EEL 4924 Electrical Engineering Design (Senior Design)

Digital Strobe Tuner

Final Design Report

Team Members:
Name: David Barnette
Email: dtbarn@ufl.edu

Name: Jamie Lin
Email: jlquads@ufl.edu

Table of Contents

1.0 Project Abstract

The purpose of our project is to create a digitally implemented strobe tuner for onstage use. Our device provides an accurate and accessible tuning functionality ideal for live performances. The device is separated into 2 modules, the onstage display unit and the floor unit. The floor unit has a true bypass mute function in order for silent tuning. A digital signal processing unit, in the floor unit, employs a zero crossing algorithm in order to accurately determine musical frequencies. A 4-wire resistive LCD touch screen, on the display unit, provides a graphical user interface easily understood by musicians. Both units communicate through a serial peripheral interface.

2.0 Introduction

Analog strobe (stroboscopic) tuners have been around since 1936. They typically take a disk with lines on it and spin it mechanically at a very specific frequency. The audio input is then flashed behind the disk using a light source and any deviation from the desired pitch can be seen by the rotating motion of the lines. When the input is at the desired pitch the lines do not move, and as you move away from the pitch the lines move left or right depending on whether or not you or over or under the pitch. Strobe tuners are the most accurate tuners but they are significantly more expensive than other available tuners.

This project's goal is to create an accurate digital strobe tuner that will be smaller, lighter, and cheaper than analog strobe tuners. There are already electronic strobe tuners available on the market, but ours will be unique in its functionality and shape. We will be using a stomp box guitar pedal interface with a satellite display for tuning information. This will make our tuner perfect for live performance settings when tuning on stage is required. The touch screen interface will allow for the user to intuitively interact with the different functions of the tuner.

Figure 1 Design Model

2.1 Features

Stroboscope Tuner

- Accurate real time frequency differentiation
- Simple and Informative Tuning interface
- On stage hidden presence, with foot pedal muter
- Satellite display allows for anywhere placement

2.2 Design Overview

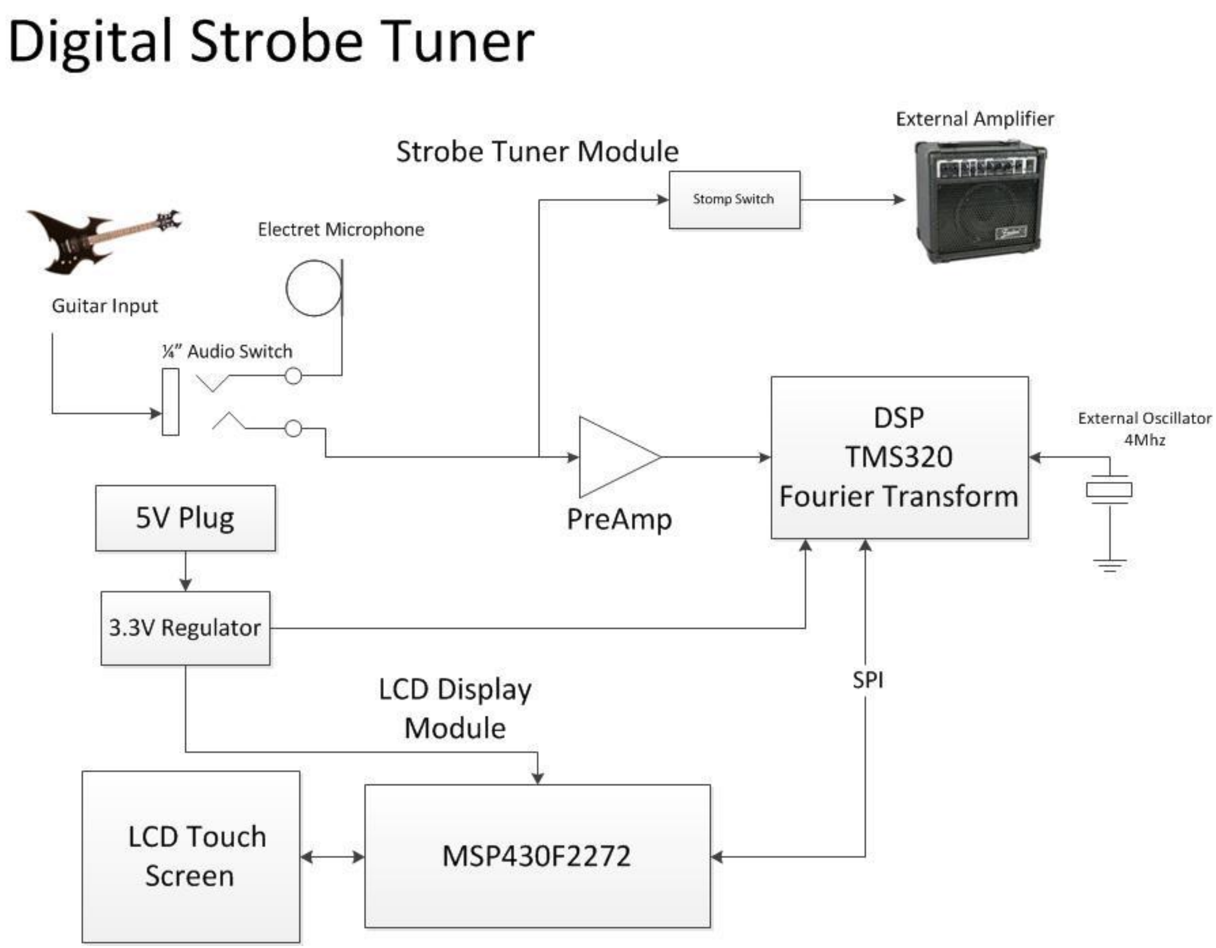

Figure 2 Design Overview

3.0 Floor & Display Units

3.1 Floor Unit

The floor unit houses the audio processing, audio line input/output, stomp switch, and power supply. A power supply of +5V is needed to power the floor unit and display unit however; the power comes from a DC plug on the floor unit. As stated previously the PS-2 cable provides power to the display unit. A +3.3V voltage regulator (LM3940) was used for low voltage powering off of the +5V.

3.1.1 Pre-Amp Design

The single supply preamplifier design allowed for the input signal to be raised from the average guitar/microphone output of ±100 mV to 0-+3.3V. This amplifier insures a clean crossing value for the DSP without worry of overcharging voltages.

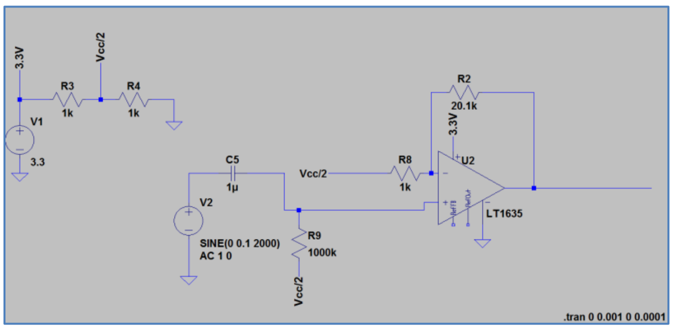

Figure 3 Pre-Amplifier design

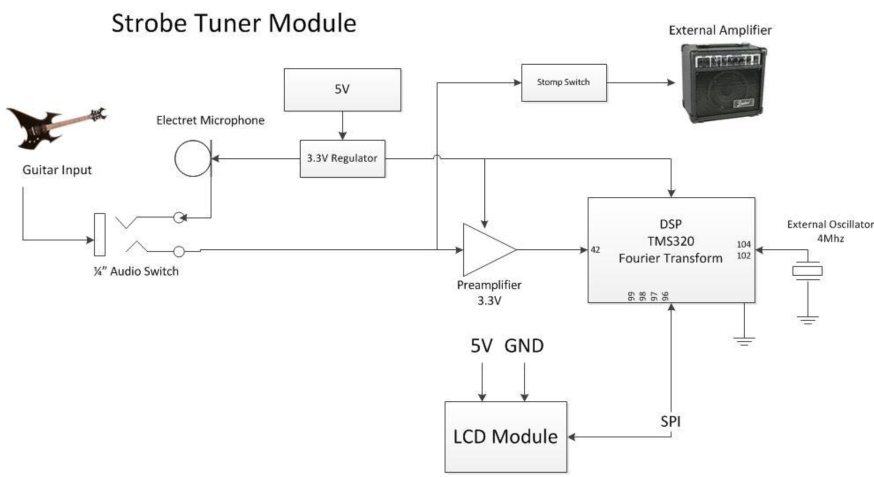

Figure 4 Floor unit hardware overview

3.1.2 Tuner Module

The tuner module, in the floor unit, processes all the incoming audio data. The input runs through a preamp to increase the strength of the signal. The preamp is then connected to the ADC on the DSP. The ADC in the chosen DSP is 12-bits so the output ranges from 0 to 4095. During the sampling section in the software a timer interrupt runs at 30khz and records 3000 samples. As it records these samples in records "zero" crossings from negative to positive. The zero is determined after each completed sampling section by:

$$Zero = (1.2)\frac{max \times min}{2} \quad (1)$$

This offset zero is designed to be far enough away from the DC value to reduce noise around the DC value. Also, after every zero crossing there is a debounce counter to make sure there isn't another zero crossing within 3 samples, which will weed out any high frequency noise (above 10khz, which does weed out some audible frequencies, but not ones frequently used for music). The parameters recorded to determine the frequency are the number of zero crossings, the indexes of the first and last zero crossing, and the sampling frequency and are used in:

$$Frequency = \frac{(zero\ crossing - 1)F_S}{last\ index - first\ index} \quad (2)$$

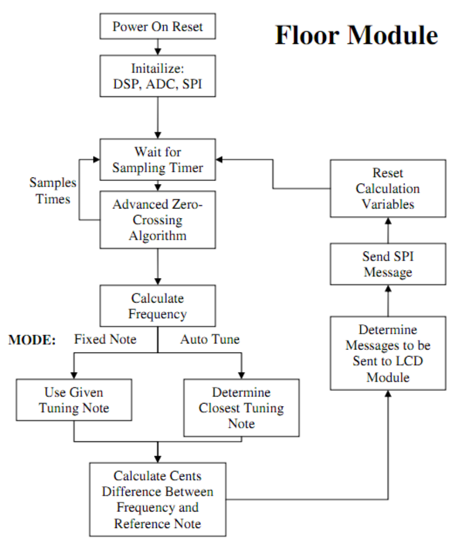

Figure 5Tuner software diagram

3.1.3 Tuning Data

This frequency is determined by the overall average of all the oscillation periods sampled. This note is then compared to the selected tuning note or the closest musical note if in auto-tune. To determine the variation in pitch between the calculated frequency and the desired frequency a log scale is used to allow for a uniform representation across the musical spectrum. The unit of this log scale is called cents and ranges from -50 to 50 around every note. The conversion is:

$$Cents = 1200 \log_2 \frac{frequency\ estimate}{frequency\ ideal} \quad (3)$$

The human ear can only hear pitch differences with a 5 cent magnitude. With the used zero crossing algorithms and some multiple-cycle averaging to determine pitch stability, the tuner module can determine the pitch within an average of 1 cent accuracy. Most stock tuners have a 5 cent accuracy, but top of the line tuners claim to have a 0.1 cent accuracy. The tuner module then waits for the display module to initiate SPI communication and sends its updated cents value.

When a new note is chosen to tune the software runs through the whole pitch detection algorithm three times to ensure that a new note has settled and is being determined correctly, not just a noisy reading. Also, if the range of the incoming single is not great enough, the module sends a "No Signal" message to be displayed on the screen.

3.2 Serial Peripheral Interface (SPI)

The communication between the two modules is sent through a PS-2 cable. The PS-2 cable is a 6-pin Mini-DIN connector that is mostly used for connecting mice and keyboards to computers. The 6-pins allow us to send power, ground, and four signals for SPI communication (clock, enable, MISO, MOSI). The power is sent from the tuner module to the display module so that the power source can be plugged in without regard for the placement of the display module. The SPI is set up for 8 bit communication at a slow clock rate to help reduce false readings over the 5 feet of cable.

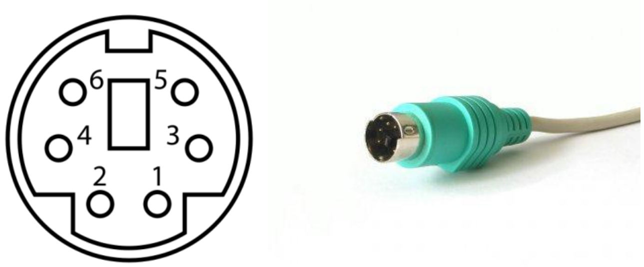

Figure 6 SPI PS/2 cable

The display module is the master for the SPI communication. When a timer interrupt is reached the display module initiates communication between the two modules. It sends the tuning mode to the tuner module in the form of 0xFF for auto-tune or a corresponding index for a specific note if in select tune mode. The tuner module, which is in slave mode, responds by sending over the cents difference and the determined note when in auto-tune mode. Because the tuner module can only send one message at a time it must only send the note index every few cycles, so for a short time the display module may be displaying the incorrect note but is updated as soon as the note index is sent (usually about half a second).

3.2.1 Decode Data

In order to communicate between the devices we have developed a decode method for the 8-bit data sent through the SPI. The table below summarizes the decode, for example for A4 the value is determined by multiplying 12 by 4, 12*4 = 48 then adding to the base octave of A, A=0 so, 0+48 = 48. Therefore to indicate an A we must send 48 in hex over.

Note	Octave 1 (base)	Add multiples of 12	Octave 5
A	0	…+12 …	60
A#	1	…+12 …	61
B	2	…+12 …	62
C	3	…+12 …	63
C#	4	…+12 …	64
D	5	…+12 …	65
D#	6	…+12 …	66
E	7	…+12 …	67
F	8	…+12 …	68
F#	9	…+12 …	69
G	10	…+12 …	70
G#	11	…+12 …	71

Table 1Values are in decimal base

For cent difference decode we simply send a value between 100 and 200 in hex. A value of 150 indicates the note being played is in tune while anything below 150 is –(150-value) cents off and above is +(value-150) cents off.

3.3 Display Unit

The active display unit provides a simple touch screen interface for the musician when tuning. There are 2 active tuning options provided, auto tune and select tune. Both options are included to cater to user preferences. The display unit is a graphical LCD (GLCD) which provides the interface and custom animation to convey tuning information. Figure 7 shows the user display projected on the display screen. The MSP430F2272 provides the necessary peripherals to operate the touchscreen, display and SPI communication with the DSP. The MSP430 provides a fast controller that can easily handle the multiple tasks needed from the device. The MSP430 is powered via +3.3 voltage regulator (LM3940).

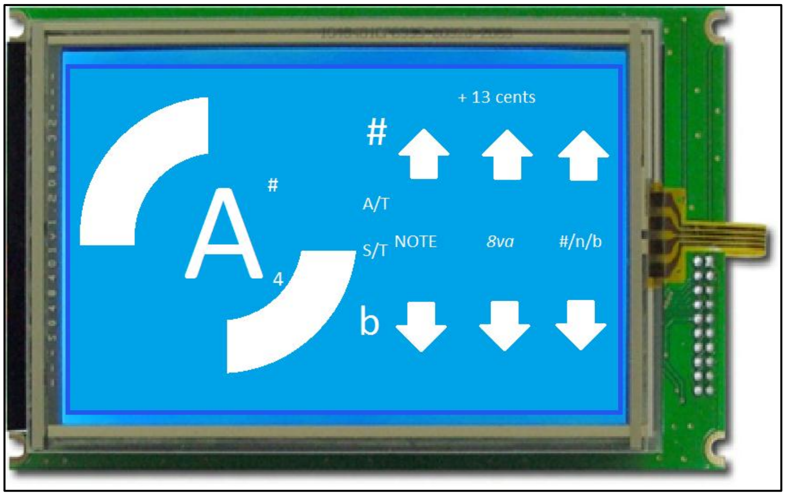

Figure 7 GLCD Display

3.3.1 Tuning Options

In order to be accessible to any musician's preferences we provided 2 methods of tuning. The musician can simply switch from one to other by selecting it on the touch screen.

3.3.1.1 Auto Tuning

Auto tuning provides an open approach for users to tune. The musician simply has to play the note into the floor module the closest note will be projected along with the instrument's cent deviation. The user therefore does not need to continually adjust the tuner when calibrating the instrument.

3.3.1.2 Select Tuning

The select tuning allows for a much faster cent deviation acquisition. Before playing the user must select the note using the touch screen interface and then play the note. The only difference between this method and auto tuning is that the user must select each note before trying to tune.

3.3.2 User Interface

The user interface allows the user to select S/T for select tuning or A/T for auto tuning. The device defaults to select tuning at A4. When in select tuning the user can choose the note, octave, and accidentals providing a wide range of tunable notes. The LCD screen also has an adjustable contrast and backlight allowing for adjustments for all lighting situations.

3.3.2.1 Touch Screen

A 4-wire resistive touch screen provides the touch interface that the user can uses to control the tuner. The touch screen is controlled through the MSP430F2272 analog ports A0-A3. A resistive touch screen was chosen due to its simplicity and quick response.

3.3.3 Display Unit Software

The display unit software basically runs through a main program with 2 separate timer interrupts that draws the animation and communicates with the floor module. The program runs through main loop that initializes the necessary peripherals on power up and continuously controls the touch screen and updates the interface display (for select tuning only). TimerA0 constantly updates the strobe animation and cent difference on the screen and TimerB0 communicates with the DSP/Floor Unit in order to receive the tuning information.

3.3.3.1 Main

The main program defaults to the select tune option which waits for a touch by the user. If no touch is detected the timer interrupts allow for tuning acquisition while the main program waits for a touch. When a touch is detected the program acquires the location of the touch and changes the display appropriately. If auto tune is pressed the program goes to auto tune mode until select tune is reselected again. Much of the GLCD software was taken from previous open source code online and modified to suit our needs. The header file of the T6963 controller allows for simpler LCD code.

3.3.3.2 TimerA0

TimerA0 constantly updates the strobe animation and cent deviation when in select tuning. When in Auto Tune only the strobe animation updates. The cent deviation data determines the speed of the animation and timer up count value. As the deviation grows smaller the animation simply revolves slower. To project this effect the timer interrupt’s 16-bit counter is increased to a higher value each time. Once in tune, or no cent deviation, the animation halts. The animation is a series of 12 distinct bit maps that are drawn continuously as the tuner operates.

3.3.3.3 TimerB0

TimerB0 communicates via SPI every 0.5 seconds in order to poll updated information from the floor unit. The local variables are updated only is the data is valid. If the data is invalid the previous value is retained. For auto tune the value is read and printed onto the display while local variables are changed. In select tune it only changes the local variables.

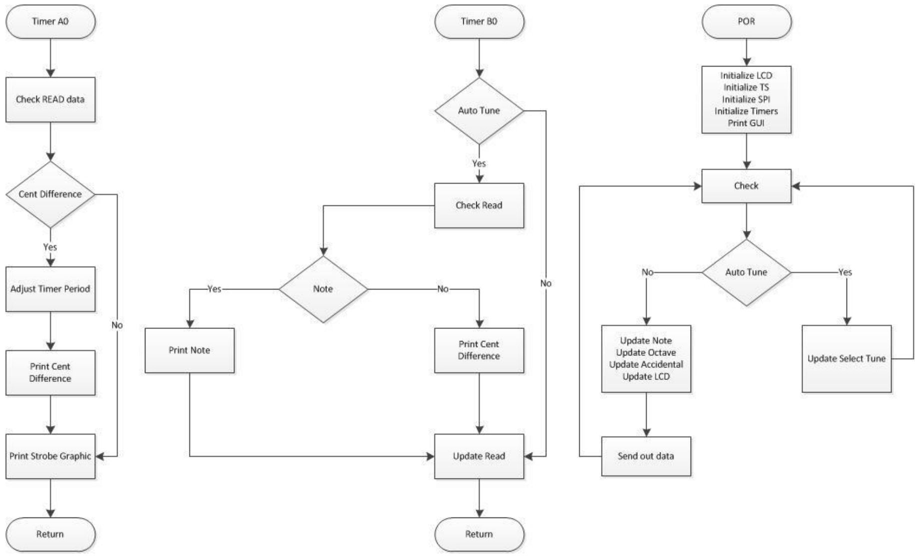

Figure 8 Display Unit Software

3.3.4 GLCD Interface

The GLCD provides 20 pins for both communication and display options. There are 4 controls pins and 8 data pins that are connected to the MSP430's GPIOs (P1.0-1.3, P4.0-4.7). Using bit banging, simple functions could be written in order to communicate and update the GLCD accordingly. The advantage of a CFAG240128L provided by Crystalfontz was the fast communication clock (12MHz). This allowed for quick graphical setups for the user interface and animation. The GLCD screen is powered via 5V from the floor unit which powers both the backlight and GLCD unit.

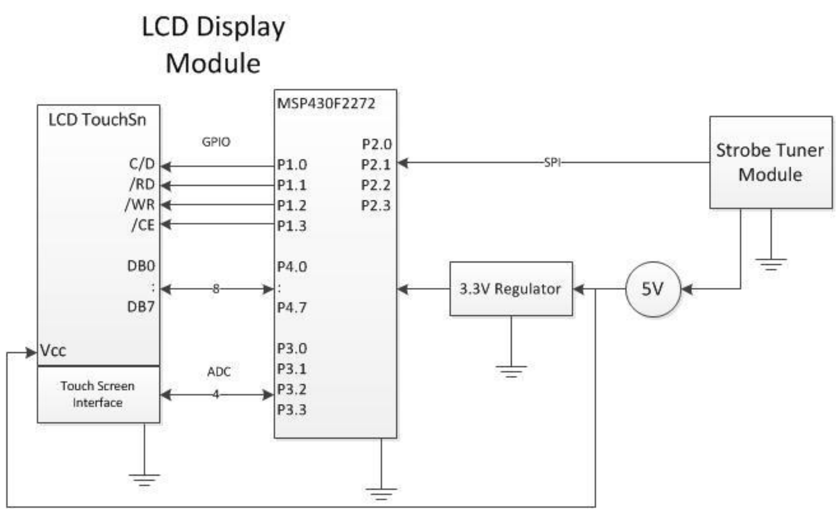

Figure 2: Hardware Diagram

3.3.5 Diagrams and Housing Units

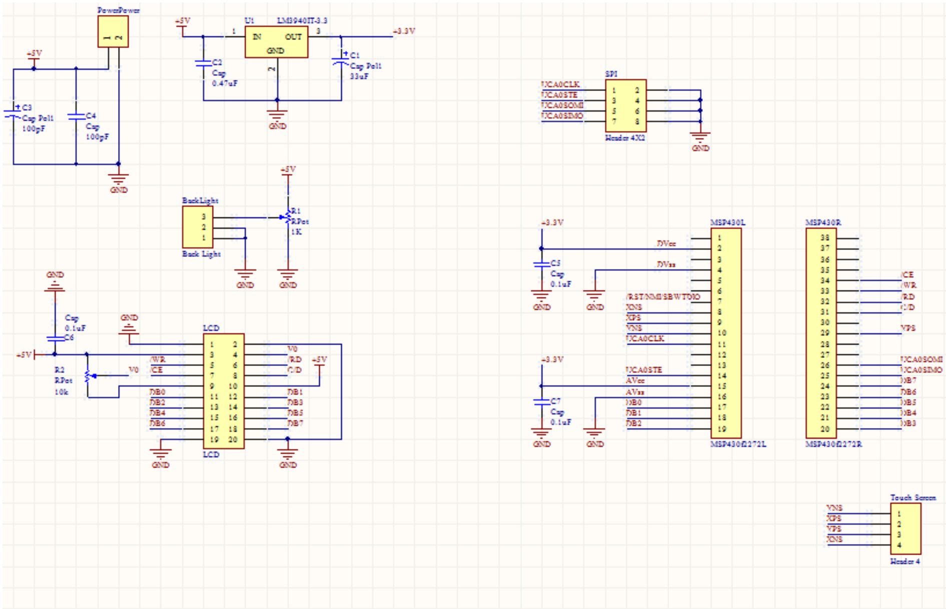

Figure 9 Display Unit Schematic

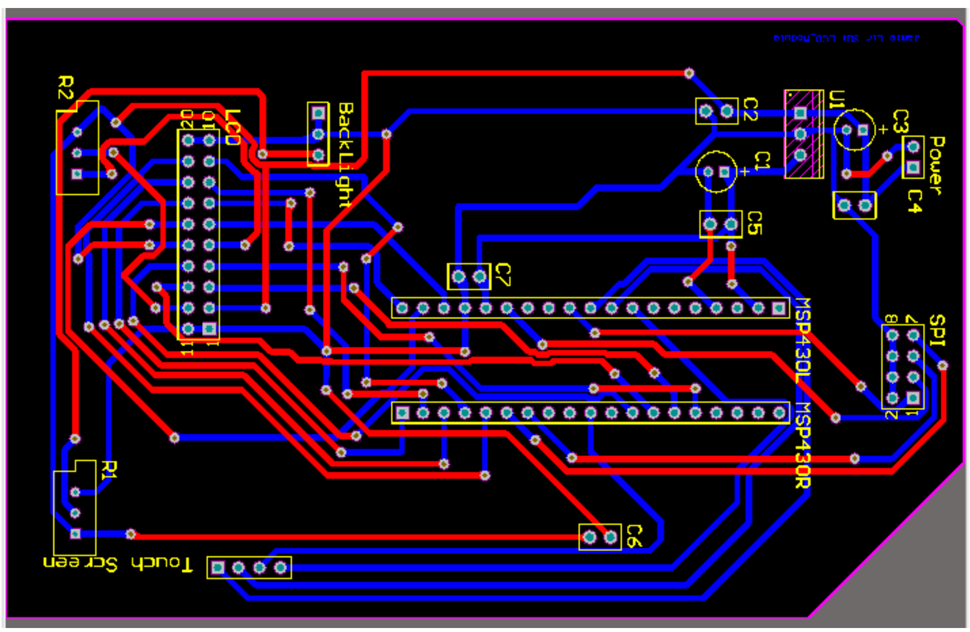

Figure 10 Display Unit PCB

Final Design Report

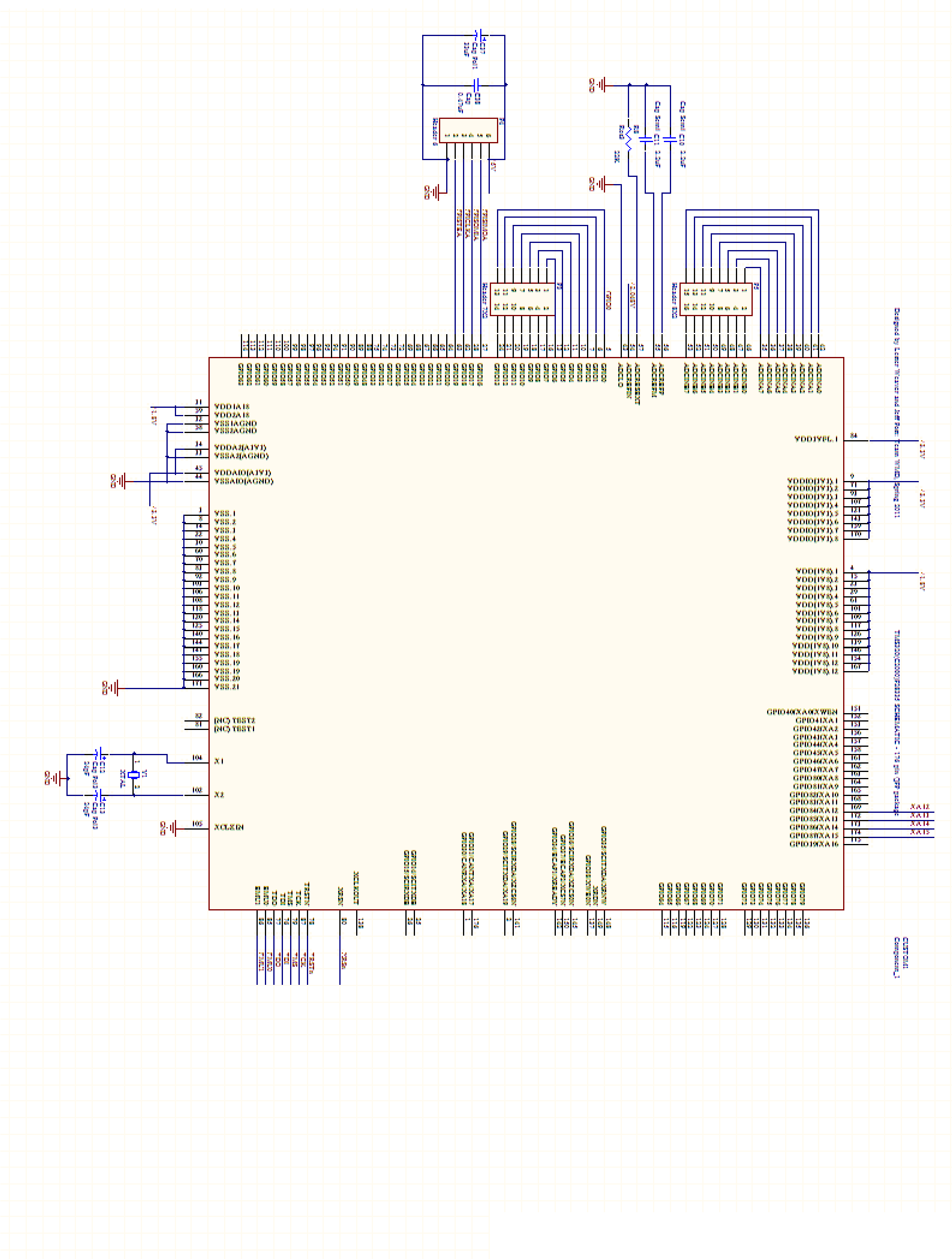

Figure 11 DSP Pin IN/OUT

Final Design Report

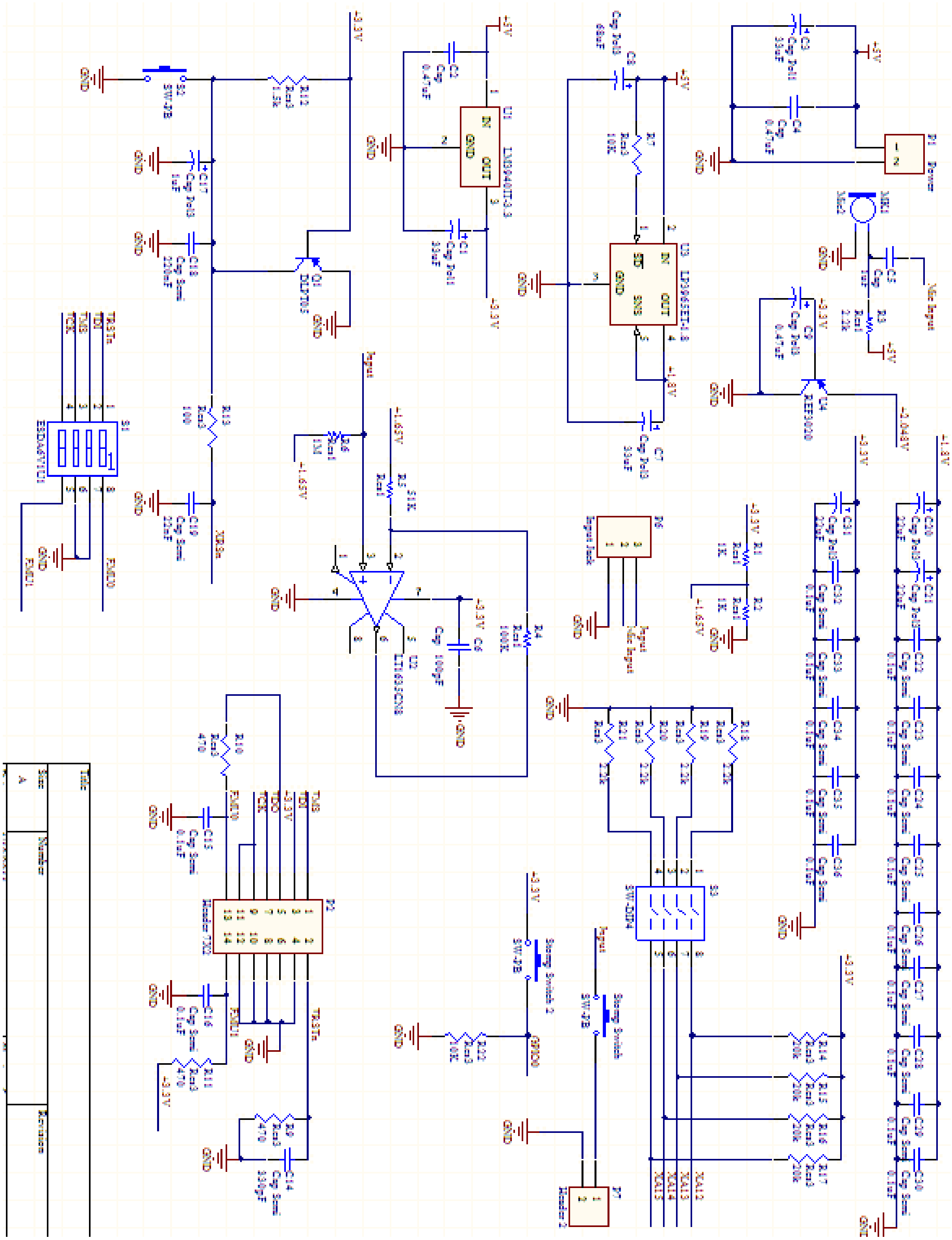

Figure 12 Floor Unit Schematic

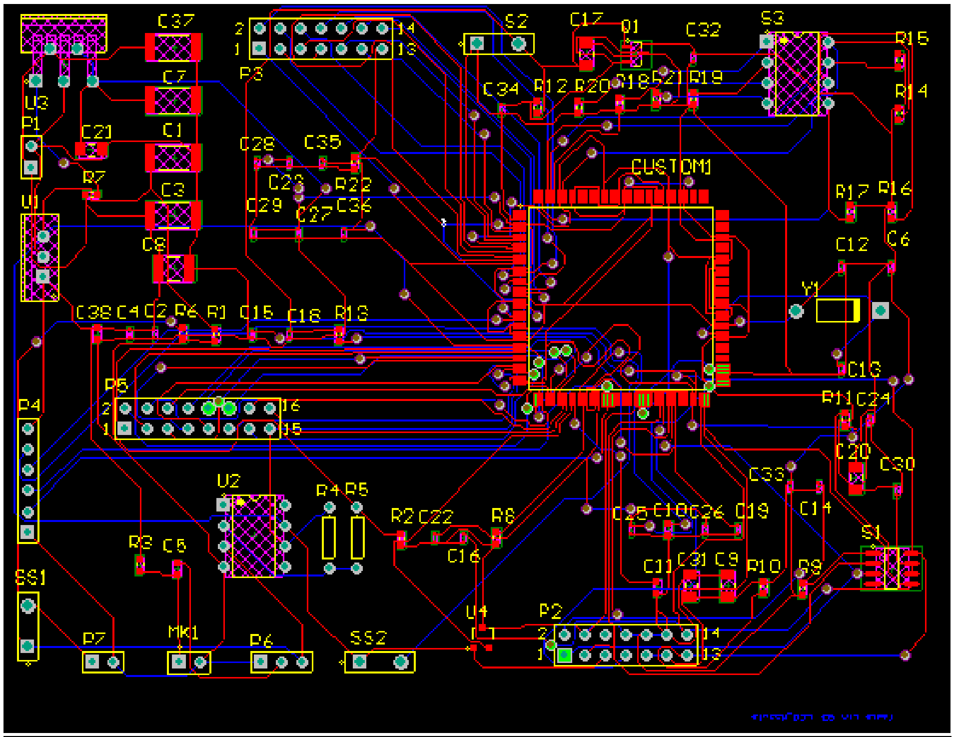

Figure 13 Floor Unit PCB

Figure 14 Display Unit

Final Design Report

Figure 15 Left Side of Display Unit

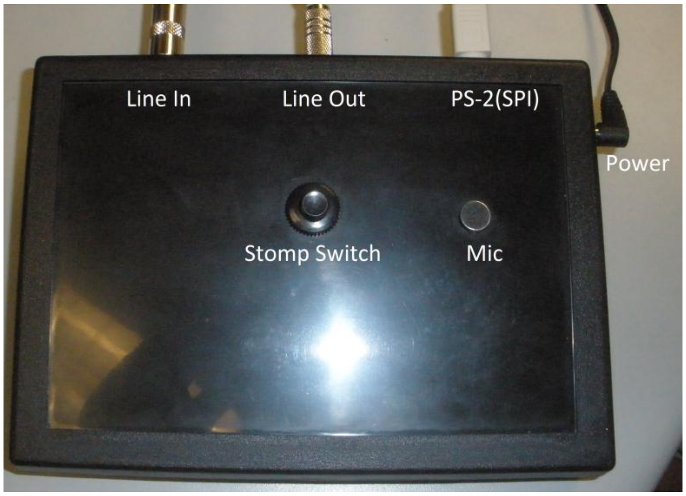

Figure 16 Floor Unit Housing

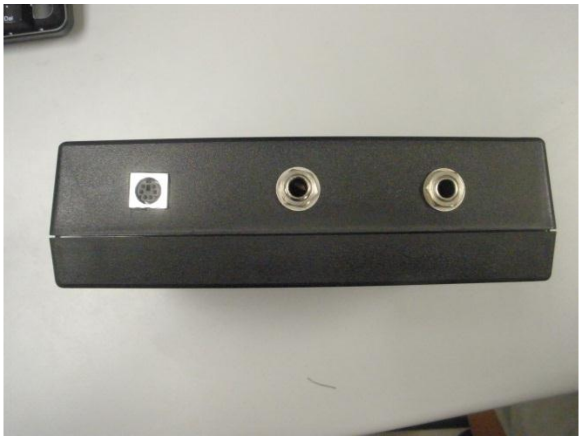

Figure 17 Floor Unit side view

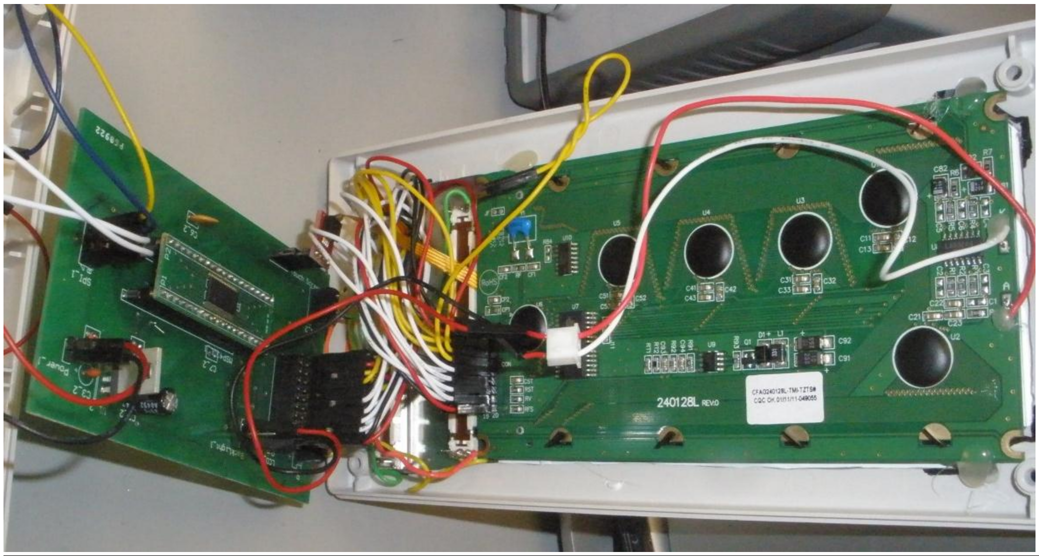

Figure 18 Display Unit Interior

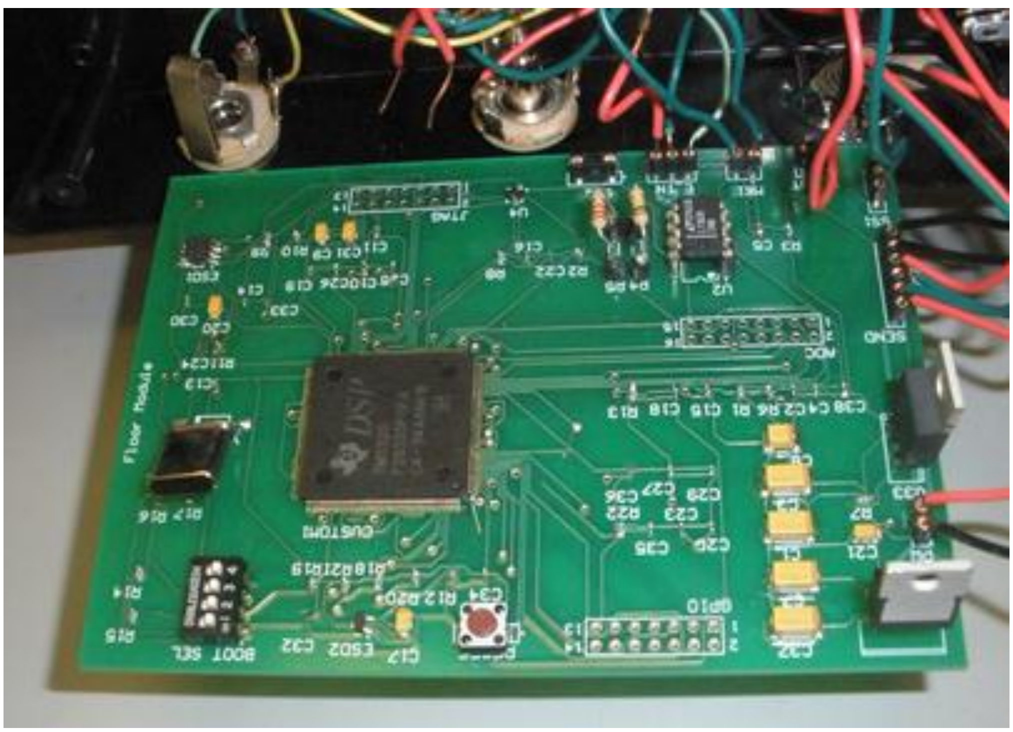

Figure 19 Floor Unit Interio

4.0 Charts

4.1 Division of Labor

Item	David	Jamie
Research & Design	50%	50%
Choose/Order Parts	50%	50%
Input/Preamp Design	35%	65%
ADC/Pitch Software	100%	0%
LCD/Touch Interface	0%	100%
Floor Module PCB	80%	20%
LCD Module PCB	0%	100%
SPI Interface	50%	50%
Final Construction	50%	50%

Table 2Division of Laboe

4.2 Gantt Chart

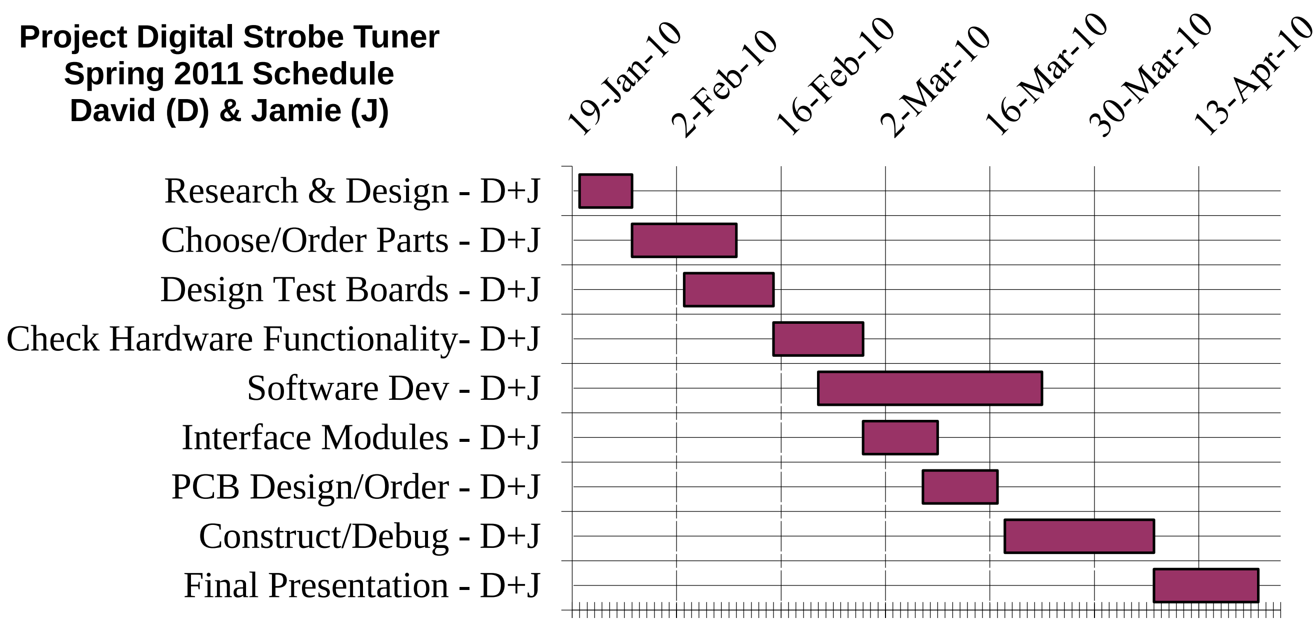

Table 3 Gantt Chart/Time Table

4.3 Parts and Cost

Part	Part Number	Cost
DSP	TMS320F28335	$20.54
Microcontroller	MSP430F2722	$1.59
Touch LCD Screen	CFAG240128L-TMI-TZTS	$87.08
Faceplate	Custom	$32.05
Capacitors	Various	$ 5.48
PCB	Custom	$33.00
Regulators	LM3940	$ 1.00
LCD Housing	HM211-ND	$ 7.90
Tuner Housing	PS-11502-G-ND	$10.10
Preamp	MAX9813	$ 0.57
Opamps	LT1630	$ 4.32
Audio Jack	¼”	$ 2.10
Condenser Mic	CMA-6542PF	$ 1.03
24.000MHz Crystal	ECS-240-S-1	$ 0.40
IC Switch	ADG444BNZ	$ 2.25
Push Button Switch	KS-03Q-03	$ 0.66
	Total:	$170.59

Table 4 Parts and Costs

5.0 References

1. CFAG240128L-TMI-TZ-TS Data Sheet February 2010 http://www.crystalfontz.com/products/document/2752/CFAG240128LTMITZTS_vPreliminary_3.1.pdf
2. LT1635 Data Sheet http://cds.linear.com/docs/Datasheet/1635fa.pdf
3. LCD.h, T6963.h Aprilc24, 2008 http://www.wpi.edu/Pubs/E-project/Available/E-project-042108-083837/unrestricted/MQP-FAADR-Report-final.pdf
4. www.petersontuners.com/
5. http://en.wikipedia.org/wiki/Electronic_tuner
6. 4-wire and 8-wire Resistive Touch Screen controller using MSP430 Rev. A, 29 Nov 2010 http://focus.ti.com/mcu/docs/litabsmultiplefilelist.tsp?sectionId=96&tabId=1502&literatureNumber=slaa384a&docCategoryId=1&familyId=342
7. MSP430F2272 Data Sheet http://focus.ti.com/lit/ds/symlink/msp430f2272.pdf
8. MSP430x2xx User Guide 2008 http://focus.ti.com/lit/ug/slau144h/slau144h.pdf
9. TMS320F28335 User Guide http://focus.ti.com/lit/ds/symlink/tms320f28335.pdf

6.0 Source Code

6.1 Display Unit

TouchScreen.h

```
//*****************************************************************************
//   MSP430 8 and 4-wire resistive touchscreen application
//   By: Jamie Lin
//   Modified from: TI MSP430 8 and 4-wire resistive touchscreen notes code
//   Febraury 2011
//
//*****************************************************************************

// Pin Definitions
// Port 1

#define XNS         (0x01)
#define XPS         (0x02)
#define YNS         (0x04)
#define YPS         (0x08)



#define XND         XNS
#define XPD         XPS
#define XPDDIR      P2DIR
#define XPDOUT      P2OUT
#define YND         YNS
#define YPD         YPS


#define YPIIE       P2IE
#define YPIIFG      P2IFG
#define YPIIES      P2IES
#define YPIDIR      P2DIR
#define YPIIN       P2IN
#define YPIOUT      P2OUT
#define YPIREN      P2REN
#define YPI         YPS
#define YPVECTOR    PORT2_VECTOR

#define CHAN_XPS    INCH_1
#define CHAN_YPS    INCH_3


// ADC Averaging Values
#define NUMSAMPLES  6
#define SAMPLESHIFT 2



// Global Variables
char touched;                              // Flag for touch status
unsigned int samples[NUMSAMPLES];          // ADC sample results
unsigned int x[2],y[2];                    // X,Y touch position variables

// System Routines
void initTS(void);                     // Configure modules & control Registers
void waitForTouch(void);                   // Waits for a touch on the screen
```

```
unsigned int getSamples(unsigned int);      // Get samples from ADC


void readXY(unsigned char index);           // Read X,Y coordinates
void setXDrive(void);                       // Drive X wires
void setYDrive(void);                       // Drive Y wires
void clearDrives(void);                     // Stop driving X and Y
void setTouchDrives(void);                  // Drive touch test wires
void delay(char);                           // Software delay



// Initializing Device
void initTS(void)
{
  BCSCTL3 |= LFXT1S_2;                      // LFXT1 = VLO
  // 16x S&H, MSC, ADC10ON, interrupt enabled
  ADC10CTL0 = SREF0 + ADC10SHT_2 + ADC10ON + ADC10IE; // Enable ADC
  //ADC10DTC1 = NUMSAMPLES;                   // Set number of conversions


  P2OUT = 0;                                // Clear outputs
  P2DIR = 0;                                // Touch screen pins input
  YPIDIR &= ~(YPI);                         // Touch screen pins input
  YPIIES = YPI;                             // Y+drive high to low transition int
  ADC10AE0 = (XNS+XPS+YNS+YPS);             // Enable analog input on all ADC pins
}

// Wait for Touch on Screen
void waitForTouch(void) {
  setTouchDrives();
  while((YPIIN & YPI)) {             // Y+ high?
    YPIIFG = 0;                             // Clear interrupt flags
    YPIIE |= YPI;                           // Enable Y+ interrupt on H-L
    __bis_SR_register(GIE);                 // Wait for interrupt
  }
  YPIREN &= ~YPI;                           // Disable pull-up resistor
  clearDrives();                            // Clear drive wires
}

// Get ADC Samples of Specified Channel and Average
unsigned int getXloc(unsigned int chan)
{

  unsigned int avg = 0;
  ADC10CTL0 &= ~ENC;
  ADC10CTL0 |= ADC10ON;
  P2SEL = 0x08;       // Measure the x distance
  ADC10CTL1 = chan;
  ADC10CTL0 &= ~ADC10SC;
  ADC10CTL0 &= 0x1FFF;
  ADC10CTL0 |= ADC10SC + ENC;
  while(ADC10CTL1 & BUSY);
  // Disable conversion
  avg = ADC10MEM;
  ADC10CTL0 &= ~ADC10ON;
  ADC10CTL0 &= ~ENC;
  return avg;
}

unsigned int getYloc(unsigned int chan)
{
  unsigned int avg = 0;
  ADC10CTL0 &= ~ENC;
  ADC10CTL0 |= ADC10ON;
  P2SEL = 0x010;      // Measure the x distance
  ADC10CTL1 = chan;
  ADC10CTL0 &= ~ADC10SC;
```

```
  ADC10CTL0 |= ADC10SC + ENC;
  while(ADC10CTL1 & BUSY);
  // Disable conversion
  ADC10CTL0 &= ~ADC10ON;
  ADC10CTL0 &= ~ENC;
  avg = ADC10MEM;
  return avg;
}

// Read X and Y Coordinates of Touch
void readXY(unsigned char index) {
  // Find X Coordinate
  setXDrive();                            // Set X wires to drives
  x[index] = getXloc(CHAN_YPS);        // Sample Y+ to get X coordinate
  clearDrives();                          // Clear drive wires


  // Find Y Coordinate
  setYDrive();                            // Set Y wires to drives
  y[index] = getYloc(CHAN_XPS);        // Sample X+ to get Y coordinate
  clearDrives();                          // Clear drive wires


  // Setup pins to check if screen is still touched
  setTouchDrives();                       // See if screen still touched

  if ((YPIIN&YPI)) {                      // Y+ high?
    touched = 0;                          // Not touched
  } else {                                // Y+ low
    touched = 1;                          // Touched
  }
  YPIREN &= ~YPI;                         // Disable pull-up resistor
  clearDrives();                          // Clear drive wires
}

// Drive X+drive High and X-drive Low, Configure ADC Pins
void setXDrive(void) {
  XPDOUT |= XPD;                          // X+drive = high
  XPDDIR |= XPD;                          //
  P2OUT &= ~XND;                          // X-drive = low
  P2DIR |= XND;                           // X+D,X-D = outputs  - P1.0

  P2DIR |= YND;
  P2OUT &= ~YND;                          // Y-drive = low - to discharge pin
  P2DIR &= ~YND;
  ADC10AE0 &= ~(XPS+XNS);                 // Enable GPIO Function
  delay(1);                               // Delay to allow settling
}

// Drive Y+drive High and Y-drive Low, Configure ADC Pins
void setYDrive(void) {
  YPIOUT |= YPI;                          // Y+drive = high
  P2OUT &= ~YND;                          // Y-drive = low
  P2DIR |= YND;                           // Y+D,Y-D = outputs
  YPIDIR |= YPI;

  P2DIR |= XND;                           // X-drive = low - to discharge pin
  P2OUT &= ~XND;
  P2DIR &= ~XND;
  ADC10AE0 &= ~(YPS+YNS);                 // Enable GPIO Function
  delay(1);                               // Delay to allow settling
}

// Stop Driving X and Y, Configure ADC Pins
void clearDrives(void) {
  P2OUT &= ~(YND+XND);                    // Y+D,Y-D,X+D,X-D = output low
  P2DIR &= ~(YND+XND);                    // Y+D,Y-D,X+D,X-D = inputs
  XPDOUT &= ~XPD;                         //
```

```
  XPDDIR &= ~XPD;
  YPIOUT &= ~(YPI);
  YPIDIR &= ~(YPI);
  ADC10AE0 = (YPS+YNS+XPS+XNS);             // Enable analog inputs
}

// Set Y+drive to input with pull-up, X-drive to output low
void setTouchDrives(void) {
  YPIDIR &= ~YPI;                           // Y+drive = input
  YPIOUT |= YPI;                            // Resistor pull-up
  YPIREN |= YPI;                            // Enable resistor
  P2OUT &= ~XND;                            // X-drive = low
  P2DIR |= XND;                             // X-drive = output
  ADC10AE0 &= ~(YPS+XNS);                   // Enable GPIO Function
  delay(1);                                 // Wait for pin to settle
}

// SW Delay
void delay(char times) {
  volatile unsigned int z;
  while(times > 0) {                        // SW delay loop
    for(z=0;z<0xFF;z++);
    times--;
  }
}
#pragma vector=ADC10_VECTOR
__interrupt void adc10(void) {
  //__bic_SR_register_on_exit(GIE); // Wake up from LPM0
}

// Port Interrupt Service Routine
#pragma vector=YPVECTOR
__interrupt void port(void) {
  YPIIE &= ~YPI;                            // Disable Y+drive interrupt
  YPIIFG = 0;                               // Clear interrupt flags
  //__bic_SR_register_on_exit(GIE);           // Wake up from LPM4
}
```

SPI.h

```
void initSPI (void);
void DSP_com(char temp);


void initSPI(void)
{
       P3SEL |= 0x31;                                      // P3.0,4,5 USCI_A0 option select
       P3DIR |= 0x08;                                              // P3.3 SPI enable
    //UCA0CTL0 |=  UCCKPH + UCMSB + UCMST + UCSYNC + UCMODE1; // 4-pin, 8-bit SPI master,
    UCA0CTL0 |=  UCCKPH + UCCKPL + UCMSB + UCMST + UCSYNC;         // 3-pin, 8-bit SPI master,
    UCA0CTL1 |= UCSSEL_1;                                // running off 12 MHZ
    UCA0BR0 = 0x01;                            //Leave CLK at 16 MHz
    UCA0BR1 = 0;
    UCA0MCTL = 0;
    UCA0CTL1 &= ~UCSWRST;
    P3OUT |= 0x08;                                                 // Set SPI high

}


unsigned int DSP_com(char temp)
{

    while (!(IFG1 & UCA0TXIFG));
       UCA0TXBUF = temp;                     // Dummy write to start SPI
       IFG2 &= ~UCA0RXIFG;                  // Clear int flag
```

```
    while (!(IFG2 & UCA0RXIFG));            // RXBUF ready?
    return UCA0RXBUF;                       // Move value

}
```

Main.h

```
/*
 Digital Strobe Tuner S.C.  ver 1.0
 UF Senior Design 2011 Spring
 MSP430F2272
 By: Jamie Lin
*/
#include "msp430x22x2.h"
#include "LCD.h"
#include "TouchScreen.h"
#include "SPI.h"
unsigned int loc1=0, loc2=0, note = 1, octave = 4, temp_str = 1, snf= 2;
unsigned int auto_sel = 0, prev_val = 0;
unsigned int WRITE = 0;
unsigned int READ = 0x2D;
unsigned int prev_READ = 0;
unsigned int no_sig = 1;
LCD lcd;

int main( void )
{
  // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    DCOCTL = 0x00;
    BCSCTL1 = CALBC1_12MHZ;
    DCOCTL  = CALDCO_12MHZ;
    BCSCTL2 = SELM_0 + DIVM_0;
    BCSCTL3 |= (LFXT1S_2);
    TACCTL0  = CCIE;
    TACTL = TASSEL_1 +MC_1 +TACLR; // Ext. INCLK, interrupt
    TACCR0 = 1000;
    TBCCTL0  = CCIE;
    TBCTL = TBSSEL_1 +MC_1 +TBCLR; // Ext. INCLK, interrupt
    TBCCR0 = 20500;                 //SPI transmit frequency
    for(int q = 0; q<30000; q++){
        for(int t = 0; t<3000; t++){
     }
    }
    initTS();
    lcd.init();
    lcd.draw_graphic(16,5,autotune);
        lcd.draw_graphic(16,9,seltune);
        lcd.draw_graphic(23,3,up);
        lcd.draw_graphic(23,11,down);
        lcd.draw_graphic(20,3,up);
        lcd.draw_graphic(20,11,down);
        lcd.draw_graphic(26,3,up);
        lcd.draw_graphic(26,11,down);
        lcd.draw_graphic(16,1,sharp);
        lcd.draw_graphic(16,13,flat);
        lcd.draw_graphic(26,7,natural);
        lcd.draw_graphic(23,7,octave_1);
        lcd.draw_graphic(20,7,note_1);
        lcd.draw_letter(1);
        lcd.write_text(10,11,"4");
```

```
        lcd.write_text(20,1,"+");
        lcd.write_num_text(21,1,0);
        lcd.write_num_text(22,1,0);
        lcd.write_text(24, 1, "cents");
        initSPI();
        WRITE = lcd.parse_data(1, 4, 2);

    __bis_SR_register(GIE);
        while(1){
                waitForTouch();
                    touched = 1;                            // Screen is touched
                    while(touched == 1)
                    {
                                                         // Loop while screen is touched
                     readXY(0);                             // Get X,Y coordinates index 0
                     readXY(1);                             // Get X,Y coordinates index 1

                      // Take 2 values and see the difference between them.
                      // If value is greater than 5 steps, discard packet.
                      // This code is designed to take a majority vote of 2 packets.

                      signed int temp = x[0] - x[1];        // Diff 2 of x-coordinates
                      if (temp & 0x8000)                    // Convert 2's complement to
                      {                                     // positive value
                        temp = ~temp;
                        temp++;
                      }

                      if (temp <= 5)                        // Check if diff value is smaller
                      {                                     // than 5
                       loc1 = x[0];
                       loc2 = y[0];
                      }

                    }
                    if(auto_sel == 0){
                            if(loc1 < 710 && loc1 > 600){
                                    if( loc2 > 670 && loc2 < 740){
                                            if(note >= 7){
                                                    note=7;
                                            }
                                            else{
                                                    note++;
                                            }
                                            __disable_interrupt();
                                            lcd.draw_letter(note);
                                            __enable_interrupt();
                                    }
                            }
                            if(loc1 < 350 && loc1 > 200){
                                    if( loc2 > 640 && loc2 < 740){
                                            if(note <= 1){
                                                    note=1;
                                            }
                                            else{
                                                    note--;
                                            }
                                            __disable_interrupt();
                                            lcd.draw_letter(note);
                                            __enable_interrupt();
                                    }
                            }
                            if(loc1 > 600 && loc1 < 730){
                                    if( loc2 < 820 && loc2 > 760){
                                            if(octave >= 7){
                                                    octave = 7;
                                            }
                                            else{
```

Final Design Report

```
                        octave++;
                    }
                    __disable_interrupt();
                    lcd.draw_octave(octave);
                    __enable_interrupt();
                }
            }
            if(loc1 > 220 && loc1 < 400){
                if( loc2 < 840 && loc2 > 770){
                    if(octave <= 1){
                        octave=1;
                    }
                    else{
                        octave--;
                    }
                    __disable_interrupt();
                    lcd.draw_octave(octave);
                    __enable_interrupt();
                }
            }

            if(loc1 > 650 && loc1 < 700){
                if( loc2 < 960 && loc2 > 860){
                    if(snf >= 3){
                        snf=3;
                    }
                    else{
                        snf++;
                    }
                    __disable_interrupt();
                    lcd.draw_accidental(snf);
                    __enable_interrupt();
                }

            }

            if(loc1 > 200 && loc1 < 350){
                if( loc2 < 960 && loc2 > 850){
                    if(snf <= 1){
                        snf=1;
                    }
                    else{
                        snf--;
                    }
                    __disable_interrupt();
                    lcd.draw_accidental(snf);
                    __enable_interrupt();
                }
            }
            WRITE = lcd.parse_data(note, octave, snf);
    }
if(loc2 > 540 && loc2 < 630){
    if(loc1 > 500 && loc1 < 608){
            auto_sel = 1;
                __disable_interrupt();
                lcd.write_text(5,5,"      ");
                lcd.draw_graphic(16,9,seltune);
                lcd.draw_graphic(16,5,autotune_inv);
                __enable_interrupt();
    }
}
    if(auto_sel == 1){
            if(loc2 > 540 && loc2 < 630){
                if(loc1 > 310 && loc1 < 430){
                    auto_sel = 0;
                    __disable_interrupt();
                    WRITE = lcd.parse_data(1, 4, 2);
                    lcd.draw_letter(1);
```

```
                                        lcd.draw_octave (4);
                                        lcd.draw_accidental(2);
                                        lcd.draw_graphic(16,5,autotune);
                                        lcd.draw_graphic(16,9,seltune_inv);
                                        __enable_interrupt();
                                        P3OUT &= ~0x08;                          //
Enable SPI
                                        DSP_com(WRITE);
                                        P3OUT |= 0x08;                       //
Disable SPI
                                        __delay_cycles(100);
                                   }
                              }
                         }
                    loc1 = 0;
                    loc2 = 0;
  return 0;
}

//interrupt A
#pragma vector=TIMERA0_VECTOR
__interrupt void TimerA0_interrupt (void)
{
        __disable_interrupt();
    if(READ != 253 && READ != 254){
        if(no_sig == 1){
            lcd.draw_letter(note);
            lcd.draw_octave(octave);
            lcd.draw_accidental(snf);
            no_sig = 0;
        }

    if( READ == 150){
      TACCR0 = 2000;
      lcd.write_text(5,5,"      ");
      lcd.write_num_text(21,1,0);
          lcd.write_num_text(22,1,0);
          lcd.draw_graphic(16,13,flat);
          lcd.draw_graphic(16,1,sharp);
    }
        else if(READ < 199 && READ > 150){
                lcd.write_text(5,5,"      ");
                unsigned int h = READ;
                if(READ > 175){
                        h = 1000;
                }
                else{
                h = h- 150;
                        h = -392*h + 10408;
                }
                TACCR0 = h;
                temp_str = lcd.strobe_cw(temp_str);
           }
        else if(READ < 150 && READ > 101){
                lcd.write_text(5,5,"      ");
                unsigned int h = READ;
                if(READ < 125){
                        h = 1000;
                }
                else{
                h = 150 - h;
                h = -392*h + 10408;
                }
                TACCR0 = h;
                temp_str = lcd.strobe_ccw(temp_str);
           }
        else if((READ > 185 || READ < 115) && READ > 100){
                lcd.write_text(5,5,"      ");
```

Final Design Report

```
            if(temp_str==1){
                TACCR0 = 2000;
                 lcd.fstrobe_1();
                 temp_str=2;
            }else{
                 TACCR0 = 2000;
                 lcd.fstrobe_2();
                 temp_str=1;
            }
        }
        if(READ > 150 && READ < 200){
                lcd.write_text(20,1,"+");
                unsigned int k = READ - 150;
                k = k/10;
                lcd.write_num_text(21,1,k);
                k = READ - 150;
                k = k%10;
                lcd.write_num_text(22,1,k);
                lcd.draw_graphic(16,13,flat);
                lcd.draw_graphic(16,1,sharp_inv);
        }
        else if (READ <150 && READ > 100){
                lcd.write_text(20,1,"-");
                unsigned int k = 150-READ;
                k = k/10;
                lcd.write_num_text(21,1,k);
                k = 150-READ;
                k = k%10;
                lcd.write_num_text(22,1,k);
                lcd.draw_graphic(16,1,sharp);
                lcd.draw_graphic(16,13,flat_inv);
        }
        else if(READ > 200){
                lcd.write_text(20,1,"+");
                lcd.write_num_text(21,1,5);
                lcd.write_num_text(22,1,0);
        }
        else if(READ < 100 && READ > 72){
                lcd.write_text(20,1,"-");
                lcd.write_num_text(21,1,5);
                lcd.write_num_text(22,1,0);
        }
    }

        else if( READ == 254){
                lcd.write_text(5,5,"NO SIG");
                lcd.clear_graphic(10,4,s_sharp);
            lcd.clear_graphic(5,5,B);
            lcd.clear_graphic(10,4,s_flat);
            lcd.write_text(10,11," ");
            no_sig = 1;

        }
        else if(READ == 253){
                lcd.write_num_text(21,1,0);
            lcd.write_num_text(22,1,0);
            lcd.clear_graphic_a(1,8, arc3a1);
         lcd.clear_graphic_a(1,11,arc3b1);
         lcd.clear_graphic_a(5,12,arc3c1);
         lcd.clear_graphic_a(8,1,arc2a1);
         lcd.clear_graphic_a(11,1,arc2b1);
         lcd.clear_graphic_a(12,5,arc2c1);
         lcd.clear_graphic_a(12,8,arc4a1);
         lcd.clear_graphic_a(11,11,arc4b1);
         lcd.clear_graphic_a(8,12,arc4c1);
         lcd.clear_graphic_a(5,1,arc1a1);
         lcd.clear_graphic_a(1,1,arc1b1);
         lcd.clear_graphic_a(1,5,arc1c1);
```

```
        }
        __enable_interrupt();
}

#pragma vector=TIMERB0_VECTOR
__interrupt void TimerB0_interrupt (void)
{
            if(auto_sel == 1){
            P3OUT &= ~0x08;                                     // Enable SPI
                prev_READ = DSP_com(0xFF);
                P3OUT |= 0x08;                                  // Disable SPI
            __delay_cycles(100);
            if(prev_READ <73 and prev_READ != 0){
                            lcd.data_print(prev_READ-1);
                }
                if(prev_READ != 0 && prev_READ > 73){
                  READ = prev_READ;
                }
    }
    else{
        P3OUT &= ~0x08;                                         // Enable SPI
        __delay_cycles(100);
                prev_READ = DSP_com(WRITE);
                __delay_cycles(10000);
                prev_READ = DSP_com(WRITE);
                __delay_cycles(100);
                P3OUT |= 0x08;
                __delay_cycles(100);
                if(prev_READ != 0){
                  READ = prev_READ;
                }

    }
}
```

6.2 Floor Unit

Main.h

```
//////////////////////////////////////////////////////////////////////////
//// TUNER MODULE MAIN CODE //////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
// David Barnette

#include "F28335_example.h"                         // Main include file
#include "math.h"
#include "keys.h"
#include "AtoD.h"
#include "DSP28x_Project.h"

#define F_CPU 74.8
#define F_SAMPLE 30000
#define SAMPLES 3000
#define ZERO 2500
#define ZERO_R 1.2
#define BUFFER 7
#define BUFFER_ADC 5
#define DBNC 5
#define THRESHOLD 700
#define AUTOTUNE 255
#define DEBUG_NOTE 466
#define DEBUG_SWEEP 15
```

```
#define DEBUG_SPEED 0.5
#define NO_SIG 254
#define UNSETTLED 253



//Freq of notes C1 -> C7
float freq[] = {32.70, 34.65, 36.71, 38.89, 41.20, 43.65, 46.25, 49.00,    51.91, 55.00,
                              58.27, 61.74, 65.41, 69.30, 73.42, 77.78, 82.41, 87.31, 92.50, 98.00,
                              103.83, 110.00, 116.54, 123.47, 130.81, 138.59, 146.83, 155.56, 164.81,
174.61,
                              185.00, 196.00, 207.65, 220.00, 233.08, 246.94, 261.63, 277.18, 293.66,
311.13,
                              329.63, 349.23, 369.99, 392.00, 415.30, 440.00, 466.16, 493.88, 523.25,
554.37,
                              587.33, 622.25, 659.26, 698.46, 739.99, 783.99, 830.61, 880.00, 932.33,
987.77,
                              1046.50, 1108.73, 1174.66, 1244.51, 1318.51, 1396.91, 1479.98, 1567.98,
1661.22,
                              1760.00, 1864.66, 1975.53, 2093.00};

////////////////////////////////////////////////////////////////////////////////////////
//// Variables Section /////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
float freq_est, f_ideal, f_diff, diff_calc, cents;
float f_buff[BUFFER];
float f_calc[BUFFER];
int update_cents, Read;

Uint16 i, j, debounce;
Uint16 note_index, update_note;
float first_zc, last_zc, zc, calc_zero;

Uint16 tune_note = AUTOTUNE, rec;  // 100 for auto tune, else index for note being tuned
Uint16 ADC_samples[SAMPLES];
Uint16 max, min;

Uint16 messages[3] = {150, 0, 80}; // 0-cents, 1-note, 2-mute
Uint16 m_update[3] = {0,0,0}; // 0-no signal, 1-note, 2-mute, default if all '0'-cents
int tuner_count = 0;

float debug_freq = DEBUG_NOTE, debug_dir = DEBUG_SPEED;


//--- Global Variables
Uint16 AdcBuf[ADC_BUF_LEN];                           // ADC data buffer allocation
Uint32 PwmDuty;                                       // Measured PWM duty cycle
Uint32 PwmPeriod;                                     // Measured PWM period

void InitMicroP() // Turn off Watchdog timer, EALLOW
{
        InitSysCtrl();
        EALLOW;
        SysCtrlRegs.PLLCR.bit.DIV = 0;                //Pll=bypass
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 2;    //SysClkOut=OscClk/2
        SysCtrlRegs.LOSPCP.all = 0;
}


void InitIO() // Initialize GPIO, PCLKCR3
{
        GpioCtrlRegs.GPBMUX1.all = 0xFFFFC3C0;
        GpioCtrlRegs.GPCMUX1.all = 0xFFFF0000;
        GpioCtrlRegs.GPCMUX2.all = 0x0000FFFF;
        SysCtrlRegs.PCLKCR3.bit.XINTFENCLK = 1;
}

void InitTimers() // Initialize timers
```

```
{
	GpioCtrlRegs.GPAMUX1.all = 0x00000000;

}

Uint16 SendMessage( Uint16 messages[], Uint16 m_update[] ) // SPI message
{
	// Uint16 messages[3]  // 0-cents, 1-note, 2-mute
	// bool m_update[3]    // 0-no signal, 1-note, 2-mute, default if all '0'-cents
	Uint16 r_val, send;

	if	( m_update[0] == 1 )
		send = NO_SIG;			//no signal
	else if( m_update[2] == 1)
		send = messages[2];		//mute message
	else if( m_update[1] == 1)	{
		send = messages[1] + 1;		//note message
		m_update[1] = 0;
	}
	else
		send = messages[0];		 //cents message

	SendSPI( send );

	if(send == 128)
		send = 0;

	SpiaRegs.SPISTS.bit.OVERRUN_FLAG = 1;
	r_val = SpiaRegs.SPIRXBUF & 0xFF;

	if( r_val > 72 && r_val != AUTOTUNE )
		r_val = 255;
	else
		r_val = r_val;

	return r_val;

}
void main(void)
{
//--- CPU Initialization
	InitSysCtrl();				// Initialize the CPU
	InitPieCtrl();				// Initialize and enable the PIE
	InitWatchdog();				// Initialize the Watchdog Timer
	InitGpio();				// Initialize the shared GPIO pins
	InitXintf();				// Initialize the external memory interface

#ifdef EXAMPLE_FLASH		// EXAMPLE_FLASH, if defined, is in CCS project options

//--- Copy all Flash sections that need to run from RAM (use memcpy() from RTS library)

// Section secureRamFuncs contains user defined code that runs from CSM secured RAM
	memcpy(&secureRamFuncs_runstart, &secureRamFuncs_loadstart, (Uint32)&secureRamFuncs_loadsize);

//--- Initialize the Flash and OTP
	InitFlash();						// Initialize the Flash

#endif

//--- Peripheral Initialization
	InitAdc();				// Initialize the ADC
	InitEPwm();				// Initialize the PWM
	InitECap();				// Initialize the Capture units

	////////////////////////////////////////////////////////////////////////////////    ////
Intializing Section /////////////////////////////////////////////////////////
	////////////////////////////////////////////////////////////////////////////////
	InitMicroP();
```

```
InitIO();
//InitLCD();
InitAtoD();
InitSpi();

EALLOW;
InitCpuTimers();
ConfigCpuTimer( &CpuTimer0, F_CPU, 100000.0/F_SAMPLE);

calc_zero = ZERO;

        EALLOW;
GpioCtrlRegs.GPAMUX1.all = 0x00000000;        //set GPIO31:0 pins to GPIO
GpioCtrlRegs.GPADIR.all = 0x000000F0; //set GPIO0 as output

i = 0;

////////////////////////////////////////////////////////////////////////////////
//// Main Section ////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////

while(1)
{
        //initialize zero count variables
        first_zc = 0;
        last_zc = 0;
        zc = 0;
        debounce = 0;
        max = 0;
        min = 4096;

        ////////////////////////////////////////////////////////////////////////////////
        //// Sampling Section ///////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

        for( i=0 ; i<SAMPLES-1 ; i++ ){


                while(CpuTimer0Regs.TCR.bit.TIF  == 0);
                CpuTimer0Regs.TCR.bit.TIF = 1;

                ADC_samples[i] = GetAnalogInput();

                if( ADC_samples[i] > max ) max = ADC_samples[i];
                if( ADC_samples[i] < min ) min = ADC_samples[i];
                if( i>0 && ADC_samples[i] >= calc_zero && ADC_samples[i-1] < calc_zero ){
                        if( first_zc == 0 ) first_zc = i;

                        if( debounce == 0 ){ //if it hasn't been triggered recently
                                zc++;
                                last_zc = i;
                                debounce = DBNC;
                        }
                }

                if( debounce != 0) debounce--;
        }
        freq_est = (zc-1)*F_SAMPLE/(last_zc-first_zc);

        f_ideal = 0;
        f_diff = F_SAMPLE/2;

        if( debug_freq < DEBUG_NOTE-DEBUG_SWEEP ) debug_dir = DEBUG_SPEED;
        else if( debug_freq > DEBUG_NOTE+DEBUG_SWEEP ) debug_dir = -DEBUG_SPEED;

        debug_freq += debug_dir;

        ////////////////////////////////////////////////////////////////////////////////
```

Final Design Report

```
        //// Tuning Section ////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////


        //tune_note = AUTOTUNE;
        //determine tuning frequency
        if( tune_note == AUTOTUNE){
        // calculate closest frequency
                for( i=0 ; i<73 ; i++ ){
                        diff_calc = fabs( freq_est-freq[i] );
                        if(  diff_calc < f_diff ){
                                f_diff = diff_calc;
                                note_index = i;
                                f_ideal = freq[i];
                        }
                        else
                                break;
                }
        }
        else{ // if user selected
                f_ideal = freq[tune_note];
                note_index = tune_note;
        }

        calc_zero = ZERO_R*(max+min)/2;
        cents = 1200 * log(freq_est/f_ideal)/log(2);

        // round to nearest cent
        if( cents > 0 )
                cents = floor(cents);
        else if (cents < 0 )
                cents = -floor(fabs(cents));

        // max out cents difference
        if( cents > 48 )
                cents = 49;
        else if (cents < -48 )
                cents = -49;

        //check to see if the same value is being calculated
        j=1;
        f_calc[0] = freq_est;
        f_buff[0] = f_ideal;
        for(i=1; i<BUFFER; i++){
                if( f_buff[0] == f_buff[i] ) j++;
        }


        ////////////////////////////////////////////////////////////////////////////////
        //// Message Section ///////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

        //check for signal
        if( (max-min) < THRESHOLD || freq_est < 20 || freq_est > 20000 ){
                update_cents = NO_SIG - 150;
                messages[0] = update_cents;
                update_note = 0;
                m_update[0] = 1;        //send no signal message
        }
        // if signal has settled and needs to update
        else if( j==2 && tune_note != AUTOTUNE){
                update_cents = UNSETTLED;
                messages[0] = update_cents;
        }
        else if(j>=((BUFFER/2)) && update_cents != cents){
                m_update[0] = 0;        // don't send no signal  message
                update_cents = cents;
                update_note = note_index;
```

```
        if( messages[1] != update_note && tune_note == AUTOTUNE ){
            messages[1] = update_note;
            m_update[1] = 1;              // send new note message
        }

        messages[0] = 150+update_cents;
    }

    if(tune_note == AUTOTUNE ){
        messages[1] = update_note;
        if( tuner_count == 2 ){ // send note on auto tune every 10 cycles
            m_update[1] = 1;
            tuner_count = 0;
        }
        else tuner_count++;
    }

    if(tune_note != AUTOTUNE ){
        m_update[1] = 0;
    }

    for(i=BUFFER-1; i>0; i--) f_buff[i] = f_buff[i-1];
    for(i=BUFFER-1; i>0; i--) f_calc[i] = f_calc[i-1];

    // send 0-77 for note index
    // send 80 + mute_toggle for mute info
    // send 150 + cents for tuning info
    // send 256 for no read

    //SPI update for LCD module
    //tune_note = SendMessage( messages, m_update );

    rec = SendMessage( messages, m_update );
    if(rec != tune_note)
        tune_note = rec;

    //messages[0]
    GpioDataRegs.GPADAT.all = 0x00;
    if(messages[0] < 110){
        GpioDataRegs.GPATOGGLE.bit.GPIO4 = 0;
        GpioDataRegs.GPATOGGLE.bit.GPIO5 = 0;
        GpioDataRegs.GPATOGGLE.bit.GPIO6 = 0;
        GpioDataRegs.GPATOGGLE.bit.GPIO7 = 1;
    }
    else if(messages[0] < 130){
        GpioDataRegs.GPATOGGLE.bit.GPIO4 = 0;
        GpioDataRegs.GPATOGGLE.bit.GPIO5 = 0;
        GpioDataRegs.GPATOGGLE.bit.GPIO6 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO7 = 1;
    }
    else if(messages[0] < 140){
        GpioDataRegs.GPATOGGLE.bit.GPIO4 = 0;
        GpioDataRegs.GPATOGGLE.bit.GPIO5 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO6 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO7 = 1;
    }
    else if(messages[0] < 160){
        GpioDataRegs.GPATOGGLE.bit.GPIO4 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO5 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO6 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO7 = 1;
    }
    else if(messages[0] < 170){
        GpioDataRegs.GPATOGGLE.bit.GPIO4 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO5 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO6 = 1;
        GpioDataRegs.GPATOGGLE.bit.GPIO7 = 0;
```

```
		}
		else if(messages[0] < 180){
			GpioDataRegs.GPATOGGLE.bit.GPIO4 = 1;
			GpioDataRegs.GPATOGGLE.bit.GPIO5 = 1;
			GpioDataRegs.GPATOGGLE.bit.GPIO6 = 0;
			GpioDataRegs.GPATOGGLE.bit.GPIO7 = 0;
		}
		else{
			GpioDataRegs.GPATOGGLE.bit.GPIO4 = 1;
			GpioDataRegs.GPATOGGLE.bit.GPIO5 = 0;
			GpioDataRegs.GPATOGGLE.bit.GPIO6 = 0;
			GpioDataRegs.GPATOGGLE.bit.GPIO7 = 0;
		}

		i=0;
		while( SpiaRegs.SPISTS.bit.BUFFULL_FLAG == 1 && i<10000) i++;
		GpioDataRegs.GPADAT.all = 0x00;
	}
} // end of main()
```