

Lab Report

|  |  |
| --- | --- |
| Name: Alexis Steven Garcia | Date: October 28, 2018 |
| Course: EGCP-450 | Lab #: 6 |

Grading Criteria:

|  |  |  |
| --- | --- | --- |
| **Section** | **Earned Points** | **Possible Points** |
| Problem/Objective: |  | 10 |
| Background: |  | 15 |
| Questions/Deliverables: |  | 15 |
| Program Code: |  | 30 |
| Demo: |  | 30 |
| Total: | 0 | 100 |

**PLEASE UPLOAD YOUR REPORT IN TITANIUM. NO PAPER REPORTS.**

Professor Comments:

# Problem/Objective

State the problem statement and/or objective of the lab. This must be a complete paragraph (i.e., at least 5 sentences).

The objective of this lab was to familiarize ourselves with a few topics. For example, we gained a better understanding of how digital-to-analog converters (DAC) work. Also, we were made aware of how digital data stored in computers can be used to represent sounds and music. In addition, we were briefed on how a DAC can function to create sounds. With this knowledge, our goal was to create a simple sound generation system that plays musical notes.

# Background

Briefly describe what you did in the lab including technical detail. It must be at least two **complete** paragraphs to receive full credit.

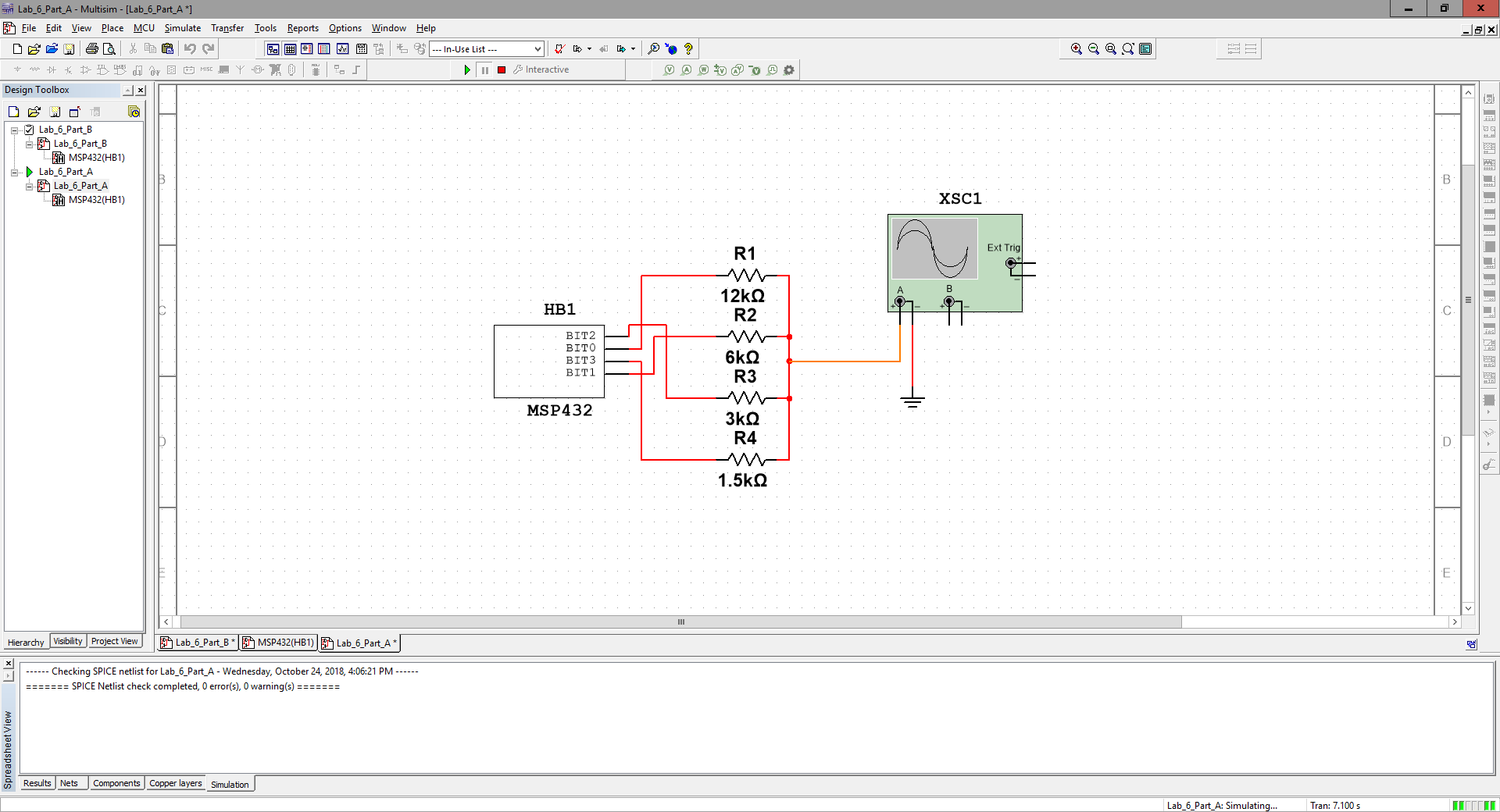
This lab assisted in understanding how to design a sound generation system that plays musical notes using a digital-to-analog converter (DAC). The DAC was first simulated on the Multisim program to give us an idea of how it operates. The schematic provided a four-bit output and we were made aware that we would have three switches to act as inputs. When connected to an oscilloscope, the simulation produced a staircase looking waveform. This waveform depicted output voltages equal to the number value of the input. In other words, each output voltage corresponded to a binary digital code such as “000” and up to “111”.

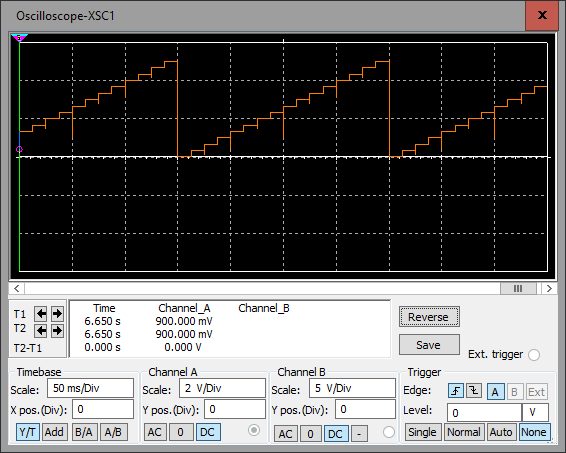
With the simulation working as intended, the following step was to build the circuit. The hardware included in this system included the MSP432, three switches, resistors, and a headphone audio connector. First, the three switches were connected to separate pins in port 5 to represent the input. Then, four outputs were branched from separate pins of port 4 and connected to one node. The node would then represent one single output and would be connected to the correct corresponding pin of the audio connector. At this point the configuration needed resistors set up in a specific pattern to finish the DAC. In other words, each individual output bit was supposed to be connected resistors of different values. The most significant output bit would need the smallest resistance of the four while the least significant bit would need the highest resistance. The configuration was set up in a way where the most significant bit was connected to a resistor that is [1/8](https://www.learnenglish.de/sounds/Numbers/1eighth.rm) of the resistance of the resistor connected to the least significant bit, the next bit was [1/4](https://www.learnenglish.de/sounds/Numbers/1quarter.rm) of the resistance, and the following bit was [1/2](https://www.learnenglish.de/sounds/Numbers/1half.rm) of the resistance. For example, the least significant bit had a 12kΩ resistor connected to it; therefore, the most significant bit would have a 1.5kΩ resistor connected, the next bit would have a 3kΩ connected, and the following bit would have a 6kΩ. Correspondingly, since the resistors we had were scarce, we would need to connect some of our resistors in series or parallel to get these values. Finally, the equipment was connected to ground as needed and the supply voltage was set up to be 3.3V.

The final step was to create driver software for the system and program the board. We were instructed to create device drivers for the DAC, the switches that would act as our “Piano”, and the sound driver that would take care of the playing the note. The DAC drivers’ function would be to output to the audio connector, the piano driver would allow us to get the input of the switches, and the sound driver would iterate through the sinewave given, playing the sound over a period that was calculated from the frequency of the sound we are trying to play. In other words, the period for the sounds we attempt to play are calculated from the following equation:. The final objective was to create a main loop that would handle which sound would play depending on the input.

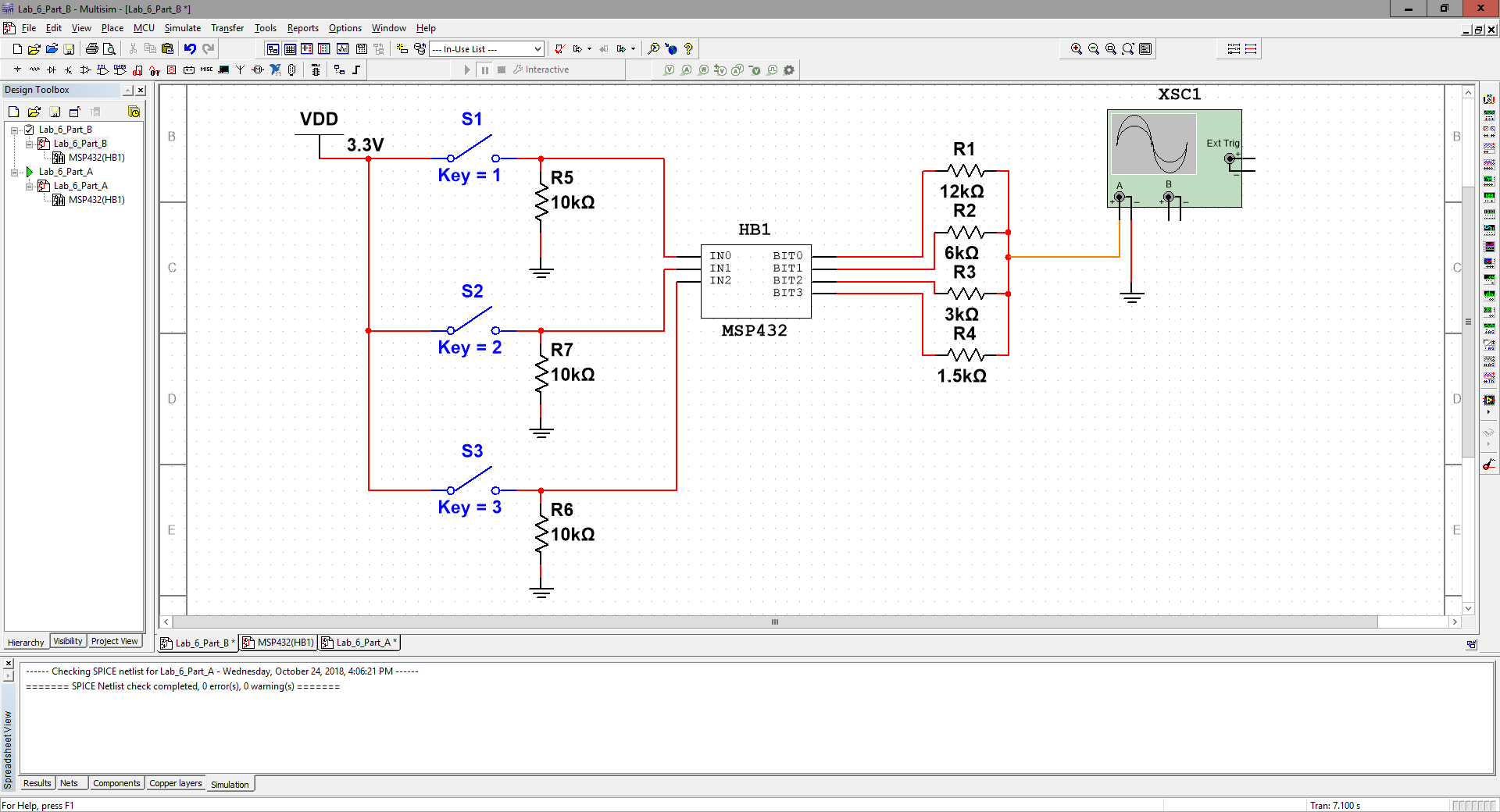
# Questions/Deliverables:

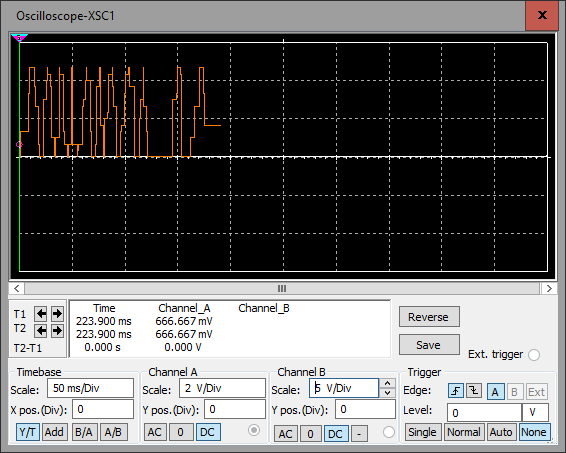
1. Screenshots from Part A of the instructions.





1. Screenshots from Part B of the instructions.





1. Using a voltmeter, complete the following table for the static performance evaluation of the DAC. Also, calculate resolution, range, precision and accuracy (show your work).

|  |  |  |
| --- | --- | --- |
| MSP432 Output Value (Decimal) | Theoretical DAC voltage | Measured DAC voltage |
| 0 | 0 | 0 |
| 1 | .22 | .224 |
| 2 | .44 | .432 |
| 3 | .66 | .662 |
| 4 | .88 | .894 |
| 5 | 1.1 | 1.117 |
| 6 | 1.32 | 1.330 |
| 7 | 1.52 | 1.554 |
| 8 | 1.76 | 1.744 |
| 9 | 1.98 | 1.967 |
| 10 | 2.2 | 2.179 |
| 11 | 2.42 | 2.402 |
| 12 | 2.64 | 2.631 |
| 13 | 2.86 | 2.854 |
| 14 | 3.08 | 3.066 |
| 15 | 3.3 | 3.289 |

1. This lab mentions 32 samples per cycle. Increasing the DAC output rate and the number of points in the table is one way of smoothing out the “steps” that in the DAC output waveform. If we double the number of samples from 32 to 64 to 128 and so on, keeping the DAC precision at 4-bit, will we keep getting a corresponding increase in quality of the DAC output waveform?

Yes, we will receive a corresponding increase in quality of the DAC output waveform because by increasing the number of samples, we decrease the period in which we cycle through each sample.

1. As you increase the number of bits in the DAC you expect an increase in the quality of the output waveform. If we increase the number of bits in the DAC from 4 to 6 to 8 and so on, keeping the number of points in the table fixed at 32, will we keep getting a corresponding increase in quality of the DAC output waveform?

Yes, increasing the number of input bits for a DAC will get a corresponding increase in quality of the DAC output waveform because the waveform depicts output voltages equal to the number value of the input. In other words, each output voltage corresponded to a binary digital code so instead of “000” up to “111”, we can get steps to represent from “0000” up to “1111” and “00000000” up to “11111111” with 4 and 8 inputs respectively.

# Program Code

Copy your code here. Please provide comments in your code. This will help me analyze your code and remove any ambiguity. **Provide your code as text, not as a screenshot/image**. Also, provide your flowchart here. Again, you can hand write your work, take a picture, and paste the image here. One app that I would suggest to easily do this is “CamScanner”.

**Main.c:**

**#include** <stdint.h>

**#include** "msp432p401r.h"

**#include** "SysTickInts.h"

**#include** "DAC.h"

**#include** "Piano.h"

**#include** "Sound.h"

**#include** "Led.h"

**void** **EnableInterrupts**();

**void** **DisableInterrupts**();

//Points per 1 wave length = 32

//Bus clock frequency = 3 MHz

//Bus clock frequency = 1/3 MHz = 333.33ns

//Frequency of wave = 1/Period of wave = 1/(333.33ns\*period(values defined below)\*32)

**#define** A 213 //440 Hz

**#define** F 269 //349 Hz

**#define** D 319 //294 Hz

**void** **main**(**void**){

/\*

\* 0 to 15 DAC output (0, 1, ..., 15, 0, 1, 2)

\* int Data = 0;

\*/

**unsigned** **long** input, previous;

Piano\_Init();

Sound\_Init();

LED\_Init();

EnableInterrupts();

previous = Piano\_In()&0x07;

**while**(1){

/\*

\* DAC\_Out(Data);

\* Data = 0x0F&(Data+1);

\*/

// Get input of switches

input = Piano\_In();

// If pressed -> No need to check

**if**(input&&(previous!=input)){

**switch**(input){

**case** 0x01:

Sound\_Play(A);

**break**;

**case** 0x02:

Sound\_Play(F);

**break**;

**case** 0x04:

Sound\_Play(D);

**break**;

**default**:

**break**;

}

}

/\*

\* If button is released

\* Stop the sound

\*/

**if**((input==0) && previous){

Sound\_Off();

}

//Store previous input to keep playing if needed

previous = input;

}

}

**DAC.h:**

**#ifndef** DAC\_H\_

**#define** DAC\_H\_

**#include** <stdint.h>

**#include** "msp432p401r.h"

//Address of Port 4

**#define** SPEAKER (\*((**volatile** uint8\_t \*)0x40004C23))

//Initialize DAC

**void** **DAC\_Init**();

//Output to Speaker(P4->Out)

**void** **DAC\_Out**(**unsigned** **char** Data);

**#endif** /\* DAC\_H\_ \*/

**DAC.c:**

**#include** "DAC.h"

**void** **DAC\_Init**(){

/\*

\* To Do:

\* 1.) Initialize P4.3 to P4.0 as output

\*/

P4->SEL0 &= ~0x0F; //make P4.3-P4.0 GPIO Outputs

P4->SEL1 &= ~0x0F;

P4->DIR |= 0x0F;

}

**void** **DAC\_Out**(**unsigned** **char** Data){

**int** br;

/\*

\* To Do:

\* 1.) Output value

\*/

P4->OUT = Data;

br = 0;

}

**Piano.h:**

**#ifndef** PIANO\_H\_

**#define** PIANO\_H\_

**#include** <stdint.h>

**#include** "msp432p401r.h"

//Address of Port 5

**#define** SENSOR (\*((**volatile** uint8\_t \*)0x40004C40))

//Initialize Piano

**void** **Piano\_Init**();

//Return Input of Switches

**unsigned** **long** **Piano\_In**();

**#endif** /\* PIANO\_H\_ \*/

**Piano.c:**

**#include** "Piano.h"

**unsigned** **long** Temp;

**unsigned** **long** Input;

**void** **Piano\_Init**(){

/\*

\* To Do:

\* 1.) Initialize P5.2-P5.0 as input

\* P5.0 was working last time so use work around

\*/

P5->SEL0 &= ~0x16; //make P5.2-P5.0 GPIO Inputs

P5->SEL1 &= ~0x16;

P5->DIR &= ~0x16;

}

**unsigned** **long** **Piano\_In**(){

/\*

\* To Do:

\* 1.) Return Input Values

\*/

Temp = (SENSOR&0x10)>>4; //Get P5.4 and shift it to be in position 5.0

Input = (SENSOR&0x06)|Temp;

**return** Input;

}

**Sound.h:**

**#ifndef** SOUND\_H\_

**#define** SOUND\_H\_

**#include** "DAC.h"

**#include** "SysTickInts.h"

**unsigned** **short** Index;

**unsigned** **short** Sound\_Flag;

//Initialize Sound

**void** **Sound\_Init**();

//Play sound over a period that was calculated

**void** **Sound\_Play**(**unsigned** **long** period);

//Turn sound off

**void** **Sound\_Off**();

**void** **Sound\_Start**(**unsigned** **short** num);

**#endif** /\* SOUND\_H\_ \*/

**Sound.c:**

**#include** "Sound.h"

**#include** "Led.h"

/\* Sinewave was causing problems when placed

\* in other files

\* Given Sinewave:

\*/

**const** **unsigned** **char** SineWave[32] = {

8,9,11,12,13,14,14,15,15,15,14,

14,13,12,11,9,8,7,5,4,3,2,

2,1,1,1,2,2,3,4,5,7};

**void** **Sound\_Init**(){

/\*

\* To Do:

\* 1.) Initialize the Data Structures

\* 2.) Call DAC\_Init

\* 3.) Initializes the SysTick Interrupt

\* Done in main.c

\*/

DAC\_Init();

SysTick\_Init(1);

Index = 0;

}

**void** **Sound\_Play**(**unsigned** **long** period){

/\*

\* To Do:

\* 1.) Starts sound output at the specified pitch

\* 2.) Go for 'period' without interrupting

\*/

Sound\_Flag = 1;

SysTick->LOAD = period;

}

**void** **Sound\_Off**(){

/\*

\* Turn of sound

\*/

Sound\_Flag = 0;

Index = 0;

}

**void** **Sound\_Start**(**unsigned** **short** num){

/\*

\* Index increments from 0 to 31 and then starts back at 0 again

\* while sound flag is true, else output nothing

\*/

**if** (Sound\_Flag){

LED\_Blink();

DAC\_Out(SineWave[num]);

}

**else**{

LED\_Stop();

}

}

**SysTickInts.h:**

**#ifndef** SYSTICKINTS\_H\_

**#define** SYSTICKINTS\_H\_

**#include** "Sound.h"

**void** **SysTick\_Init**(**unsigned** **long** period);

**void** **SysTick\_Handler**();

**#endif** /\* SYSTICKINTS\_H\_ \*/

**SysTickInts.c:**

**#include** "SysTickInts.h"

**void** **DisableInterrupts**(**void**); // Disable interrupts

**void** **EnableInterrupts**(**void**); // Enable interrupts

**long** **StartCritical** (**void**); // previous I bit, disable interrupts

**void** **EndCritical**(**long** sr); // restore I bit to previous value

**void** **WaitForInterrupt**(**void**); // low power mode

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*SysTick\_Init\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Initialize SysTick periodic interrupts

// Input: interrupt period

// Units of period are 333ns (assuming 3 MHz clock)

// Maximum is 2^24-1

// Minimum is determined by length of ISR

// Output: none

**volatile** uint32\_t Counts;

**void** **SysTick\_Init**(**unsigned** **long** period) {

**long** sr = StartCritical();

Counts = 0;

SysTick->CTRL = 0; // disable SysTick during setup

SysTick->LOAD = period - 1; // maximum reload value

SysTick->VAL = 0; // any write to current clears it

SCB->SHP[3] = (SCB->SHP[3]&0x00FFFFFF)|0x40000000; // priority 2

SysTick->CTRL = 0x00000007; // enable SysTick with no interrupts

EndCritical(sr);

}

**void** **SysTick\_Handler**(){

**if** (Counts == 32)

Counts = 0;

Sound\_Start(Counts);

Counts++;

}

**LED.h:**

**#ifndef** LED\_H\_

**#define** LED\_H\_

**#include** <stdint.h>

**#include** "msp432p401r.h"

**void** **LED\_Init**(**void**);

**void** **LED\_Blink**(**void**);

**void** **LED\_Stop**(**void**);

**#endif** /\* LED\_H\_ \*/

**LED.c:**

**#include** "Led.h"

**volatile** **unsigned** **long** LED\_ticks = 0;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*Led\_Init\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Initialize Port 2.1 RGB LED for RED LED Output.

// Also initializes RGB LED to off

// Input: none

// Output: none

**void** **LED\_Init**(**void**){

P2SEL0 &= ~0x01;

P2SEL1 &= ~0x01;

P2DIR |= 0x01;

P2OUT = 0x00;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*LED\_Blink\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Blinks the LED while button is pressed

// Input: none

// Output: none

**void** **LED\_Blink**(**void**){

LED\_ticks++;

**if**(LED\_ticks == 800){

LED\_ticks = 0;

P2OUT = ~P2OUT;

}

}

**void** **LED\_Stop**(**void**){

P2OUT &= 0x00;

}

