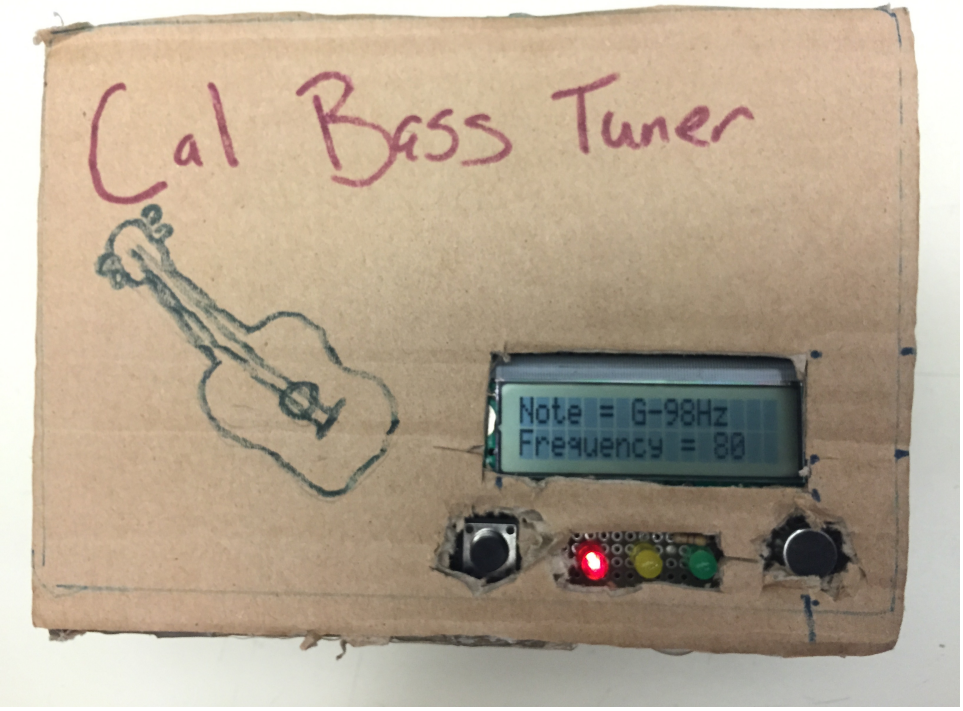
Project 45:
Bass Guitar Tuner!



Miguel Villafuerte
Alfredo Medina
Brian Holland

CPE 329 - 01
Jeff Gerfen

Fall 2014

YouTube Link: http://youtu.be/NAs-ECH_yFA

Purpose

The purpose of this project is to interface the MSP430 microcontroller to an analog sensor. In this project, the Cal Bass Tuner utilizes a condenser microphone to sense the acoustic waves produced by a 4-string bass guitar when a single note is played. The fundamental frequency at which the note resonates is then converted to a digital value and displayed on a LCD screen using the MSP430. The microcontroller compares the read frequency to the expected frequency of the note and three LEDs are used as reference for the accuracy of the bass note. A green-lit LED indicates an error of less than ±5%; a yellow-lit LED indicates an error of less than ±10%, and a red lit LED represents an error less than ±25%. All in all, the system provides a solution for tuning a 4 string bass guitar to the fundamental frequency associated with each note.

System Requirements

The Cal Bass Tuner is designed for bass guitarists that require a fast portable solution for tuning a 4-string bass guitar. LED indicators allow for easy interpretation and calibration of the desired note. The Cal Bass Tuner supports four different notes that may be calibrated. These notes include the E, A, D, and G; which are the common notes associated with a 4-string bass guitar. Figure 1 is high level block diagram of the system, which includes all peripherals of the tuner. Table 1 demonstrates the system requirements of the Cal Bass Tuner.

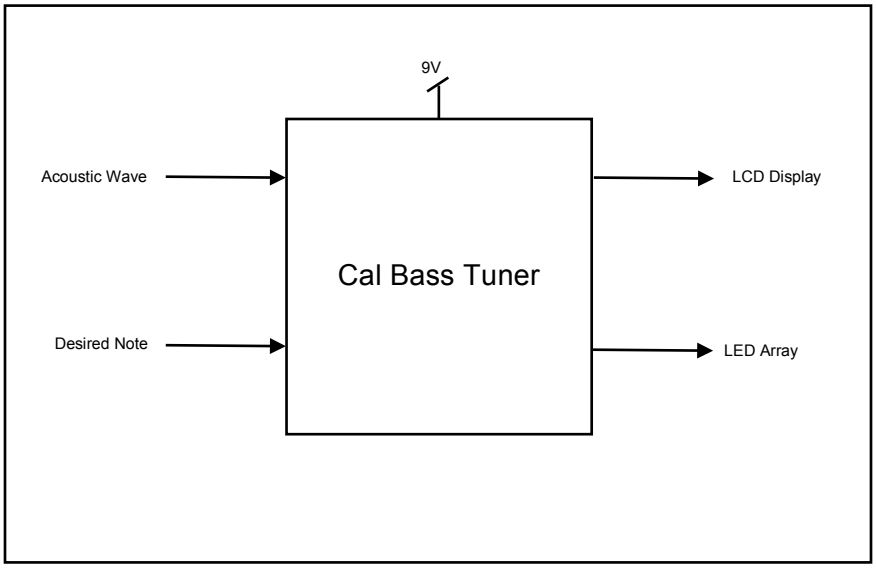


Figure 1: Cal Bass Tuner level 0 block diagram

Table 1: Cal Bass Tuner system requirements

System Requirements
Operates autonomously on a 9 Volt battery supply
Sense and isolate the fundamental frequency of four bass notes (E = 41.2Hz, A = 55.0Hz, D = 73.2Hz, G = 98Hz) [1]
Display the desired bass note and frequency on a LCD display
Display the measured frequency on a LCD display
Use button to select the desired bass note
Illuminate a red LED when the measured frequency is between ±10-25% of the desired frequency
Illuminate a yellow LED when the measured frequency is between ±5-10% of the desired frequency
Illuminate a green LED when the measured frequency is between ±0-5% of the desired frequency

System Specification

The Cal Bass Tuner's detailed specifications and operating conditions are shown in table 2.

Table 2: Cal Bass Tuner system specifications

Description	Specification
Power Supply	9V Battery
Current Consumption*	20.1mA
Typical battery life*	12,000 hours
Operating Range	Up to 1 meter
Maximum Accuracy	95%
Supported Notes/Frequencies	
	E = 41.2Hz
	A = 55.0Hz
	D = 73.2Hz
	G = 98.0Hz
Detection Range	20 – 150 Hz (For All Notes)
Ideal Operating Temperature	25ºC
Dimensions	16x11x4.5cm
Weight	255 g

*See Battery Powered Operation for details

System Architecture

The Cal Bass Tuner is broken up into two portions consisting of hardware and software. Table 3 demonstrates the roles each has on the overall system. The hardware is composed of a condenser microphone, amplifier stage, low pass filter stage, and a hysteretic comparator. The MSP430 receives a square wave proportional to the frequency of the acoustic wave sensed by the condenser microphone. The MSP430 then process the signal and displays the measured frequency on the LCD. In addition, a 5V regulator is used to supply power to the LCD screen and a 3.6V regulator supplies power to the MSP430. The MSP430 is interfaced with the LCD screen, LEDs, and buttons to allow the user to provide input to the system (button) and receive feedback (LED's and LCD).

Table 3: Hardware and software tasks

Hardware	Software
Supplies voltage to circuit	Set Clocks, Interrupts ISRs
Amplifies signal from condenser microphone	Set up and run LCD
Filters out unwanted harmonics	Set and control LEDs
Convert analog signal to square wave	Count the incoming frequency
	Compare in a loop amplitude values

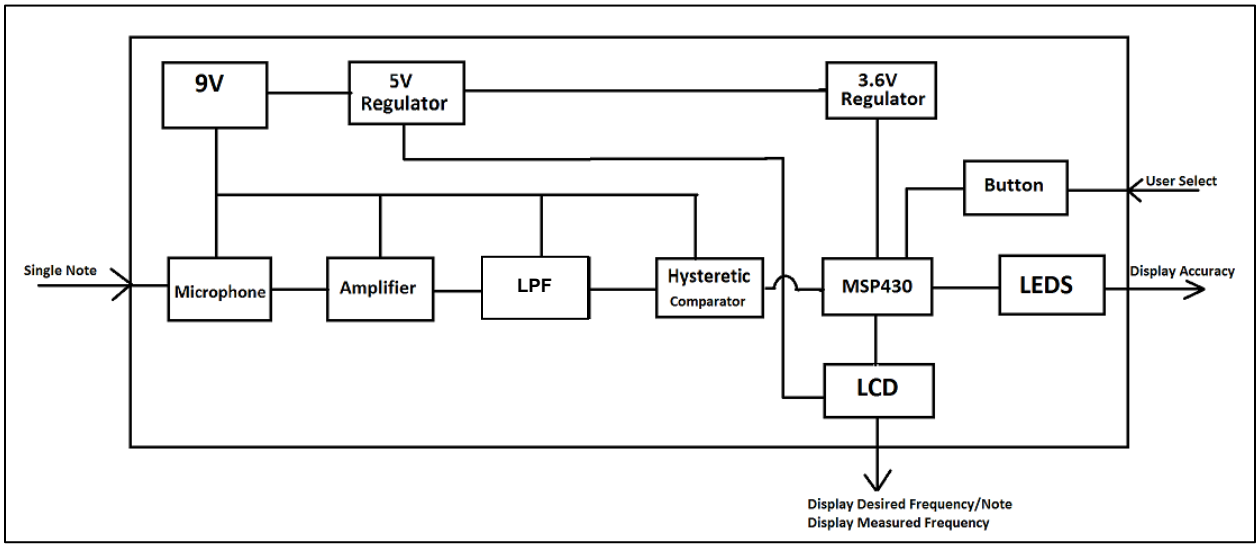


Figure 2: Cal Bass Tuner level 1 block diagram

Component Design

The design of the Cal Bass Tuner consists of a software portion where the MSP430 microcontroller is used to count the frequency of the signal processed by the hardware portion of the system. First, the hardware design of the system will be discussed and the expected signal going to the MSP430 will be described. The software portion of the system is then discussed, where the section describes the pseudo code used to analyze the signal processed by the hardware portion of the system.

Hardware

The hardware of the MSP430 consists of three main stages that include an amplification stage, a filtering stage, and a comparator stage. In addition, the design of the voltage regulators that provide appropriate voltages to the various components will be discussed. Figure 3 demonstrates the schematic of the three main stages associated with the hardware portion of the system design.

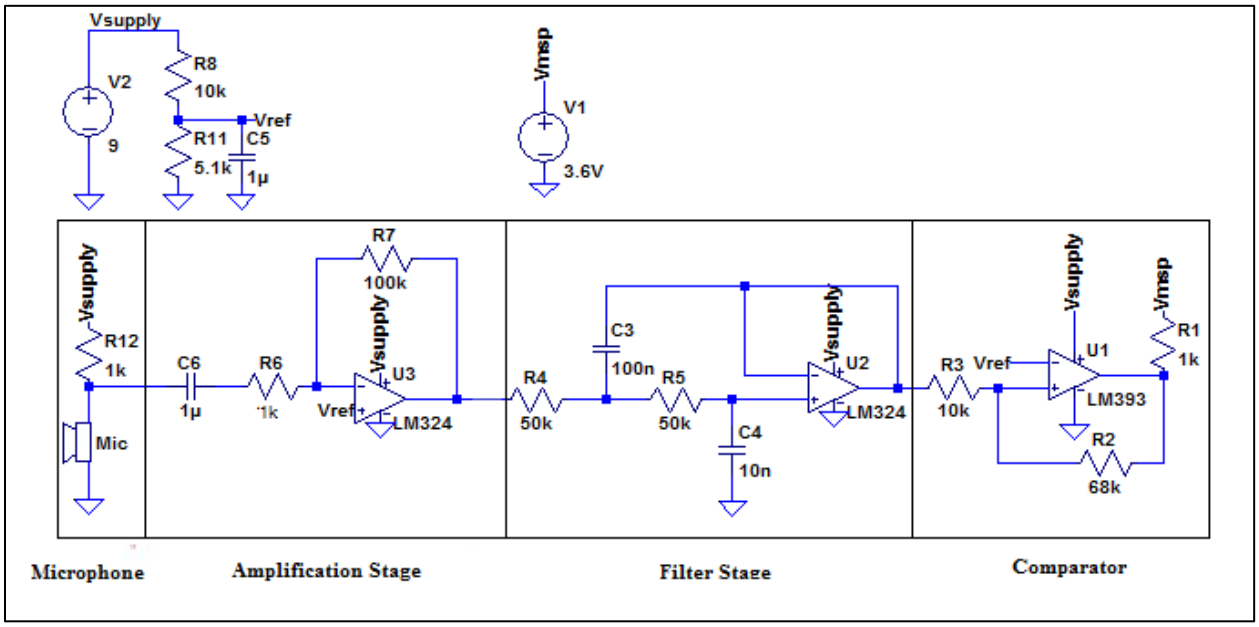


Figure 3: Cal Bass Tuner Hardware Schematic

Microphone:

The first step to the Cal Bass Tuner design was to assemble the microphone circuit based on the design specified on the condenser microphone data sheet. The condenser microphone is a two terminal device, where one node is grounded and the other is the output and uses a 1kΩ pull up resistor to V_{supply}, as specified on the condenser microphone data sheet [2]. After the circuit was assembled data was taken to determine the expected waveform of the output. Figure 4 demonstrates the measured waveform when the G note (≈98Hz) is played.

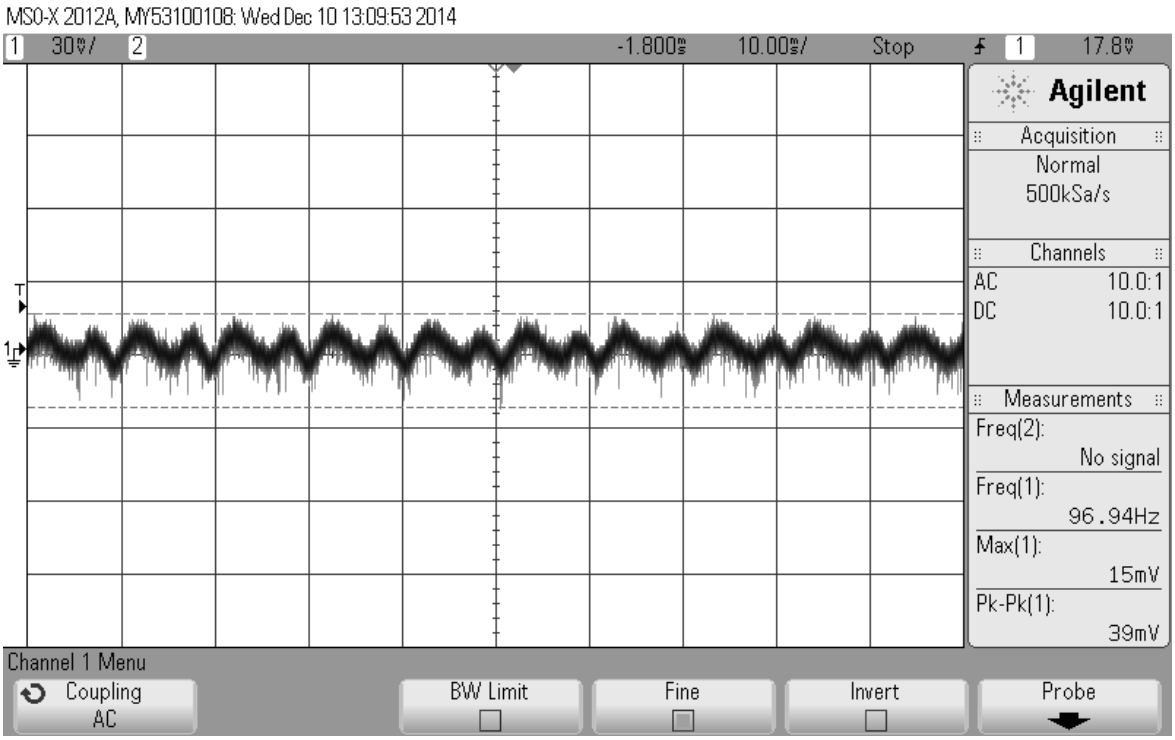


Figure 4: Expected Amplitude of signal when G note is played

Amplification Stage:

After determining the expected input signal, amplification is required to bring the voltage to a desirable level that is easier to process. To accomplish this task an inverting amplifiers is used with a gain of 100. Since we are only concerned with the frequency of the signal, inverting the waveform will not cause any detrimental effects. Equation 1 shows the transfer function of an inverting amplifier and therefor the equation used to choose the values of R7 and R6 of figure 3. In addition, the positive terminal of the op-amp is offset by 3V to allow the op-amp to operate on a single 9V supply without clipping the output. This is accomplished via a resistive divider as shown in figure 3.

Equation 1: $$A_v = -\frac{R_7}{R_6} = -\frac{100k\Omega}{1k\Omega} = -100$$

Figure 5 demonstrates the output of the amplification stage. When a note is played on the bass guitar at first the sound is loud then it slowly attenuates. This translates to voltage levels proportional to the loudness of the sound. Thus, in figure 5 we see a 5 V_{pp} wave as opposed to the expected 4V_{pp} calculated previously. The signal is still great enough to interpret and therefore no disastrous effects will occur. In addition, the

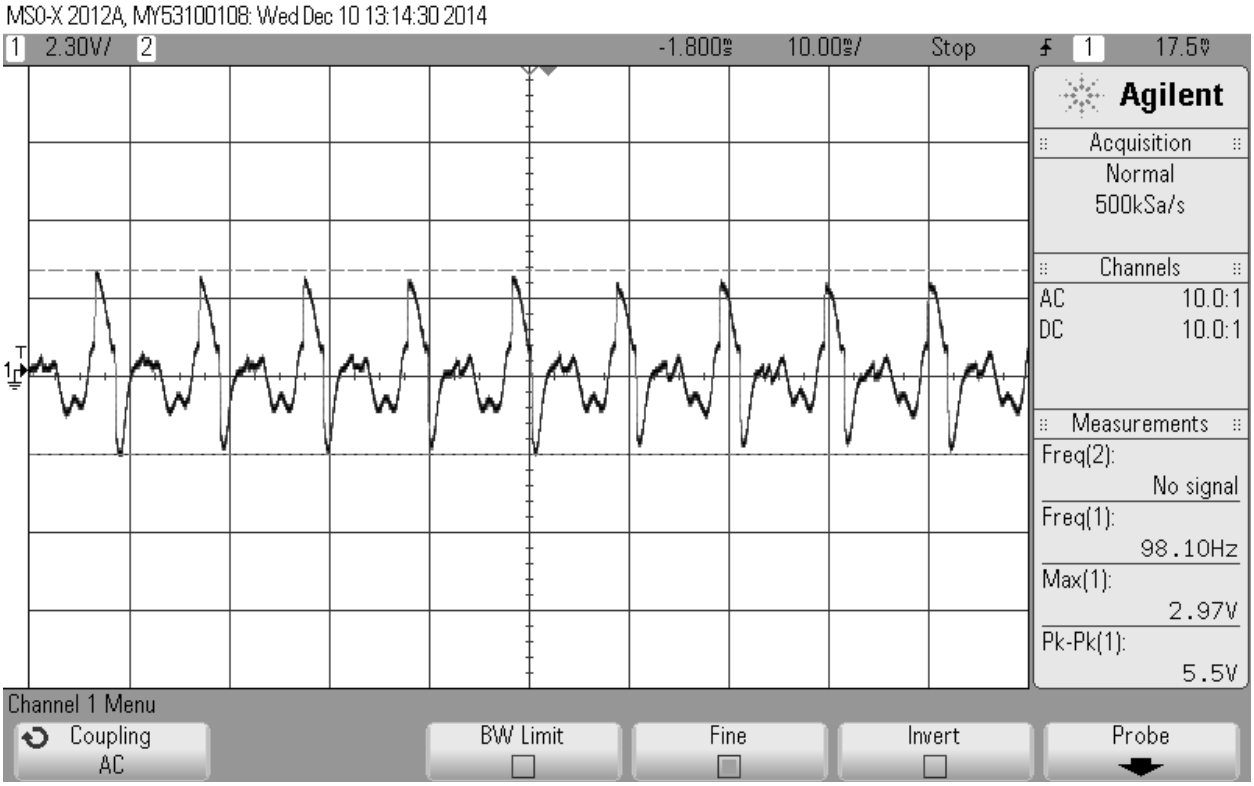


Figure 5: Signal after the amplification stage when the G note is played

Filter Stage:

The signal of figure 5 has the fundamental frequency of the 98 Hz signal associated with the G note; however, unwanted harmonics affect the signals sinusoidal quality. Harmonics are expected at integer multiples of the fundamental frequency, therefore a low pass filter with a cut-off frequency of approximately 100 Hz is required to block these frequencies.

A sallen-key low pass filter is used to accomplish the task of filtering. The sallen-key low pass filter offers a cut off of 40dB per decade allowing any frequencies above 100Hz to be attenuated by a factor greater than a passive low pass filter, which only offers a cut off of 20dB per decade. Equation 2 is used to determine the cut-off frequency of the sallen-key low pass filter, where R4 and R5 are chosen to be equivalent. Figure 6 demonstrates the output of the low-pass filter stage when integrated with the amplification stage.

Equation 2: $$f_c = \frac{1}{2\pi\sqrt{C_3 C_4 R_4 R_5}} = \frac{1}{2\pi\sqrt{(100nF)(10nF)(50k\Omega)(50k\Omega)}} = 101Hz$$

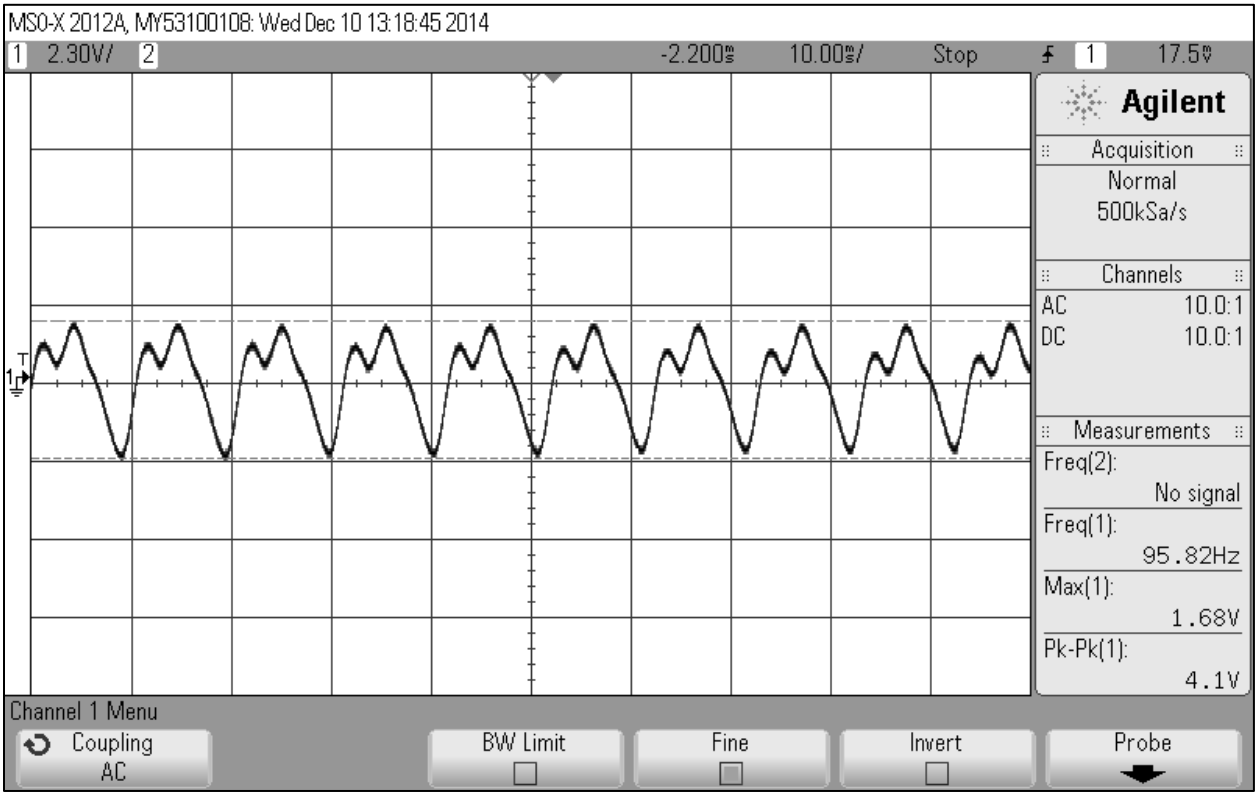


Figure 6: Signal after the filtering stage when the G note is played

Comparator:

The last stage to the signal processing is converting the analog waveform of figure 6 to a square wave that may be easily interfaced with the MSP430. A comparator is used to accomplish the conversion. Since the expected signal is relatively slow (100 Hz and below), we can expect there to be noise that may couple to the signal. This noise can cause a traditional comparator to chatter (or oscillate) at the output, since voltages close to the threshold may be misinterpreted due to the noise. Instead, a hysteretic comparator is used to give the comparator two thresholds.

Below equations 3 and 4 are used to set the thresholds to 3.4V and 2.9V. The LM393 comparator has an open collector at the output and therefore requires a 1k pull up resistor as specified by the data sheet [3]. The resistor is pulled up to the MSP430's 3.6V supply to avoid overdriving the input pins, since the signal is expected to be interfaced with a port on the MSP430. Figure 7 demonstrates the output of the hysteretic comparator, and thus the overall hardware processing portion of the system.

Equation 3: $$V_{TH} = \left(1 + \frac{R_2}{R_3}\right) x V_{ref} - \frac{R_2}{R_3} x V_{OL} = \left(1 + \frac{10k}{68k}\right) x(3V) - \frac{10k}{68k} x(0V) = 3.44V$$

Equation 4: $$V_{TL} = \left(1 + \frac{R_2}{R_3}\right) x V_{ref} - \frac{R_2}{R_3} x V_{OH} = \left(1 + \frac{10k}{68k}\right) x(3V) - \frac{10k}{68k} x(3.6V) = 2.91V$$

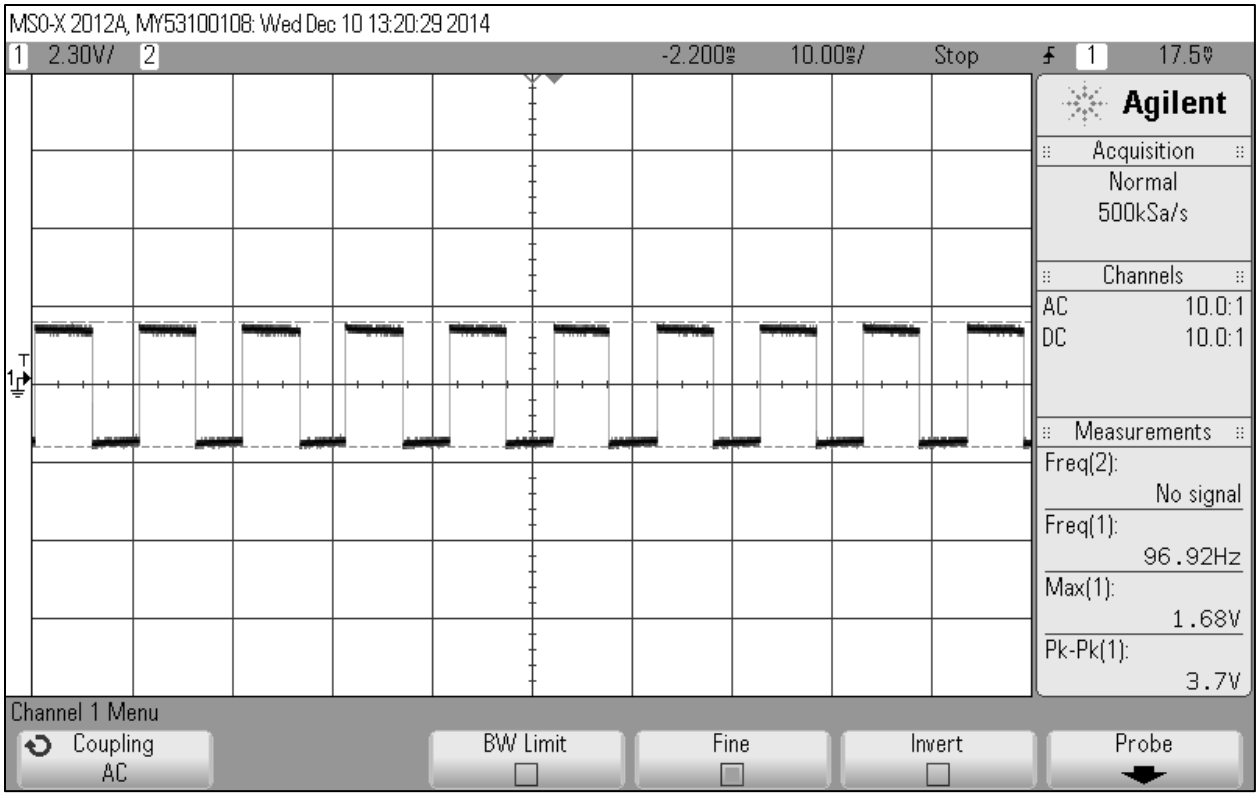


Figure 7: Signal after the comparator stage when the G note is played

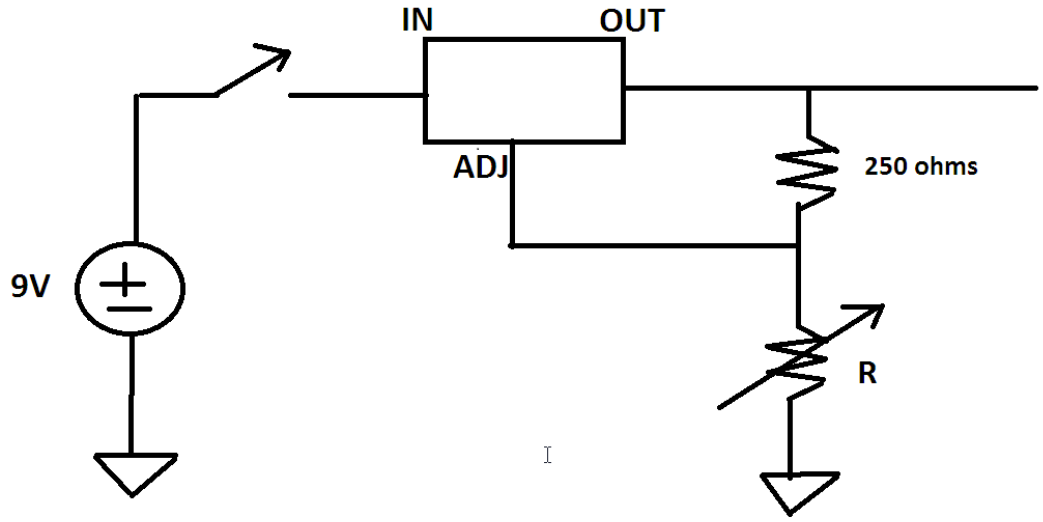


Figure 8: Voltage Regulator Functionality LM317LZ

Voltage Regulator Sample Calculations:

$$\text{Vout} = \left(1.25\text{x}\frac{R}{250}\right) + 1.25, \quad V_{outmin} = 1.25\text{V}$$

For,

Vout= 5V then R= 470Ω

Vout= 3.6V then R=750Ω

Software

- The functions watchDog, clockSet, and buttonInterruptEnable sent the clock speed and interrupt setting of the MSP430.
- The setLEDS function sets up the pins used to run the LEDS.
- The functions initialize_LCD, read_flag, write, write_char, write_word, and clear_disp handle all of the functions of running the LCD.
- The show_param function is the main part of the program that displays information on the LCD.
- The itoa function takes an integer and turns it into the ascii code for that integer.
- The diffFrq function takes in the frequency of the signal and compares it to three different levels of accuracy to the fundamental frequency. If the frequency is within 25% but not within 10% of the fundamental frequency a red LED lights up. If the frequency is within 10% but not within 5% of the fundamental frequency an yellow LED lights up. But if the frequency is within 5% a green LED lights up.
- The main function runs the startup functions and enables the interrupts for the three different ISRs. It also have a built in delay statement that runs each time one of the lower frequencies has an interrupt occur. This delay statement is used to remove some of the harmonics that the signal can get from the band pass filter.
- The main functionality of this program is within the three ISRs.
- The Timer_A ISR interrupts every quarter of a second. It runs the functions that display the information on the LCD screen and runs the diffFrq function.
- The PORT1_VECTOR ISR counts the rising edges of the incoming square ware that is the frequency of the sound the microphone picked up.
- The PORT2_VECTOR ISR checks to see if the button has been pushed. If the button has been pushed it changes the program to search for a different frequency.
- The flowchart in **Figure 8** shows the top-level design for the main routine.

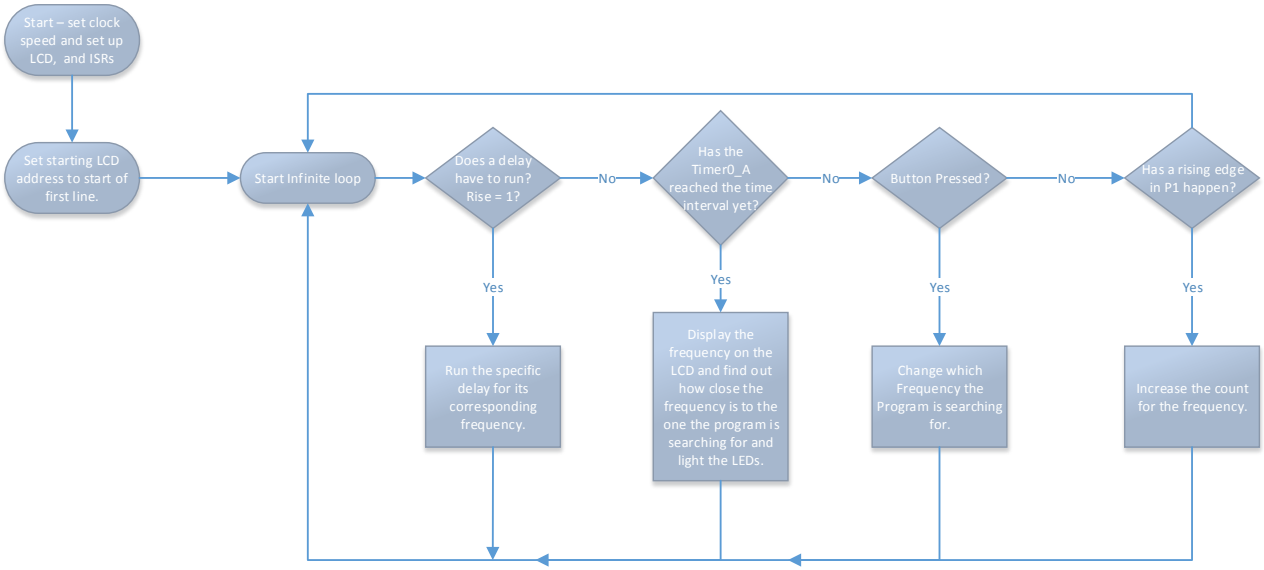


Figure 8: Flow Diagram for Main Routine

Bill of Materials

Table 4 shows the parts used in the design and the cost of the parts.

Table 4: Bill of Materials

Part #	Part Name	Description	Qty	Unit Cost	Cost
	MSP430G2553	MSP430 Microcontroller	1	$ 9.99	$ 9.99
LCM-SO1602DTR/M	PmodCLP	Parallel LCD Display	1	$ 9.99	$ 9.99
COM-09190 ROHS	Square Push Button Switch	Push Button 12mm	1	$ 0.50	$ 0.50
	50k Resistor	50k Ohm 1/4W	2	$ 0.10	$ 0.20
	1K Resistor	1k Ohm 1/4W	3	$ 0.10	$ 0.30
	5.1k Resistor	5.1K Ohm 1/4W	1	$ 0.10	$ 0.10
	68k Resistor	68k Ohm 1/4W	1	$ 0.10	$ 0.10
CFR-50JB-52	10K Resistor	10K Ohm 1/4W	8	$ 0.10	$ 0.80
352	Prototyping Breadboard		1	$ 4.95	$ 4.95
	Capactior 10nF	10nF	1	$ 0.25	$ 0.25
	Capacitor 1uF	1uF	2	$ 0.25	$ 0.50

	LM393	Low Power Low Offset Voltage Dual Comparator	1	$ 1.75	$ 1.75
MLM324	LM324	Low Power Quad Operational Amplifier	1	$ 1.74	$ 1.74
270-0090	PC-Mount Condenser Microphone Element	PC-Mount Condenser Microphone Element	1	$ 2.99	$ 2.99
431397930l06	MN1604B1	9V Duracell Battery	1	$ 3.59	$ 3.59
279-1851	8-Pin Retention Contact	8-Pin Retention Contact	1	$ 0.49	$ 0.49
279-1991	20-Pin Retention Contact	20-Pin Retention Contact	1	$ 0.49	$ 0.49
279-1925	14-Pin Retention Contact	14-Pin Retention Contact	2	$ 0.49	$ 0.98
	LM317	Voltage Requator	2	$ 0.49	$ 0.98
	Battery Holder	Battery Holder	1	$ 1.00	$ 1.00
276-010	5mm Yellow LED	5mm Yellow LED	1	$ 1.99	$ 1.99
276-209	5mm Red LED	5mm Red LED	1	$ 1.99	$ 1.99
276-022	5mm Green LED	5mm Green LED	1	$ 1.99	$ 1.99
	Total		**35**		**$ 47.66**

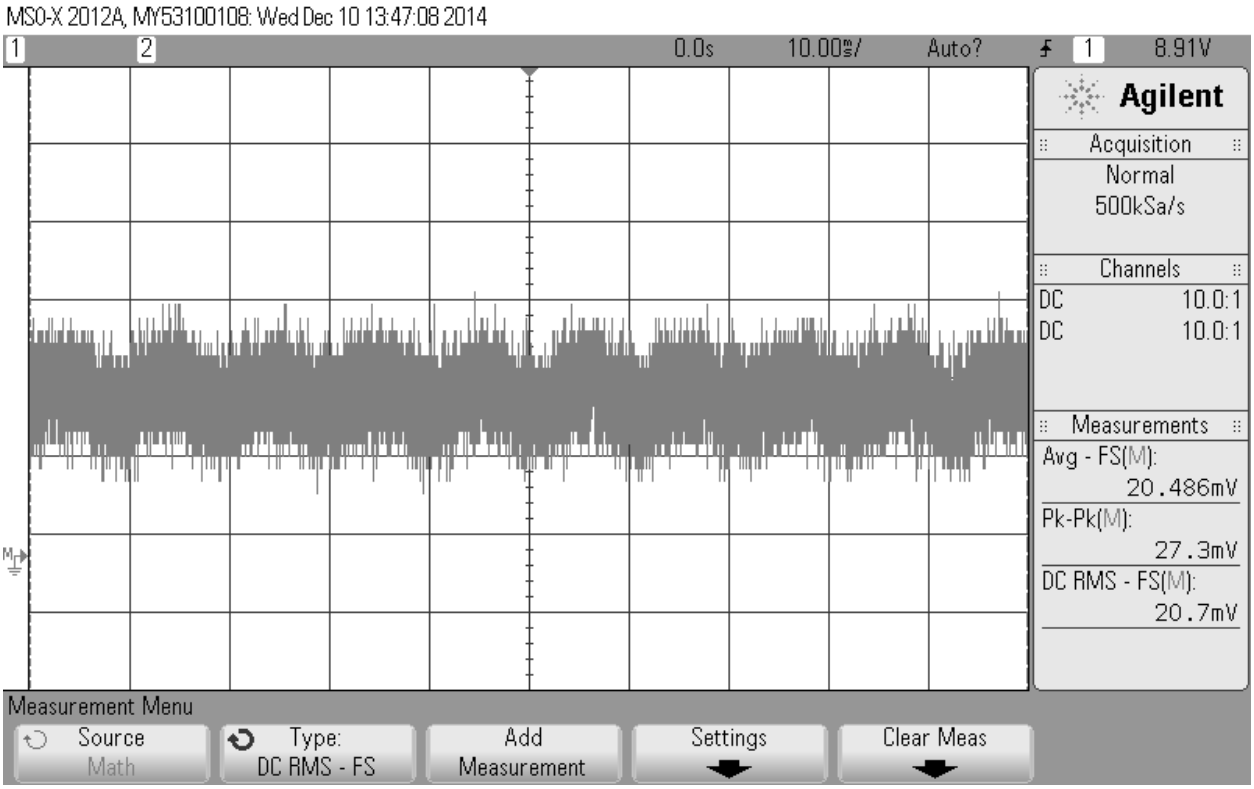


Figure 9: Voltage across 1Ω output resistor from Battery supply

Sample calculation Iav and Power in :

$I_{av} = \frac{V_{test}}{1\Omega} = 20.7mA$

$P_{in} = V_{in} \times \mathrm{I_{av}} = 9 \times 0.0207 = 186.3\mathrm{mW}$

Table5: Power Dissipated in the System

Device/Component	Current (mA)	Voltage (V)	Power (mW)
MSP430	0.4	3.6	1.44
Microphone	1	9	9
LM324	4	9	36
R1	1.5	3.6	5.4
Isupply	0.6	9	5.4
Comparator LM393	3	9	27
LCD	2	5	10
LED	3.404	2	6.8
LM317LZ	3.5	9	31.5
Total:	19.404	9V	132.54

Sample Calculations

Isupply=Vsupply/(R8+R11)= 9/(15.1K)=0.6mA , Vsupply = 9V

$\mathrm{P_{out_{supply}}} = \mathrm{VxI} = 1.44\mathrm{mW}$

$\mathrm{n_{efficiency}} = \frac{P_{out}}{Pin} = \frac{132.54mW}{186.3mW} = 0.71$

$Battery_{Life} = \frac{Battery\ Capacity\ (AH)}{I_{av}} = \frac{13000mAH}{10.7mA} = 1214.9\ hours$

System Integration

1. The process to build this system started with reading the data sheets and the Project 45 the Whole Enchilada, pdfs.
2. Next the schematic of the connections between the LM324, MSP430, LM393, LEDs, LCD, Push Button, 9V battery, Microphone, and the voltage regulators was developed.
3. The flow charts for the software were then completed before the coding was started.
4. The LM324, LM393, voltage regulators, and 9V battery were then connected together. This took some checking and rechecking to make sure it was connected correctly. Most of the testing for this project was done at this point.
5. Then the LCD and microphone were connected to the other components. Tests were executed to determine the expected signal was being output by the amplifier, LPF, and comparator.

6. The code for the program was then completed.
7. One bug found was a transmission gate CD4016 connected to the circuit and for some reason the whole circuit would not work with it connected. The circuit had to be redesign to work without it.
8. Another bug was when the battery was connected to the circuit the battery gave off a 100Hz frequency that the microphone picked up. A capacitor was added to the design to eliminate the problem.

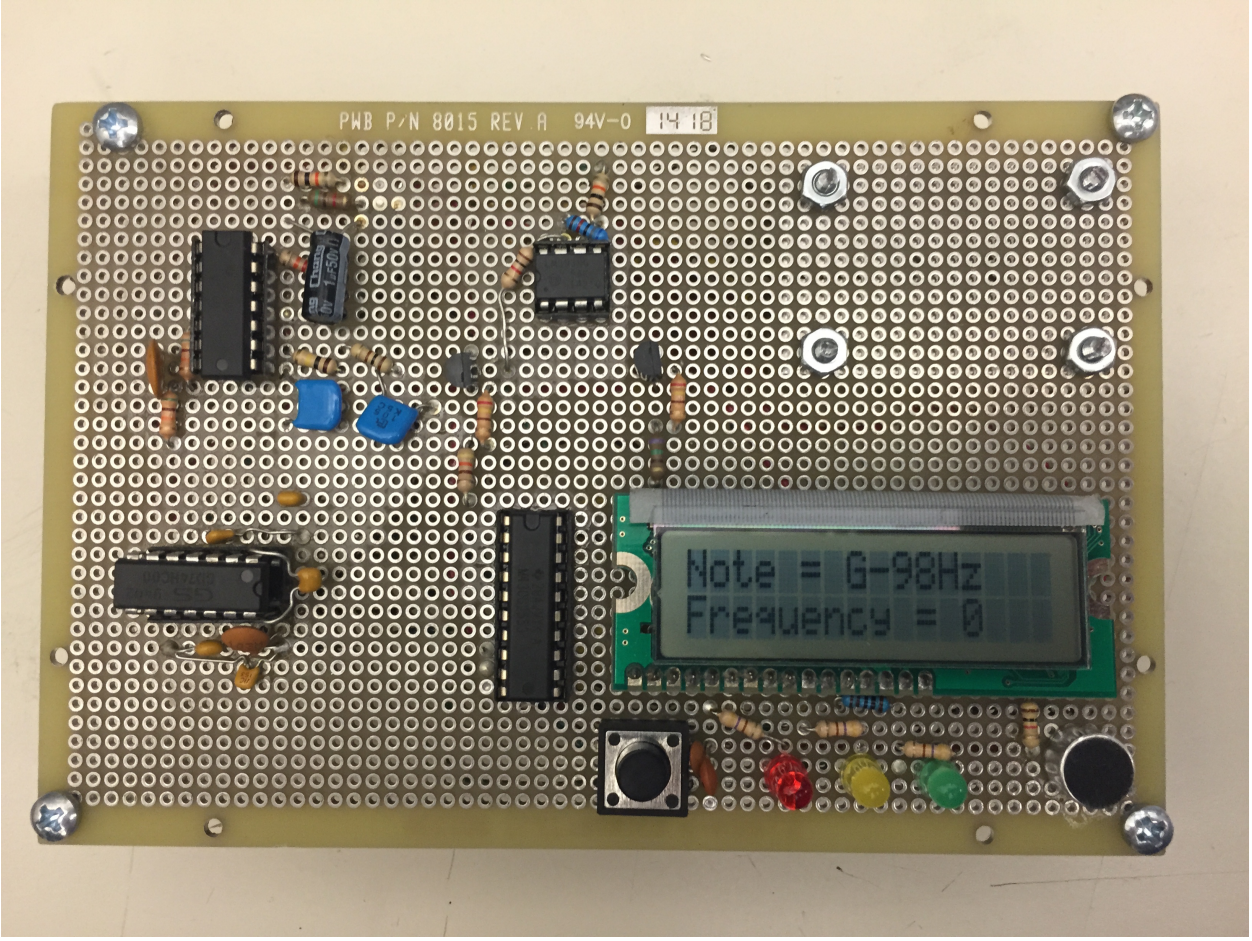


Figure 6: Picture of connected Bass Guitar Tuner Circuit

Conclusion

The overall project was a success as we were able to meet the requirements set by the project proposal. Most importantly, the user is able to tune a 4-string bass guitar to an accuracy of 95%. Though the project was a success, improvements can be made to increase quality and performance for the user.

The hardware employed a simple technique of processing the acoustic wave sensed by the condenser microphone. To achieve a more desirable signal, techniques may be used to better filter the fundamental frequencies of the bass guitar. Our design employs one filter that removes harmonics above 100 Hz, the frequency of the signal is

then measured by the MSP430 based on the selected note. A better technique that would yield a more appropriate fundamental frequency would be to isolate the fundamental frequency in hardware, so that the wave coming into the MSP430 would not need to be re-processed in code. In addition, more sophisticated techniques could also be applied. There exist IC packages that are able to perform FFT operations. Using these embedded systems we could have isolated the fundamental frequency without the need of any sort of filtering. This would decrease cost of our system, since it would require fewer components, and would also decrease power consumption. For future reference, more research will be done on implementing IC's containing embedded systems that can aide in our project.

One major problem we came across in writing the code for this project was how to make sure the code was counting the rising edges of the frequency accurately and efficiently. We had to find a time length when the count of the frequency would be accurate. It had to be long enough to get a good count but not too long, because the base string would start to distort the frequency. We ended up using a time length of a quarter of a second to get the frequency sample. Another problem was that even through the LPF we were still getting some harmonics. We had to get rid of them, so we put a delay in the code to pass over these rising edges and not count them towards our frequency.

The system is fully functional and may be used to tune a 4 string bass guitar with open string notes A, D, G, and E. The system provides an easy to reference LED array and enables the user to quickly tune their bass guitar to 95% accuracy. Overall the Cal Bass Tuner prototype is a building block to a system that may be improved on to achieve greater accuracies and a better experience for the user. The project met its requirements and our team is pleased with the final result.

References

[1] Bass Guitar Frequency Spectrum - http://www.bass-guitar-info.com/FrequencySpectrum.html

[2] Condenser Microphone Datasheet - http://www.datasheetarchive.com/dlmain/Datasheets-24/DSA-475102.pdf

[3] LM393 Comparator Datasheet - http://www.ti.com/lit/ds/symlink/lm393.pdf

[4] MSP430G2553 Datasheet - http://www.ti.com/lit/ds/symlink/msp430g2553.pdf

[5] PmodCLP LCD Datasheet - http://www.digilentinc.com/Data/Products/PMOD-CLP/PmodCLP_rm_RevA.pdf

[6] LM324 Datasheet http://www.ti.com/lit/ds/symlink/lm124-n.pdf

[7] LCD Display - LCM-s01602DTR+M https://polylearn.calpoly.edu/AY_2014-2015/mod/resource/view.php?id=43673

[8] LCD datasheet KS0066U https://polylearn.calpoly.edu/AY_2014-2015/mod/resource/view.php?id=43674

Appendix A – Code

```
#include <msp430g2553.h>

#define MSB 0xF0
#define LSB 0x0F


//Interrupt Buttons
#define freqButton BIT2

//Frequences of Notes
#define Freq_G_100HZ 98
#define Freq_D_75HZ 74
#define Freq_A_55HZ 55
#define Freq_F_41HZ 41

//Input Frequency
#define Freq_In BIT5

//LED PINS
#define LED_R BIT4
#define LED_Y BIT0
#define LED_G BIT1

//Clock function
void watchDog();
void clockSet();

//Timer A interrupt function
void TimerSet();

//Button and Frequency Interrupt function
void buttonInterruptEnable();

//Set up of LED pins
void setLEDS();

//Function that checks how close the
//incoming frequency is to the frequency
//that the program is set to find.
void diffFrq(int);

//Functions to run the LCD
void initialize_LCD();
void read_flag();
void write(char);
void write_char(char);
void write_word(char*);
void clear_disp();
void show_param(int);
void itoa();

//Constants
```

```
volatile unsigned int ITimer = 3906;
volatile unsigned int count = 0;
volatile int func = 0;
volatile int rise = 0;

int main(void)
{
    watchDog();
    clockSet();
    initialize_LCD();
    Transgate();
    setLEDS();
    TimerSet();
    buttonInterruptEnable();
    P1OUT |= Trans100;

    _enable_interrupts();

    while(1){
             if (rise == 1){
                    _disable_interrupts();
                    if (func == 1){
                           __delay_cycles(1188);
                    }
                    else if (func == 2){
                           __delay_cycles(1925);
                    }
                    else if(func == 3){
                           __delay_cycles(2250);
                    }
                    rise = 0;
                     _enable_interrupts();
             }
    }
}



/*******************************
 * Writes a char array to LCD   *
 *******************************/
void write_word(char *phrase)
{
    int i=0;

    while(phrase[i] != '\0')
    {
        write_char(phrase[i++]);
    }
}

/*******************************
 * Writes character to LCD      *
 *******************************/
void write_char(char letter)
```

```
{
    char temp = 0x00;

    temp =  (letter >> 4) & LSB;
    temp |= BIT4;
    write(temp);

    temp = letter & LSB;
    temp |= BIT4;
    write(temp);

    read_flag();

}

/*****************************
 * Writes data to lcd        *
 *****************************/
void write(char data)
{
    //Set data outputs
    P1DIR |= (LSB);

    //output data
    P1OUT &= ~LSB;
    P1OUT |= data & LSB; //Output data bits

    P2OUT &= ~BIT4;
    P2OUT |= (BIT4 & data) >> 1;//Output RS Bits


    //Enable & Output data
    P2OUT |= BIT5; //Set Enable

    P2OUT &= ~BIT5;//Lower Enable
}

void show_param(int freq)
{
       char temp[3];

       clear_disp();

       if (func == 0){
              write_word("Note = G-98Hz");
       }
       else if (func == 1){
              write_word("Note = D-73Hz");
       }
       else if (func == 2){
              write_word("Note = A-55Hz");
       }
       else{
              write_word("Note = E-41Hz");
       }
```

```
      write(0x0C);
      write(0x00);
      read_flag();


      write_word("Frequency = ");
      itoa(freq, temp);
      write_word(temp);

 }

/*****************************
 * Reads flag generated by LCD *
 *****************************/
void read_flag(void)
{
    P1DIR &= ~LSB;//Set lower bits as inputs

    //busy flag(Bit7), AC Bits(Bit6-Bit0)
    char flag = 0x00;//used to read busy flag

    //read flag til busy flag is cleared
    do{

        flag = 0x00;//reset busy flag

        P2OUT &= ~BIT3;//Clear RS

        P2OUT |= BIT4;//Set R/W

        P2OUT |= BIT5; //Set Enable

        flag = P1IN;//read flag + AC

        P2OUT &= ~BIT5;//Lower Enable

        //Second instruction for reading busy flag
        P2OUT &= ~BIT3;
        P2OUT |= (BIT4);//Output R/W

        P2OUT |= BIT5; //Set Enable

             flag &= BIT3;

        P2OUT &= ~BIT5;//Lower Enable

    }while(flag >> 3);

    return;
}

void clear_disp(void)
{
    write(0x00);
```

```
    write(0x01);
    read_flag();
}

/*****************************
 * Initialize LCD             *
 *****************************/
void initialize_LCD(void)
{
       //Set Control Outputs
       //BIT3(RS) BIT4(R/W) BIT5(E)
       P2DIR |= (BIT3 + BIT4 + BIT5);
       P2OUT &= ~(BIT3 + BIT4 + BIT5);

    //Function Set
    __delay_cycles(500000); //Wait for more than 30ns
    write(0x02);
    write(0x02);
    write(0x08);

    //Display ON/OFF Control
    __delay_cycles(800); // wait for more than 39us
    write(0x00);
    write(0x0C);

    //Display Clear
    __delay_cycles(800); // wait for more than 39us
    write(0x00);
    write(0x01);

    //Entry Mode Set
    __delay_cycles(50000); //wait for more than 1.54ms
    write(0x00);
    write(0x06);
    read_flag();

    return;

}

/*****************************************
Modified:_PROTOTYPE( char *itoa, (int n));*
*****************************************
* The code below is a modified version of minix: lib/other/itoa.c
* intended to store the input int n into a buffer of size 8 bytes;
*
* VAR n = number to conver to ASCII Char
* VAR qbuf = buffer[8] to store input number. Supports up to 7 chars long
*
* Caution: Does a max of 14 divides and 7 multiplies (hopefully done with
* a hardware multiplier)
*/
void itoa(int n,char *qbuf)
{
    volatile int r, k;
```

```
    volatile char flag = 0;
    volatile char next = 0;

    if(n < 0)
    {
        qbuf[next++] = '-';
        n = -n;
    }

    if(n == 0)
    {
        qbuf[next++] = '0';
    }
    else
    {
        k = 10000;
        while(k > 0)
        {
            r = n / k;

            if(flag || r > 0)
            {
                qbuf[next++] = '0' + r;
                flag = 1;
            }

            n -= r * k;
            k = k / 10;
        }
    }

    qbuf[next] = '\0';
}

/*****************************
 * Stop Watchdog Timer       *
 *****************************/
void watchDog(){
    WDTCTL = WDTPW | WDTHOLD;   }

/*****************************
 * Set Clock Frequency       *
 *****************************/
void clockSet(void)
{
      // 16Mhz SMCLK
      if (CALBC1_16MHZ==0xFF)         // If calibration constant erased
      {
                  while(1);               // do not load, trap CPU!!
      }

   DCOCTL = 0;
   BCSCTL1 = CALBC1_1MHZ;
   DCOCTL = CALDCO_1MHZ;
   BCSCTL2 |= (BIT1 + BIT2);
```

```
}

/*****************************
 * Timer Configuration    *
 *****************************/
void TimerSet(void)
{
    CCTL0 = CCIE;    // CCR0 interrupt enabled
    CCR0 = ITimer;   // Set the interrupt to 160 cycles.
    TACTL = TASSEL_2 + MC_1 + ID_3;   // SMCLK, contmode
}


/*****************************
 * Interrupt Initialization   *
 *****************************/
void buttonInterruptEnable()
{
      //Enable Button used as interrupt
              P1DIR &= ~Freq_In;
              P1IE |= Freq_In;   // Enable interrupts
              P1IES &= ~Freq_In; // Select low to high edge Interrupts
              P1IFG &= ~Freq_In; // Clear the interrupt flags to ensure system

              P2DIR &= ~freqButton;
              P2IE |= freqButton;      // Enable interrupts
              P2REN |= freqButton;     // Sets a pull-up resistor
              P2IES |= freqButton;     // Select high to low edge Interrupts
              P2IFG &= ~freqButton;    // Clear the interrupt flags to ensure system
     // starts with no interrupts
}
/*****************************
 * LEDs Initialization   *
 *****************************/
void setLEDS(){
      P1DIR |= LED_R;
      P1OUT &= ~LED_R;
      P2DIR |= LED_Y + LED_G;
      P2OUT &= ~(LED_Y + LED_G);
}
/***********************************
 *Timer A0 interrupt service routine*
 ***********************************/
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
      _disable_interrupts();

      if (func == 0){
                  show_param(count << 2);
                  diffFrq(count << 2);
            }
            else if (func == 1){
                  show_param(count << 1);
                  diffFrq(count << 1);
```

```
            }
            else if (func == 2){
                  show_param(count << 1);
                  diffFrq(count << 1);

            }
            else{
                  show_param((count << 2)/3);
                  diffFrq((count << 1)/3);
            }

            count = 0;

      _enable_interrupts();
}

/*********************************
 *Port1 interrupt service routine  *
 *********************************/
#pragma vector = PORT1_VECTOR
__interrupt void PORT1(void)
{
      _disable_interrupts();
      count++;
      rise = 1;
      P1IFG &= ~Freq_In; // Clear the interrupt flags to ensure system
      _enable_interrupts();
}
/*********************************
 *Port2 interrupt service routine  *
 *********************************/
#pragma vector = PORT2_VECTOR
__interrupt void PORT2(void)
{
      if (func == 3){
            func = 0;
      }
      else{
            func++;
      }
      P2IFG &= ~freqButton; // Clear the interrupt flags to ensure system

}
/*******************************
 * Function that checks for     *
 * the difference between the   *
 * frequency from the square    *
 * wave and the frequency that  *
 * the program is searching for *
 *******************************/
void diffFrq(int fq){
      if (func == 0){//100
            if ((fq >= 74 && fq < 91) || (fq > 108 && fq <= 123)){
                  P1OUT |= LED_R;
                  P2OUT &= ~(LED_Y + LED_G);
```

```
            }
            else if ((fq >= 91 && fq < 93) || (fq > 103 && fq <= 108)){
                  P2OUT |= LED_Y;
                  P1OUT &= ~LED_R;
                  P2OUT &= ~LED_G;
            }
            else if (fq >= 93 && fq <= 103){
                  P2OUT |= LED_G;
                  P2OUT &= ~LED_Y;
                  P1OUT &= ~LED_R;
            }
            else{
                  P2OUT &= ~(LED_Y + LED_G);
                  P1OUT &= ~LED_R;
            }
      }
      else if (func == 1){//75
            if ((fq >= 55 && fq < 68) || (fq > 82 && fq <= 91)){
                  P1OUT |= LED_R;
                  P2OUT &= ~(LED_Y + LED_G);
            }
            else if ((fq >= 68 && fq < 70) || (fq > 76 && fq <= 82)){
                  P2OUT |= LED_Y;
                  P1OUT &= ~LED_R;
                  P2OUT &= ~LED_G;
            }
            else if (fq >= 70 && fq <= 76){
                  P2OUT |= LED_G;
                  P2OUT &= ~LED_Y;
                  P1OUT &= ~LED_R;
            }
            else{
                  P2OUT &= ~(LED_Y + LED_G);
                  P1OUT &= ~LED_R;
            }
      }
      else if (func == 2){//55
            if ((fq >= 42 && fq < 50) || (fq > 60 && fq <= 68)){
                  P1OUT |= LED_R;
                  P2OUT &= ~(LED_Y + LED_G);
            }
            else if ((fq >= 50 && fq < 52) || (fq > 58 && fq <= 60)){
                  P2OUT |= LED_Y;
                  P1OUT &= ~LED_R;
                  P2OUT &= ~LED_G;
            }
            else if (fq >= 52 && fq <= 58){
                  P2OUT |= LED_G;
                  P2OUT &= ~LED_Y;
                  P1OUT &= ~LED_R;
            }
            else{
                  P2OUT &= ~(LED_Y + LED_G);
                  P1OUT &= ~LED_R;
            }
```

```
        }
        else if (func == 3){//41
                if ((fq >= 31 && fq < 37) || (fq > 45 && fq <= 51)){
                        P1OUT |= LED_R;
                        P2OUT &= ~(LED_Y + LED_G);
                }
                else if ((fq >= 37 && fq < 39) || (fq > 43 && fq <= 45)){
                        P2OUT |= LED_Y;
                        P1OUT &= ~LED_R;
                        P2OUT &= ~LED_G;
                }
                else if (fq >= 39 && fq <= 43){
                        P2OUT |= LED_G;
                        P2OUT &= ~LED_Y;
                        P1OUT &= ~LED_R;
                }
                else{
                        P2OUT &= ~(LED_Y + LED_G);
                        P1OUT &= ~LED_R;
                }
        }
}
```