Lab Assignment # 4: Simon Game with Sound

Due Date: Week of October 16th, 2023,
at noon on the day of your lab session
on myMason (Blackboard)

Objectives:

- Learn how to use timers to create sounds.
- Learn how to use timers to accomplish delays.
- Be introduced to more intricate ISR functionality.
- Lean usage of Low Power Modes.

Parts:

Please make sure that you have the following parts before you start the lab.

- MSP-EXP430FR6989 LaunchPad
- 4 push buttons
- Pin headers
- 4 LEDs: red, yellow, green, blue
- 4 1 kΩ resistors
- Piezo Buzzer

Introduction:

Lab 4 will be an extension of Lab 3, by adding sound to your Simon game using a Piezo Buzzer. In this lab we will use timers to play different notes on a buzzer and to accomplish all required delays.

You will need to connect the four external LEDs, four buttons and the piezo buzzer to GPIO pins shown in Table 1. This table also shows which timer outputs the LEDs and the Piezo buzzer are connected to. If you have already completed the Simon game, then you only have to connect the Piezo buzzer.

In this lab, you will be introduced to a Piezo Buzzer Fig 1.

Background:

Now that you have mastered receiving a user's input via a standard push button, let's move onto something a little more interesting. For this lab, you will be given the task of generating a tone based on the buttons that was pressed and use low power mode.

Table 1: Port mappings for LEDs, the Buzzer, and Buttons

Device	**Type**	**Port**	**Timer**
LED	yellow	P2.6	TB0.5
LED	green	P2.7	TB0.6
LED	red	P3.6	TB0.2
LED	blue	P3.7	TB0.3
Buzzer		P1.5	TA0.0
Button	left	P2.1	
Button	up	P2.2	
Button	down	P2.3	
Button	right	P2.4	

Low Power Modes

Low power modes are used for:

Infrequent tasks: Rather than waste power running the CPU when the software is just waiting for an event, the software can put the CPU to sleep and wait for the event to trigger an interrupt.

Piezo Buzzer:

Piezo elements convert vibration to voltage or voltage to vibration. That means you can use this as a buzzer for making beeps, tones and alerts.

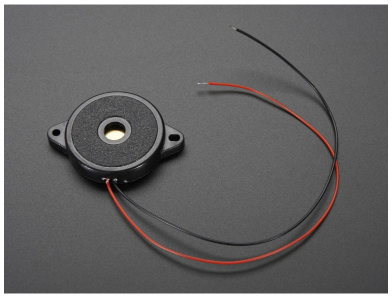

Figure 1: Piezo Buzzer

Code Walk Through:

The provided code for this lab shows how to make use of the MSP430's timer module to produce and output a waveform whose frequency can be changed by a user. Each timer module (*Timer A0*, *Timer B0*, etc.) contains a primary 16-bit counter (*TA0R*, *TB0R*, etc.) and multiple capture/compare (CC) registers (*TA0CCR0*, *TB0CCR1*, etc.). Each one of these CC registers is controlled by their own control register (*TA0CCTL0*, *TB0CCTL1*, etc.) and provides useful functions.

For this lab, we will make use of *Output Modes*, which are specific to CC registers. Each CC register is connected, via a special function pin, to a typical GPIO pin. By loading certain 16-bit values into these CC registers, you will be able to determine the time at which each CC register's dedicated pin is HIGH or LOW. In the code below, the CC registers are being configured and connected to their specific I/O pins.

C code	Explanations

`P1DIR  |= BIT5;`
`P1SEL1 |= BIT5;`
`P1SEL0 |= BIT5;`
	Set pin 1.5 to be an output; Buzzer connect buzzer to TA0.0 which can be driven by outmode from TA0CCR0

Output Modes, or *OUTMODS*, allow CC registers to determine when specific pins (P1.5 in this case) are HIGH or LOW. Additionally, the values put into the CC registers must be relative to some minimum and maximum counter value. These 16-bit counters are capable of taking on any unsigned value from 0 to $2^{16}-1$. Another way to think of it is that these counters can take on 2^{16} or 65,536 different values. If the counters are being driven by a clock that is 32 kHz (or more precisely 32768 Hz $= 2^{15}$ Hz), it will take 2 seconds ($2^{16}[ticks] \cdot \frac{1[tick]}{2^{15}[ticks]} = 2^{16} - 2^{15} = 2\,\mathrm{s}$) for the counter to count from 0 to its maximum value. In the provided code, we will use a driving clock that is 1 MHz and, instead of counting from 0 to 65,536, we will count from 0 to whatever value is stored in TA0CCR0 using the counter mode *UP*. This is shown in the two code segments below.

C code	Explanations
`unsigned int note=851;`	
`TA0CCR0 = note;`	Set timer to "note" frequency

Before getting into the significance of *TA0CCR0*, it is important to pay some attention to how the provided code sets *Timer A0's* control register.

C code	Explanations
`TA0CTL = MC_0 | ID_0 | TASSEL_2 | TACLR;`	info. on pg 465 in the device family guide (slau367f)

TA0CTL is the control register that determines the functionality of the *Timer A0 module*. We configure the timer to be in "`MC_0`)" mode, which is the *stop* mode. As you will be attaching a piezo speaker to it, you would want the functionality to turn the sound off.

Button *S1* on the launchpad serves as the On/Off button for the sound. In its interrupt service routine we assign "`MC_1`)", i.e., the *up* mode to *TA0CTL* is , which tells the *TA0R* 16-bit counter to count from 0x0000 to *TA0CCR0* instead of 0x0000 to 0xFFFF. Therefore, the value in *TA0CCR0* now determines the period of time it takes for *TA0R* to count from its minimum to maximum value.

With *CCR0* (CC Register 0) set to 851, *TA0R* counts from 0 to 851 at a rate of 1 MHz. Coming back to *OUTMODS*, the code below shows that we are using *OUTMOD* Toggle. That means, that for one full timer period, i.e., for the time it takes to count from 0 to the value stored in *TA0CCR0*, the output on P1.5 is **on**, and for one full timer period it is **off**. Hence, the resulting clock period on P1.5 is twice the timer period.

C code	Explanations
`TA0CCTL0 = OUTMOD_4;`	Setup CCR 0 in outmode toggle

To save power, the processor is put to sleep by enabling low power mode.

C code	Explanations
`while(1)`	Make sure the processor stays asleep normally
`    __low_power_mode_2();`	Enter low power mode with SMCLK active

In-lab Exercise (week of October 2nd):

1. Plug in your MSP430 Launchpad and run lab4.c in CCS.
2. Connect the negative (black) lead from an oscilloscope to the MSP430's GND pin.
3. Connect the positive (red) lead from an oscilloscope to pin P1.5.
4. Press button S1 on the launchpad to start and stop the waveform generation.
5. Adjust the oscilloscope's settings until you are able to observe the waveform being generated.
6. What frequency does the waveform have?
7. Press buttons *Up* (P2.2) to increase the frequency and *Down* (P2.3) to decrease the frequency and observe the frequencies on the oscilloscope.
8. Now, connect the Piezo Buzzer to P1.5. Observe the change in tone when you press the buttons.
9. Demonstrate to the GTA that your buzzer makes a sound and that you can change the frequency using the buttons.
10. Explain to the GTA how the value in *TA0CCR0* corresponds to the frequency observed on the oscilloscope.

Exercises:

Table 2: LED / Button - Note Mapping

Event	Approximate Musical Note	Duration
Red Button Press	D#_4	While Button Pressed
Blue Button Press	G#_3	While Button Pressed
Green Button Press	G#_4	While Button Pressed
Yellow Button Press	B_3	While Button Pressed

1. Generate tones for each button press with the musical notes mentioned in table 2. The frequencies for all the musical notes are shown in table 3.

Table 3: Musical Note - Frequency Mapping

Musical Note	Frequency
G#_3	207.65 Hz
A_3	220.00 Hz
B_3	246.94 Hz
C_4	261.63 Hz
D_4	293.66 Hz
D#_4	311.13 Hz
E_4	329.63 Hz
F_4	349.23 Hz
G_4	392.00 Hz
G#_4	415.30 Hz
A_4	440.00 Hz
A#_4	466.16 Hz

2. Generate tones for each LED blink with the corresponding musical notes shown in table 2.
3. The Peizo Buzzer should play a tone with frequency 82Hz for 1.5 seconds when the user loses the game.
4. The Piezo Buzzer should play a winning melody using the notes and **durations** provided in table 4 when the user wins the game. Here all LEDs should blink at the same time for 0.5 second on and 0.25 second off until the melody is over. **Hint**: You may use another timer to blink the LEDs.
5. Replace all delay loops in your program with timers.
6. Modify your program so that it is in a low power mode whenever possible. For example, when waiting for a button press, all clocks should be off. When showing the sequence of LED lights, only the appropriate clocks should run.
7. **Extra Credit: (10 points)** Use a transistor amplifier to make the piezo speaker louder.

Assignment/Submission Instructions:

Please submit your code and report for Lab 4 on Blackboard, make sure you follow the lab report format uploaded on Blackboard. With respect to the exercises, you may put all of your answers/-comments in a separate section 'Questions' after the 'Conclusions' section.

1. If your code doesn't work with LFSR, then please mention so in your report.
2. The lab report must be uploaded in .pdf format.
3. Provide the formula you used for calculating the timer count values for TA0CCR0 for the notes.

Table 4: Winning Melody

Approximate Musical Note	Duration
A_3	0.125 second
C_4	0.125 second
D_4	0.25 second
D_4	0.25 second
D_4	0.125 second
E_4	0.125 second
F_4	0.25 second
F_4	0.25 second
F_4	0.125 second
G_4	0.125 second
E_4	0.25 second
E_4	0.25 second
D_4	0.125 second
C_4	0.125 second
C_4	0.125 second
D_4	0.5 second
A_3	0.125 second
C_4	0.125 second
D_4	0.25 second
D_4	0.25 second
D_4	0.125 second
F_4	0.125 second
G_4	0.25 second
G_4	0.25 second
G_4	0.125 second
A_4	0.125 second
A#_4	0.25 second
A#_4	0.25 second
A_4	0.125 second
G_4	0.125 second
A_4	0.125 second
D_4	0.5 second

4. Provide a table that extends table 3 with the TA0CCR0 values for all notes.

5. Please change your code such that the winning score is 5 for the video demonstration.

Points to be covered in the demo:

1. Show that an LED stays on and the corresponding note is played as long as the button for the LED is pressed. Demonstrate that this works for short and long presses and for all 4 LEDs.

2. Show that after the start of the game, the first element of a sequence is displayed and the corresponding note is played for the same duration as the LED is lit.
3. Show that when a sequence of LEDs is lit and the same LED lights up twice in a row, a short gap in the sound can be heard as the LED is briefly turned off.
4. Show that loosing results in the correct blinking pattern on the LEDs and sound (see Exercise 3).
5. Show that winning results in the correct blinking pattern on the LEDs and playing of the melody with correct duration for each note. (see Exercise 4).
6. Explain while showing the code how you replaced delay functions with timer functions.
7. Explain while showing the code how and under which conditions you put the MSP430 into sleep mode.
8. Explain how you computed the value for TA0CCR0 for the note `D#_4` at 311.13 Hz.