CECS 447 Project 1

Digital Piano

Prepared By: Andrew Miyaguchi

2/10/2021

REVISION HISTORY

Revision Version	Revision Date	Release Date	Description	Author
1.0 A	2/10/2021	2/10/2021	Initial document created	A. Miyaguchi

TABLE OF CONTENTS

1. INTRODUCTION 5
2. OPERATION 5
3. THEORY 5
Timing 5
Tone Generator 5
Debounce 5
Pinout Selection 6
3. HARDWARE DESIGN 8
Pin Assignment 8
Enclosure 9
Schematic 12
4. SOFTWARE DESIGN 13
Source Code 13
Part 1 13
Part 2 21
5. CONCLUSION 35
6. References 36

TABLE OF FIGURES

Figure 1. Pong Chu Debounce FSM... 6
Figure 2. Launchpad ICDI pinouts (highlighted)... 6
Figure 3. LM Flash Programmer Debug Port Unlock 7
Figure 4. TM4C123G Launchpad Evaluation Board 8
Figure 5. Enclosure 3D Model Render 9
Figure 6. Microswitch and speaker wiring inside enclosure 10
Figure 7. Photo of actual system 11
Figure 8. Schematic 12

1. INTRODUCTION

Project 1 reviews materials such as GPIO, timers, and interrupt. Building upon that knowledge we create a music box and digital piano which utilizes a Digital to Analog Converter (DAC).

Microcontrollers are primarily digital devices, meaning that producing analog values can be a challenge. Most microcontrollers do not have the circuitry built in for analog out and therefore external circuitry must be added.

2. OPERATION

In part 1, we design a music box which contains three songs. Onboard switch 1 (left switch) is the play and stop button. When this button is pressed, the current selected song either starts or stops.

Switch 2 (right switch) changes which song is played. This is a simple index which wraps back around, otherwise known as round robin order.

In part 2, we take the music box a step further and a add a DAC. Rather than output square waves, we output sine waves. Additionally, we include seven more push buttons which act as an octave on a piano keyboard. Switch 1 turns the piano on and off while switch 2 changes between piano and auto-play mode.

Demonstrations for both part 1 and part 2 can be found at the following link:

https://photos.app.goo.gl/qDbo8QQ6ducDmzov5

3. THEORY

With my modifications to part 2, multiple keys can be pressed and parsed at the same time. I also added two more tone generators meaning that the piano can play three tones at the same time. The output value from each tone generator is summed before being written to the DAC.

Timing

The system clock is initialized at 50Mhz which is then fed into various other systems. One system would be the timers which are used to keep track of time and generate the tones.

Due to a misunderstanding on my part, I believed that we were using the peripheral general-purpose timers instead of systick. Fortunately, the timers have the same principles of operation.

General purpose Timer0A is initialized to keep track of time since the micro-controller started. This is the underlying timer for the millis() function, which returns an unsigned long with the time since the last start. The returned value has a resolution of 1/10 of a milli-second.

General purpose Timer1A, Timer1B, and Timer2A are utilized as Tone Generators

Tone Generator

To generate a tone, I've built pseudo classes which are essentially named structs. Public functions utilize these structs as data input which allow me to keep track of multiple outputs without writing redundant code.

Excel is used to generate the 64-sample sine wave array. Each value is the amplitude of the sine wave at a given time. By replaying these values in order, we can output an approximated sine wave.

Debounce

The debounce logic is based off Pong Chu's Finite State Machine which we made in Verilog for previous FPGA classes. The code was translated into ansi C and proper handlers and structs were made to allow the same class to be utilized multiple times. This type of debounce was much needed due to the poor performance of time out debounces, which tended to trigger randomly or not trigger at all.

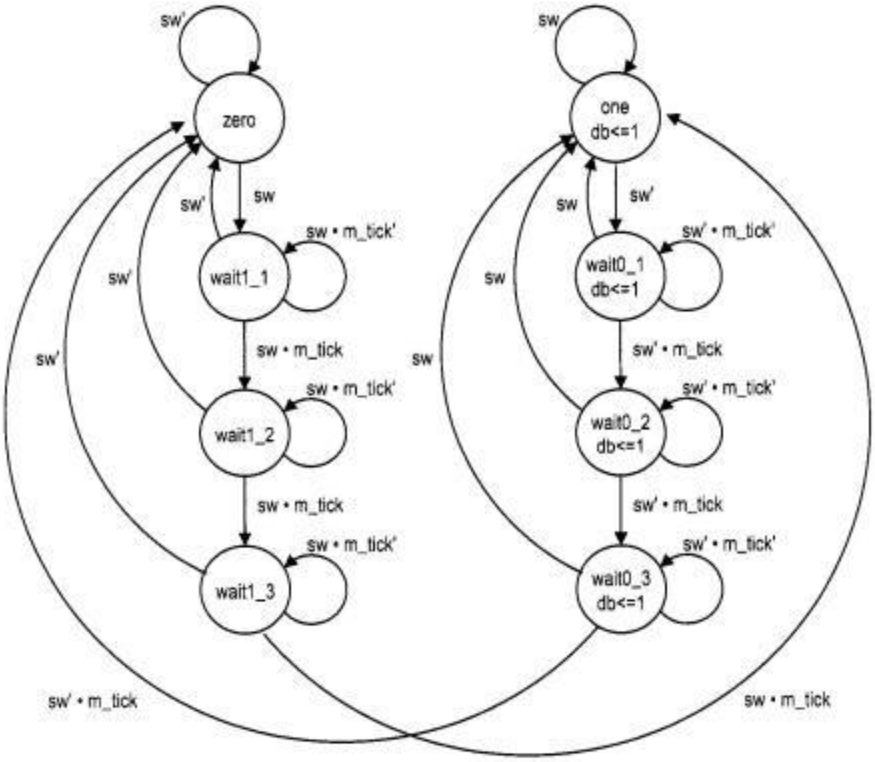


Figure 1. Pong Chu Debounce FSM

Pinout Selection

One must be careful when setting up the pins for PortC. It came to my attention that some of my fellow classmates were having issues with their boards immediately after flashing their program, and I determined that this was because their code touched PC0, PC1, PC2, and PC3 which are used for the ICDI debug interface.

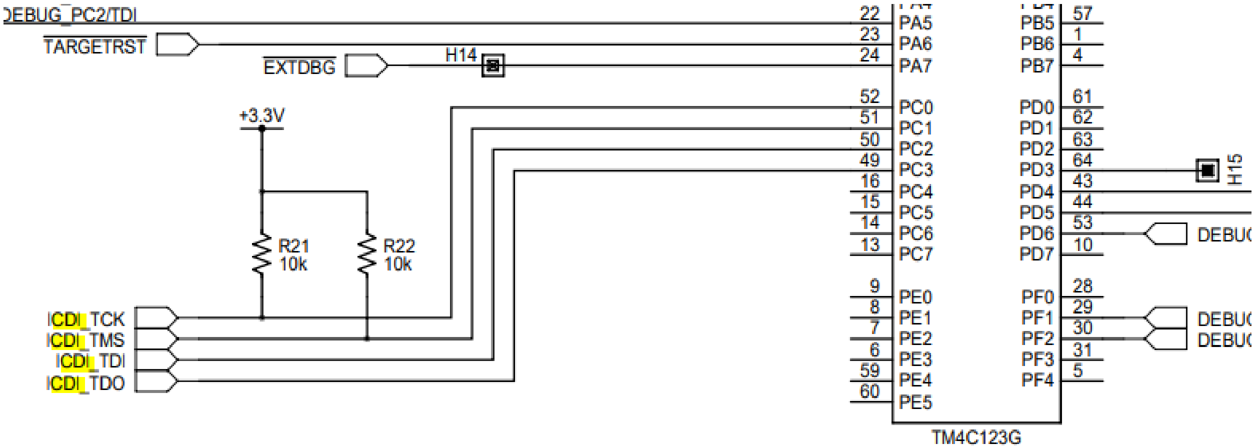
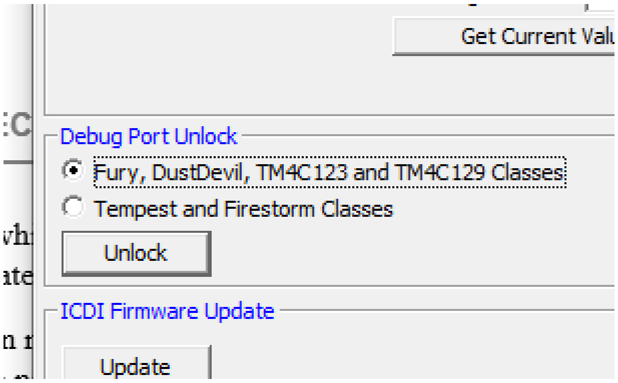


Figure 2. Launchpad ICDI pinouts (highlighted)

This prevented future use of the board without wiping the firmware through LM Flash Programmer and a special bootloader sequence. Perhaps the functionality is intended for production use but could be a serious blunder for a developer who cannot find the necessary tools to fix their board.



Debug Port Unlock

● Fury, DustDevil, TM4C123 and TM4C129 Classes

○ Tempest and Firestorm Classes

Figure 3. LM Flash Programmer Debug Port Unlock

3. HARDWARE DESIGN

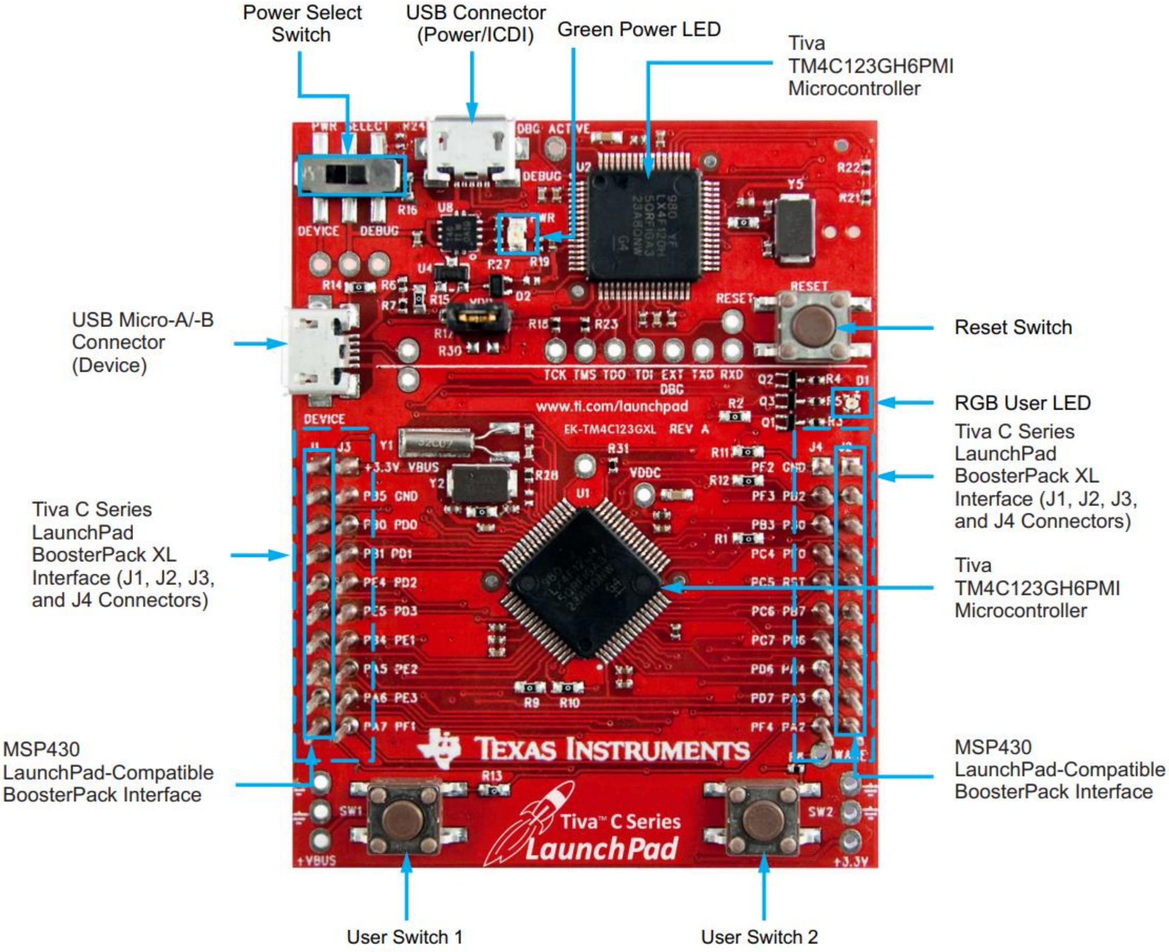

Power Select Switch
USB Connector (Power/ICDI)
Green Power LED
Tiva TM4C123GH6PMI Microcontroller
USB Micro-A/-B Connector (Device)
Reset Switch
RGB User LED
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva TM4C123GH6PMI Microcontroller
MSP430 LaunchPad-Compatible BoosterPack Interface
MSP430 LaunchPad-Compatible BoosterPack Interface
User Switch 1
User Switch 2

Figure 4. TM4C123G Launchpad Evaluation Board
via Texas Instruments Launchpad User's Guide [1]

Pin Assignment

Pin	Name	I/O	Description
PF4	SW1	I	User Switch 1 (Left) Part 1: Play/Stop Part 2: On/Off
PF0	SW2	I	User Switch 2 (Right) Part 1: Change songs Part 2: Autoplay/Piano mode
PD2	SPEAKER	O	Speaker output, only for Part 1

Pin	Name	I/O	Description
PD0	PIANO_C	I	Piano key for tone C
PD1	PIANO_D	I	Piano key for tone D
PD2	PIANO_E	I	Piano key for tone E
PD3	PIANO_F	I	Piano key for tone F
PC4	PIANO_G	I	Piano key for tone G
PC5	PIANO_A	I	Piano key for tone A
PC6	PIANO_B	I	Piano key for tone B
PE0	DAC0	O	DAC Output Bit 0
PE1	DAC1	O	DAC Output Bit 1
PE2	DAC2	O	DAC Output Bit 2
PE3	DAC3	O	DAC Output Bit 3
PE4	DAC4	O	DAC Output Bit 4
PE5	DAC5	O	DAC Output Bit 5

Enclosure

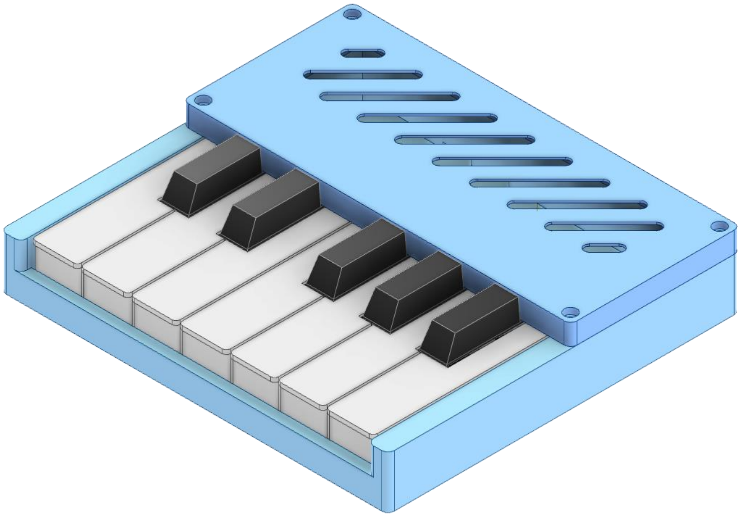


Figure 5. Enclosure 3D Model Render

I decided to have a little more fun and put my CAD modeling skills to the test. The following design was created in around 3 hours in Autodesk Inventor and printed on a Lulzbot Taz6 3D printer. A cheap $1 speaker was bought from Torrance Electronics and hot glued into place. In the figure below, the speaker may appear to be glued flush, but it is in face on three standoffs allowing for airflow and vibrations. The positioning is less than optimal but saved space due to the protrusion of the speaker driver coil.

Wires are run to the microswitches normally open pin with a common ground. No resistor is utilized since I planned to use the internal pullups.

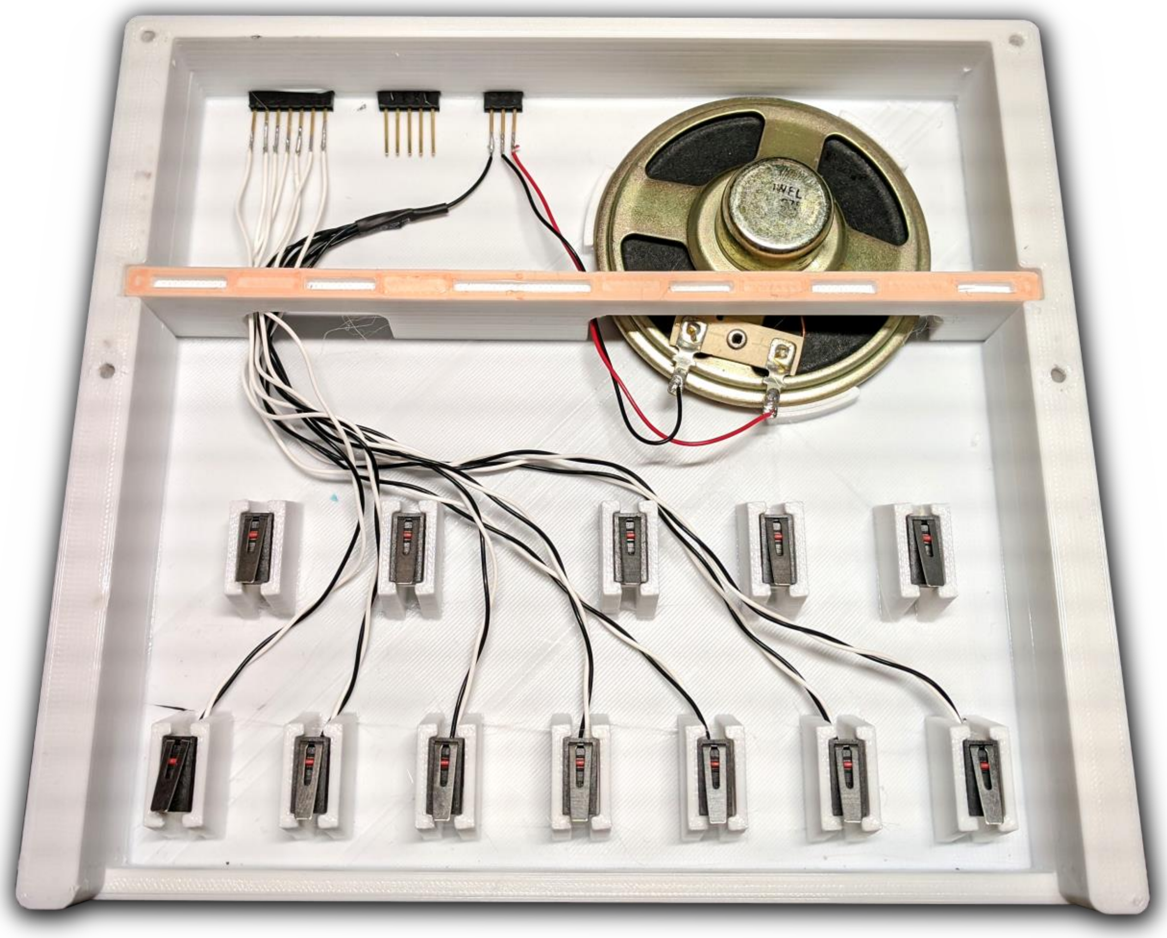


Figure 6. Microswitch and speaker wiring inside enclosure

You may also notice that the switches for the sharps are populated, but not wired in. This was implemented in case I had time to play with extra keys. In the case of Project 1, I simply did not put in the time or effort to allow the sharp keys to work.

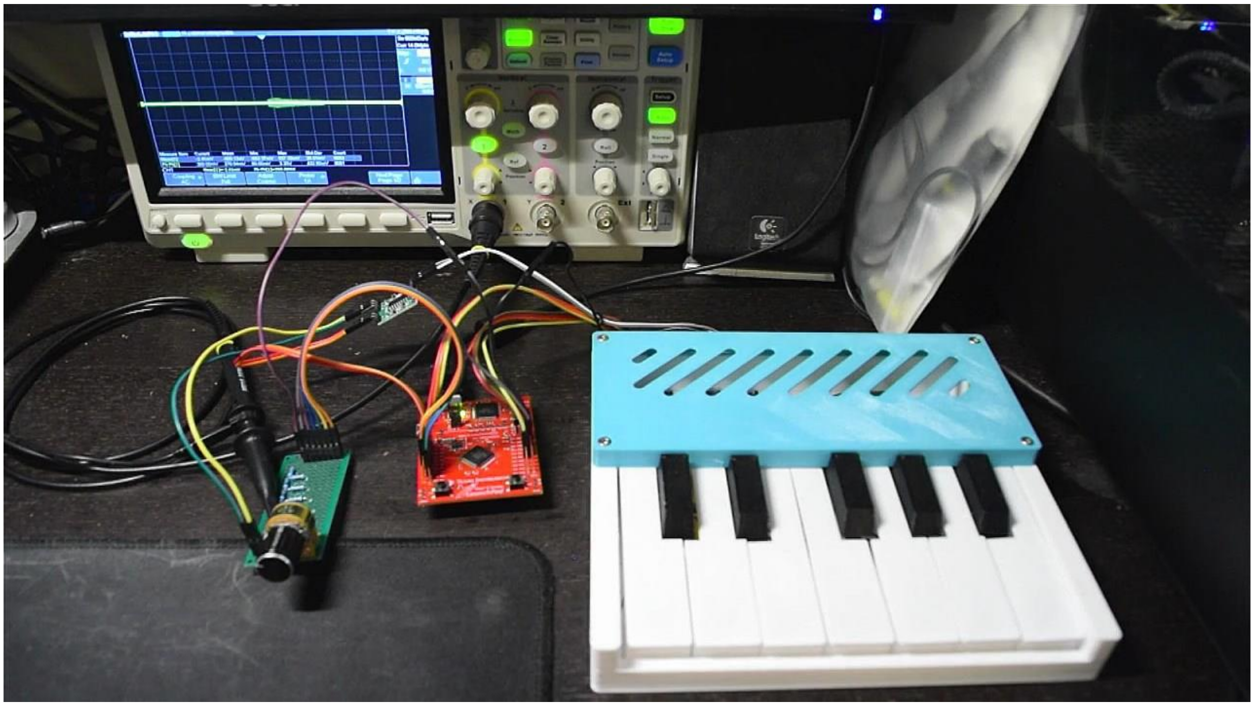


Figure 7. Photo of actual system

Schematic

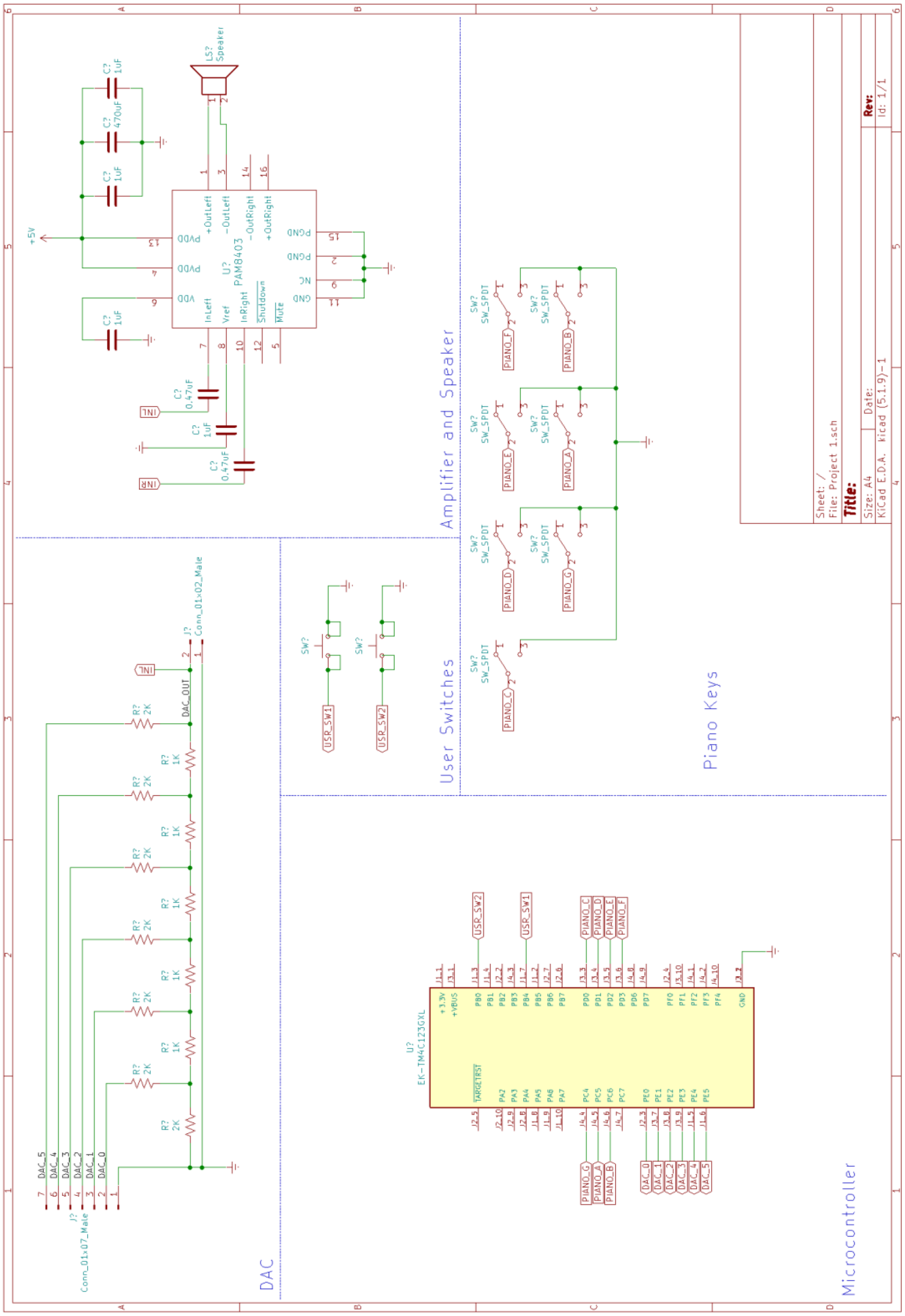


Figure 8. Schematic

4. SOFTWARE DESIGN

The initialization sequence starts with the PLL at 50mhz, and timer0 at 100hz. The software is designed to run asynchronously with a focus on object orientated programming. Objects in ansi C are just structs renamed as such, but the functionality is still there.

The program is constantly checking for flags flipped and watchdogs to timeout in the super loop. During a hardware interrupt, a flag gets flipped in the classes which allows the super loop to get work done. This can be seen in the debounce class, which only triggers after the Debounce_notify() function is called on a debounce object. Not only does this save time, it saves cpu cycles which can be used elsewhere.

The sound class acts as a supervisor class for the three tone generators. It contains the timer handlers which call on their respected tone generator class, and the sound class sums the outputs of the tone generators which is then written to the DAC. This way we can sum the outputs without requiring additional DAC and analog summation circuitry.

The music box is like the tone generator in that it has a constant array which is indexed in sequential order. I needed to add a timing array to get the special music which I extracted from a midi file to play correctly.

Source Code

Part 1

util.h
timer.h
Debounce.h
MusicBox.h

```
/*
    Project 1 - Part 1 Music Box

    Copyright (c) 2021 Andrew Miyaguchi. All rights reserved.

    I've broken out the debounce code into its own class since it's
    reused a lot

    Part I: Music Box
    Build an embedded system that plays the following three songs:
    1. Mary had a little lamb
    2. Twinkle twinkle little star
    3. Happy birthday

    SW1 (Left) switch turns the music box on/off
    SW2 (Right) switch toggles between songs

    Deliverable:
    1. Demonstrate both parts on board
    2. Submit a project report
    3. Submit links to videos
    HID Pinout:
    PF4 SW1 (Left)
    PF0 SW2 (Right)

    Speaker Pinout:
    PD2 Speaker

 */

#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "math.h"
#include "PLL.h"
#include "util.h"
#include "timer.h"
#include "Debounce.h"
#include "MusicBox.h"

// Tone table
#define C4 95420
#define D4 85034
#define E4 75758
#define F4 71633
#define G4 63776
#define A4 56818
#define B4 50607
#define C5 (C4/2)

#define lamb_size 27
uint32_t lamb_notes[lamb_size] = {E4, D4, C4, D4, E4, E4, E4, D4, D4, D4
, E4, G4, G4, E4, D4, C4, D4, E4, E4, E4, D4, D4, E4, D4, C4, 0, 0};
float lamb_beats[lamb_size] = {1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1,
 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 4};

#define star_size 43
uint32_t star_notes[star_size] = {C4, C4, G4, G4, A4, A4, G4, F4, F4, E4
, E4, D4, D4, C4, G4, G4, F4, F4, E4, E4, D4, G4, G4, F4, F4, E4, E4, D4
, C4, C4, G4, G4, A4, A4, G4, F4, F4, E4, E4, D4, D4, C4, 0};
float star_beats[star_size] = {4, 4, 4, 4, 4, 4, 8, 4, 4, 4, 4, 4, 4, 8,
 4, 4, 4, 4, 4, 4, 8, 4, 4, 4, 4, 4, 4, 8, 4, 4, 4, 4, 4, 4, 8, 4, 4, 4,
 4, 4, 4, 8, 8};

#define bday_size 26
```

```
uint32_t bday_notes[bday_size] = {C4, C4, D4, C4, F4, E4, C4, C4, D4, C4
, G4, F4, C4, C4, C5, A4, F4, E4, D4, B4, B4, A4, F4, G4, F4, 0};
float bday_beats[bday_size] = {3, 1, 4, 4, 4, 8, 3, 1, 4, 4, 4, 8, 3, 1,
 4, 4, 4, 4, 4, 3, 1, 4, 4, 4, 12, 12};

// Song object is reused form part 2
#define num_songs 3
Song songs[3] = {
  {lamb_notes, lamb_beats, lamb_beats, lamb_size, 128},
  {star_notes, star_beats, lamb_beats, star_size, 128},
  {bday_notes, bday_beats, lamb_beats, bday_size, 128}
};
uint8_t song_idx = 0;

MusicBox musicbox0;

/**
 * Initializes the timer and interrupt for the tone generator
 */
void Tone_init();

/**
 * Plays a specified tone using the period as an input
 * @param period input period, use the const at the top of the file
 * @param timeout how long to play a tone. 0 to play forever
 */
void Tone_playTone();

/**
 * Starts the tone generator timer
 */
void Tone_start();

/**
 * Stops the tone generator timer
 */
void Tone_stop();

/**
 * Initialize PF4 and PF0 for falling edge interrupts
 * Initialize PF3, PF2, PF1 as digital outputs
 */
void setup_switches(void);

/**
```

```
 * Initialize PD2 as speaker output
 */
void setup_speaker(void);

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

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

/** Halts the main clock and enters powersave mode until an interrupt */
extern void WaitForInterrupt(void);


// Switch debounce
Debounce sw1_db_obj;
uint8_t sw1_input_handler(void) {re-
turn ((~GPIO_PORTF_DATA_R & 0x10) >> 4);};
void sw1_pressed_handler(void) {
  if(musicbox0->mb_flag == 1) {
    // Stop the music box
    MusicBox_stop(musicbox0);

    // Stop the tone generator
    Tone_stop();
  } else {
    MusicBox_play(musicbox0, &(songs[song_idx]));
  }
}

Debounce sw2_db_obj;
uint8_t sw2_input_handler(void) {re-
turn ((~GPIO_PORTF_DATA_R & 0x01) >> 0);};
void sw2_pressed_handler(void) {
  if(musicbox0->mb_flag == 1) {
    // Increment the song index
    song_idx ++;
    if(song_idx >= num_songs) {
      song_idx = 0;
    }

    // play function will automatically reset the music box
    MusicBox_play(musicbox0, &(songs[song_idx]));
  } else {
    // do nothing
```

```c
  }
}

uint32_t tone_timeout;  // tone length
uint32_t tone_watchdog; // current time
uint8_t tone_flag;      // is playing

extern void DisableInterrupts(void); // Disable interrupts
extern void EnableInterrupts(void);  // Enable interrupts
extern void WaitForInterrupt(void);  // low power mode
unsigned long volatile clockDelay;

int main(void){
  uint8_t i;

  // Begin critical section
  DisableInterrupts();
  PLL_Init();       // Initialize 50Mhz system clock
  Timer_Time_Init(5000);  // Initialize timer0 for use with millis() 1/10ms

  // Intialize onboard switches
  setup_switches();
  sw1_db_obj = new_Debounce(&sw1_input_handler, &sw1_pressed_handler, NULL);
  sw2_db_obj = new_Debounce(&sw2_input_handler, &sw2_pressed_handler, NULL);

  setup_speaker();

  // Initialize the tone generator
  Tone_init();

  // Initialize the musicbox
  musicbox0 = new_MusicBox(&Tone_playTone);

  // End critical section
  EnableInterrupts();

  // Superloop
  while(1)
  {
    // run the debounce routines
    run_Debounce(sw1_db_obj);
    run_Debounce(sw2_db_obj);
    // run the music box routine
    MusicBox_run(musicbox0);

    // run the tone routine
    if(tone_flag) {
      // stop a currently playing tone if we exceed its timeout
      if(millis() - tone_watchdog > tone_timeout) {
        Tone_stop();
      }
    }

  }

}

/**
 * Initializes the timer and interrupt for the tone generator
 */
void Tone_init(){
  // Initialize Timer1
  SYSCTL_RCGCTIMER_R |= 0x02;     // 0) activate TIMER
  clockDelay = SYSCTL_RCGCTIMER_R;     //    delay by assigning a regis-
ter
  TIMER1_CTL_R    = 0x00000000;   // 1) disable TIMER during setup
  TIMER1_CFG_R    = 0x00000000;   // 2) configure for 32-bit mode

  // Configure Timer1A for tonegen0
  TIMER1_TAMR_R   = 0x00000002;           // 3) configure for peri-
odic mode, default down-count settings
  TIMER1_TAPR_R   = 0x00000000;           // 5) no prescaler
  TIMER1_ICR_R    |= TIMER_ICR_TATOCINT;  // 6) clear TIMER timeout flag
  TIMER1_IMR_R    |= TIMER_IMR_TATOIM;    // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0xFFFF00FF)|0x00006000; // 8) priority 3
  NVIC_EN0_R |= (0x00000001 << 21);       // 9) enable interrupt in NVIC
}

/**
 * Plays a specified tone using the period as an input
 * @param period input period, use the const at the top of the file
 * @param timeout how long to play a tone. 0 to play forever
 */
void Tone_playTone(uint32_t period, uint32_t timeout) {
  if(period == 0) {
    Tone_stop();
    return;
  }

  TIMER1_TAILR_R = period - 1;
  tone_timeout = timeout;
  tone_watchdog = millis();
  Tone_start();
}

/**
 * Starts the tone generator timer
 */
void Tone_start() {
  TIMER1_CTL_R |= TIMER_CTL_TAEN;
  tone_flag = 1;
}

/**
 * Stops the tone generator timer
 */
void Tone_stop() {
  TIMER1_CTL_R &= ~TIMER_CTL_TAEN;
  tone_flag = 0;
}

/**
 * Initialize PF4 and PF0 for either edge interrupt
 */
void setup_switches(void) {
  SYSCTL_RCGCGPIO_R |= 0x20;        // activate port
  clockDelay = SYSCTL_RCGCGPIO_R;   // allow time to finish activating
  GPIO_PORTF_LOCK_R = 0x4C4F434B;   // unlock GPIO Port
  GPIO_PORTF_CR_R = 0x11;           // allow changes
  GPIO_PORTF_DIR_R &= ~0x11;      // (c) make PF4,0 in (built-in button)
  GPIO_PORTF_AFSEL_R &= ~0x11;    //     disable alt funct
  GPIO_PORTF_DEN_R |= 0x11;       //     enable digital I/O
  GPIO_PORTF_PCTL_R &= ~0x000F000F; //  configure as GPIO
  GPIO_PORTF_AMSEL_R &= ~0x11;    //     disable analog functionality

  GPIO_PORTF_PUR_R |= 0x11;       //     enable weak pull-up
  GPIO_PORTF_IS_R &= ~0x11;       // (d) is edge-sensitive
  GPIO_PORTF_IBE_R |= 0x11;       //     is both edges
  GPIO_PORTF_ICR_R = 0xFF;        // (e) clear all flags
  GPIO_PORTF_IM_R |= 0x11;        // (f) arm interrupt
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFFF)|0x00400000; // (g) bits:23-
21 for PORTF, set priority to 2
  NVIC_EN0_R |= (0x00000001 << 30);       // (h) enable inter-
rupt 30 in NVIC
}

/**
 * Initialize PD2 as speaker output
 */
void setup_speaker(void) {
  SYSCTL_RCGCGPIO_R |= 0x08;        // activate port
  clockDelay = SYSCTL_RCGCGPIO_R;   // allow time to finish activating
  GPIO_PORTD_LOCK_R = 0x4C4F434B;   // unlock GPIO Port
  GPIO_PORTD_CR_R = 0x04;           // allow changes
  GPIO_PORTD_DIR_R |= 0x04;       // (c) make PD2 output
  GPIO_PORTD_AFSEL_R &= ~0x04;    //     disable alt funct
  GPIO_PORTD_DEN_R |= 0x04;       //     enable digital I/O
  GPIO_PORTD_PCTL_R &= ~0x00000F00; //  configure as GPIO
  GPIO_PORTD_AMSEL_R &= ~0x04;    //     disable analog functionality
  GPIO_PORTD_DATA_R &= ~0x04;   //    clear output
  GPIO_PORTD_DR8R_R |= 0x04;        // enable 8 mA drive
}

// Handle PORTF interrupts
void GPIOPortF_Handler(void){   // called on touch of either SW1 or SW2
  if(GPIO_PORTF_RIS_R&0x01){    // SW2 touched
    GPIO_PORTF_ICR_R |= 0x01;   // acknowledge flag0
      notify_Debounce(sw2_db_obj);
  } else if(GPIO_PORTF_RIS_R&0x10){     // SW1 touched
    GPIO_PORTF_ICR_R |= 0x10;   // acknowledge flag4
      notify_Debounce(sw1_db_obj);
  }
}

// Handle Timer1A interrupts
void Timer1A_Handler(void){
  TIMER1_ICR_R |= TIMER_ICR_TATOCINT;  // acknowledge TIMER1A timeout
  GPIO_PORTD_DATA_R = (~GPIO_PORTD_DATA_R) & 0x04;
}
```

Part 2

timer.h
util.h
Debounce.h
Sound.h
MusicBox.h
ToneGenerator.h

```
/*
    Project 1 - Part 2 Digital Piano

    Copyright (c) 2021 Andrew Miyaguchi. All rights reserved.

    I've broken out the debounce code into its own class since it's
    used 9 times

    Part II: Digital Piano
    Build a piano using a 6-bit DAC
    Features:
    - Sinusoidal fade
    - Three simultaneous tones (single chord)
    - Round robin tone generator selection
    - Adjustable tone length
    - FSM-based interrupt-driven debounce
    - Accurate time keeping (timer0A)

    SW1 (Left) switch turns the piano on/off
    SW2 (Right) switch toggles between piano mode and auto-play mode

    Deliverable:
    1. Demonstrate both parts on board
    2. Submit a project report
    3. Submit links to videos

    Piano Key Pinout:
    PD0 C
    PD1 D
    PD2 E
    PD3 F
    PC4 G
    PC5 A
    PC6 B

    HID Pinout:
    PF4 SW1 (Left)
    PF0 SW2 (Right)

    Digital to Analog Converter Pinout:
    PE0 DAC0
    PE1 DAC1
    PE2 DAC2
    PE3 DAC3
    PE4 DAC4
    PE5 DAC5
 */

#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "math.h"
#include "PLL.h"
#include "timer.h"
#include "util.h"
#include "Debounce.h"
#include "Sound.h"
#include "MusicBox.h"

// CONFIGURATION

// Uncomment to disable the fancy stuff and just do the bare minimum
//#define MEET_REQ

// Uncomment to disable key fade. Outputs a wave as long as a key is pressed
//#define NO_FADE

/**
 * Initialize PF4 and PF0 for falling edge interrupts
 */
void setup_switches(void);

/**
 * Initialize PD and PC for piano input
 */
void setup_piano(void);

/**
 * Initialize PB4 and PB5 as direction pins
 */
void setup_dir_pins(void);

/** Disable interrupts */
```

```
extern void DisableInterrupts(void);

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

/** Halts the main clock and enters powersave mode until an interrupt */
extern void WaitForInterrupt(void);


// Tone table
#define C  2982
#define Cs 2820
#define Db Cs
#define D  2657
#define Ds 2512
#define Eb Ds
#define E  2367
#define F  2239
#define Fs 2111
#define Gb Fs
#define G  1993
#define Gs 1883
#define Ab Gs
#define A  1776
#define As 1677
#define Bb As
#define B  1581

#define C3  (C*2)
#define Cs3 (Cs*2)
#define Db3 (Eb*2)
#define D3  (D*2)
#define Ds3 (Ds*2)
#define Eb3 (Eb*2)
#define E3  (E*2)
#define F3  (F*2)
#define Fs3 (Fs*2)
#define Gb3 (Gb*2)
#define G3  (G*2)
#define Gs3 (Gs*2)
#define Ab3 (Ab*2)
#define A3  (A*2)
#define As3 (As*2)
#define Bb3 (Bb*2)
#define B3  (B*2)
```

```
#define C5  (C/2)
#define Cs5 (Cs/2)
#define Db5 (Eb/2)
#define D5  (D/2)
#define Ds5 (Ds/2)
#define Eb5 (Eb/2)
#define E5  (E/2)
#define F5  (F/2)
#define Fs5 (Fs/2)
#define Gb5 (Gb/2)
#define G5  (G/2)
#define Gs5 (Gs/2)
#define Ab5 (Ab/2)
#define A5  (A/2)
#define As5 (As/2)
#define Bb5 (Bb/2)
#define B5  (B/2)

#define C6  (C/4)
#define Cs6 (Cs/4)
#define Db6 (Eb/4)
#define D6  (D/4)
#define Ds6 (Ds/4)
#define Eb6 (Eb/4)
#define E6  (E/4)
#define F6  (F/4)
#define Fs6 (Fs/4)
#define Gb6 (Gb/4)
#define G6  (G/4)
#define Gs6 (Gs/4)
#define Ab6 (Ab/4)
#define A6  (A/4)
#define As6 (As/4)
#define Bb6 (Bb/4)
#define B6  (B/4)

#define NUM_TRACKS 3
MusicBox track[NUM_TRACKS];

#ifdef MEET_REQ
  #define lamb_size 27
  uint32_t lamb_notes[lamb_size] = {E,D,C,D,E,E,E,D,D,D,E,G,G,E,D,C,D,E,E,E,D,D
,E,D,C,0,0};
```

```
  float lamb_beats[lamb_size] = {1,1,1,1,1,1,2,1,1,2,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,2,4};
  Song lamb = {lamb_notes, lamb_beats, lamb_beats, lamb_size, 128};
#else
  #define melody1_size 158
  uint32_t melody1_notes[melody1_size] = {F5,Cs5,Eb5,C5,Cs5,C5,Bb,C5,Gs,Gs,A,Bb,Cs5,Bb,Bb5,Gs5,Fs5,F5,Fs5,Gs5,F5,Eb5,F5,Cs5,Eb5,C5,Cs5,C5,Bb,C5,Gs,Gs,A,Bb,Cs5,Bb,Bb5,Gs5,Fs5,Gs5,Bb5,C6,Cs6,Eb6,Cs6,F6,Cs6,C6,Eb6,C6,Gs5,A5,Bb5,Cs6,Bb5,C6,Eb6,Gs5,Bb5,B5,C6,Cs6,F6,Cs6,C6,Eb6,C6,Gs5,A5,Bb5,Cs6,Bb5,C6,Eb6,F6,Eb6,Cs6,C6,Gs5,F5,Cs5,Eb5,C5,Cs5,C5,Bb,C5,Gs,Gs,A,Bb,Cs5,Bb,Bb5,Gs5,Fs5,F5,Fs5,Gs5,F5,Eb5,F5,Cs5,Eb5,C5,Cs5,C5,Bb,C5,Gs,Gs,A,Bb,Cs5,Bb,Bb5,Gs5,Fs5,Gs5,Bb5,C6,Cs6,Eb6,Cs6,F6,Cs6,C6,Eb6,C6,Gs5,A5,Bb5,Cs6,Bb5,C6,Eb6,Gs5,Bb5,B5,C6,Cs6,F6,Cs6,C6,Eb6,C6,Gs5,A5,Bb5,Cs6,Bb5,C6,Eb6,F6,Eb6,Cs6,C6,Gs5};
  float melody1_beats[melody1_size] = {1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1.5,1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1.5,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,1,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1.5,1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1.5,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,1,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,0.5,0.5};
  float melody1_delta[melody1_size] = {1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,2,1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,2,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,1,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,0.5,0.5,1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,2,1,1,1,1,0.5,0.5,0.5,0.5,1.5,0.25,0.25,1,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,2,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,1,1.5,1.5,1,1.5,1.5,0.5,0.25,0.25,1.5,1.5,1,1,0.5,0.5,0.5,0.5,0.5,0.5};

  #define melody2_size 331
  uint32_t melody2_notes[melody2_size] = {0, Cs5,Gs,Gs,C5,Gs,Gs,Bb,F,Cs,F,Eb,C,Cs,Eb,Fs,Cs,Cs,Fs,Cs,Fs,Bb,Cs,Eb,Gs,Cs5,Gs,C5,Eb5,Fs5,Eb5,Fs5,Gs5,C6,F5,Cs5,Gs,Cs5,F5,Gs5,F5,Gs5,Cs5,Gs,F,Gs,Cs5,F5,Cs5,F5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,Fs5,C5,Gs,Fs,Gs,C5,Eb5,C5,Eb5,Cs5,Bb,Fs,Bb,Cs5,F5,Cs5,F5,Bb,Fs,Cs,Fs,Bb,Cs5,Bb,Cs5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,F5,Fs5,Gs5,F5,Cs5,Gs,Cs5,F5,Gs5,F5,Gs5,Cs5,Gs,F,Gs,Cs5,F5,Cs5,F5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,Fs5,C5,Gs,Fs,Gs,C5,Eb5,C5,Eb5,Cs5,Bb,Fs,Bb,Cs5,F5,Cs5,F5,Bb,Fs,Cs,Fs,Bb,Cs5,Bb,Cs5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,Fs5,C6,Gs5,Fs5,Eb5,Cs5,Gs,Gs,C5,Gs,Gs,Bb,F,Cs,F,Eb,C,Cs,Eb,Fs,Cs,Cs,Fs,Cs,Fs,Bb,Cs,Eb,Gs,Cs5,C5,Gs,C5,Cs5,Gs,Gs,C5,Gs,Gs,Bb,F,Cs,F,Eb,C,Cs,Eb,Fs,Cs,Cs,Fs,Cs,Fs,Bb,Cs,Eb,Gs,Cs5,Gs,C5,Eb5,Fs5,Eb5,Fs5,Gs5,C6,F5,Cs5,Gs,Cs5,F5,Gs5,F5,Gs5,Cs5,Gs,F,Gs,Cs5,F5,Cs5,F5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,F
```

```
s5,C5,Gs,Fs,Gs,C5,Eb5,C5,Eb5,Cs5,Bb,Fs,Bb,Cs5,F5,Cs5,F5,Bb,Fs,Cs,Fs,Bb,Cs5,Bb,C
s5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,F5,Fs5,Gs5,F5,Cs5,Gs,Cs5,F5,Gs5,F5,Gs5,Cs5,Gs,F,Gs,
Cs5,F5,Cs5,F5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,Fs5,C5,Gs,Fs,Gs,C5,Eb5,C5,Eb5,Cs5,Bb,Fs,
Bb,Cs5,F5,Cs5,F5,Bb,Fs,Cs,Fs,Bb,Cs5,Bb,Cs5,Eb5,C5,Gs,C5,Eb5,Fs5,Eb5,Fs5,C6,Gs5,
Fs5,Eb5};
  float melody2_beats[mel-
ody2_size] = {0,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5
,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.
25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0
.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,
0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25
,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.5,0.5,1,0.25,0.25,0.25,0.25,0.25,0.25,0.25
,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.2
5,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.
25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0
.25,0.25,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.
5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5
,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.
25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0
.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,
0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25
,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.5,
0.5,1,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.2
5,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.
25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0
.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.5,0.5,0.5};
  float melody2_delta[mel-
ody2_size] = {16,0.5,1,0.5,0.5,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1,0.5,
0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.
25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0
.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,
0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25
,0.25,0.25,0.25,0.25,0.25,0.5,0.5,0.5,1,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25
,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.2
5,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.
25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0
.25,0.5,0.5,0.5,0.5,0.5,1,0.5,0.5,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1,0
.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,1,0.5,0.5,1,0.5,0.5,0.5,0.5,0.
5,0.5,0.5,0.5,0.5,0.5,1,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.25,0.25,0.25,0.25
,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.2
5,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.
25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0
.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.5,0.5,1,0.25,0.25,0
.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,
0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25
```

```
,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.25,0.2
5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.5,0.5,0.5};

  #define bass_size 182
  //uint32_t bass_notes[bass_size] = {Cs3,Cs3,C3,C3,Bb2,Bb2,Gs2,Gs2,Fs2,Fs2,Eb3
,Eb3,Gs3,Gs3,Gs2,Gs2,Gs3,Cs3,Cs3,C3,C3,Bb2,Bb2,Gs2,Gs2,Fs2,Fs2,Eb3,Eb3,Gs3,Gs3,
Gs2,Gs2,Gs3,Cs3,Cs,Cs3,Cs3,C,Cs,Gs3,C3,C,C3,C3,B3,C,Gs3,Bb2,Bb3,Bb2,Bb2,A3,Bb3,
Fs3,Gs2,Gs3,Gs2,Gs2,C3,Eb3,G3,Eb3,Cs3,Cs,Cs3,Cs3,C,Cs,Gs3,C3,C,C3,C3,B3,C,Gs3,B
b2,Bb3,Bb2,Bb2,A3,Bb3,Fs3,Gs2,Gs3,Gs2,Gs2,Gs3,Eb3,Gs2,Cs3,Cs3,C3,C3,Bb2,Bb2,Gs2
,Gs2,Fs2,Fs2,Eb3,Eb3,Gs3,Gs3,Gs2,Gs2,Gs3,Cs3,Cs3,C3,C3,Bb2,Bb2,Gs2,Gs2,Fs2,Fs2,
Eb3,Eb3,Gs3,Gs3,Gs2,Gs2,Gs3,Cs3,Cs,Cs3,Cs3,C,Cs,Gs3,C3,C,C3,C3,B3,C,Gs3,Bb2,Bb3
,Bb2,Bb2,A3,Bb3,Fs3,Gs2,Gs3,Gs2,Gs2,C3,Eb3,G3,Eb3,Cs3,Cs,Cs3,Cs3,C,Cs,Gs3,C3,C,
C3,C3,B3,C,Gs3,Bb2,Bb3,Bb2,Bb2,A3,Bb3,Fs3,Gs2,Gs3,Gs2,Gs2,Gs3,Eb3,Gs2};
  uint32_t bass_notes[bass_size] = {Cs,Cs,C,C,Bb3,Bb3,Gs3,Gs3,Fs3,Fs3,Eb,Eb,Gs,
Gs,Gs3,Gs3,Gs,Cs,Cs,C,C,Bb3,Bb3,Gs3,Gs3,Fs3,Fs3,Eb,Eb,Gs,Gs,Gs3,Gs3,Gs,Cs,Cs5,C
s,Cs,C5,Cs5,Gs,C,C5,C,C,B,C5,Gs,Bb3,Bb,Bb3,Bb3,A,Bb,Fs,Gs3,Gs,Gs3,Gs3,C,Eb,G,Eb
,Cs,Cs5,Cs,Cs,C5,Cs5,Gs,C,C5,C,C,B,C5,Gs,Bb3,Bb,Bb3,Bb3,A,Bb,Fs,Gs3,Gs,Gs3,Gs3,
Gs,Eb,Gs3,Cs,Cs,C,C,Bb3,Bb3,Gs3,Gs3,Fs3,Fs3,Eb,Eb,Gs,Gs,Gs3,Gs3,Gs,Cs,Cs,C,C,Bb
3,Bb3,Gs3,Gs3,Fs3,Fs3,Eb,Eb,Gs,Gs,Gs3,Gs3,Gs,Cs,Cs5,Cs,Cs,C5,Cs5,Gs,C,C5,C,C,B,
C5,Gs,Bb3,Bb,Bb3,Bb3,A,Bb,Fs,Gs3,Gs,Gs3,Gs3,C,Eb,G,Eb,Cs,Cs5,Cs,Cs,C5,Cs5,Gs,C,
C5,C,C,B,C5,Gs,Bb3,Bb,Bb3,Bb3,A,Bb,Fs,Gs3,Gs,Gs3,Gs3,Gs,Eb,Gs3};
  float bass_beats[bass_size] = {1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.
25,0.25,1,0.5,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,0.25,1,0.5,0.25,
0.25,0.25,0.25,0.25,0.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.25,0.25,0.25,0
.25,0.25,0.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.5,0.25,0.25,0.25,0.25,0.2
5,0.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,
0.25,0.25,0.25,0.25,1,0.5,0.5,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,
0.25,1,0.5,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,1,0.25,0.25,1,0.5,0.25,0.2
5,0.25,0.25,0.25,0.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.25,0.25,0.25,0.25
,0.25,0.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.5,0.25,0.25,0.25,0.25,0.25,0
.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.25,0.25,0.25,0.25,0.25,0.25,0.5,0.2
5,0.25,0.25,0.25,1,0.5,0.5};
  float bass_delta[bass_size] = {1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.
5,1.5,0.5,0.5,1,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,0.5
,1,0.5,0.5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,0.25
,0.75,1.5,0.25,0.25,0.5,0.5,0.25,0.75,1,0.25,0.25,0.5,0.5,0.5,0.25,0.75,1.5,0.2
5,0.25,0.5,0.5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,
0.25,0.75,0.5,1,0.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5
,0.5,1,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,1.5,0.5,0.5,1,0.5,0.
5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,0.25,0.75,1.5
,0.25,0.25,0.5,0.5,0.25,0.75,1,0.25,0.25,0.5,0.5,0.5,0.25,0.75,1.5,0.25,0.25,0.
5,0.5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,0.25,0.75,1.5,0.25,0.25,0.5,0.5,0.25,0.75
,0.5,1,0.5,0.5};

  Song goldenrodcity[3] = {
    {melody1_notes, melody1_beats, melody1_delta, melody1_size, 128},
    {melody2_notes, melody2_beats, melody2_delta, melody2_size, 128},
    {bass_notes, bass_beats, bass_delta, bass_size, 128}
  };
#endif

/**
 * Switch debounces
 */
Debounce sw1_db_obj;
uint8_t sw1_input_handler(void) {return ((~GPIO_PORTF_DATA_R & 0x10) >> 4);};
void sw1_pressed_handler(void) {
  uint8_t i;
  if(Sound_status()) {
    // Turn off the sound driver
    Sound_disable();

    // stop all tracks
    for(i = 0; i < NUM_TRACKS; i ++) {
      MusicBox_stop(track[i]);
    }
  } else {
    // Turn on the sound driver
    Sound_enable();
#if defined(MEET_REQ) || defined(NO_FADE)
    // no fade for bare-minimum
    Sound_fade(0);
#else
    Sound_fade(1);
#endif
  }
}

Debounce sw2_db_obj;
uint8_t sw2_input_handler(void) {return ((~GPIO_PORTF_DATA_R & 0x01) >> 0);};
void sw2_pressed_handler(void) {
  uint8_t i;

  if(Sound_status() == 0) {
    // Do nothing
    return;
  }

  if(track[0]->mb_flag) {
    // Stop all tracks
    for(i = 0; i < NUM_TRACKS; i ++) {
      MusicBox_stop(track[i]);
      Sound_stop();
    }

    Sound_fade(1);  // re-enable fade
  } else {
    Sound_fade(0);  // disable tone fade out, too much overhead
#ifdef MEET_REQ
    MusicBox_play(track[0], &(lamb));
#else
    MusicBox_play(track[0], &(goldenrodcity[0]));
    MusicBox_play(track[1], &(goldenrodcity[1]));
    MusicBox_play(track[2], &(goldenrodcity[2]));
#endif
  }
}

Debounce C_db_obj;
uint8_t C_input_handler(void) {return ((~GPIO_PORTD_DATA_R & 0x01) >> 0);};

Debounce D_db_obj;
uint8_t D_input_handler(void) {return ((~GPIO_PORTD_DATA_R & 0x02) >> 1);};

Debounce E_db_obj;
uint8_t E_input_handler(void) {return ((~GPIO_PORTD_DATA_R & 0x04) >> 2);};

Debounce F_db_obj;
uint8_t F_input_handler(void) {return ((~GPIO_PORTD_DATA_R & 0x08) >> 3);};

Debounce G_db_obj;
uint8_t G_input_handler(void) {return ((~GPIO_PORTC_DATA_R & 0x10) >> 4);};

Debounce A_db_obj;
uint8_t A_input_handler(void) {return ((~GPIO_PORTC_DATA_R & 0x20) >> 5);};

Debounce B_db_obj;
uint8_t B_input_handler(void) {return ((~GPIO_PORTC_DATA_R & 0x40) >> 6);};

#if defined(MEET_REQ) || defined(NO_FADE)
  // in bare-minimum mode, we play a tone until we let go
  void C_pressed_handler(void) {Sound_playTone(C, 0);}
  void D_pressed_handler(void) {Sound_playTone(D, 0);}
  void E_pressed_handler(void) {Sound_playTone(E, 0);}
  void F_pressed_handler(void) {Sound_playTone(F, 0);}
  void G_pressed_handler(void) {Sound_playTone(G, 0);}
  void A_pressed_handler(void) {Sound_playTone(A, 0);}
  void B_pressed_handler(void) {Sound_playTone(B, 0);}
  void C_released_handler(void) {Sound_stop();}
  void D_released_handler(void) {Sound_stop();}
  void E_released_handler(void) {Sound_stop();}
  void F_released_handler(void) {Sound_stop();}
  void G_released_handler(void) {Sound_stop();}
  void A_released_handler(void) {Sound_stop();}
  void B_released_handler(void) {Sound_stop();}
#else
  // in fancy mode, we let the tone fade out by itself, regardless if we let go
  void C_pressed_handler(void) {Sound_playTone(C, 750);}
  void D_pressed_handler(void) {Sound_playTone(D, 750);}
  void E_pressed_handler(void) {Sound_playTone(E, 750);}
  void F_pressed_handler(void) {Sound_playTone(F, 750);}
  void G_pressed_handler(void) {Sound_playTone(G, 750);}
  void A_pressed_handler(void) {Sound_playTone(A, 750);}
  void B_pressed_handler(void) {Sound_playTone(B, 750);}
#endif

extern void DisableInterrupts(void); // Disable interrupts
extern void EnableInterrupts(void);  // Enable interrupts
extern void WaitForInterrupt(void);  // low power mode
unsigned long volatile clockDelay;

int main(void){
  uint8_t i;

  // Begin critical section
  DisableInterrupts();
  PLL_Init();       // Initialize 50Mhz system clock
  Timer_Time_Init(5000);  // Initialize timer0 for use with millis() 1/10ms

  // Intialize onboard switches
  setup_switches();
  setup_piano();

  for(i = 0; i < NUM_TRACKS; i++) {
    track[i] = new_MusicBox(&Sound_playTone);
  }

  sw1_db_obj = new_Debounce(&sw1_input_handler, &sw1_pressed_handler, NULL);
  sw2_db_obj = new_Debounce(&sw2_input_handler, &sw2_pressed_handler, NULL);
```

```
#if defined(MEET_REQ) || defined(NO_FADE)
  C_db_obj = new_Debounce(&C_input_handler, &C_pressed_handler, &C_released_handler);
  D_db_obj = new_Debounce(&D_input_handler, &D_pressed_handler, &D_released_handler);
  E_db_obj = new_Debounce(&E_input_handler, &E_pressed_handler, &E_released_handler);
  F_db_obj = new_Debounce(&F_input_handler, &F_pressed_handler, &F_released_handler);
  G_db_obj = new_Debounce(&G_input_handler, &G_pressed_handler, &G_released_handler);
  A_db_obj = new_Debounce(&A_input_handler, &A_pressed_handler, &A_released_handler);
  B_db_obj = new_Debounce(&B_input_handler, &B_pressed_handler, &B_released_handler);
#else
  C_db_obj = new_Debounce(&C_input_handler, &C_pressed_handler, NULL);
  D_db_obj = new_Debounce(&D_input_handler, &D_pressed_handler, NULL);
  E_db_obj = new_Debounce(&E_input_handler, &E_pressed_handler, NULL);
  F_db_obj = new_Debounce(&F_input_handler, &F_pressed_handler, NULL);
  G_db_obj = new_Debounce(&G_input_handler, &G_pressed_handler, NULL);
  A_db_obj = new_Debounce(&A_input_handler, &A_pressed_handler, NULL);
  B_db_obj = new_Debounce(&B_input_handler, &B_pressed_handler, NULL);
#endif

  DAC_init();
  Sound_Init(); // Initialize at 440 hz

  // End critical section
  EnableInterrupts();

  // Superloop
  while(1)
  {
    run_Debounce(sw1_db_obj);
    run_Debounce(sw2_db_obj);

    // Do not parse keyboard input during playback mode
    if(track[0]->mb_flag == 0) {
      run_Debounce(C_db_obj);
      run_Debounce(D_db_obj);
      run_Debounce(E_db_obj);
      run_Debounce(F_db_obj);
      run_Debounce(G_db_obj);
      run_Debounce(A_db_obj);
      run_Debounce(B_db_obj);
    }

    for(i = 0; i < NUM_TRACKS; i++) {
      MusicBox_run(track[i]);
    }

    Sound_run();
  }
}

/**
 * Initialize PF4 and PF0 for falling edge interrupts
 */
void setup_switches(void) {
  SYSCTL_RCGCGPIO_R |= 0x20;        // activate port
  clockDelay = SYSCTL_RCGCGPIO_R;   // allow time to finish activating
  GPIO_PORTF_LOCK_R = 0x4C4F434B;   // unlock GPIO Port
  GPIO_PORTF_CR_R = 0x11;           // allow changes
  GPIO_PORTF_DIR_R &= ~0x11;      // (c) make PF4,0 in (built-in button)
  GPIO_PORTF_AFSEL_R &= ~0x11;    //     disable alt funct
  GPIO_PORTF_DEN_R |= 0x11;       //     enable digital I/O
  GPIO_PORTF_PCTL_R &= ~0x000F000F; //  configure as GPIO
  GPIO_PORTF_AMSEL_R &= ~0x11;    //     disable analog functionality
  GPIO_PORTF_DATA_R &= ~0x0E;   //    clear output

  GPIO_PORTF_PUR_R |= 0x11;       //     enable weak pull-up
  GPIO_PORTF_IS_R &= ~0x11;       // (d) is edge-sensitive
  GPIO_PORTF_IBE_R |= 0x11;       //     is both edges
  GPIO_PORTF_ICR_R = 0x11;        // (e) clear all flags
  GPIO_PORTF_IM_R |= 0x11;        // (f) arm interrupt
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFFF)|0x00400000; // (g) bits:23-
21 for PORTF, set priority to 2
  NVIC_EN0_R |= (0x00000001 << 30);       // (h) enable interrupt 30 in NVIC
}

/**
 * Initialize PD and PC for piano input
 */
void setup_piano(void) {
  SYSCTL_RCGCGPIO_R |= 0x04;        // activate portC
  clockDelay = SYSCTL_RCGCGPIO_R;   // allow time to finish activating

  GPIO_PORTC_LOCK_R = 0x4C4F434B;   // unlock GPIO Port
  GPIO_PORTC_CR_R = 0x70;           // allow changes
  GPIO_PORTC_DIR_R &= ~0x70;        // (c) make PF4,0 in
  GPIO_PORTC_AFSEL_R &= ~0x70;    //     disable alt funct
  GPIO_PORTC_DEN_R |= 0x70;       //     enable digital I/O
  GPIO_PORTC_PCTL_R &= ~0x01110000; //  configure as GPIO
  GPIO_PORTC_AMSEL_R &= ~0x70;    //     disable analog functionality
  GPIO_PORTC_DATA_R &= ~0x70;   //    clear output
  GPIO_PORTC_PUR_R |= 0x70;       //     enable weak pull-up

  GPIO_PORTC_IS_R &= ~0x70;       // (d) is edge-sensitive
  GPIO_PORTC_IBE_R |= 0x70;       //     is both edges
  GPIO_PORTC_ICR_R = 0x70;        // (e) clear all flags
  GPIO_PORTC_IM_R |= 0x70;        // (f) arm interrupt

  NVIC_PRI0_R = (NVIC_PRI0_R&0xFF00FFFF)|0x00A00000; // (g) set priority to 5
  NVIC_EN0_R |= (0x00000001 << 2);        // (h) enable interrupt 2 in NVIC

  SYSCTL_RCGCGPIO_R |= 0x08;        // activate portD
  clockDelay = SYSCTL_RCGCGPIO_R;   // allow time to finish activating

  GPIO_PORTD_LOCK_R = 0x4C4F434B;   // unlock GPIO Port
  GPIO_PORTD_CR_R = 0x0F;           // allow changes
  GPIO_PORTD_DIR_R &= ~0x0F;        // (c) make PF4,0 in
  GPIO_PORTD_AFSEL_R &= ~0x0F;    //     disable alt funct
  GPIO_PORTD_DEN_R |= 0x0F;       //     enable digital I/O
  GPIO_PORTD_PCTL_R &= ~0x0000FFFF; //  configure as GPIO
  GPIO_PORTD_AMSEL_R &= ~0x0F;    //     disable analog functionality
  GPIO_PORTD_DATA_R &= ~0x0F;   //    clear output
  GPIO_PORTD_PUR_R |= 0x0F;       //     enable weak pull-up

  GPIO_PORTD_IS_R &= ~0x0F;       // (d) is edge-sensitive
  GPIO_PORTD_IBE_R |= 0x0F;       //     is both edges
  GPIO_PORTD_ICR_R = 0xFF;        // (e) clear all flags
  GPIO_PORTD_IM_R |= 0x0F;        // (f) arm interrupt

  NVIC_PRI0_R = (NVIC_PRI0_R&0x00FFFFFF)|0x80000000; // (g) set priority to 4
  NVIC_EN0_R |= (0x00000001 << 3);        // (h) enable interrupt 3 in NVIC
}


// Handle PORTD Interrupts
void GPIOPortD_Handler(void){   // called on touch of either SW1 or SW2
  if(GPIO_PORTD_RIS_R&0x01){        // note_C
    GPIO_PORTD_ICR_R |= 0x01;
    notify_Debounce(C_db_obj);
  } else if(GPIO_PORTD_RIS_R&0x02){ // note_D
```

```
    GPIO_PORTD_ICR_R |= 0x02;
    notify_Debounce(D_db_obj);
  } else if(GPIO_PORTD_RIS_R&0x04){ // note_E
    GPIO_PORTD_ICR_R |= 0x04;
    notify_Debounce(E_db_obj);
  } else if(GPIO_PORTD_RIS_R&0x08){ // note_F
    GPIO_PORTD_ICR_R |= 0x08;
    notify_Debounce(F_db_obj);
  }
}

// Handle PORTC Interrupts
void GPIOPortC_Handler(void){   // called on touch of either SW1 or SW2
  if(GPIO_PORTC_RIS_R&0x10){ // note_G
    GPIO_PORTC_ICR_R |= 0x10;
    notify_Debounce(G_db_obj);
  } else if(GPIO_PORTC_RIS_R&0x20){ // note_A
    GPIO_PORTC_ICR_R |= 0x20;
    notify_Debounce(A_db_obj);
  } else if(GPIO_PORTC_RIS_R&0x40){ // note_B
    GPIO_PORTC_ICR_R |= 0x40;
    notify_Debounce(B_db_obj);
  }
}

// Handle PORTF Interrupts
void GPIOPortF_Handler(void){   // called on touch of either SW1 or SW2
  if(GPIO_PORTF_RIS_R&0x01){    // SW2 touched
    GPIO_PORTF_ICR_R |= 0x01;   // acknowledge flag0
      notify_Debounce(sw2_db_obj);
  } else if(GPIO_PORTF_RIS_R&0x10){     // SW1 touched
    GPIO_PORTF_ICR_R |= 0x10;   // acknowledge flag4
      notify_Debounce(sw1_db_obj);
  }
}
```

5. CONCLUSION

It was fun creating more than what was necessary. It gave me some nostalgia from when I made music cards back in high school. Albeit the piano I created can play multiple tones at the same time.

I was also finally able to implement the FSM debounce circuit. Some may say that it’s overkill, but it is honestly the best debounce I’ve created in my entire hobby career. Simple timeout debounces just aren’t enough when it gets this complex.

I found it really cool to build our our DAC. It reminds me of the DAC that can be found on the VGA output of our FPGAs. I never realized how they worked until this class.

6. References

[1] “Tiva™ C Series TM4C123G LaunchPad Evaluation Board User’s Guide,” April 2013. [Online]. Available: https://www.ti.com/lit/ug/spmu296/spmu296.pdf.

[2] “FT232R USB UART IC Datasheet,” [Online] Available: https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf, pp. 19.