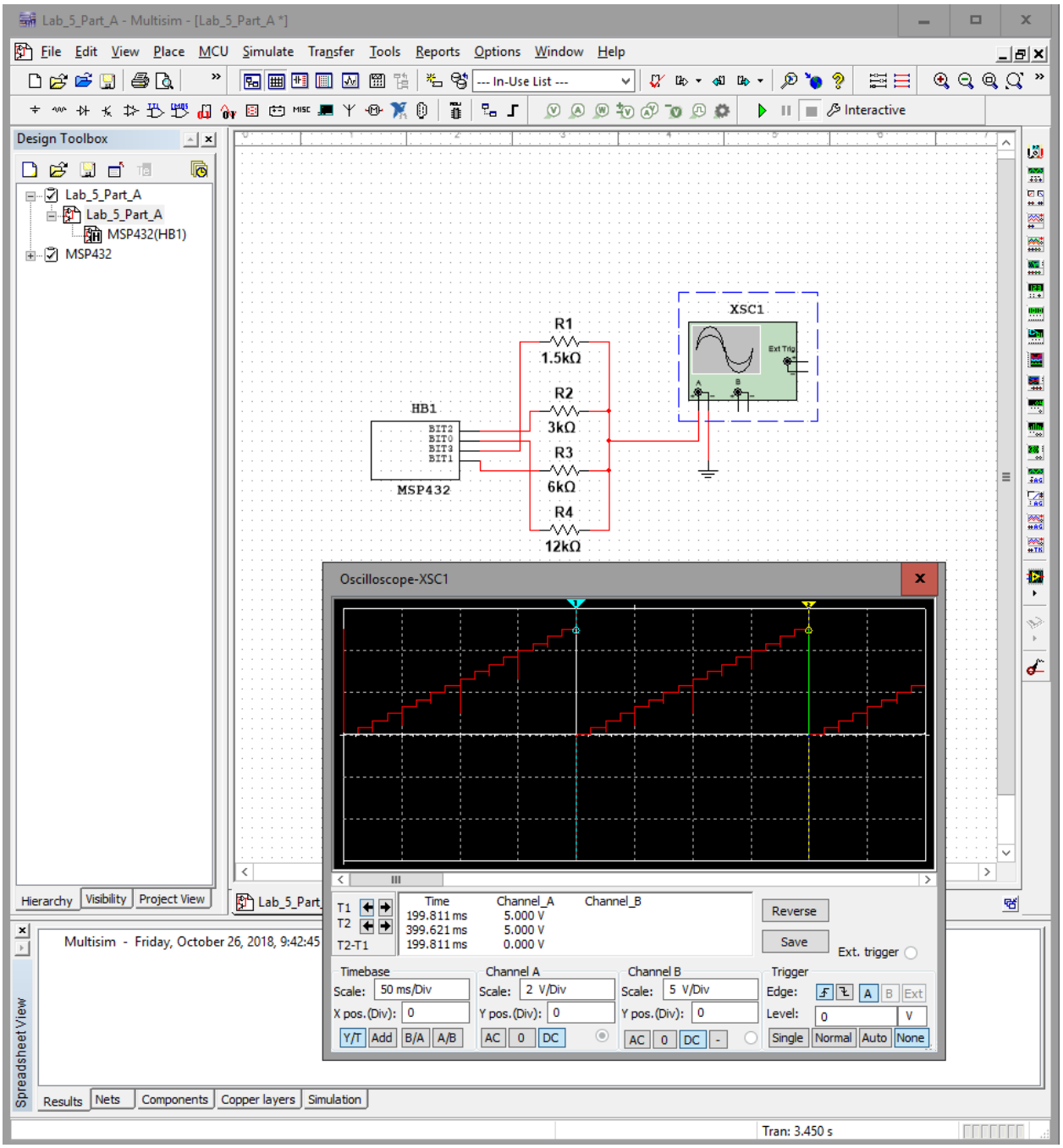
Introduction:

During this lab, we will be looking at implementing a digital-to-analog converter, otherwise known as a DAC. This circuit is made up of a network of resistors in order to manipulate sine/cos waveforms in order to output different frequency sounds to an audio jack.

Procedure:

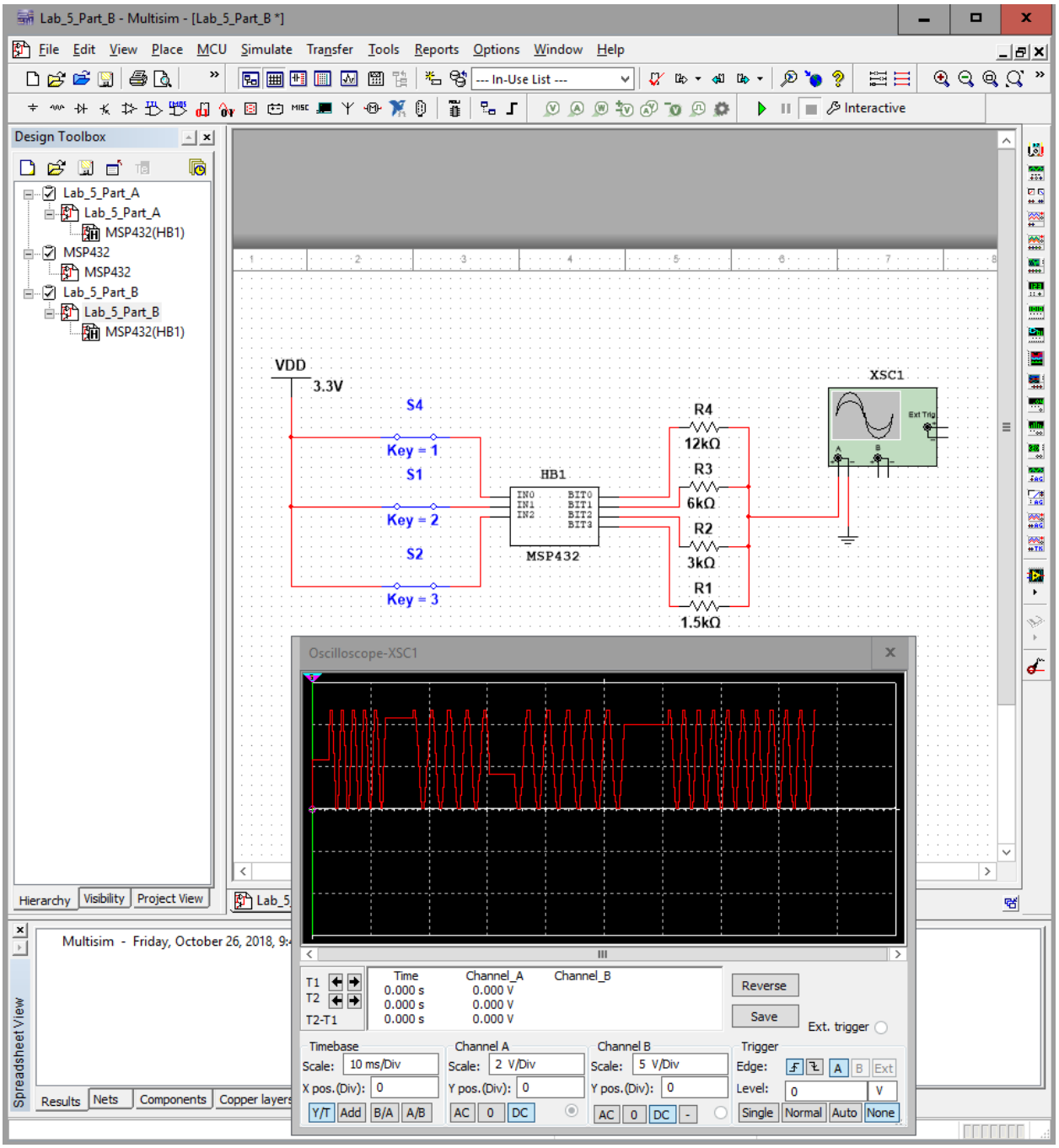
Part A:

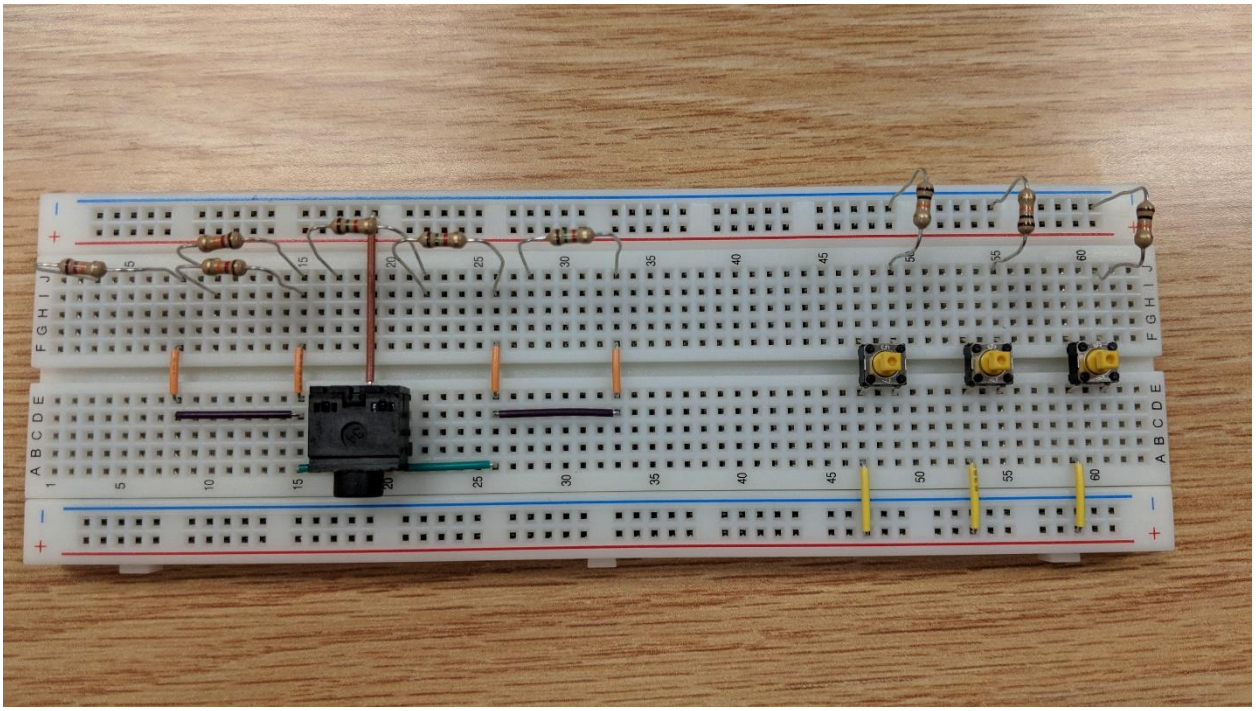


For part A, I simulated the circuit using multisim and verified what the output should look like using the resistance values 1.5k, 3k, 6k, and 12k.

Part B:

I setup the board in which the 3 buttons on the right will be the inputs to port 5.2-5.0 then we tested the output using an oscilloscope in multisim.





Part C:

For part C, I wrote the functions for the DAC interface, which included DAC_Init(), and DAC_Out();

Part D:

For part D, I wrote a static test to make sure the output of our DAC was working properly. This included hard coding the output to just play a static tone to the headphone jack.

Part E:

For part E, I wrote the functions for the piano to simulate the keys, this included Piano_Init(), and Piano_In(). This function was to initialize the inputs, and the other is to read the inputs from the push buttons.

Part F:

For part F, I wrote the functions for the sound device driver software. This module is what will produce the sounds and contain the sine waveform that will depict which note to play. This also includes the functions that are part of SysTick.

Part G:

For part G, we implemented the main function to determine the input combination in order to select and play the right note, this involved writing a while loop to keep the program constantly running and enable/disable the output depending on the input combination.

Conclusion:

During this lab, we were able to learn how to configure a DAC and use it to produce sound waveforms using Fourier Transform methods. At first, we had to read a sine wave to get the inputs for the waveform to base the sounds off of. Then we needed to determine what frequency to play each note at depending on the system clock speed of 3MHz and the sampling period of 32. Once this was done, we used those frequencies to set the sample for the notes to be played. Then during the main function, we had to do a while loop in order to keep reading for inputs every cycle and change the output depending on which combination of inputs is being read.

References:

During this lab, I used the notes from the lecture slides, as well as the textbook, and some assistance from other classmates.

Appendix:

Code:

Main.c:

```
#include <stdint.h>
#include "msp432p401r.h"
#include "DAC.h"
#include "Piano.h"
#include "Sound.h"
#include "SysTickInts.h"


void main(void){

  Sound_Init();
  Piano_Init();

  while(1){
    Piano_In();
    if( Piano_In() == 0x01)
      Sound_Play(LowC);
    else if( Piano_In() == 0x02)
      Sound_Play(D);
    else if( Piano_In() == 0x03)
      Sound_Play(E);
    else if( Piano_In() == 0x04)
      Sound_Play(F);
    else if( Piano_In() == 0x05)
      Sound_Play(G);
    else if( Piano_In() == 0x06)
      Sound_Play(A);
    else if( Piano_In() == 0x07)
```

```
        Sound_Play(B);
      else
        Sound_Play(0);

  }
}
```

DAC.h:

```
/*
 * DAC.h
 *
 *  Created on: Oct 26, 2018
 *      Author: Cory Longshore 891158180
 */

#ifndef DAC_H_
#define DAC_H_

void DAC_Init();

void DAC_Out(unsigned char data);

#endif /* DAC_H_ */
```

DAC.c:

```
/*
 * DAC.c
 *
 *  Created on: Oct 26, 2018
 *      Author: Cory Longshore 891158180
 */

#include <stdint.h>
#include "msp432p401r.h"
#include "DAC.h"

void DAC_Init(void){      // Initializes the device
  P4DIR |= 0x1E;     // Enable DAC; P4.4-P4.1 = 1 1110
}

void DAC_Out(unsigned char data){   // Transfers data to device
  P4OUT = data;
}
```

Piano.h:

```
/*
 * Piano.h
 *
 *  Created on: Oct 26, 2018
 *      Author: Cory Longshore 891158180
 */

#ifndef PIANO_H_
#define PIANO_H_

void Piano_Init();

unsigned char Piano_In(void);

#endif /* PIANO_H_ */
```

Piano.c:

```
/*
 * Piano.c
 *
 *  Created on: Oct 26, 2018
 *      Author: Cory Longshore 891158180
 */

#include <stdint.h>
#include "msp432p401r.h"
#include "Piano.h"
#include "DAC.h"


void Piano_Init(){
   P5DIR  &= ~0x07;     // Enable Buttons; P5.2-P5.0 = 0111
}

unsigned char Piano_In(void){
   return(P5IN & 0x07);
}
```

Sound.h:

```
/*
 * Sound.h
 *
 *  Created on: Oct 26, 2018
```

```
 *      Author: Cory Longshore 891158180
 */
#define HighC 179    // 3MHz/523Hz/32
#define B     190    // 3MHz/494Hz/32
#define A     213    // 3MHz/440Hz/32
#define G     239    // 3MHz/392Hz/32
#define F     269    // 3MHz/349Hz/32
#define E     284    // 3MHz/330Hz/32
#define D     319    // 3MHz/294Hz/32
#define LowC  358    // 3MHz/262Hz/32

#ifndef SOUND_H_
#define SOUND_H_

void Sound_Init();

void Sound_Play(unsigned long period);

void Sound_Int();

#endif /* SOUND_H_ */
```

Sound.c:

```
/*
 * Sound.c
 *
 *  Created on: Oct 26, 2018
 *      Author: Cory Longshore 891158180
 */

#include "Sound.h"
#include "DAC.h"
#include "msp432p401r.h"

const unsigned char Wave[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};

unsigned char Index = 0;

void Sound_Init(void) {
   DAC_Init();
   SysTick_Init(Index);
}
```

```
void Sound_Play(unsigned long period){
  SysTick->LOAD = period;
}

void Sound_Int(){
  if (Index > 31)
    Index = 0;
  DAC_Out(Wave[Index]);
  Index++;
}
```

SysTick.h

```
// SysTickInts.h
// Runs on MSP432
// Use the SysTick timer to request interrupts at a particular period.
// Jonathan Valvano
// June 1, 2015

/* This example accompanies the books
  "Embedded Systems: Introduction to MSP432 Microcontrollers",
  ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
  Volume 1 Program 9.7

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// P4.0 is an output to profiling scope/logic analyzer

// **************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
```

```
void SysTick_Init(uint32_t period);
```

SysTick.c:

```
#include <stdint.h>
#include "msp432p401r.h"
#include "DAC.h"
#include "Piano.h"
#include "Sound.h"
#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;

int Index = 0;

void SysTick_Init(uint32_t 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(void){
          Sound_Int();
}
```