Lab 04
Timer Interrupts

Goals:

- Learn about Timer Interrupts.
- Learn how to interface a 16x2 LCD Display with the MSP430.
- Build a simple 3 button stopwatch with: Start, Stop and Reset; stopwatch should measure minutes and seconds.

Bonus:

Have you timer display milliseconds as well. +10

Use Port Interrupts for the button functionality, milliseconds and add Lap Functionality (display on second line). +20

Pre Lab Questions:

- Compare and contrast using Timer Interrupts and simply using the Timer Count Register.
- What Timer Count Mode did you use?
- Which port did you use for your inputs; outputs?
- Did you use an infinite loop; why or why not?
An important ability in embedded design is being able to keep track of time. We can accomplish various precise timing tasks using the Timer peripheral of the MSP430.

**Timer**

The registers that control the Timer Interrupts are 16 bit. There are two timer modules; Timer A0 and Timer A1. Below are the pertinent registers:

- **TAxCTL** - Timer A Control; \( x = 0 \) or \( 1 \) (Timer A0 or Timer A1)
- **TAxCCTLy** - Timer A Capture/Compare Control
- **TAxCCRY** - Timer A Capture/Compare Module Register
- **TAXR** - Timer A Count Register
- **TAIV** - Timer A Interrupt Vector Value

Where \( x \) is the Timer A Module number, \( 0 \) or \( 1 \) (Timer A0 or Timer A1) and \( y \) is the Timer A Capture/Control Module number: \( 0 \), \( 1 \) or \( 2 \). For the MSP430G2553, we have 2 Timer A Modules with 3 Capture/Compare modules for a total of 8 interrupt trigger sources: 2 Timer Overflows and 6 Capture/Compare flags (there is no Capture function for Module 2 for Timer A0).

Shorthand to refer to a Timer module and its capture/compare submodule exists. For example, **TA0.1** refers to Timer A0 – Capture/Control Module 1. In addition, the MSP430G2553 header file has Control Bits defined for the Timer Control Registers. See the **Interrupts Guide** for more details.
LCD

The **LCD Display** that we will be using is a 16x2 character screen. This means that we have two rows and 16 columns in which we can display a character, for a total of 32 characters. A driver library has been provided for you to interface with the LCD screen right away. Below are the more useful functions that you will be using:

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

**Pin Connections:**

<table>
<thead>
<tr>
<th>MICRO</th>
<th>LCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND, 10k Potentiometer Output (V₁)</td>
<td>1</td>
</tr>
<tr>
<td>VCC, 10k Potentiometer Output (V₂)</td>
<td>2</td>
</tr>
<tr>
<td>10k Potentiometer Output (Vₒ)</td>
<td>3</td>
</tr>
<tr>
<td>P2.5</td>
<td>4</td>
</tr>
<tr>
<td>GND</td>
<td>5</td>
</tr>
<tr>
<td>P2.4</td>
<td>6</td>
</tr>
<tr>
<td>Not Connected</td>
<td>7</td>
</tr>
<tr>
<td>Not Connected</td>
<td>8</td>
</tr>
<tr>
<td>Not Connected</td>
<td>9</td>
</tr>
<tr>
<td>Not Connected</td>
<td>10</td>
</tr>
<tr>
<td>P2.0</td>
<td>11</td>
</tr>
<tr>
<td>P2.1</td>
<td>12</td>
</tr>
<tr>
<td>P2.2</td>
<td>13</td>
</tr>
<tr>
<td>P2.3</td>
<td>14</td>
</tr>
<tr>
<td>100Ω Resistor to VCC</td>
<td>15</td>
</tr>
<tr>
<td>GND</td>
<td>16</td>
</tr>
</tbody>
</table>

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.
Notes:

- Make sure that the `lcdLib.h` and `lcdLib.c` source files are in the same folder as your `main.c` source file in your Code Composer Project.
- Some LCD Displays are meant to work with 5V only, so your LCD Display might be dim.
- In later Labs, it is a good idea to clear the LCD Display before sending new information to display, otherwise you will write over what is currently on the display, resulting in possible mixed words/data. In the case of this Lab, we will be updating the current time elapsed, so we can overwrite the seconds or minutes characters only without clearing the whole display.
- You can connect the LCD VCC and backlight anode to a Port Pin if you want to be able to turn it on or off through software (remember to reinitialize the LCD Display in software).
You may poll the buttons in this lab but you must use a **Timer Interrupt** to keep track of time. You do not need to employ an **LPM** in this Lab (only for extra credit).

It is recommended that you use global variables, for example: `tic` to keep track of the number of times the **Timer ISR** was entered; `sec` to keep track of seconds and `min` to keep track of minutes.

Try and use a nice round number, `x`, such as 10,000 or 50,000, as the number of clock cycles to enter the **Timer ISR** (see the **Interrupts Guide** for details on how to do this). By default, you know that a clock cycle is 1µs because the micro CPU is running at 1MHz. From here, the `tic` will give you `sec`, and `sec` will give you `min`.

\[
\text{tic} = \text{number of times the Timer ISR was entered after } x \text{ clock cycles}
\]

\[
\text{sec} = \text{tic} \times x \text{ clock cycles}
\]

\[
\text{min} = \frac{\text{sec}}{60}
\]

For example, if we set the **Timer ISR** to trigger every 20,000 clock cycles, and increment `tic` every time the **Timer ISR** is triggered, we can increment `sec` every time `tic` equals 50 (50tics x 20,000cc = 1s); we can then increment `min` every time `sec` equals 60.

*See the next page for the Program Flow Chart*
INITIAL SETUP
(WDT, Ports, Timer, Interrupts)

Poll Buttons

Button 0 Pushed?
Yes → Start TIMER
No → Button 1 Pushed?

Button 1 Pushed?
Yes → Stop TIMER
No → Button 2 Pushed?

Button 2 Pushed?
Yes → Reset TIMER

Jump to TIMER ISR Depending on Value stored in TACCR0

TIMER ISR

tic enough for a sec?
Yes → Increment sec
No → sec enough for a min?

sec enough for a min?
Yes → Increment sec
No → Output to LCD
Exit TIMER ISR
Sample LCD Program

Use the sample program to test the LCD Display:

```c
#include <msp430g2553.h>
#include "lcdLib.h"

void main(void) {
  WDTCTL = WDTPW + WDTHOLD; // Stop Watchdog
  lcdInit(); // Initialize LCD

  char* msg = "Livin' on the...";
  int i = 0;

  while(1){
    lcdClear();
    lcdSetText(msg, i, 0); // Main Print
    lcdSetText(&msg[16 - i], 0, 0); // Reprint what has 'fallen off'.
    delay_ms(500);
    i = ++i % 16;
  }
}
```
Timer Interrupt Code Framework

Use the code framework below to help you get started; you will need to read the **Interrupts Guide** to fill in the gaps:

```c
#include <msp430g2553.h>
#include "lcdLib.h"

unsigned short int tic = 0, sec = 0, min = 0; //Example of Global variables.

int main (void)
{
    //Setup
    WDTCTL = WDTPW + WDTHOLD; //Stop Watchdog Timer
    lcdInit(); // Initialize LCD

    //Setup Buttons
    P1DIR = ;
    P1REN = ; //Optional.
    P1OUT = ; //Optional.

    //Setup Timer A0
    TA0CTL = ; //Setup Timer/Start Timer
    TA0CCTL0 = ; //Enable interrupt on TA0.0
    TA0CCR0 = ; //Period of 50ms or whatever interval you like.

    _enable_interrupts(); //Enable General Interrupts. Best to do this last.

    while (1)
    {
    }
}

// Timer 0 A0 Interrupt Service Routine
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer0_A0_ISR( void)
{
    /* INCREMENT GLOBAL VARIABLES HERE:

        Will jump in here when TA0R reaches the value stored in TA0CCR0 during setup.
        Count the number of tics to equal a second, then increment seconds.
        Count the number of seconds to increment minutes. You get it.

        Don’t forget to update the LCD Display with the current time here if you are
        not doing that in the main loop.
    */

}
Timer Interrupts
Post Lab

Lab Notebook Questions:

• How do we start, stop or reset the Timer?
• What is the maximum number of clock cycles that we can count?
• What is the largest amount of time we can measure assuming we use only 3 registers; one for hours, minutes and seconds?
• What characters are we not able to display on the LCD screen?
• Explain the potentiometer and its other possible uses.
Can I manage two or more timings at once?

Yes, you can manage two different timings. The MSP430 has one **TimerA** peripheral but two **TimerA** modules. So in other words we have two different counters that we can use to keep track of time.

Will the two TIMERA modules interfere with each other?

As far as resources, although they can be made to share the same timer source (such as the CPU), they will not interfere with each other’s operation. However, each **Timer Module** and **Submodule** has its own priority. See the **Interrupts Guide** for more details.

How do I connect my potentiometer!?

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

![Potentiometer Diagram](image)

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.