**ECE2049 A-2022 Lab 2**

**Sign-off Sheet**

***Bonus Sign-off***: **Friday 9/23/22 *Report due***: **Tuesday 9/28/22**

**Student 1:** Cristobal Rincon Rogers **Lab Board #**: 72

**Student 2:** Lili Loughlin

| ***Pre-lab*** (students receive points individually)  Signed paper submissions to be attached to lab report | 10    10 | Student 1    Student 2 |
| --- | --- | --- |
| Power Up and Reset to Welcome mode (Display welcome message) | 5 |  |
| Timer A2 properly configured with resolution set to 5 ms or less | 10 |  |
| Countdown to start of game w/ Launchpad LEDs flashing using Timer A2 | 5 |  |
| Buzzer routines with pitch and duration (min of 8 pitches) | 10 |  |
| Playback of song at least 28 notes long using Timer A2 to measure note duration | 20 |  |
| Playing game with timing of button (4 buttons) presses using Timer A2 | 20 |  |
| Proper player humiliation on losing | 10 |  |
| Proper celebration for winning | 5 |  |
| Answer to TA Questions at Sign-off  (students receive points individually) | 5 pts    5 pts | Student 1    Student 2 |
| Report (answering ***all*** questions from the requirements section) | 50 |  |
| ***Total points*** | 150 |  |

***BOTH partners MUST be present for ALL sign-offs!!***

**TA’s signature:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Date:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Table of Contents

[**Table of Contents**](#_vfak4q6bqnai) **2**

[**Introduction**](#_sygs27tj9773) **3**

[**Discussion/Questions**](#_c2pcz1ltq3ki) **4**

[**Conclusion**](#_4tldmhpd6yql) **7**

[**Pre-Labs**](#_xgabbpjz9ok) **8**

# 

# Introduction

“Guitar Hero” is a popular and fun music video game released in 2005, giving users the opportunity to play familiar rock songs from the comfort of their homes. The goal is conceptually simple: press the notes on a peripheral I/O device that corresponds with specific flashing lights.

In this lab, our objective is to use the MSP430 board and Code Composer Studio (CCS) 10.4 to create a basic version of the game. The buzzer on the board plays a certain note that corresponds with one of the four LED lights. Then, the player must press the button associated with said LED for as long as the light flashes. If the player reaches 28 notes without making more than three errors, they win. This lab is necessary to gain experience with digital I/O devices, understand the operation and execution of MSP430 ‘s timers, and become more comfortable writing software dependent on time.

# Discussion/Questions

Figure 1: State Diagram for Guitar Hero Program

*2) In the final version, the countdown must be measured by Timer A2 and NOT implemented using software delays. Explain the difference between event (or interrupt) driven code and polling. Is your final code strictly event driven or will you use a mix of interrupts and polling? Explain in your report.*

Event driven code generates an interruption to the main code to run the interruption function the instant that the desired event happens. Polling on the other hand is the act of repeatedly checking for something/event. For some applications, the choice of using event driven code or polling is arbitrary. However, for some it is not. Event driven is considered to be superior for some applications because it ensures that you do not *miss* a given event. Whereas with polling, there is a chance that you might miss the event that you are looking for.

Our final code is not strictly event driven, but rather a mix of interrupts and polling. Interrupts are used for time dependent operations. Polling is used to determine button presses. The buttons are given a start timer count timestamp and an end timer count time stamp. The difference in between these determines the length of time that the button was pressed.

*4) You will need to do some math to convert these frequencies to the number of ACLK tics. Discuss your conversion of frequency in Hz to Timer B CCR0 settings in your report.*

To convert from notes to ACLK tics, we used the following function:

| noteInACLKTics = ( fACLK / fNote ) - 1; |
| --- |

The derivation for this function is as follows:

tNote = 1 / fNote;

tNote = ( MAX\_CNT + 1) \* ( 1 / fCLK )

1 / fNote = ( MAX\_CNT + 1) \* ( 1 / fCLK )

fCLK / fNote = MAX\_CNT + 1

fCLK / fNote - 1= MAX\_CNT

*5) How will you control the duration of your notes? Will you do this within the buzzer function or within the main game loop? Remember you will need to be checking buttons during the notes. Explain your choice in your report.*

Each note's data will be contained in a struct. This struct is called Notes and has the following fields: pitch, duration, and LED to light. The duration of a note is handled in the function called PlayNote, which also determines the interval the corresponding button is pressed. Each LED has a corresponding LED to light up, so we check the output of P6 and & it with the result of the readButtons() function which formats the input so that each corresponds to the correct LED (i.e button S1 corresponds to the first LED on our lab board).

We implemented a timer to check the elapsed time since the function playNote was called. And if the amount of time of the timer becomes greater than or equal to the note, then we know it is time to stop the note from playing on the buzzer.

*6) Explain in your report why software delay would then no longer work and why you must implement note duration using the timer interrupts.*

A software delay would no longer work because we would not be able to register button presses since the program would be suspended/stuck in the loops.

*7) Explain in your report how you setup Timer A2 and why Timer A2's resolution should be several times smaller than the duration of a note.*

To set up Timer A2, we had a function called startTimerA2() that configured the timer register [insert register name here] to 163 clock ticks for an interrupt. The resolution was set several times smaller than the duration of the note to improve the resolution of button presses and to achieve more accuracy in timing.

*8) Explain your rules for scoring and losing and how you implemented them in your report.*

In order to win the game, the player must play 28 notes of the song and make less than four errors. If the player holds the corresponding button for too long or too short of a period, or they press the wrong button, an error is added. If the player makes four errors before they play 28 notes, they lose the game. We implement this in our code by using a “checkError” state that specifically checks the number of mistakes made while the game is running. If the value is greater than or equal to 4, the state switches to “loser” and the string “LOL LOSER” is displayed on the screen.

# Conclusion

In conclusion, we designed and implemented a simple version of “Guitar Hero” on the MSP430 board using a state machine and timers. When one of the four LED lights flashes and a certain note is played, the player must press the corresponding button for the specific amount of time that the buzzer is on. If the player is able to reach 28 notes played without making four mistakes, they win the game.

# Pre-Labs

Cristobal Rincon Rogers

Pre Lab for Lab 2

ECE 2049

Lili Loughlin

Pre Lab for Lab 2

ECE 2049

2) Write a function to configure the 4 lab board buttons, S1 through S4 See the Lab Board

schematics, Homework 2 and Lectures 6-8 for information on digital IO and the buttons.

3) Write a function that returns the state of the lab board buttons with 1=pressed and 0=not

pressed. For example if S1 alone is pressed then the function should return 00000001b = 0x01. If only S3 is pressed, then the function should return 00000100b = 0x04. If both S1 and S3 are

pressed then the function should return 00000101b = 0x05, etc. Remember that the buttons are

not on contiguous I/O port pins. You will have to check each button individually and combine their states before returning from your function.

void initButtons(void)

{

//Configure buttons as outputs using internal pull up resistors

// Button S1: P7.0

P7SEL &= ~BIT0;

P7DIR &= ~BIT0;

P7REN |= BIT0;

P7OUT |= BIT0;

// Button S2: P3.6

P3SEL &= ~BIT6;

P3DIR &= ~BIT6;

P3REN |= BIT6;

P3OUT |= BIT6;

// Button S3: P2.2

P2SEL &= ~BIT2;

P2DIR &= ~BIT2;

P2REN |= BIT2;

P2OUT |= BIT2;

// Button S4: P7.4

P7SEL &= ~BIT4;

P7DIR &= ~BIT4;

P7REN |= BIT4;

P7OUT |= BIT4;

}

5) Write a complete C function to configure and light 2 user LEDs on the MSP430F5529 Launchpad board based on the char argument passed. If BIT0 of the argument = 1, LED1 is lit and if BIT0=0 then LED1 is off. Similarly, if BIT1 of the argument = 1, LED2 is lit and if BIT1=0 then LED2 is off. Again, see the MSP430F5529 Launchpad User's Guide (Useful Links), HW#2 and Lecture 8 for information on the user LEDs and functions configuring and the using other LEDs.

void led()

{

char val;

val = sw();

switch(val)

{

case 1:

P6OUT |= BIT0; //turn on LED1

P6OUT &= ~BIT1; //turn off LED2

break;

case 2:

P6OUT |= BIT1; //turn on LED2

P6OUT &= ~BIT0; //turn off LED1

break;

default:

P6OUT &= ~(BIT0 | BIT1); //turns both leds off

break;

}

}

4) In order to play a song, you will need to find a way to give each note of your song both pitch and duration. 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 but that is the case in Guitar Hero, too. What data structure(s) will you use to store pitch, duration and the corresponding LED? What length of songs will you eventually want to play? Given how you choose to save your notes, etc., how much memory will that require?

I think we should use int arrays to store pitch, duration, and the corresponding LED.