EE445L – Lab5: Music Player and Audio Amp

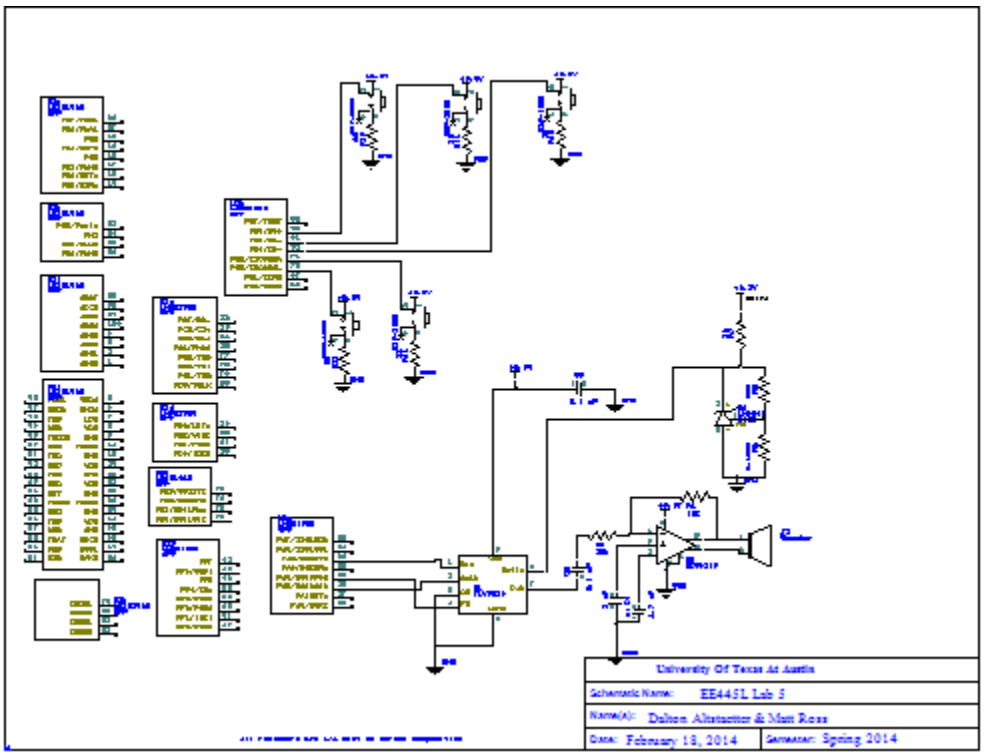
Harley Ross and Dalton Altstaetter

2/18/14

GOALS

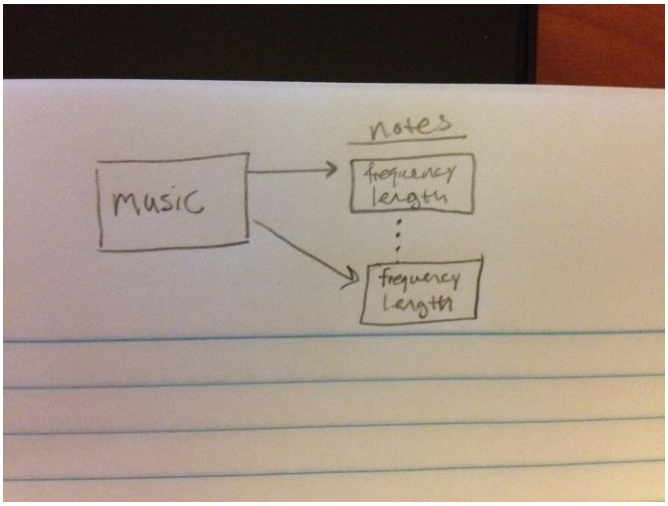
This objectives of the project are learning DAC conversion, learning SPI/SSI interface, design a data structure to represent music, and develop a system that plays sound.

HARWARE DESIGN



SOFTWARE DESIGN

Music



MEASUREMENT DATA

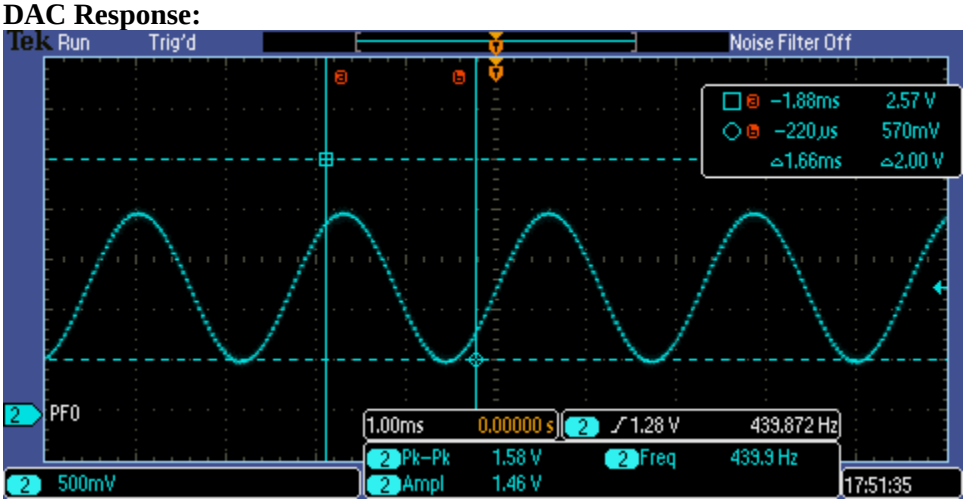
Calculated Data:

Resolution: 8.057*10^-4

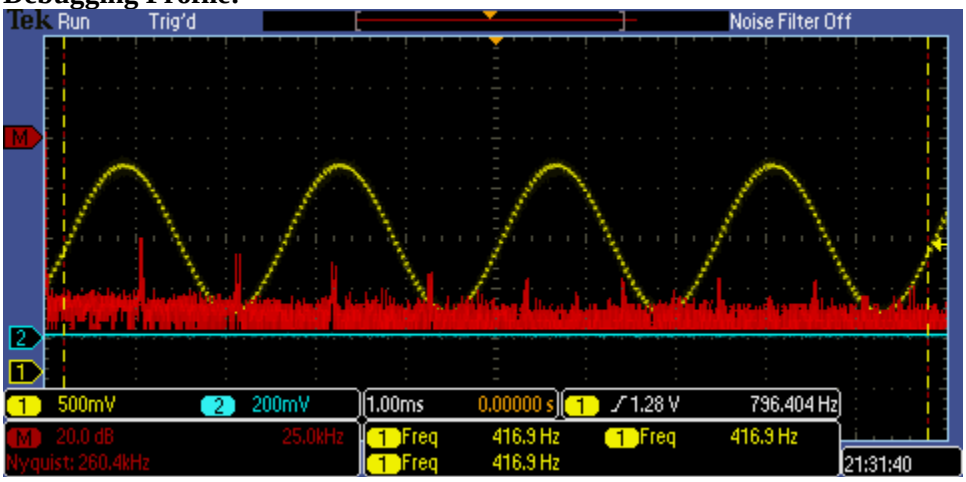
Range: 3.3V

Accuracy: .02%

DAC Response:



Debugging Profile:



Current required to run the system:

Without music playing:

With music playing:

ANALYSIS AND DISCUSION

1. Three DAC errors are the offset error, the span error, and the gain error. The offset error are the values that produce a 0 voltage that are not 0. The span error is the difference between the precision and the offset error. The gain error is the actual span divided by the expected span.
2. The data available time should overlap the data required time so the it is latched correctly. We selected the SSI frequency so that it overlaps the required time by the DAC.
3. The frequency range of the spectrum analyzer is determined by the cursors and max sampling rate on the oscilloscope.
4. We needed more voltage given by the amplifier in order to power the speaker.

SOURCE CODE

Timer

```
// Runs on LM3S1968
// Use Timer0A in periodic mode to request interrupts at a particular
// period.
// Daniel Valvano
// September 14, 2011

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
  Program 7.5, example 7.6

 Copyright 2011 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 "timer0A.h"
#include "Systick.h"
#include "DAC.h"
#include "lm3s1968.h"
#include "Music.h"

#define MAXVOLUME 20
#define DUMPSIZE 175

extern const unsigned short* instrumentPtr;
extern NotePtr songPtr; // this is no longer a pointer to a pointer
extern int wait0;
extern int volume;
extern int songNumber;
extern const unsigned short SinWave1[128];

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
void (*PeriodicTask)(void);   // user function
void Debugging_Profile(void);

static unsigned char index = 0;
static unsigned char index2 = 0;
static unsigned char index3 = 0;
unsigned long debugArray[DUMPSIZE];
unsigned long songNote;

void Timer2A_Handler(void)
{
#ifndef CHORD
	if(wait0 == -1)
	{
		wait0 = songPtr->duration;
	}

	TIMER2_ICR_R = TIMER_ICR_TATOCINT;// acknowledge timer2A timeout

	if(wait0 == 0)
	{
		songNote++; // move to the next note if duration = 0
		wait0 = songPtr[songNote].duration;
		Debugging_Profile();
	}
	else
	{wait0--;}

	TIMER2_TAILR_R = 62500; // interrupt every 62.5 ms

	if(songNote > ENDSONG)
	{
		songNote = ENDSONG;
	}
#else
	TIMER2_ICR_R = TIMER_ICR_TATOCINT;
	TIMER2_TAILR_R = 62500;
#endif
}


void Timer0B_Handler(void)
{
	TIMER0_ICR_R = TIMER_ICR_TBTOCINT;
	TIMER0_TBILR_R = G;
#ifdef CHORD
	DAC_Out(instrumentPtr[index]*volume/MAXVOLUME+instrumentPtr[index2]*volume/MAXVOLU
ME+instrumentPtr[index3]*volume/MAXVOLUME);
	index3 = ((index3+1)&0x7F); // [0,127]
#endif
}

void Timer1A_Handler(void)
{
	TIMER1_ICR_R = TIMER_ICR_TATOCINT;
	TIMER1_TAILR_R = C; // reload value
```

```
#ifdef CHORD
	DAC_Out(instrumentPtr[index]*volume/MAXVOLUME+instrumentPtr[index2]*volume/MAXVOLU
ME+instrumentPtr[index3]*volume/MAXVOLUME);
	index2 = ((index2+1)&0x7F); // [0,127]
#endif
}

void Timer0A_Handler(void)
{
	GPIO_PORTG_DATA_R ^= 0x04;
  TIMER0_ICR_R = TIMER_ICR_TATOCINT;// acknowledge timer0A timeout

#ifdef CHORD
	DAC_Out(instrumentPtr[index]*volume/MAXVOLUME+instrumentPtr[index2]*volume/MAXVOLU
ME+instrumentPtr[index3]*volume/MAXVOLUME);
	TIMER0_TAILR_R = E;
	index = ((index+1)&0x7F);
#else
	TIMER0_TAILR_R = songPtr[songNote].frequency;
	if(songPtr[songNote].frequency != REST)
	{
		DAC_Out(instrumentPtr[index]*volume/MAXVOLUME);
		index = ((index+1)&0x7F); // index = [0,127]
	}
	else
	{
		return;
	}
#endif

/*
//if(!GPIO_PORTF_DATA_R)
//{
//	TIMER0_TAILR_R = 1000;
//	return;
//}

//DAC_Out(SinWave1[index]);
//	//DAC_Out((SinWave1[index]));/envelope[index])/16);
//	//DAC_Out(Bassoon[index]);
//	//DAC_Out(Horn[index]*3-1063);
//
//	//DAC_Out(Flute[index]);
//	//DAC_Out(Trumpet[index]);
//	//DAC_Out(Oboe[index]*((index%8)/8));
//		if(GPIO_PORTF_DATA_R &0x01)
//		{
//			DAC_Out(SinWave1[index]*6/10);
//		  TIMER0_TAILR_R = Ab;//200;
//		}
//		else if(GPIO_PORTF_DATA_R &0x02)
//		{
//			DAC_Out(SinWave1[index]*6/10);
//			TIMER0_TAILR_R = Bb;
//		 }
//		else if(GPIO_PORTF_DATA_R &0x04)
//		{
//			DAC_Out(SinWave1[index]*6/10);
```

```
//                  TIMER0_TAILR_R = Db;
//            }
//            else if(GPIO_PORTF_DATA_R &0x08)
//            {
//                  DAC_Out(SinWave1[index]*6/10);
//                  TIMER0_TAILR_R = Eb;
//                  }
//            else if(GPIO_PORTF_DATA_R &0x10)
//            {// this one just dont work
//                  DAC_Out(SinWave1[index]*6/10);
//                  TIMER0_TAILR_R = D;
//            }
//            else if(GPIO_PORTF_DATA_R &0x20)
//            {
//                  DAC_Out(SinWave1[index]*6/10);
//                  TIMER0_TAILR_R = E;
//            }
//            else if(GPIO_PORTF_DATA_R &0x40)
//            {
//                  DAC_Out(SinWave1[index]*6/10);
//                  TIMER0_TAILR_R = F;
//            }
//            else if(GPIO_PORTF_DATA_R &0x80)
//            {
//                  DAC_Out(SinWave1[index]*6/10);
//                  TIMER0_TAILR_R = Gb;
//            }


      //index = ((index+1)&0x3F); // 0-63
      //index = ((index+1)&0x1F); // 0-31
      */
}


// ***************** Timer0A_Init ***************
// Activate Timer0A interrupts to run user task periodically
// Inputs:  task is a pointer to a user function
//          period in usec
// Outputs: none
// timer0A is for outputting the sine Wave and creating sound
void Timer0A_Init(void(*task)(void), unsigned short period)
{
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER0; // 0) activate timer0
  PeriodicTask = task;             // user function
  TIMER0_CTL_R &= ~0x00000001;     // 1) disable timer0A during setup
  TIMER0_CFG_R = 0x00000004;       // 2) configure for 16-bit timer mode
  TIMER0_TAMR_R = 0x00000002;      // 3) configure for periodic mode
  TIMER0_TAILR_R = period-1;       // 4) reload value
  TIMER0_TAPR_R = 0;              // 5) 20 ns timer0A //49=>1 us timer0A
  TIMER0_ICR_R = 0x00000001;       // 6) clear timer0A timeout flag
  TIMER0_IMR_R |= 0x00000001;      // 7) arm timeout interrupt
  NVIC_PRI4_R = (NVIC_PRI4_R&0x01FFFFFF)|0x40000000; // 8) priority 2
  NVIC_EN0_R |= NVIC_EN0_INT19;    // 9) enable interrupt 19 in NVIC
  TIMER0_CTL_R |= 0x00000001;      // 10) enable timer0A

//     SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER0; // 0) activate timer0
//  PeriodicTask = task;             // user function
```

```
//  TIMER0_CTL_R &= ~TIMER_CTL_TAEN; // 1) disable timer0A during setup
//  TIMER0_CFG_R = TIMER_CFG_16_BIT; // 2) configure for 16-bit timer mode
//  TIMER0_TAMR_R = TIMER_TAMR_TAMR_PERIOD;  // 3) configure for periodic mode
//  TIMER0_TAILR_R = period-1;       // 4) reload value
//  TIMER0_TAPR_R = 0;               // 5) 20 ns timer0A //49=>1 us timer0A
//  TIMER0_ICR_R = TIMER_ICR_TATOCINT;       // 6) clear timer0A timeout flag
//  TIMER0_IMR_R |= TIMER_IMR_TATOIM;      // 7) arm timeout interrupt
//  NVIC_PRI4_R = (NVIC_PRI4_R&0x01FFFFFF)|0x40000000; // 8) priority 2
//  NVIC_EN0_R |= NVIC_EN0_INT19;    // 9) enable interrupt 19 in NVIC
//  TIMER0_CTL_R |= TIMER_CTL_TAEN;      // 10) enable timer0A
//


}

void Timer0B_Init(void(*task)(void), unsigned short period)
{
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER0; // 0) activate timer0
  PeriodicTask = task;             // user function
  TIMER0_CTL_R &= ~TIMER_CTL_TBEN;     // 1) disable timer0B during setup
  TIMER0_CFG_R = TIMER_CFG_16_BIT;       // 2) configure for 16-bit timer mode
  TIMER0_TBMR_R = TIMER_TBMR_TBMR_PERIOD;      // 3) configure for periodic mode
  TIMER0_TBILR_R = period-1;       // 4) reload value
  TIMER0_TBPR_R = 0;               // 5) 20 ns timer0A //49=>1 us timer0A
  TIMER0_ICR_R = TIMER_ICR_TBTOCINT;       // 6) clear timer0A timeout flag
  TIMER0_IMR_R |= TIMER_IMR_TBTOIM;      // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0xFFFFFF1F)|0x00000040; // 8) priority 2
  NVIC_EN0_R |= NVIC_EN0_INT20;    // 9) enable interrupt 20 in NVIC
  TIMER0_CTL_R |= TIMER_CTL_TBEN;      // 10) enable timer0B
}

// timer1A is for the chord to play
void Timer1A_Init(void(*task)(void), unsigned short period)
{
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER1; // 0) activate timer0
  PeriodicTask = task;             // user function
  TIMER1_CTL_R &= ~0x00000001;     // 1) disable timer1A during setup
  TIMER1_CFG_R = 0x00000004;       // 2) configure for 16-bit timer mode
  TIMER1_TAMR_R = 0x00000002;      // 3) configure for periodic mode
  TIMER1_TAILR_R = period-1;       // 4) reload value
  TIMER1_TAPR_R = 0;               // 5) 20 ns timer1A //49=>1 us timer0A
  TIMER1_ICR_R = 0x00000001;       // 6) clear timer1A timeout flag
  TIMER1_IMR_R |= 0x00000001;      // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0xFFFF1FFF)|0x00004000; // 8) priority 2
  NVIC_EN0_R |= NVIC_EN0_INT21;    // 9) enable interrupt 19 in NVIC
  TIMER1_CTL_R |= 0x00000001;      // 10) enable timer1A
}

// timer2A is for the the length of the note to play
void Timer2A_Init(void(*task)(void), unsigned short period)
{
      SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER2; // 0) activate timer2
  PeriodicTask = task;             // user function
  TIMER2_CTL_R &= ~0x00000001;     // 1) disable timer2A during setup
  TIMER2_CFG_R = 0x00000004;       // 2) configure for 16-bit timer mode
  TIMER2_TAMR_R = 0x00000002;      // 3) configure for periodic mode
  TIMER2_TAILR_R = period-1;       // 4) reload value
  TIMER2_TAPR_R = 49;              // 5) 1us timer2A
```

```
  TIMER2_ICR_R = 0x00000001;       // 6) clear timer2A timeout flag
  TIMER2_IMR_R |= 0x00000001;      // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0x1FFFFFFF)|0x40000000; // 8) priority 2
  NVIC_EN0_R |= NVIC_EN0_INT23;    // 9) enable interrupt 23 in NVIC
  TIMER2_CTL_R |= 0x00000001;      // 10) enable timer2A
}

void Debugging_Profile(void)
{
       static unsigned long ind = 0;
       if(ind < DUMPSIZE)
       {
              debugArray[ind] = NVIC_ST_CURRENT_R;
              ind++;
       }
  return;
}
```

SysTick

```
// SysTick.c
// Runs on LM3S1968
// Provide functions that initialize the SysTick module, wait at least a
// designated number of clock cycles, and wait approximately a multiple
// of 10 milliseconds using busy wait.  After a power-on-reset, the
// LM3S1968 gets its clock from the 12 MHz internal oscillator, which
// can vary by +/- 30%.  If you are using this module, you probably need
// more precise timing, so it is assumed that you are using the PLL to
// set the system clock to 50 MHz.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// February 22, 2012

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
   Program 2.11, Section 2.6

 Copyright 2011 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 "lm3s1968.h"
#include "SysTick.h"

void SysTick_Wait(unsigned long delay);
void SysTick_Wait10ms(unsigned long delay);
void SysTick_Init(void);
```

```
// Initialize SysTick with busy wait running at bus clock.
void SysTick_Init(void)
{
  NVIC_ST_CTRL_R = 0;             // disable SysTick during setup
  NVIC_ST_RELOAD_R = 0x00FFFFFF;  // MAX 24-bit reload value
  NVIC_ST_CURRENT_R = 0;          // any write to current clears it
                                  // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC;
}
// Time delay using busy wait.
// The delay parameter is in units of the core clock. (units of 20 nsec for 50 MHz clock)
void SysTick_Wait(unsigned long delay)
{
  volatile unsigned long elapsedTime;
  unsigned long startTime = NVIC_ST_CURRENT_R;
  do{
    elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
  }
  while(elapsedTime <= delay);
}
// Time delay using busy wait.
// This assumes 50 MHz system clock.
void SysTick_Wait10ms(unsigned long delay)
       {
  unsigned long i;
  for(i=0; i<delay; i++)
       {
    SysTick_Wait(500000);  // wait 10ms (assumes 50 MHz clock)
  }
}
```

Switches

```
//Switches.c

#include "Switches.h"
#include "lm3s1968.h"
#include "timer0A.h"
#include "music.h"

#define PLAYING 0
#define PAUSE   1

extern int volume;
extern unsigned long songNote;
extern const unsigned short* instrumentPtr;
extern Instruments* instrumentStruct;
extern const unsigned short* instrArray[6];

extern void DisableInterrupts(void); // Disable interrupts
extern void EnableInterrupts(void);  // Enable interrupts
extern long StartCritical (void);    // previous I bit, disable interrupts
extern void EndCritical(long sr);    // restore I bit to previous value

unsigned long* MusicPlayOrPause(unsigned long* songIndex);
void DecreaseVolume(int* volumePtr);
void IncreaseVolume(int* volumePtr);
void Rewind(unsigned long* songIndex, unsigned long* enableSong);
void Change_Instruments(const unsigned short* instrumentsPtr);
static void Delay(unsigned long count);
```

```
unsigned long* MusicPlayOrPause(unsigned long* songIndex)
{
	static unsigned long enableDisable=0;
	Delay(100000);
	if(GPIO_PORTB_DATA_R & 0x04)
	{
		if(songNote < ENDSONG)
		{
			if(enableDisable == PLAYING)
			{
				// disable Timer intrpts
				NVIC_DIS0_R = NVIC_DIS0_INT19; // Timer0A
				NVIC_DIS0_R = NVIC_DIS0_INT20; // Timer0B
				NVIC_DIS0_R = NVIC_DIS0_INT21; // Timer1A
				NVIC_DIS0_R = NVIC_DIS0_INT23; // Timer2A
				enableDisable = PAUSE;
			}
			else
			{
				// enable Timer intrpts
				NVIC_EN0_R = NVIC_EN0_INT19; // Timer0A
				NVIC_EN0_R = NVIC_EN0_INT20; // Timer0B
				NVIC_EN0_R = NVIC_EN0_INT21; // Timer1A
				NVIC_EN0_R = NVIC_EN0_INT23; // Timer2A
				enableDisable = PLAYING;
			}
		}
		else
		{
			*songIndex = 0;
			// this is just to be safe but shouldn't be necessary
			if(enableDisable == PAUSE) // re-enable all interrupts
			{
				NVIC_EN0_R = NVIC_EN0_INT19;
				NVIC_EN0_R = NVIC_EN0_INT23;
				enableDisable = PLAYING;
			}
		}
	}
	Delay(5000000);
	return &enableDisable;
}

void DecreaseVolume(int* volumePtr)
{
	Delay(100000);
	if(GPIO_PORTB_DATA_R & 0x08)
	{
		(*volumePtr)--;
		if( *volumePtr < MIN_VOLUME)
		{
			*volumePtr = MIN_VOLUME;
		}
	}
	Delay(5000000);
}
```

```
void IncreaseVolume(int* volumePtr)
{
	Delay(100000);
	if(GPIO_PORTB_DATA_R & 0x10)
	{
#ifndef CHORD
		(*volumePtr)++;
		if((*volumePtr) > MAX_VOLUME)
		{
			(*volumePtr) = MAX_VOLUME;
		}
#else // too prevent clippng on the chords
		(*volumePtr)++;
		if((*volumePtr) > MAX_VOLUME/3)
		{
			(*volumePtr) = MAX_VOLUME/3-1;
		}

#endif
	}
	Delay(5000000);
}



void Rewind(unsigned long* songIndex,unsigned long* enableSong)
{
	Delay(100000);
	if(GPIO_PORTB_DATA_R & 0x20)
	{
		*songIndex = 0;
		// enable Timer intrpts
		NVIC_EN0_R = NVIC_EN0_INT19;
		NVIC_EN0_R = NVIC_EN0_INT23;
	}
	*enableSong = 0; // used in MusicPlayOrPause
	Delay(5000000);
}

void Change_Instruments(const unsigned short* instrumentsPtr)
{
	static unsigned long j = 0;

	Delay(10000);
	if(GPIO_PORTB_DATA_R & 0x40)
	{
		j = (j+1)%6;
		instrumentPtr = instrArray[j];
	}
	Delay(5000000);
}
```

```
static void Delay(unsigned long count)
{
	while(count)
	{
		count--;
	}
}

void PortC_Init(void)
{
	volatile unsigned long dummyInstr;
	SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOC; // activate PortC

	dummyInstr = SYSCTL_RCGC2_R;
	dummyInstr = SYSCTL_RCGC2_R;

	// used for the three buttons for play,stop,rewind
	GPIO_PORTC_DIR_R &= ~0xE0; // PC5,6,7 are inputs
	GPIO_PORTC_DEN_R |= 0xE0;  // enable digital inputs
	GPIO_PORTC_AFSEL_R &= ~0xE0; // disable alt function
}

void PortF_Init(void)
{
	volatile unsigned long dummyInstr;
	SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF; // activate PortC

	dummyInstr = SYSCTL_RCGC2_R;
	dummyInstr = SYSCTL_RCGC2_R;
	dummyInstr = SYSCTL_RCGC2_R;
	dummyInstr = SYSCTL_RCGC2_R;

	// used for the three buttons for play,stop,rewind
	GPIO_PORTF_DIR_R &= ~0xFF; // PC5,6,7 are inputs
	GPIO_PORTF_DEN_R |= 0xFF;  // enable digital inputs
	GPIO_PORTF_AFSEL_R &= ~0xFF; // disable alt function
}



// used for the three buttons for play,stop,rewind
void PortB_Init(void)
{
	volatile unsigned long dummyInstr;
	SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB; // activate PortB

	dummyInstr = SYSCTL_RCGC2_R;
	dummyInstr = SYSCTL_RCGC2_R;
	dummyInstr = SYSCTL_RCGC2_R;

	GPIO_PORTB_DIR_R &= ~0x7C; // PB2-6 inputs
	GPIO_PORTB_DEN_R |= 0x7C;  // enable digital inputs
	GPIO_PORTB_AFSEL_R &= ~0x7C; // disable alt function

	GPIO_PORTB_DATA_R = 0;
```

```
	GPIO_PORTB_IS_R &= ~0x7C;    // makes PB4-6 level-triggered interrupts
	GPIO_PORTB_IBE_R &= ~0x7C;   //sets it so it looks at GPIO_IEV
	GPIO_PORTB_ICR_R = 0x7C;     // clear flag4-6, do this every ISR call
	GPIO_PORTB_IEV_R |= 0x7C;    // interrupt triggers on HIGH level
	GPIO_PORTB_IM_R |= 0x7C;     // arm interrupt
	NVIC_PRI0_R = NVIC_PRI0_R&0xFFFF1FFF; // PL bits 13-15  pri = 0;
	NVIC_EN0_R = NVIC_EN0_INT1; // enables intrpts in PB, its a friendly operation
// NVIC_DIS0_R disables interpts for that particular port letter

}

void PortG_Init(void)
{
	volatile unsigned long dummyInstr;
	SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOG; // activate PortG

	dummyInstr = SYSCTL_RCGC2_R;
	dummyInstr = SYSCTL_RCGC2_R;

	GPIO_PORTG_DIR_R &= ~0x04; // PG2 Output
	GPIO_PORTG_DEN_R |= 0x04;  // enable digital IO
	GPIO_PORTG_AFSEL_R &= ~0x04; // disable alt function
}

void GPIOPortB_Handler(void)
{
	//unsigned long iBits;
	//iBits = StartCritical();
	unsigned long* songEnableDisablePtr;
	DisableInterrupts();

	if(GPIO_PORTB_DATA_R & 0x04) // PB2
	{ // this pauses the song
		songEnableDisablePtr = MusicPlayOrPause(&songNote);
		GPIO_PORTB_ICR_R = 0x04; // acknowledge interrupt
	}
	else if(GPIO_PORTB_DATA_R & 0x08) // PB3
	{// turn down the volume
		DecreaseVolume(&volume);
		GPIO_PORTB_ICR_R = 0x08; // acknowledge interrupt
	}
	else if(GPIO_PORTB_DATA_R & 0x10) // PB4
	{ // turn up the volume
		IncreaseVolume(&volume);
		GPIO_PORTB_ICR_R = 0x10; // acknowledge interrupt
	}
	else if(GPIO_PORTB_DATA_R & 0x20) // PB5
	{ // start song from beginning
		Rewind(&songNote,songEnableDisablePtr);
		GPIO_PORTB_ICR_R = 0x20; // acknowledge interrupt
	}
	else if(GPIO_PORTB_DATA_R & 0x40) // PB6
	{ // change instrument
		Change_Instruments(instrumentPtr);
		GPIO_PORTB_ICR_R = 0x40; // acknowledge interrupt
	}
```

```
      //EndCritical(iBits);
      GPIO_PORTB_ICR_R = 0x7C; // clear all interrupt ack bits again
      EnableInterrupts();
}
```

Periodic Timer

```
// PeriodicTimer0AInts.c
// Runs on LM3S1968
// Use Timer0A in periodic mode to request interrupts at a particular
// period.
// Daniel Valvano
// September 14, 2011

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvanto, copyright (c) 2011
  Program 7.5, example 7.6

 Copyright 2011 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/
 */

// oscilloscope connected to PC5 for period measurement
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "timer0A.h"
#include "lm3s1968.h"
#include "Music.h"
#include "DAC.h"
#include "Systick.h"
#include "Switches.h"

extern const unsigned short* instrArray[6];
extern Note* SongAlbum[5];
extern const unsigned short* instrumentPtr;
//^^^ the current output waveform sent to DAC

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
void UserTask(void);
int wait0 = -1;
//Instruments instrument;
//Instruments* instrumentStruct = &instrument;

extern NotePtr songPtr; // this is no longer a pointer to a pointer

int main(void)
```

```
{
      volatile unsigned long delay;
  instrumentPtr = instrArray[SINE];
      songPtr = SongAlbum[0]; // this is a pointer to a pointer
      DisableInterrupts();

  SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                 SYSCTL_XTAL_8MHZ); // 50 MHz Clock

  Timer0A_Init(&UserTask,200);  // timer0A->DAC output
#ifdef CHORD
      Timer0B_Init(&UserTask,800);  // timer0B->DAC output
      Timer1A_Init(&UserTask,1200); // timer1A->DAC output
#else
      Timer2A_Init(&UserTask,600);  // timer2A->changes note
#endif
      //PortF_Init();
      PortB_Init(); // used for switch interrupts
      PortG_Init();
      SysTick_Init(); // used for timing of debugging profile
  DAC_Init();

      EnableInterrupts();

      while(1)
      {
/*
//            if(GPIO_PORTF_DATA_R &0x01)
//            {
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(1019/2);//noteArray[NOTE_Gb]/2);
                  // outputs at 703 should be 740
//            }
//            else if(GPIO_PORTF_DATA_R &0x02)
//            {
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(1020/2);//noteArray[NOTE_F]/2);
                  // outputs at 703 should be 698
//            }
//            else if(GPIO_PORTF_DATA_R &0x04)
//            {
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(1075/2);//noteArray[NOTE_E]/2);
                  // outputs at 624 should be 659
                  // it plays an Eb, Eb =
//            }
//            else if(GPIO_PORTF_DATA_R &0x08)
//            {
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(1260/2);//noteArray[NOTE_D]/2);
                  // outputs at 546 should be 587
                  // it plays an Eb, Eb =
//            }
//            else if(GPIO_PORTF_DATA_R &0x10)
//            {// this one just dont work
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(1412/2);//noteArray[NOTE_C]/2);
                  // outputs at 547 should be 523
```

```
//                  // it plays an Eb, Eb =
//            }
//            else if(GPIO_PORTF_DATA_R &0x20)
//            {
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(1386/2);//noteArray[NOTE_B]/2);
//                  // outputs at 468 should be 659
//                  // it plays an Bb, Bb = 1582
//            }
//            else if(GPIO_PORTF_DATA_R &0x40)
//            {
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(noteArray[NOTE_A]/2);
//                  // outputs at 468 should be 440
//                  // it plays an Bb, Bb = 1776
//            }
//            else if(GPIO_PORTF_DATA_R &0x80)
//            {
//                  DAC_Out(SinWave1[i]);
//              SysTick_Wait(1730/2);//noteArray[NOTE_G]/2);
//                  // outputs at 391 should be 392
//                  // it plays an Eb, Eb =
//            }




              //GPIO_PORTG_DATA_R &= ~0x04;
    //GPIO_PORTC4 ^= 0x10;
//DAC_Out(SinWave[i]);
//SysTick_Wait(noteArray[NOTE_G]);
              //SysTick_Wait(2000);
              //DAC_Out(i*10);


//            i = (i+1)%64;
//            i = (i+1)&0x7F;

//
//            if(i == 63)
//            {
//                  j++;
//                  if (j%60 == 0)
//                  {
//                        GPIO_PORTG_DATA_R ^= 0x04;
//                  }
//            }
*/
  }
}

// This is never used and came default in the starter code
void UserTask(void)
{
  //GPIO_PORTC5 ^= 0x20;
       GPIO_PORTG_DATA_R ^= 0x04;
}
```

Music

```
// Music.c

#include "Music.h"

/* these are the number of clock cycles to wait before the
// next interrupt its calculated from 50e6/64/DesiredFrequency
//#define NOTE_Ab1      940 // 831
//#define NOTE_G1       996 // 784
//#define NOTE_Gb1     1056 // 740
//#define NOTE_F1      1119 // 698
//#define NOTE_E1      1186 // 659
//#define NOTE_Eb1     1256 // 622
//#define NOTE_D1      1331 // 587
//#define NOTE_Db1     1410 // 554
//#define NOTE_C1      1494 // 523
//#define NOTE_B1      1582 // 494
//#define NOTE_Bb0     1677 // 466
//#define NOTE_A0      1776 // 440
//#define NOTE_Ab0     1883 // 415
//#define NOTE_G0      1993 // 392
//#define NOTE_Gb0     2112 // 370
//#define NOTE_F0      2239 // 349
//#define NOTE_E0      2367 // 330
//#define NOTE_Eb0     2512 // 311
//#define NOTE_D0      2657 // 294
//#define NOTE_Db0     2820 // 277
//#define NOTE_C0      2982 // 262
//#define NOTE_B0      3163 // 247

// playing this sequence at difference interrupt periods
// creates different tones */


const unsigned short SinWave[64] = {
                           2048, 2249, 2448, 2643, 2832, 3013, 3186,  3347,
             3496, 3631, 3751, 3854, 3940, 4008, 4057,  4086,
             4096, 4086, 4057, 4008, 3940, 3854, 3751,  3631,
             3496, 3347, 3186, 3013, 2832, 2643, 2448,  2249,
             2048, 1847, 1648, 1453, 1264, 1083,  910,   749,
              600,  465,  345,  242,  156,   88,   39,    10,
                0,   10,   39,   88,  156,  242,  345,   465,
              600,  749,  910, 1083,  1264, 1453, 1648, 1847};


const unsigned short SinWave1[128] = {
2048,
2148,
2249,
2349,
2448,
2546,
2643,
2738,
2832,
2924,
```

3013,
3101,
3186,
3268,
3347,
3423,
3496,
3565,
3631,
3693,
3751,
3805,
3854,
3899,
3940,
3976,
4008,
4035,
4057,
4074,
4086,
4094,
4094,
4094,
4086,
4074,
4057,
4035,
4008,
3976,
3940,
3899,
3854,
3805,
3751,
3693,
3631,
3565,
3496,
3423,
3347,
3268,
3186,
3101,
3013,
2924,
2832,
2738,
2643,
2546,
2448,
2349,
2249,
2148,
2048,
1948,
1847,
1747,

1648,
1550,
1453,
1358,
1264,
1172,
1083,
995,
910,
828,
749,
673,
600,
531,
465,
403,
345,
291,
242,
197,
156,
120,
88,
61,
39,
22,
10,
2,
0,
2,
10,
22,
39,
61,
88,
120,
156,
197,
242,
291,
345,
403,
465,
531,
600,
673,
749,
828,
910,
995,
1083,
1172,
1264,
1358,
1453,
1550,
1648,
1747,

```
1847,
1948,
};



const unsigned short Horn[128] = {
//  1063,1082,1119,1275,1678,1748,1275,755,661,661,703,
//  731,769,845,1039,1134,1209,1332,1465,1545,1427,1588,
//  1370,1086,708,519,448,490,566,684,802,992

            2490,
        2502,
        2514,
        2523,
        2535,
        2556,
        2579,
        2600,
        2621,
        2713,
        2804,
        2896,
        2987,
        3224,
        3460,
        3694,
        3931,
        3973,
        4013,
        4055,
        4095,
        3819,
        3542,
        3263,
        2987,
        2682,
        2378,
        2073,
        1769,
        1715,
        1659,
        1605,
        1549,
        1549,
        1549,
        1549,
        1549,
        1574,
        1598,
        1623,
        1647,
        1663,
        1680,
        1696,
        1712,
        1736,
        1757,
```

1780,
1802,
1846,
1891,
1935,
1980,
2094,
2207,
2322,
2434,
2490,
2546,
2600,
2657,
2701,
2746,
2788,
2832,
2905,
2978,
3048,
3120,
3198,
3277,
3355,
3432,
3479,
3526,
3573,
3619,
3551,
3481,
3413,
3343,
3437,
3533,
3626,
3720,
3594,
3465,
3338,
3209,
3043,
2877,
2710,
2544,
2324,
2101,
1881,
1659,
1549,
1438,
1326,
1216,
1174,
1134,
1092,
1050,

```
        1075,
        1099,
        1124,
        1148,
        1192,
        1237,
        1281,
        1326,
        1396,
        1464,
        1534,
        1602,
        1673,
        1741,
        1811,
        1879,
        1991,
        2101,
        2214,
        2324,
        2366,
        2408,
        2448,

};



const unsigned short Flute[128] = {
//  1007,1252,1374,1548,1698,1797,1825,1797,1675,1562,1383,
//  1219,1092,1007,913,890,833,847,810,777,744,674,
//  598,551,509,476,495,533,589,659,758,876

            2260,
        2396,
        2536,
        2672,
        2809,
        2879,
        2946,
        3016,
        3083,
        3182,
        3278,
        3377,
        3473,
        3559,
        3642,
        3727,
        3810,
        3866,
        3922,
        3976,
        4032,
        4048,
        4064,
        4079,
        4095,
```

4079,
4064,
4048,
4032,
3965,
3895,
3828,
3758,
3696,
3633,
3568,
3505,
3404,
3305,
3204,
3103,
3011,
2919,
2827,
2735,
2663,
2594,
2522,
2450,
2403,
2356,
2307,
2260,
2208,
2154,
2102,
2049,
2035,
2024,
2010,
1997,
1966,
1934,
1901,
1869,
1878,
1885,
1894,
1901,
1880,
1860,
1838,
1818,
1800,
1782,
1761,
1743,
1726,
1708,
1687,
1669,
1631,
1591,

```
        1553,
        1512,
        1470,
        1427,
        1384,
        1342,
        1315,
        1290,
        1263,
        1236,
        1214,
        1189,
        1167,
        1142,
        1124,
        1106,
        1086,
        1068,
        1079,
        1091,
        1099,
        1111,
        1133,
        1153,
        1176,
        1196,
        1227,
        1259,
        1290,
        1322,
        1362,
        1400,
        1441,
        1479,
        1535,
        1591,
        1645,
        1701,
        1768,
        1833,
        1901,
        1966,
        2040,
        2114,
        2185,


};


const unsigned short Trumpet[128] = {
//  1007,1088,1156,1194,1067,789,303,99,789,1510,1476,
//  1173,1067,1037,1084,1062,1011,1015,1045,1062,1011,1011,
//  1058,1113,1084,1075,1079,1105,1088,1049,1015,1045
        2731,
        2785,
        2842,
        2896,
```

2951,
2997,
3043,
3089,
3135,
3111,
3086,
3059,
3035,
2999,
2964,
2929,
2894,
2706,
2517,
2330,
2140,
1812,
1481,
1153,
 822,
 683,
 545,
 407,
 268,
 738,
1204,
1673,
2140,
2628,
3119,
3607,
4095,
4073,
4049,
4027,
4003,
3797,
3593,
3387,
3181,
3111,
3037,
2967,
2894,
2875,
2853,
2834,
2812,
2845,
2877,
2907,
2940,
2926,
2910,
2896,
2880,
2845,

2812,
2777,
2742,
2744,
2747,
2750,
2753,
2774,
2793,
2815,
2834,
2845,
2858,
2869,
2880,
2845,
2812,
2777,
2742,
2742,
2742,
2742,
2742,
2774,
2807,
2837,
2869,
2907,
2945,
2980,
3018,
2999,
2980,
2959,
2940,
2934,
2929,
2921,
2915,
2918,
2921,
2923,
2926,
2945,
2961,
2980,
2997,
2986,
2975,
2961,
2951,
2923,
2899,
2872,
2845,
2823,
2799,
2777,

```
        2753,
        2774,
        2793,
        2815,
        2834,
        2810,
        2782,
        2758,
};


const unsigned short Bassoon[128]={
//1068, 1169, 1175, 1161, 1130, 1113, 1102, 1076, 1032, 985, 963, 987, 1082, 1343, 1737,
1863,
//1575, 1031, 538, 309, 330, 472, 626, 807, 1038, 1270, 1420, 1461, 1375, 1201, 1005,
819, 658,
//532, 496, 594, 804, 1055, 1248, 1323, 1233, 1049, 895, 826, 826, 850, 862, 861, 899,
961, 1006,
//1023, 1046, 1092, 1177, 1224, 1186, 1133, 1098, 1102, 1109, 1076, 1027, 1003};
//
        2348,
        2459,
        2570,
        2576,
        2583,
        2567,
        2552,
        2518,
        2484,
        2465,
        2446,
        2434,
        2422,
        2394,
        2365,
        2317,
        2268,
        2217,
        2165,
        2141,
        2117,
        2143,
        2169,
        2274,
        2378,
        2665,
        2952,
        3385,
        3818,
        3957,
        4095,
        3778,
        3462,
        2864,
        2266,
        1724,
        1183,
         931,
```

679,
702,
725,
881,
1037,
1207,
1376,
1575,
1774,
2028,
2282,
2537,
2792,
2956,
3121,
3166,
3211,
3117,
3022,
2831,
2640,
2424,
2209,
2005,
1800,
1623,
1446,
1308,
1169,
1130,
1090,
1198,
1306,
1536,
1767,
2043,
2319,
2531,
2743,
2826,
2908,
2809,
2710,
2508,
2306,
2137,
1967,
1891,
1816,
1816,
1816,
1842,
1868,
1882,
1895,
1894,
1893,
1934,

```
        1976,
        2044,
        2112,
        2162,
        2211,
        2230,
        2249,
        2274,
        2299,
        2350,
        2400,
        2494,
        2587,
        2639,
        2690,
        2649,
        2607,
        2549,
        2490,
        2452,
        2413,
        2418,
        2422,
        2430,
        2438,
        2401,
        2365,
        2311,
        2257,
        2231,
        2205,
        2276
};


const unsigned short Oboe[128]={
//1024, 1024, 1014, 1008, 1022, 1065, 1093, 1006, 858, 711, 612, 596, 672, 806, 952,
1074, 1154, 1191,
//1202, 1216, 1236, 1255, 1272, 1302, 1318, 1299, 1238, 1140, 1022, 910, 827, 779, 758,
757, 782, 856,
//972, 1088, 1177, 1226, 1232, 1203, 1157, 1110, 1067, 1028, 993, 958, 929, 905, 892,
900, 940, 1022,
//1125, 1157, 1087, 965, 836, 783, 816, 895, 971, 1017};
             3182,
        3180,
        3176,
        3166,
        3150,
        3141,
        3132,
        3154,
        3175,
        3242,
        3309,
        3352,
        3396,
        3261,
```

3126,
2896,
2666,
2437,
2209,
2055,
1901,
1877,
1852,
1970,
2088,
2296,
2504,
2731,
2958,
3147,
3337,
3461,
3585,
3643,
3700,
3718,
3735,
3756,
3778,
3809,
3840,
3870,
3899,
3926,
3952,
3999,
4045,
4070,
4095,
4065,
4036,
3941,
3846,
3694,
3542,
3359,
3175,
3001,
2827,
2698,
2569,
2495,
2420,
2388,
2355,
2354,
2352,
2391,
2430,
2545,
2660,
2840,

```
        3020,
        3200,
        3380,
        3519,
        3657,
        3733,
        3809,
        3818,
        3828,
        3783,
        3738,
        3666,
        3595,
        3522,
        3449,
        3382,
        3315,
        3255,
        3194,
        3140,
        3085,
        3031,
        2976,
        2931,
        2886,
        2849,
        2812,
        2792,
        2771,
        2784,
        2796,
        2858,
        2921,
        3048,
        3175,
        3335,
        3495,
        3545,
        3595,
        3486,
        3377,
        3188,
        2998,
        2798,
        2597,
        2515,
        2433,
        2484,
        2535,
        2658,
        2781,
        2899,
        3017,
        3088,
        3160,
        3171,
};
```

```
/*
const unsigned short envelope[128] = {
10,
10,
10,
10,
10,
10,
10,
10,
10,
9,
9,
9,
9,
9,
9,
9,
9,
9,
7,
7,
7,
7,
7,
7,
7,
7,
6,
6,
6,
6,
6,
6,
6,
6,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
5,
4,
4,
4,
4,
4,
4,
4,
```

```
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
```

```
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
4,
};




const unsigned short duration[4] = {8000,16000,32000,64000};

//unsigned char anvelope[128] = {
//       26,
//    26,
//    27,
//    27,
//    28,
//    29,
//    30,
//    30,
//    31,
//    31,
//    32,
//    32,
//    32,
//    32,
//    32,
//    32,
//    32,
//    32,
//    32,
//    31,
//    31,
//    30,
//    30,
//    29,
//    29,
//    28,
//    28,
//    27,
//    26,
//    26,
//    25,
//    24,
//    24,
//    23,
```

```
//    22,
//    22,
//    21,
//    20,
//    20,
//    19,
//    18,
//    18,
//    17,
//    17,
//    16,
//    15,
//    15,
//    14,
//    14,
//    13,
//    13,
//    12,
//    12,
//    11,
//    11,
//    10,
//    10,
//    10,
//     9,
//     9,
//     8,
//     8,
//     8,
//     7,
//     7,
//     7,
//     7,
//     6,
//     6,
//     6,
//     5,
//     5,
//     5,
//     5,
//     5,
//     4,
//     4,
//     4,
//     4,
//     4,
//     3,
//     3,
//     3,
//     3,
//     3,
//     3,
//     3,
//     2,
//     2,
//     2,
//     2,
//     2,
```

```
//     2,
//     2,
//     2,
//     2,
//     2,
//     2,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     1,
//     0,
//     0,
//     0,
//     0,
//     0,
//     0,
//  ,   0,
    65,
    66,
    67,
    69,
    70,
    72,
    74,
    76,
    77,
    78,
    79,
    80,
    80,
    81,
    81,
    81,
    80,
    80,
    79,
    78,
    77,
    76,
```

```
75,
74,
72,
71,
69,
68,
66,
64,
63,
61,
59,
58,
56,
54,
53,
51,
49,
48,
46,
44,
43,
41,
40,
38,
37,
36,
34,
33,
32,
31,
29,
28,
27,
26,
25,
24,
23,
22,
21,
20,
19,
19,
18,
17,
16,
16,
15,
14,
14,
13,
12,
12,
11,
11,
10,
10,
 9,
 9,
```

```
      9,
      8,
      8,
      8,
      7,
      7,
      7,
      6,
      6,
      6,
      5,
      5,
      5,
      5,
      4,
      4,
      4,
      4,
      4,
      3,
      3,
      3,
      3,
      3,
      3,
      3,
      2,
      2,
      2,
      2,
      2,
      2,
      2,
      2,
      2,
      2,
      2,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,
      1,

         };



// t_val = 1.5*exp(-t/10)+.25*t.*exp(-t/10);
// newer one -> > plot(t,1.3*exp(-t/5)+.25*t.*exp(-t/17));
//      15.8347,
//   16.3746,
```

```
//   16.6684,
//   16.7580,
//   16.6796,
//   16.4643,
//   16.1390,
//   15.7265,
//   15.2464,
//   14.7152,
//   14.1470,
//   13.5537,
//   12.9453,
//   12.3298,
//   11.7143,
//   11.1043,
//   10.5043,
//    9.9179,
//    9.3480,
//    8.7968,
//    8.2658,
//    7.7562,
//    7.2688,
//    6.8038,
//    6.3616,
//    5.9419,
//    5.5445,
//    5.1689,
//    4.8145,
//    4.4808,
//    4.1671,
//    3.8724,*/
```

```
//void Instrument_Init(Instruments* instrument) // pted to the different instrmnts
//{
//     instrument->sinePtr = &SinWave1[0];
//     instrument->trumpetPtr = &Trumpet[0];
//     instrument->hornPtr = &Horn[0];
//     instrument->flutePtr = &Flute[0];
//     instrument->bassoonPtr = &Bassoon[0];
//     instrument->oboePtr = &Oboe[0];
```

```
//}
const unsigned short* instrArray[6] = {&SinWave1[0],&Horn[0],&Flute[0],

                                       &Trumpet[0],&Bassoon[0],&Oboe[0]};

const unsigned short* instrumentPtr;
// initial defaults
int volume = 2;
int songNumber = 0;


// Rocky's "Gonna Fly Now"
Note Song0[] = {

       {C,EIGHTH+SIXTEENTH},{REST,THIRTYSECOND},

       {C,EIGHTH},{REST,THIRTYSECOND},
       {C,SIXTEENTH},{REST,THIRTYSECOND},
       {C,SIXTEENTH},{REST,THIRTYSECOND},

       {C,EIGHTH},{REST,THIRTYSECOND}, // 10
       {C,SIXTEENTH},{REST,THIRTYSECOND},
       {C,SIXTEENTH},{REST,THIRTYSECOND},

       {E,EIGHTH},{REST,THIRTYSECOND},
       {C,SIXTEENTH},{REST,THIRTYSECOND},
       {C,SIXTEENTH},{REST,THIRTYSECOND}, // 20

       {C,QUARTER},{REST,THIRTYSECOND},

       {E,EIGHTH},{REST,THIRTYSECOND},

       {E,EIGHTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,THIRTYSECOND}, // 30

       {E,EIGHTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,THIRTYSECOND},

       {GHigh,EIGHTH},{REST,SIXTEENTH},
       {E,SIXTEENTH},{REST,THIRTYSECOND}, // 40
       {E,SIXTEENTH},{REST,THIRTYSECOND},

       {E,QUARTER},{REST,SIXTEENTH},

       {E,EIGHTH+SIXTEENTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,THIRTYSECOND}, // 50
       {E,EIGHTH+SIXTEENTH},{REST,THIRTYSECOND},

       {E,SIXTEENTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,SIXTEENTH},
       {E,SIXTEENTH},{REST,THIRTYSECOND},
       {E,QUARTER+EIGHTH},{REST,QUARTER}, // 60 at the end of this line

       {E,EIGHTH+SIXTEENTH},{REST,THIRTYSECOND},
       {E,SIXTEENTH},{REST,THIRTYSECOND},
```

```
        {E,SIXTEENTH},{REST,THIRTYSECOND},
        {E,EIGHTH+SIXTEENTH},{REST,THIRTYSECOND},

        {E,SIXTEENTH},{REST,THIRTYSECOND}, // 70
        {E,HALF},{REST,QUARTER},

/*      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,EIGHTH},{REST,SIXTEENTH},
//      {D,QUARTER},//{REST,THIRTYSECOND},
//
//  {D,EIGHTH},{REST,THIRTYSECOND},

//// this folowing part works great
//      {REST,QUARTER},
//      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,SIXTEENTH},{REST,SIXTEENTH},
//
//      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,SIXTEENTH},{REST,SIXTEENTH},
//
//      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,SIXTEENTH},{REST,SIXTEENTH},
//
//      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,HALF},{REST,SIXTEENTH},
////    ////////////////////////////
//      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,SIXTEENTH},{REST,SIXTEENTH},
//
//      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,SIXTEENTH},{REST,SIXTEENTH},
//
//      {D,SIXTEENTH},{REST,THIRTYSECOND},
//      {D,HALF},{REST,SIXTEENTH},
*/

        ////////////////////
        {E,THIRTYSECOND},{REST,SIXTEENTH},
        {GHigh,EIGHTH},{REST,SIXTEENTH+THIRTYSECOND},
        {AHigh,WHOLE},{REST,THREE_EIGHTHS},

        {AHigh,THIRTYSECOND},{REST,SIXTEENTH}, // 80
        {BHigh,EIGHTH},{REST,SIXTEENTH+THIRTYSECOND},
        {E,WHOLE},{REST,THREE_EIGHTHS},

        {E,THIRTYSECOND},{REST,SIXTEENTH},
        {GHigh,EIGHTH},{REST,SIXTEENTH+THIRTYSECOND},
        {AHigh,WHOLE},{REST,THREE_EIGHTHS}, // 90

        {CHigh,THIRTYSECOND},{REST,SIXTEENTH},
        {BHigh,EIGHTH},{REST,SIXTEENTH+THIRTYSECOND},//{REST,EIGHTH},
        {E,WHOLE+QUARTER},{REST,HALF+THREE_EIGHTHS},

        {D,SIXTEENTH},{REST,THIRTYSECOND},
        {C,SIXTEENTH},{REST,THIRTYSECOND}, // 100
        {D,SIXTEENTH+EIGHTH},{REST,THIRTYSECOND},
        {C,SIXTEENTH},{REST,THIRTYSECOND},
        {D,SIXTEENTH},{REST,THIRTYSECOND},
```

```
        {E,SIXTEENTH+EIGHTH},{REST,QUARTER+EIGHTH},



        {CHigh,SIXTEENTH},{REST,THIRTYSECOND}, // 110
        {CHigh,SIXTEENTH},{REST,THIRTYSECOND},
        {BHigh,SIXTEENTH+EIGHTH},{REST,THIRTYSECOND},

        {BHigh,SIXTEENTH},{REST,THIRTYSECOND},
        {AHigh,SIXTEENTH},{REST,SIXTEENTH},
        {AHigh,SIXTEENTH},{REST,THIRTYSECOND}, // 120
        {GHigh,HALF},{REST,THIRTYSECOND},

        {CHigh,SIXTEENTH},{REST,THIRTYSECOND+EIGHTH},
        {BHigh,WHOLE+QUARTER},{REST,HALF},

        {A,EIGHTH},{REST,SIXTEENTH},
        {A,EIGHTH},{REST,SIXTEENTH}, // 130
        {C,QUARTER+EIGHTH},{REST,SIXTEENTH+THIRTYSECOND},

        {B,EIGHTH},{REST,SIXTEENTH},
        {B,EIGHTH},{REST,SIXTEENTH},
        {D,QUARTER+EIGHTH},{REST,SIXTEENTH+THIRTYSECOND},

        {C,EIGHTH},{REST,SIXTEENTH}, // 140
        {C,EIGHTH},{REST,SIXTEENTH},
        {E,WHOLE+QUARTER},{REST,QUARTER+EIGHTH},

        {C,EIGHTH},{REST,SIXTEENTH},
        {B,THREE_FOURTHS},{REST,SIXTEENTH},
        {B,EIGHTH},{REST,SIXTEENTH}, // 150
        {B,THREE_FOURTHS},{REST,SIXTEENTH},
        {B,EIGHTH},{REST,SIXTEENTH},
        {Eb,WHOLE+EIGHTH},{REST,HALF},

        {Ab,SIXTEENTH},{REST,THIRTYSECOND},
        {Bb,QUARTER},{REST,THIRTYSECOND}, // 160
        {C,SIXTEENTH},{REST,WHOLE+WHOLE},

        {Bb,HALF+EIGHTH},{REST,EIGHTH},
        {Bb,HALF+EIGHTH},{REST,EIGHTH+QUARTER},

        {AbHigh,SIXTEENTH},{REST,THIRTYSECOND},
        {BbHigh,QUARTER},{REST,THIRTYSECOND},  // 170
        {CHigh,SIXTEENTH},{REST,WHOLE},

        {REST,QUARTER},{REST,QUARTER},
        {REST,QUARTER},{REST,QUARTER}, // 176

};



Note Song1[] = {
        {G,THIRTYSECOND},
        {Ab,THIRTYSECOND},
        {A,THIRTYSECOND},
```

```
        {Bb,THIRTYSECOND},
        {B,THIRTYSECOND},
        {C,THIRTYSECOND},
        {Db,THIRTYSECOND},
        {D,THIRTYSECOND},

        {Eb,SIXTEENTH},
        {D,SIXTEENTH},
        {Db,SIXTEENTH},
        {C,SIXTEENTH},
        {REST,QUARTER},
        {F,HALF},
        {REST,QUARTER},
        {Gb,HALF},
        {G,WHOLE},
        {B,QUARTER},
        {A,QUARTER},
        {C,QUARTER},
        {E,QUARTER},
        {D,QUARTER},
        {G,QUARTER}
};
// Flo-Rida "Right Round"
Note Song2[] = { {REST,QUARTER},
//{A,QUARTER},{REST,THIRTYSECOND},{A,QUARTER},{REST,THIRTYSECOND},{A,QUARTER},{REST,THIRT
YSECOND},
//{C,QUARTER},{REST,THIRTYSECOND},{C,QUARTER},{REST,THIRTYSECOND},{G,QUARTER},{REST,THIRT
YSECOND},
//{D,QUARTER},{REST,THIRTYSECOND},{C,EIGHTH},{REST,THIRTYSECOND},{A,EIGHTH},{REST,THIRTYS
ECOND},
//{A,EIGHTH},{REST,THIRTYSECOND},{A,EIGHTH},{REST,THIRTYSECOND},{D,QUARTER},{REST,THIRTYS
ECOND},
//{C,QUARTER},{REST,THIRTYSECOND},{REST,HALF},{REST,THIRTYSECOND},{C,QUARTER},

//{A,QUARTER},{REST,THIRTYSECOND},{A,QUARTER},{REST,THIRTYSECOND},{A,QUARTER},{REST,THIRT
YSECOND},
//{C,QUARTER},{REST,THIRTYSECOND},{C,QUARTER},{REST,THIRTYSECOND},{G,QUARTER},{REST,THIRT
YSECOND},
//{D,QUARTER},{REST,THIRTYSECOND},{C,EIGHTH},{REST,THIRTYSECOND},{A,EIGHTH},{REST,THIRTYS
ECOND},
//{A,EIGHTH},{REST,THIRTYSECOND},{A,EIGHTH},{REST,THIRTYSECOND},{D,QUARTER},{REST,THIRTYS
ECOND},
//{C,QUARTER},{REST,THIRTYSECOND},

//{G,EIGHTH},{REST,THIRTYSECOND},{G,EIGHTH},{REST,THIRTYSECOND},{A,QUARTER},{REST,THIRTYS
ECOND},
//{D,QUARTER},{REST,THIRTYSECOND},{C,QUARTER},{REST,QUARTER},


//{Ab,THIRTYSECOND},
//{A,THIRTYSECOND},
//{Bb,THIRTYSECOND},
//{B,THIRTYSECOND},
//{C,THIRTYSECOND},
//{Db,THIRTYSECOND},
//{D,THIRTYSECOND},

{REST,QUARTER},
```

```
//{Eb,SIXTEENTH},
//{D,SIXTEENTH},
//{Db,SIXTEENTH},
//{C,SIXTEENTH},
////{REST,QUARTER},
//{F,HALF},
//{REST,QUARTER},
//{Gb,HALF},
//{G,WHOLE},
//{B,QUARTER},
//{A,QUARTER},
//{C,QUARTER},
//{E,QUARTER},
//{D,QUARTER},
//{G,QUARTER}
};




Note Song3[] = {{REST,QUARTER}};
Note Song4[] = {{REST,QUARTER}};

Note* SongAlbum[5] = {Song0,Song1,Song2,Song3,Song4};
NotePtr songPtr;

DAC
// DAC.c

#include "DAC.h"
#include "lm3s1968.h"

// from the book, Sec. 7-5 pg 371
void DAC_Init(void)
{
       volatile unsigned long delay;
       SYSCTL_RCGC1_R |= SYSCTL_RCGC1_SSI0;  // activate SSI0
       SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate PortA

       delay = SYSCTL_RCGC2_R;  // allow time to finish activating
       delay = SYSCTL_RCGC2_R;  // allow time to finish activating

       GPIO_PORTA_AFSEL_R |= 0x2C; // enable alt func on PA2,3,5
       GPIO_PORTA_DEN_R |= 0x2C; // enable digital IO on PA2,3,5

       SSI0_CR1_R &= ~SSI_CR1_SSE; // disable SSI
       SSI0_CR1_R &= ~SSI_CR1_MS;  // master mode

       SSI0_CPSR_R = (SSI0_CPSR_R & ~SSI_CPSR_CPSDVSR_M) + 2; // 3MHz

       SSI0_CR0_R &= ~(SSI_CR0_SCR_M |   // SCR = 0;
                                                                  SSI_CR0_SPH   |   // SPH =
0;
```

```
                                                        SSI_CR0_SPO);     // SPO =
0;

        SSI0_CR0_R |= SSI_CR0_SPH;        // SPH = 1;

        SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_FRF_M)+SSI_CR0_FRF_MOTO; // Freescale
        SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_DSS_M)+SSI_CR0_DSS_16; // 16-bit data
        SSI0_CR1_R |= SSI_CR1_SSE; // enable SSI

        return;
}

// from the book, Sec. 7-5 pg 372
// send the 16-bit data to the SSI, return a reply
unsigned short DAC_Out(unsigned short data)
{
        while((SSI0_SR_R&SSI_SR_TNF) == 0) {}; // wait until room in FIFO
        SSI0_DR_R = (0xC000)+(data&0x0FFF);  // data out
        // ^^^ sets the speed and register select bits, look at pg 11 on tlv5618 datasheet
for more info why
        return 0;
}
```