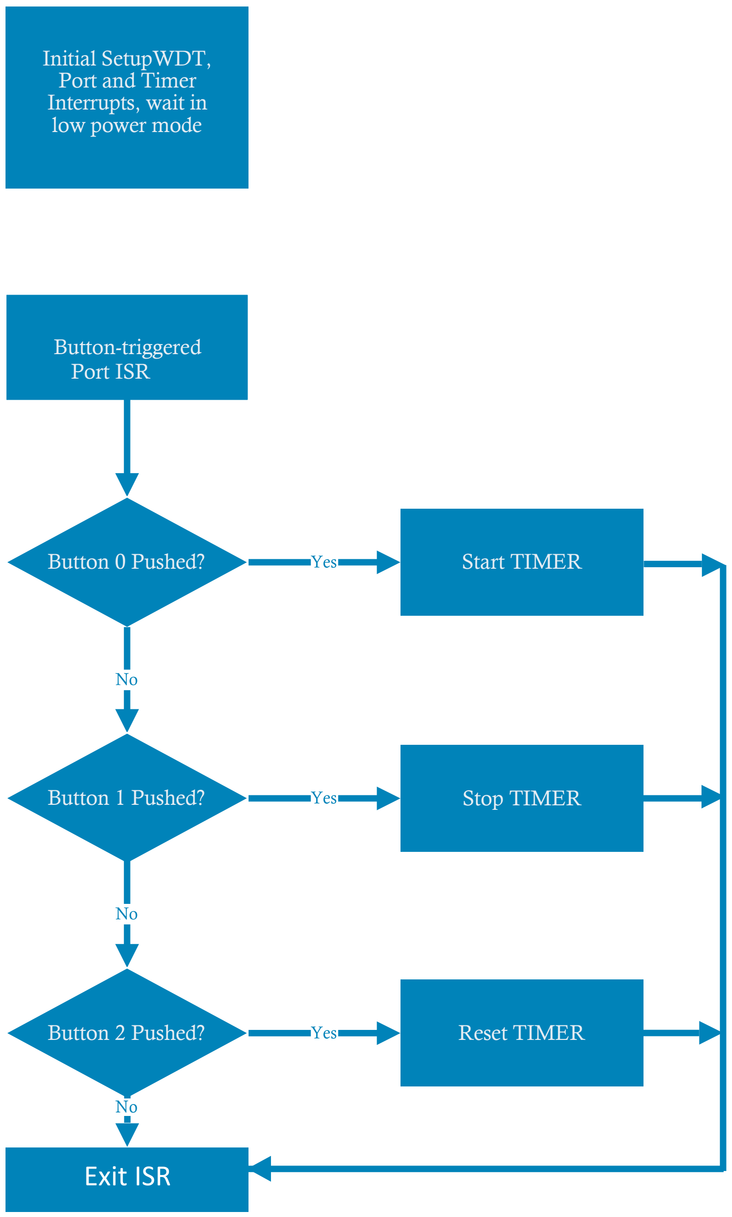
  secs = **tics** times **x** clock cycles/3 MHz, i.e., secs = tics * x /3e06.

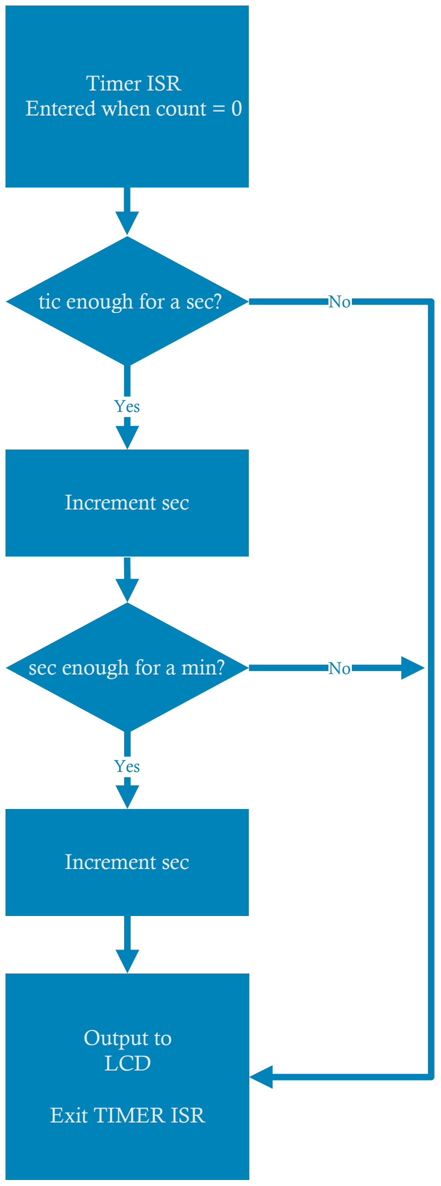
  mins = **secs/**60.

  For example, set the timer **ISR** to execute every **x**=30,000 clock cycles and increment **tics** every time the **ISR** executes. Then, **secs** must be incremented every time **tics** equals 100 (100 tics x 30,000 cycles = 1s); we can then increment **mins** every time **secs** equals 60.

- To control the stopwatch with buttons, as indicated the algorithm below, use the port interrupt example code after the sample LCD program.

System Algorithm Flowcharts





Sample LCD Program

```c
#include "msp.h"
#include "lcdLib_432.h"

/**
 * main.c
 */
void main(void)
{
        int i;
        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // stop watchdog timer
        lcdInit();
        lcdClear();
        lcdSetText("Hello!",0,0);
        i = 2000;
        lcdSetInt(i,0,1);
        delay_ms(20);
        while(1){
            lcdSetInt(i++,0,1);
            delay_ms(1000);
            if (i > 2020)
                i = 2000;
        }
 }
```

```c
//******************************************************************************
//  MSP432P401 Demo - Software Port Interrupt Service on P1.1 from LPM4
//
//  Description: MSP432 device is configured to enter LPM4 mode with GPIOs
//  properly terminated. P1.1 is configured as an input. Pressing the button
//  connect to P1.1 results in device waking up and servicing the Port1 ISR.
//  LPM3 current can be measured when P1.0 is output low (e.g. LED off).
//
//  ACLK = 32kHz, MCLK = SMCLK = default DCO
//
//               MSP432P401x
//            -----------------
//        /|\|                 |
//         | |                 |
//         --|RST              |
//     /|\   |                 |
//      --o--|P1.1         P1.0|-->LED
//     \|/
//
//   Dung Dang
//   Texas Instruments Inc.
//   Oct 2016 (updated) | November 2013 (created)
//   Built with CCSv6.1, IAR, Keil, GCC
//******************************************************************************
#include "msp.h"
int main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // Hold the watchdog

    // Configuring P1.0 as output and P1.1 (switch) as input with pull-up
    // resistor. Rest of pins are configured as output low.
    P1->DIR = ~(uint8_t) BIT1;
    P1->OUT = BIT1;
    P1->REN = BIT1;         // Enable pull-up resistor (P1.1 output high)
    P1->SEL0 = 0;
    P1->SEL1 = 0;
    P1->IES = BIT1;         // Interrupt on high-to-low transition
    P1->IFG = 0;            // Clear all P1 interrupt flags
    P1->IE = BIT1;          // Enable interrupt for P1.1

    // Enable Port 1 interrupt on the NVIC
    NVIC->ISER[1] = 1 << ((PORT1_IRQn) & 31);

    // Configure Port J
    PJ->DIR |= (BIT0| BIT1 | BIT2 | BIT3);
    PJ->OUT &= ~(BIT0 | BIT1 | BIT2 | BIT3);

    // Enable PCM rude mode to enter LPM3 without waiting for peripherals
    PCM->CTL1 = PCM_CTL0_KEY_VAL | PCM_CTL1_FORCE_LPM_ENTRY;
    // Enable global interrupt
    __enable_irq();

    // Setting the sleep deep bit
    SCB->SCR |= (SCB_SCR_SLEEPDEEP_Msk);

    // Do not wake up on exit from ISR
    SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;

    // Ensures SLEEPONEXIT takes effect immediately
    __DSB();

    // Go to LPM4
    __sleep();
}

/* Port1 Interrupt Service Routine */
void PORT1_IRQHandler(void)
{
    volatile uint32_t i;

    // Toggling the output on the LED
    if(P1->IFG & BIT1)
        P1->OUT ^= BIT0;

    // Delay for switch debounce
    for(i = 0; i < 10000; i++)

    P1->IFG &= ~BIT1; // Clear interrupt flag
}
```

Timer Interrupts

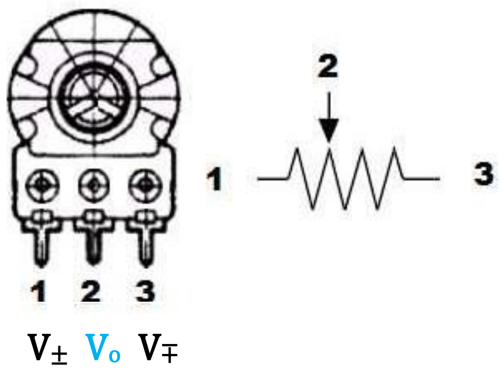
Frequently Asked Questions

Can I manage two or more timings at once?

Yes, you can manage as many different timings as needed using a combination of time-tracking variables and MSP432 timing peripherals and oscillators.

How do I connect a potentiometer!?

Checkout the diagram below if you are having trouble connecting the potentiometer:



The middle or isolated Pin (2) is always the 'output' pin. The left and right Pins (1 and 3) must each be connected to either a voltage source or ground.