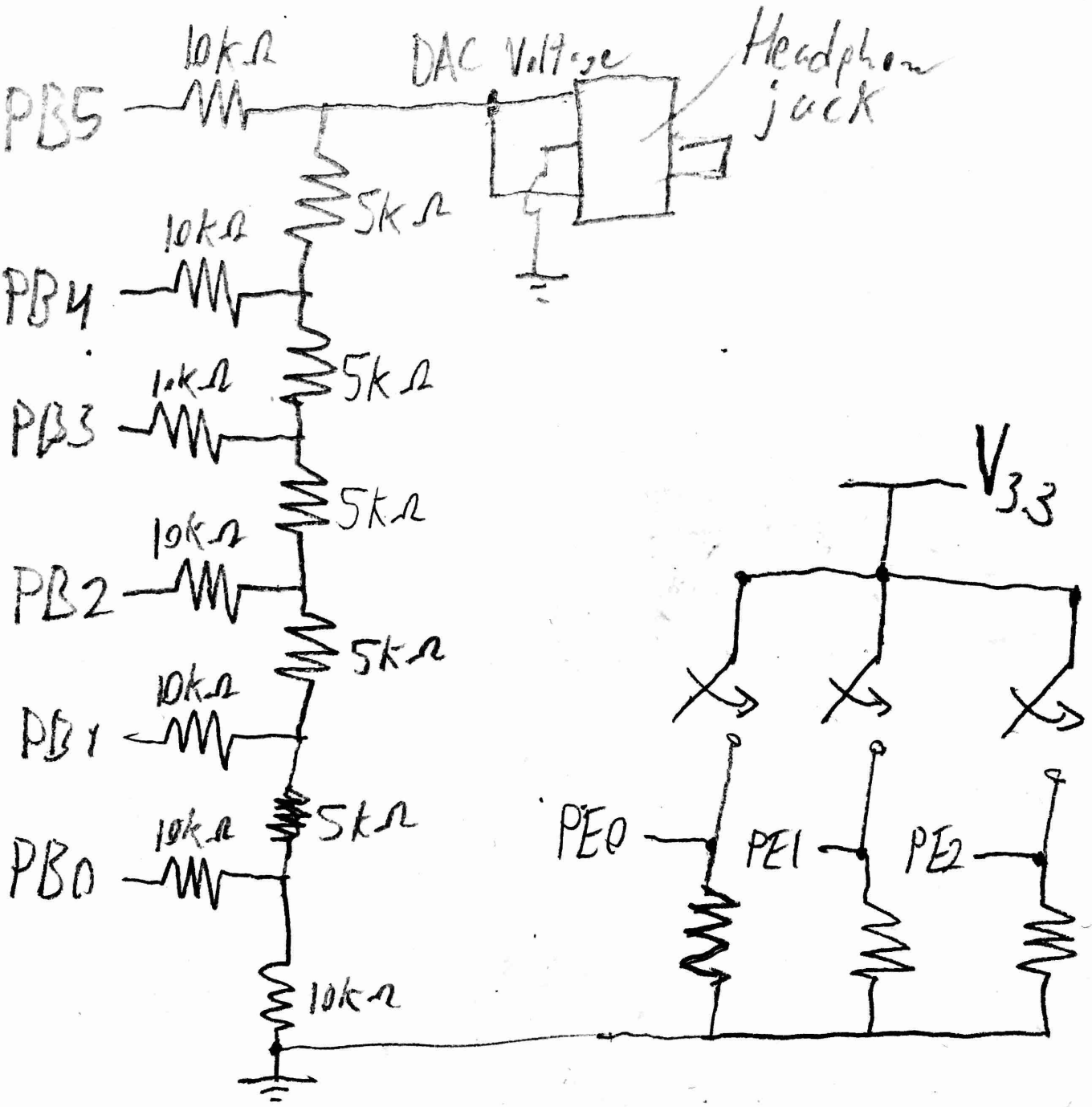
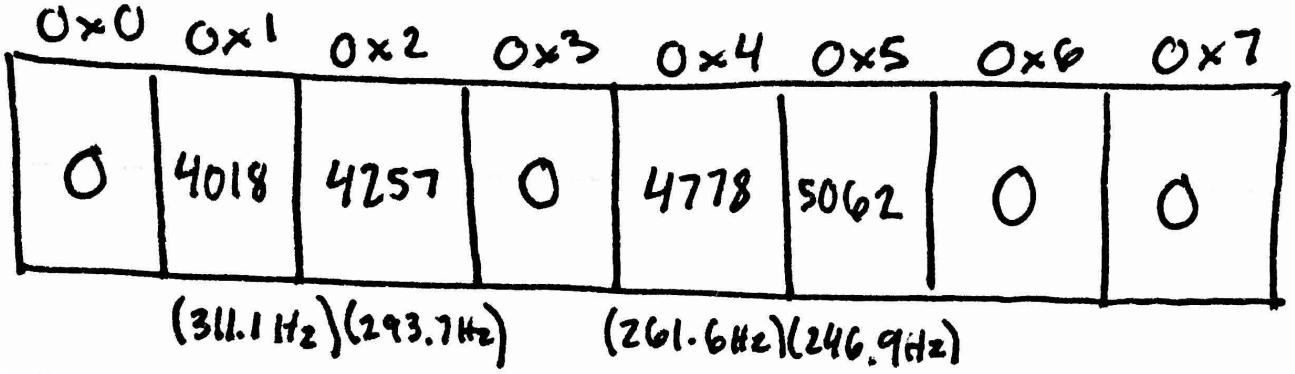

10kΩ
DAC Voltage
Headphone jack
PB5
5kΩ
10kΩ
PB4
5kΩ
10kΩ
PB3
5kΩ
10kΩ
PB2
5kΩ
10kΩ
PB1
5kΩ
10kΩ
PB0
10kΩ
V3.3
PE0
PE1
PE2

DATA STRUCTURE



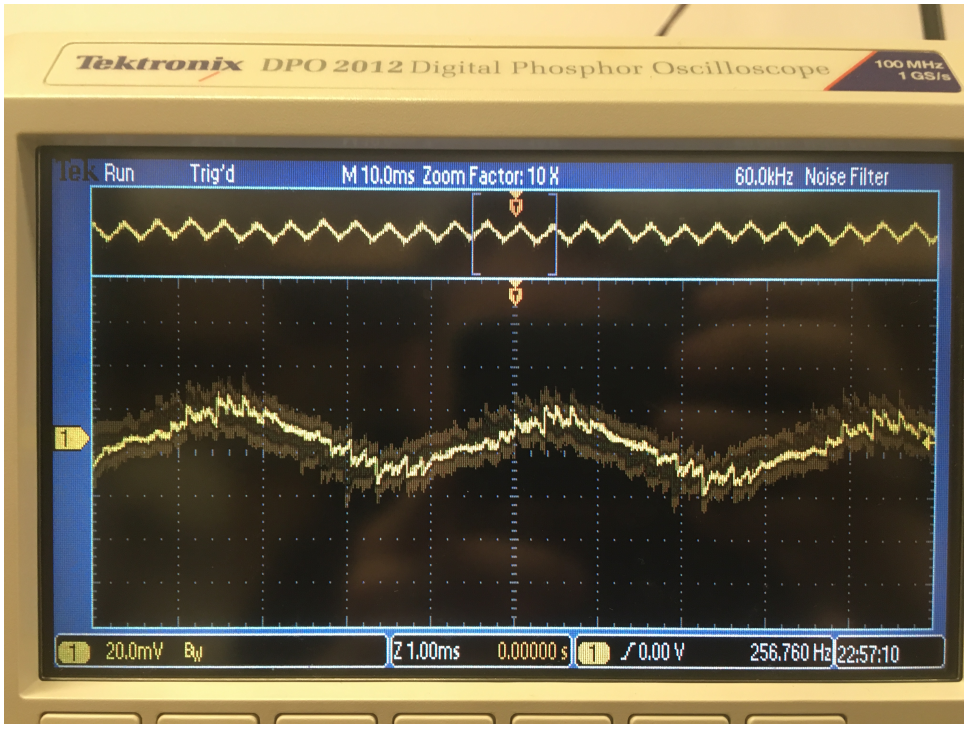
Implemented using a 6-bit 64 element sine wave

- The voltage values for this wave is contained in Sound.c under const unsigned short wave[64]

DAC Performance

Port B	Theoretical DAC Voltage	Measured DAC Voltage	Percent Error
0x0	0.000	0.0008	N/A
0x1	0.0513	0.0527	2.83
0x7	0.359	0.363	1.18
0x8	0.410	0.417	1.71
0x0F	0.769	0.779	1.33
0x10	0.820	0.828	0.98
0x11	0.871	0.879	0.89
0x12	0.922	0.932	1.03
0x1F	1.589	1.605	1.02
0x20	1.640	1.635	0.30
0x21	1.691	1.687	0.24
0x2F	2.41	2.41	0.00
0x30	2.46	2.46	0.00
0x31	2.51	2.51	0.00
0x3E	3.18	3.19	0.31
0x3F	3.28	3.24	1.22
		Range:	0.0008 to 3.24 V
		Precision:	$\pm 2^6$ levels
		Resolution:	0.052V
		Accuracy:	$\pm 5\%$
		Average accuracy:	$> 98\%$

Oscilloscope readings for C, octave 4 (ideal 261.6 Hz, actual 256.7 Hz)



Questions:

1. The interrupt trigger is periodic, determined by whatever pitch is being played.

2. The SysTick_Handler function is in SysTickInts.c, but the actual vector is defined in the TM4C header file and points to the address in the interrupt vector table where that ISR is defined.

3. The current instruction is finished. Registers 0-3, 12, LR, PC, and PSR are pushed onto the stack. LR is set to 0xFFFFFFF9. IPSR is set to the interrupt number. PC is loaded with the interrupt vector.

4. The most significant 24 bits of the LR are 1, so the program knows to pop the top 8 registers off of the stack and handle the return.

```c
// Lab6.c
// Runs on LM4F120 or TM4C123
// Use SysTick interrupts to implement a 4-key digital piano
// MOOC lab 13 or EE319K lab6 starter
// Program written by: Emily Steck and Trey Boehm
// Date Created: 2017-03-06
// Last Modified: 2017-03-23
// Lab number: 6
// Hardware connections
//     PB0 through PB5: DAC output bits
//     PE0 through PE2: Synthesizer button inputs
//     PF0 and PF4: On-board start/stop buttons


#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "Sound.h"
#include "Piano.h"
#include "TExaS.h"
#include "dac.h"
#include "SysTickInts.h"

// basic functions defined at end of startup.s
void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts

volatile uint32_t HeartBeat_Counter = 0;
void HeartBeat_Init(void);
void HeartBeat_Toggle(void);

int main(void){
    TExaS_Init(SW_PIN_PE3210, DAC_PIN_PB3210, ScopeOn);    // bus clock at 80 MHz
    DAC_Init();
    Piano_Init();
    SysTick_Init(A4);
    HeartBeat_Init();
    // other initialization
    EnableInterrupts();
    while (1) {
        HeartBeat_Counter++;
        if (HeartBeat_Counter > 600000) {
            HeartBeat_Counter = 0;
            HeartBeat_Toggle();
        }
        // Infinite loop. SysTick_Handler is called periodically.
    }
}

void HeartBeat_Init(void) {
    uint8_t i;
    SYSCTL_RCGC2_R |= 0x020;
    for (i = 0; i < 4; i++) ; // Wait for clock to stabilize
    GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY;
    GPIO_PORTF_CR_R |= 0x04;
    GPIO_PORTF_DIR_R |= 0x04;
    GPIO_PORTF_DEN_R |= 0x04;
    GPIO_PORTF_PUR_R |= 0x04;
```

```
58  }
59
60  void HeartBeat_Toggle(void) {
61      uint8_t status = GPIO_PORTF_DATA_R & 0x04;
62      if (status == 0x04) {
63          GPIO_PORTF_DATA_R &= ~0x04;
64      } else {
65          GPIO_PORTF_DATA_R |= 0x04;
66      }
67  }
68
```

```c
// Piano.c
// This software configures the off-board piano keys
// Runs on LM4F120 or TM4C123
// Program written by: Emily Steck and Trey Boehm
// Date Created: 2017-03-06
// Last Modified: 2017-03-23
// Lab number: 6
// Hardware connections
//     PB0 through PB5: DAC output bits
//     PE0 through PE2: Synthesizer button inputs

// Code files contain the actual implemenation for public functions
// this file also contains an private functions and private data
#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "Sound.h"
#include "Piano.h"

// The three pitches activated by the button presses
unsigned int pianoNotes[] = {
    0,    // 0x0: If no buttons, play no sound
    Ees4, // 0x1: Button 1
    D4,   // 0x2: Button 2
    0,    // 0x3: Button 2 and 1 simultaneously
    C4,   // 0x4: Button 3
    B3,   // 0x5: Button 3 and 1 simultaneously
    0,    // 0x6: Button 3 and 2 simultaneously
    0,    // 0x7: Button 3 and 2 and 1 simultaneously
};

// **************Piano_Init**********************
// Initialize piano key inputs, called once
// Input: none
// Output: none
void Piano_Init(void) {
    uint8_t i;
    SYSCTL_RCGC2_R |= 0x010;
    for (i = 0; i < 4; i++) ; // Wait for clock to stabilize
    GPIO_PORTE_LOCK_R = GPIO_LOCK_KEY;
    GPIO_PORTE_DIR_R &= ~0x07;
    GPIO_PORTE_DEN_R |= 0x07;
}

// **************Piano_In**********************
// Input from piano key inputs
// Input: none
// Output: 0 to 7 depending on keys
// 0x01 is just Key0, 0x02 is just Key1, 0x04 is just Key2
uint32_t Piano_In(void) {
    return (GPIO_PORTE_DATA_R & 0x07);
}

```

```c
// Sound.c
// This module contains the SysTick ISR that plays sound. Also
//     initializes the sequencer (port F buttons) and defines the song
//     data in an array of structs.
// Runs on LM4F120 or TM4C123
// Program written by: Emily Steck and Trey Boehm
// Date Created: 2017-03-06
// Last Modified: 2017-03-23
// Lab number: 6
// Hardware connections
//     PB0 through PB5: DAC output bits
//     PE0 through PE2: Synthesizer button inputs
//     PF0 and PF4: On-board start/stop buttons

// Code files contain the actual implemenation for public functions
// this file also contains an private functions and private data
#include <stdint.h>
#include "dac.h"
#include "SysTickInts.h"
#include "tm4c123gh6pm.h"

// Pointer to the next voltage level in the sine wave to output
volatile uint8_t wavePointer = 0;

// 6-bit 64-element sine wave, copy/pasted from Valvano's spreadsheet
const unsigned short wave[64] = {
    32,35,38,41,44,47,49,52,54,56,58,59,
    61,62,62,63,63,63,62,62,61,59,58,56,
    54,52,49,47,44,41,38,35,32,29,26,23,
    20,17,15,12,10,8,6,5,3,2,2,1,1,1,2,
    2,3,5,6,8,10,12,15,17,20,23,26,29
};
/*
const unsigned short wave[64] = {
    0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
    20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,
    36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,
    52,53,54,55,56,57,58,59,60,61,62,63};
*/
// **************Sound_Init*********************
// Initialize Systick periodic interrupts
// Called once, with sound initially off
// Input: interrupt period
//           Units to be determined by YOU
//           Maximum to be determined by YOU
//           Minimum to be determined by YOU
// Output: none
void Sound_Init(uint32_t period){
    SysTick_Init(period);
    DAC_Init();
}

// **************Sound_Play*********************
// Start sound output, and set Systick interrupt period
// Input: interrupt period
//           Units to be determined by YOU
//           Maximum to be determined by YOU
```

```
58  //           Minimum to be determined by YOU
59  //         input of zero disable sound output
60  // Output: none
61  void Sound_Play(uint32_t period) {
62      NVIC_ST_RELOAD_R = period-1;
63      NVIC_ST_CURRENT_R = 0;
64  }
65
66
```

```
// Sound.h
// This module contains the SysTick ISR that plays sound
// Runs on LM4F120 or TM4C123
// Program written by: Emily Steck and Trey Boehm
// Date Created: 2017-03-06
// Last Modified: 2017-03-21
// Lab number: 6
#include <stdint.h>
// Header files contain the prototypes for public functions
// this file explains what the module does

// Macros for notes. The periods and frequencies copied/pasted from
// Valvano's spreadsheet: Period = 80,000,000/64/Freq = 1,250,000/Freq
// Each macro includes the pitch, a possible modifier, and the standard
// octave designation. "is" indicates "sharp" and "es" indicates "flat"
#define REST    4444   // Musical silence is an exception
#define C7      597    // 2093 Hz
#define B6      633    // 1975.5 Hz
#define Bes6    670    // 1864.7 Hz
#define Ais6    670    // 1864.7 Hz
#define A6      710    // 1760 Hz
#define Aes6    752    // 1661.2 Hz
#define Gis6    752    // 1661.2 Hz
#define G6      797    // 1568 Hz
#define Ges6    845    // 1480 Hz
#define Fis6    845    // 1480 Hz
#define F6      895    // 1396.9 Hz
#define E6      948    // 1318.5 Hz
#define Ees6    1004   // 1244.5 Hz
#define Dis6    1004   // 1244.5 Hz
#define D6      1064   // 1174.7 Hz
#define Des6    1127   // 1108.7 Hz
#define Cis6    1127   // 1108.7 Hz
#define C6      1194   // 1046.5 Hz
#define B5      1265   // 987.8 Hz
#define Bes5    1341   // 932.3 Hz
#define Ais5    1341   // 932.3 Hz
#define A5      1420   // 880 Hz
#define Aes5    1505   // 830.6 Hz
#define Gis5    1505   // 830.6 Hz
#define G5      1594   // 784 Hz
#define Ges5    1689   // 740 Hz
#define Fis5    1689   // 740 Hz
#define F5      1790   // 698.5 Hz
#define E5      1896   // 659.3 Hz
#define Ees5    2009   // 622.3 Hz
#define Dis5    2009   // 622.3 Hz
#define D5      2128   // 587.3 Hz
#define Des5    2255   // 554.4 Hz
#define Cis5    2255   // 554.4 Hz
#define C5      2389   // 523.3 Hz
#define B4      2531   // 493.9 Hz
#define Bes4    2681   // 466.2 Hz
#define Ais4    2681   // 466.2 Hz
#define A4      2841   // 440 Hz
#define Aes4    3010   // 415.3 Hz
#define Gis4    3010   // 415.3 Hz
```

```
58  #define G4      3189   // 392 Hz
59  #define Ges4    3378   // 370 Hz
60  #define Fis4    3378   // 370 Hz
61  #define F4      3579   // 349.2 Hz
62  #define E4      3792   // 329.6 Hz
63  #define Ees4    4018   // 311.1 Hz
64  #define Dis4    4018   // 311.1 Hz
65  #define D4      4257   // 293.7 Hz
66  #define Des4    4510   // 277.2 Hz
67  #define Cis4    4510   // 277.2 Hz
68  #define C4      4778   // 261.6 Hz
69  #define B3      5062   // 246.9 Hz
70  #define Bes3    5363   // 233.1 Hz
71  #define Ais3    5363   // 233.1 Hz
72  #define A3      5682   // 220 Hz
73  #define Aes3    6020   // 207.7 Hz
74  #define Gis3    6020   // 207.7 Hz
75  #define G3      6378   // 196 Hz
76  #define Ges3    6757   // 185 Hz
77  #define Fis3    6757   // 185 Hz
78  #define F3      7159   // 174.6 Hz
79  #define E3      7584   // 164.8 Hz
80  #define Ees3    8035   // 155.6 Hz
81  #define Dis3    8035   // 155.6 Hz
82  #define D3      8513   // 146.8 Hz
83  #define Des3    9019   // 138.6 Hz
84  #define Cis3    9019   // 138.6 Hz
85  #define C3      9556   // 130.8 Hz
86
87  // **************Sound_Init*********************
88  // Initialize Systick periodic interrupts
89  // Called once, with sound initially off
90  // Input: interrupt period
91  //           Units to be determined by YOU
92  //           Maximum to be determined by YOU
93  //           Minimum to be determined by YOU
94  // Output: none
95  void Sound_Init(uint32_t period);
96
97  // **************Sound_Play*********************
98  // Start sound output, and set Systick interrupt period
99  // Input: interrupt period
100 //           Units to be determined by YOU
101 //           Maximum to be determined by YOU
102 //           Minimum to be determined by YOU
103 //         input of zero disable sound output
104 // Output: none
105 void Sound_Play(uint32_t period);
106
```

```c
// SysTickInts.c
// Runs on LM4F120/TM4C123
// Use the SysTick timer to request interrupts at a particular period.
// Daniel Valvano
// October 11, 2012

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014

   Program 5.12, section 5.7

 Copyright 2014 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/
 */

#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "SysTickInts.h"
#include "Piano.h"
#include "Sound.h"
#include "dac.h"

#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode

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 12.5ns (assuming 50 MHz clock)
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
void SysTick_Init(uint32_t period) {
    long sr;
    sr = StartCritical();
    NVIC_ST_CTRL_R = 0;         // disable SysTick during setup
    NVIC_ST_RELOAD_R = period-1;// reload value
    NVIC_ST_CURRENT_R = 0;      // any write to current clears it
    NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R & 0x00FFFFFF) | 0x40000000; // priority 2
                                // enable SysTick with core clock and interrupts
    NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC+NVIC_ST_CTRL_INTEN;
```

```
58      EndCritical(sr);
59  }
60
61  void SysTick_Handler(void) {
62      extern volatile uint8_t wavePointer;
63      extern unsigned short wave[];
64      volatile int buttons = Piano_In();
65      extern unsigned int pianoNotes[];
66      if (buttons) {
67          Sound_Play(pianoNotes[buttons]); // Get the note specified in Piano.c
68          // Only incremente the pointer if something needs to be played
69          wavePointer = (wavePointer + 1) & 0x3F;
70      }
71      DAC_Out(wave[wavePointer]);
72
73      // SysTick automatically acknowledges the ISR completion
74  }
75
```