Lab Assignment

Your game will play a tune using the buzzer and flash the 4 multi-colored LEDs with the notes. The player will play by entering the notes flashed using the buttons. If the player can’t keep up by pressing the buttons in time with the song, he or she loses the game. This lab has several objectives, including gaining further experience with digital I/O and understanding the operations of the MSP430’s timing, and developing experience writing software which depends critically on the passage of time. You have two weeks to complete this lab, but it is important that you start right away—you will need to implement a number of components in order to complete the entire project.

Pre-lab Assignment

Remember that BOTH partners are required to do the prelab—this is an individual assignment.

Before coming to lab, you should do the following:

1. **READ THE ENTIRE LAB ASSIGNMENT!** When you come to lab, you should have a good understanding of the requirements. Write down any questions you may have to ask in lab.

2. Write a complete C function\(^1\) that configures and reads the 2 user buttons on the MSP430F5529 Launchpad board. Your function should return a value corresponding to the button pressed—it is up to you to decide how to do this (look at the function `readButtons()` in the demo project for an example). See the Launchpad\(^2\) User’s Guide (found on the course webpage under Docs & Resources), HW2, and the lecture on Digital I/O for more information on the buttons.

3. Write a complete C function to configure and light the two user LEDs on the MSP430F5529 Launchpad based on input argument of type char. An example prototype for this function would be as follows:

   ```c
   void configUserLED(char inbits); // Example Prototype
   ```

   The first two bits of this argument should set the state of the LEDs: if BIT0 of the argument is 1, LED1 is on, if BIT0 is 0, the LED is off; similarly, if BIT1 is 1, LED2 is on, and if BIT1 is 0, LED2 is off. Again, you can find information on the configuration of the LEDs (like their pin numbers) in the Launchpad User’s Guide (under Docs & Resources), HW2, and the Digital I/O lecture.

4. Each note in the song you play is defined by two components: the pitch of the note, and the duration the note should be played. In order to play a song, you will need to find a way to store both properties for each note. You will also need to find a way to map notes to LEDs such that the same note always lights the same LED. With only 4 multi-colored LEDs, the same LED will need to correspond to more than one note (this is the case in Guitar Hero, too). What data structure(s) will you use to store pitch, duration, and the corresponding LED for each note? What length songs will you eventually want to play? Given how you choose to save your notes, how much memory will what require?

---

\(^1\) For this part (and the next), your code does not need to compile, and may be handwritten or typed (though you should type it because you’ll use it later!)

\(^2\) That the “Launchpad” refers to the small, red board on the left side of to our lab board, which contains the MSP430F5529.
Lab Requirements

To implement your game, you are required to complete each of the following tasks. You do not need to complete these tasks in the order listed—in fact, it is recommended that you begin with the features discussed in the “How to Start” section of this assignment first! The goal of this is to make sure you have all of the core components of the lab—which are the digital I/O functions and the operation of the timer—before you write the entire main state machine.

Finally, think back to your work on the last lab. Is there anything you could have done differently from the beginning to make your life easier? Perhaps you used too many global variables, or you found your control flow was too complicated? Maybe you realized that naming every variable “steve” was a bad idea (it is). As you approach this lab, remember what you learned about software design to help you with this one! If you have conceptual questions on how to approach specific C programming or software design concepts, please feel free to ask us—we are happy to help!

System Requirements

1. At startup, the LCD should display a MSP430 Hero welcome message which instructs the user to press the Launchpad user button S1 to begin. When your system starts or the game resets (which is triggered by losing or pressing User button S2), the game should go back to this state.

2. After Launchpad user button S1 is pressed, the game should give a 3 second countdown before playing its song. The LCD should display the time 3, then 2, then 1, then GO. While this is happening, the user LEDs should also flash something like LED1 (on 3), then LED2 (on 2) then LED1 (on 1), and then all together on GO.
   In the final version of the lab, the countdown should be measured using your timer and NOT implemented using software delays. **Explain the benefits of using the timer in this situation in your report.**

3. To produce sound, you will use the simple ceramic buzzer on the lab board. The buzzer sounds when a periodic waveform is applied to it. In our template project, this is done by generating a pulse-width modulated (PWM) waveform using TimerB0.5. See the buzzerOn() and buzzerOff() functions given in peripherals.c for more information on this.

4. In order to play different notes using the buzzer, you will need to create a new buzzer function (or modify the one in peripherals.c) that accepts an input variable for the pitch. The Timer B CCR0 register controls the period of the PWM waveform—changing the period will change the pitch of the sound. Remember that frequency of a sound is 1/pulse width and that the period is the number of ACLK ticks. Below is a table of frequencies of an octave of musical notes in Hz. You will need to do some math to convert these notes (and more, if you like) to a number of ACLK disks. **Discuss your conversion of frequency in Hz to Timer B CCR0 settings in your report.**

5. Your song should be at least 16 notes long. With each note, one of the four multi-colored LEDs should flash on for the duration of the note. As noted in the pre-lab, the same note should always flash the same LED, but with only 4 LEDs, multiple notes may be mapped to a single LED. How will you control (and store) the duration of your notes? Will you do this within the buzzer function or the main game loop? Remember that you will need to be checking the buttons for user input while playing a note. **Explain your implementation for this and how it works in your report!**
6. Implement the game requiring the player to press one of the 4 lab board buttons corresponding to the LED flashed within the assigned duration of the note. You MUST use a hardware timer to measure this interval since you will need to be checking for button presses while your song plays. In your report, explain why software delays are not feasible in this situation and why you must control the note’s duration using your timer interrupts. In this lab, you will use Timer A2 for this; the resolution of the timer should be at least 0.005 sec. In your report, explain how you configured Timer A2 and why the resolution of the timer should be several times smaller than the duration of a note.

7. While a note is playing, you will need to poll the lab board buttons to see if they have been pressed, and then determine if the press was in time with the song. It is recommended that you start Timer A2 at the beginning of the song and that you compare the duration of the note against the global count of interrupts. For example, if a note with duration 700 starts at a timer count of 32400, then it should end when the timer count is 33100. This mode of operation is much simpler than starting and stopping the timer with each note. This is important enough that I will say it again: DO NOT copy the stopwatch example we used in lecture and start and stop the timer with each note. In your report, explain how you used your timer value (ie, the global counter) to control the note’s duration.

8. When a player presses a button, you should provide the user feedback to let them know if the proper note was hit. To do this, you can use the red and green user LEDs on the Launchpad board (LED1 and LED2): when the user is pressing the correct button for the current note, light up the green LED; when the user is pressing the incorrect button, light up the red LED. Explain how you determine if a player pressed the correct button in your report.

9. While the user is playing the game itself, note that you are NOT required to display anything on the LCD—ie, you are NOT required to display the typical “note highway” found in Guitar Hero games. Instead, we display the current note that needs to be pressed on the four multi-colored LEDs. Updating the LCD in time with the song is a challenging task—if you like, you may implement this for additional bonus points, but only try it after you have the rest of the lab working!

10. In Guitar Hero games, when the player falls behind, the game plays bad notes and eventually boos them off state. How will your players lose? Implement proper humiliation for losing. Explain your rules for scoring and losing and how you implemented them in your report.

11. Conversely, you should implement proper player congratulations for winning.

12. After losing or winning, or when the Launchpad user button S1 is pressed, the game should reset to the startup screen (as stated in the first requirement).

13. Write a high quality lab report using the instructions below. Your report should include a flow chart or other type of state diagram describing your game’s functionality (you may break your game into several diagrams if that shows your functionality better). All flow charts should be computer generated.

ECE2049: Lab 2
**Bonus Points:** Present your complete game to a member of the course staff to be nominated for the Best Game awards. At the end of the lab, games that go above and the requirements will receive a 15pt bonus.

How will you make your game better than the rest? Multiple songs? Lighting effects? Improved buzzer routines? Player difficulty levels? Remember, as always, you only have 8KiB of RAM and 128 KiB of flash.

*NASA went to the moon with less on-board computing. What can you do?*

Below is a table of an octave of notes and their frequencies. You can use these notes as components of your song—feel free to add more as well!

<table>
<thead>
<tr>
<th>Note</th>
<th>Frequency (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>440</td>
</tr>
<tr>
<td>B flat</td>
<td>466</td>
</tr>
<tr>
<td>B</td>
<td>494</td>
</tr>
<tr>
<td>C</td>
<td>523</td>
</tr>
<tr>
<td>C sharp</td>
<td>554</td>
</tr>
<tr>
<td>D</td>
<td>587</td>
</tr>
<tr>
<td>E flat (Eb)</td>
<td>622</td>
</tr>
<tr>
<td>E</td>
<td>659</td>
</tr>
<tr>
<td>F</td>
<td>698</td>
</tr>
<tr>
<td>F sharp</td>
<td>740</td>
</tr>
<tr>
<td>G</td>
<td>784</td>
</tr>
<tr>
<td>A flat</td>
<td>831</td>
</tr>
<tr>
<td>A</td>
<td>880</td>
</tr>
</tbody>
</table>

**Note:** You may wish to create `#define` statements for your notes. This is a good idea, but DO NOT define them with single-letter names, like so:

```c
#define A 440
#define C 523 // <-- Will not work!
```

It turns out that the name “C” is a macro in `msp430.h`. If you try and redefine this macro, you will receive *extremely cryptic* error messages. This is why single-letter globally-defined variable names are a really bad idea!

Instead, add some kind of prefix to your notes like this:

```c
#define NOTE_A 440
#define NOTE_C 523 // OK!
```
Some example songs

- Intro to “Smoke on the Water”: C D Eb, C D F Eb, C D Eb D C, …(repeats)
How to Start

This lab has a lot of components. What is the best way to begin working on them?

This is a two-week lab, meaning you have two full lab sections to work on it. While you may have two weeks, it is important that you get started right away.

This lab has a number of “smaller” components that involve manipulating the hardware (the buttons and LEDs) as well as the operation of the timer. We recommend that you start the lab by focusing on building and testing these functions to make sure they work properly. After that, you can write your main state machine with the confidence that your peripheral functions work, allowing you to spend the rest of your time debugging your game rather than the hardware.

In the first week’s lab section, we will not have discussed interrupts in lecture yet—this is typical. Until we learn interrupts, you can use the time to focus on the other parts of the assignment. During the first lab section, we recommend starting to work on the following components:

- With your partner, write the functions you sketched in the prelab for controlling the User Buttons and LEDs on the Launchpad. Make sure the course staff signs off your prelab to ensure that you have the right idea for how to do this. To test your functions, write a simple main() that does something like the following:
  - When User Button 1 (S1 on the Launchpad) is pressed, turn on User LED1 (Red)
  - When User Button 2 (S2 on the Launchpad) is pressed, turn on User LED 2 (Green)
  - When both user buttons are pressed, both LEDs should be on
  - Otherwise, the LEDs should be off

  Note that this simple main() function ignores all of the other lab requirements—it’s just a test program! You are not required to write a program like this, but it’s a good, simple way to make sure your LED functions work.

  It is recommend that you try to build and test these functions during lab so that you can ask for help!

- Discuss your strategies for storing the information for the song from the prelab with your partner. How will you store the note and duration of the song? It is up to you how to implement this. If you are new to C programming, you should look up the concepts of “structs” and “enumerated types” (enums). You are not required to use either of these constructs, but you may do so if you wish.

- Try to play some different notes using the buzzer. The requirements explain some information on how to do this—you’ll need to do some math! Until we learn interrupts, feel free to try and play a song using swDelay() to add duration to the notes, though remember that you cannot use this in the final version of the lab.

In the second week’s lab section, try to work on the following while you are in lab:

- Build and test the LED, button, and buzzer functions if you have not done so already
- Configure the timer with the appropriate resolution and make sure it generates interrupts. Make sure you test that your interrupts are handled—you can do this by setting a breakpoint in your ISR!
- Finally, think about the logic to play the song while using interrupts to track the duration of the note and determine if a user pressed the correct button at the correct time. Be sure to think about this part during your lab section.

As always, if you have any questions about how to implement any of the requirements, always feel free to ask for help! That’s why we’re here!
Coding Standards

The game in this lab requires some element of *real-time operation* in that it requires your code to respond to the keep track of the user’s button presses within a fixed timing constraint. This constraint requires that your code to implement this part of the game be efficient—if you do too much extra work, you will notice lag in responding to the user’s button presses. The goal of this is not only to teach you how to use interrupts, but to help you build your skills in writing clean, efficient code while being mindful of timing and resource constraints.

As such, all of the previous coding standards about good control flow from lab 1 still apply to the lab. You should use the skills you learned in the last lab about control flow to help you here. In particular, try to avoid using too many nested loops or duplicating lots of code—you should be able to implement the main portion of the game with a fairly simple structure.

**Note:** To encourage you to use good software design: you **MUST** store the song as some type of array—you may **NOT** hard-code the notes of the song with a bunch of *if* statements! This would be a very poor design, as you would need to change the code to change the song!

In addition, writing code with interrupts has some specific coding requirements: as stated in lecture, your Interrupt Service Routines (ISRs) should be kept short since they interrupt the main execution of your program. For this reason, you should avoid doing expensive operations from inside an ISR, such as the following:

- Drawing any text to the display
- Updating the LCD display using `GrFlush()`
- Waiting in a loop until something happens

If you find yourself wanting to perform any of these tasks in an ISR, you should rethink your program’s design. Please feel free to ask us for help on this!
Writing your Report

Since this lab was mainly a tutorial, the report does not need to be substantial—however, we are asking you to write one as practice for future reports. Your lab should be written in a professional style. It should be an electronically-prepared technical document like what you would submit to a fellow engineer or your boss. Only one report is required per lab team. The report should include:

- **Introduction** (1-2 paragraphs max): Succinctly state the objectives of the lab and give an overview of what you accomplished.

- **Discussion and results**: Discuss what you did in each part of the lab and how you solved any problems. Describe what you did and be sure to thoroughly answer and explain the questions asked in the lab assignment. In general, this section should be as long as necessary to say what you need—no padding or fluff!

- **Summary and Conclusion** (1-2 paragraphs max): Summarize what you accomplished in the lab and what you learned. This should be a “bookend” to the introduction.

- **Appendices**: You should not need any in this lab. **DO NOT PASTE YOUR CODE INTO THE END OF THE LAB REPORT!** Instead, your code will be submitted as an archive file alongside your report, which is a lot cleaner!

Lab reports are important. In industry, the FIRST view of YOUR work by anybody other than your immediate supervisor will see will probably be in WRITING!

Learning to be an effective communicator of technical information is probably THE MOST IMPORTANT job skill you can have.

Thus, we care about lab reports. We read them. Really.

Submitting your Work

When you are done with your report, you will submit it and your code on Canvas for grading. In order to receive a grade, you must submit both your code and your report online—even though you did not write much code for this lab, we will start the submission process now. Only one member of your team needs to submit files for your lab.

In addition, you must turn in your signoff sheet to the course staff—usually, you will do this when receiving your last signoff. If not, you can turn it in by placing it in the box in the ECE office, or handing it to a member of the course staff.

To submit your code for grading, you will need to create a zip file of your CCS project so that the course staff can build it. You can also use this method to create a complete backup copy of your project (perhaps to send to your partner or save for later). Unfortunately, the only reliable method for doing this is **from inside CCS using the instructions below**—do NOT attempt to just create a zip file of your code. To export your code:

1. Inside CCS, right click on your project and select "Rename..."

2. If you are submitting your project, enter a name in the following format: `ece2049e16_lab2_username1_username2`, where username1 and username2 are the user names of you and your partner. (NOTE: Failure to follow this step will result in points deducted from your lab grade! If you don’t do it, it makes a lot of extra work for the graders!)
3. Click **OK** and wait for CCS to rename your project.
4. Right click on your project again and select "**Export...**" then select “General” and "Archive file" from the list and click **Next**.
5. In the next window, you should see the project you want to export selected in the left pane and all of the files in your project selected in the right pane. Select all. You should not need to change which files are selected.
6. Click the **Browse** button, find a location to save the archive (like your R drive) and type in a file name using the EXACT SAME NAME used in Step (2).
7. Click "**Finish**". CCS should now create a zip file in the directory you specified.
8. Go to the Assignments page on the class Canvas website. Click on the assignment for Lab 0 and attach the archive file of your project that you just created and your report. When you are ready, hit the Submit button. Only one code and report submission is required per team.