Microcontroller Instrument Tuner

Overview:

Using the Texas Instruments MSP432 microcontroller and a BOOSTEDXL peripheral board, I constructed a digital instrument tuner. Using UART communication, the tuner allows the user to select a desired pitch with which their audio input shall be compared. The 6 selectable pitches are the 6 strings of a guitar in standard tuning: E, A, D, G, B, E. Keys 1 through 6 on the user's keyboard allow for selection between these standard pitches. The BoostedXL board was used purely for its local microphone. The rest of the processing is done on the MSP board itself. Shown below are system block diagrams further illustrating the configuration.

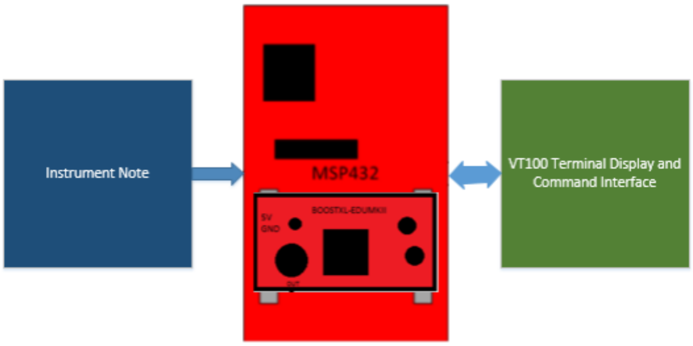


Figure 1: Block Diagram

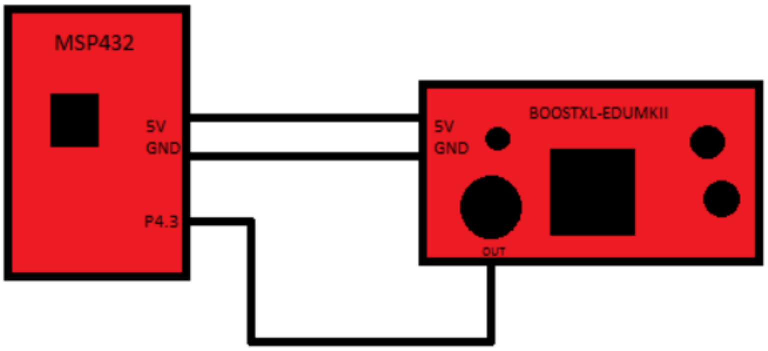


Figure 2: Wiring Schematic for Instrumental Tuner

Since the BOOSTEDXL was designed as a peripheral for the MSP 432 board, interfacing the two was very simple. The BOOSTEDXL plugs directly onto the 4 columns of leads located at the center of the MSP 432. This means that the BOOSTEDXL is powered so long as the MSP 432 itself is powered. Once secured onto the MSP 432, the BOOSTEDXL's microphone automatically samples audio and sends the transduced voltages to pin 4.3 on the MSP 432.

The MSP 432 takes in voltage samples from the microphone and stores them in an array of length 10,000. This is done using an ADC interrupt. Once all the voltage samples are accounted for, their average is calculated. Next, the code iterates through the array once more to count how many times the input waveform crosses its own average. This number is directly proportional to the frequency, so it is multiplied by the ratio of one second to the time the MSP 432 needs to gather 10,000 samples. This results in a frequency measurement that is accurate within 5 Hz. The sequential software process is shown below in figure 3.

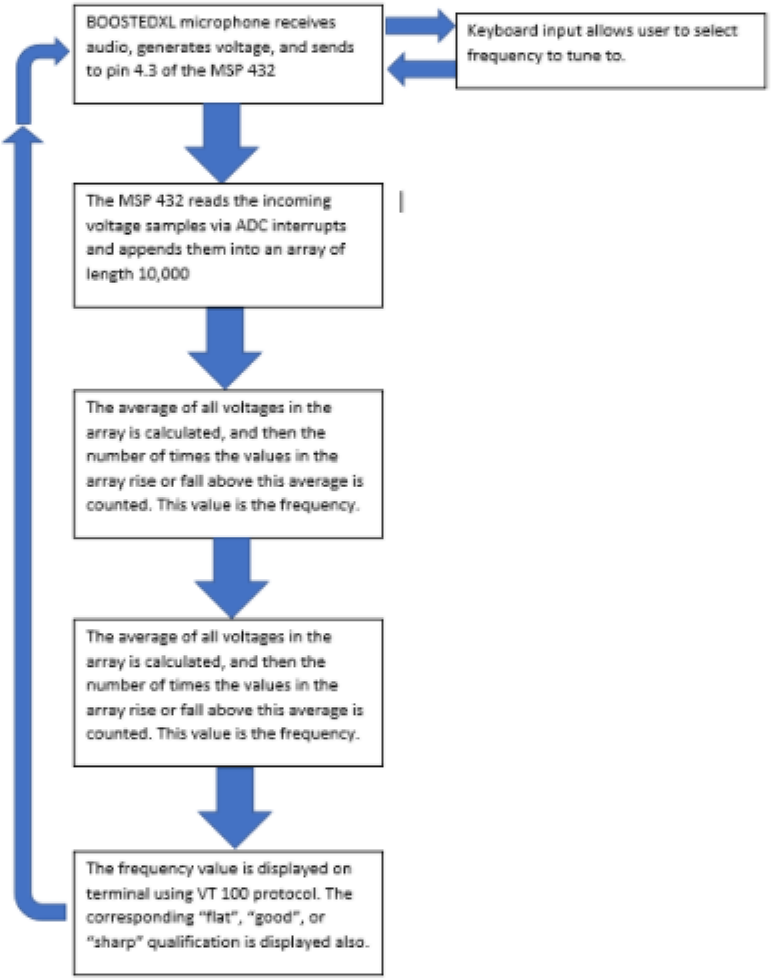
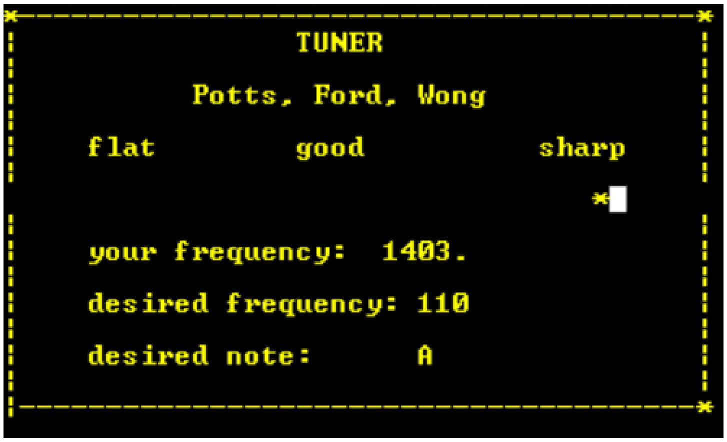

BOOSTEDXL microphone receives audio, generates voltage, and sends to pin 4.3 of the MSP 432
Keyboard input allows user to select frequency to tune to.
The MSP 432 reads the incoming voltage samples via ADC interrupts and appends them into an array of length 10,000
The average of all voltages in the array is calculated, and then the number of times the values in the array rise or fall above this average is counted. This value is the frequency.
The average of all voltages in the array is calculated, and then the number of times the values in the array rise or fall above this average is counted. This value is the frequency.
The frequency value is displayed on terminal using VT 100 protocol. The corresponding "flat", "good", or "sharp" qualification is displayed also.

Figure 3: Software Flowchart



```
*------------------------------------------------*
|                    TUNER                       |
|              Potts, Ford, Wong                 |
|      flat          good              sharp     |
                                          *
|      your frequency:  1403.                    |
|      desired frequency: 110                    |
|      desired note:       A                     |
|------------------------------------------------*
```

Figure 4: Digital Tuner Screen Display

Source Code:

```
#include "msp.h"
#include <stdio.h>

void run_avg(void);
void UART0_init(void);
void big_setup(void);
void delayMs(int n);
void term_display(void);
void send_VT(int msg);
void disp_setup();

int sample_array[10000];
int index;
int sum;
float average;
char varray[20];
char farray[20];
char desarray[10];
int nextval;
int max=0;
int min=100000;
int time1;
int time2;
int diff;
int z;
int n;
int what;
int the;
float freq;
int check;
int not_index;
float note;
int b;
char gnote;

int main(void) {

    //void UART0_init();
    __disable_irq();
    big_setup();
    __enable_irq();
    disp_setup();

    while (1) {

        if(index<10000){
        ADC14->CTL0 |= 1;          /* start a conversion */
        while (!ADC14->IFGR0);     /* wait till conversion complete */
        sample_array[index] = 0;
        sample_array[index] = ADC14->MEM[5];   /* read conversion result */
        index++;
        }
        else{
            index=0;
            sum=0;
            average=0;
            __disable_irq();
            run_avg();
            __enable_irq();
        }
    }
}

void run_avg(){
    n=0;
    for(not_index=0; not_index<10000; not_index++){
        sum += sample_array[not_index];
    }
    average = (sum/10000);

    for(not_index=0; not_index<10000; not_index++){
        if((z>average)&&(sample_array[not_index]<average)){
            n += 1;
            }
        z = sample_array[not_index];
        }

    freq = n*2.96;
    n=0;
    check += 1;

    sprintf(farray, "%f", freq);

    term_display();

}

void UART0_init(void) {
    EUSCI_A0->CTLW0 |= 1;     /* put in reset mode for config */
    EUSCI_A0->MCTLW = 0;      /* disable oversampling */
    EUSCI_A0->CTLW0 = 0x0081; /* 1 stop bit, no parity, SMCLK, 8-bit data */
    EUSCI_A0->BRW = 625;       /* 48000000 / 38400 = 1250 */ // FIX BAUD RATE
    P1->SEL0 |= 0x0C;         /* P1.3, P1.2 for UART */
    P1->SEL1 &= ~0x0C;
    EUSCI_A0->CTLW0 &= ~1;    /* take UART out of reset mode */
    EUSCI_A0->IE |= 1;        /* enable receive interrupt */
}
void EUSCIA0_IRQHandler(void) {

    delayMs(20);

    if(EUSCI_A0->RXBUF == 0x31){
       note=82;
       gnote = 'E';
    }
    else if(EUSCI_A0->RXBUF == 0x32){
       note=110;
       gnote = 'A';
    }
    else if(EUSCI_A0->RXBUF == 0x33){
       note=147;
       gnote = 'D';
    }
    else if(EUSCI_A0->RXBUF == 0x34){
       note=196;
       gnote = 'G';
    }
    else if(EUSCI_A0->RXBUF == 0x35){
       note=247;
       gnote = 'B';
    }
    else if(EUSCI_A0->RXBUF == 0x36){
       note=330;
       gnote = 'e';
    }

    sprintf(desarray, "%f", note);

    }
```

```
void delayMs(int n) {
    int i, j;

    for (j = 0; j < n; j++)
        for (i = 750; i > 0; i--);      /* Delay */
}

void big_setup(void) {

    /* Configure P2.2-P2.0 as output for tri-color LEDs */
    P2->SEL0 &= ~7;
    P2->SEL1 &= ~7;
    P2->DIR |= 7;

    ADC14->CTL0 =  0x00000010;    /* power on and disabled during configuration */
    ADC14->CTL0 |= 0x04080300;    /* S/H pulse mode, sysclk, 32 sample clocks, software trigger */
    ADC14->CTL1 =  0x00000020;    /* 12-bit resolution */
    ADC14->MCTL[5] = 10;           /* A6 input, single-ended, Vref=AVCC */
    P4->SEL1 |= 0x08;             /* Configure P4.7 for A6 */
    P4->SEL0 |= 0x08;
    ADC14->CTL1 |= 0x00050000;    /* convert for mem reg 5 */
    ADC14->CTL0 |= 2;             /* enable ADC after configuration*/

    EUSCI_A0->CTLW0 |= 1;     /* put in reset mode for config */
    EUSCI_A0->MCTLW = 0;      /* disable oversampling */
    EUSCI_A0->CTLW0 = 0x0081; /* 1 stop bit, no parity, SMCLK, 8-bit data */
    EUSCI_A0->BRW = 625;       /* 48000000 / 38400 = 1250 */ // FIX BAUD RATE
    P1->SEL0 |= 0x0C;         /* P1.3, P1.2 for UART */
    P1->SEL1 &= ~0x0C;
    EUSCI_A0->CTLW0 &= ~1;    /* take UART out of reset mode */
    EUSCI_A0->IE |= 1;        /* enable receive interrupt */

    NVIC_SetPriority(EUSCIA0_IRQn, 4); /* set priority to 4 in NVIC */
    NVIC_EnableIRQ(EUSCIA0_IRQn);      /* enable interrupt in NVIC */

}

void term_display(void){

    send_VT(27); // loading new cursor location
    send_VT('[');
    send_VT('1');
    send_VT('0');
    send_VT(';');
    send_VT('2');
    send_VT('3');
    send_VT('f');

    send_VT(farray[0]);
    send_VT(farray[1]);
    send_VT(farray[2]);
    send_VT(farray[3]);
    send_VT(farray[4]);

    send_VT(27);
    send_VT('[');
    send_VT('1');
    send_VT('2');
    send_VT(';');
    send_VT('2');
    send_VT('5');
    send_VT('f');

    send_VT(desarray[0]);
    send_VT(desarray[1]);
    send_VT(desarray[2]);
    send_VT(27);
    send_VT('[');
    send_VT('1');
    send_VT('4');
    send_VT(';');
    send_VT('2');
    send_VT('5');
    send_VT('f');

    send_VT(gnote);

    send_VT(27);
    send_VT('[');
    send_VT('8');
    send_VT(';');
    send_VT('8');
    send_VT('f');

    send_VT(27);
    send_VT('[');
    send_VT('2');
    send_VT('K');

    diff = note - freq;

    if((diff>10)&&(note!=0)){
    send_VT(27);
    send_VT('[');
    send_VT('8');
    send_VT(';');
    send_VT('8');
    send_VT('f');
    send_VT('*');
    }
    else if((diff<-10)&&(note!=0)){
    send_VT(27);
    send_VT('[');
    send_VT('8');
    send_VT(';');
    send_VT('3');
    send_VT('5');
    send_VT('f');
    send_VT('*');
    }
    else if((diff<10)&&(diff>-10)&&(note!=0)){
    send_VT(27);
    send_VT('[');
    send_VT('8');
    send_VT(';');
    send_VT('2');
    send_VT('0');
    send_VT('f');
    send_VT('*');
    }
}

void disp_setup(void){

    send_VT('*');

    for(b=0; b<39; b++){
        send_VT('-');
    }

    send_VT('*');

    for(b=1; b<16; b++){
```

```
    send_VT(27);
    send_VT('[');
    send_VT(1);
    send_VT('B');

    send_VT(27);
    send_VT('[');
    send_VT(1);
    send_VT('D');
    send_VT('|');
}

send_VT(27);
send_VT('[');
send_VT(1);
send_VT('B');
send_VT(27);
send_VT('[');
send_VT(1);
send_VT('D');
send_VT('*');

for(b=0; b<41; b++){
    send_VT(27);
    send_VT('[');
    send_VT(1);
    send_VT('D');
}

send_VT('*');

for(b=0; b<39; b++){
    send_VT('-');
}

send_VT(27);
send_VT('[');
send_VT('H');


for(b=1; b<16; b++){

    send_VT(27);
    send_VT('[');
    send_VT(1);
    send_VT('B');

    send_VT(27);
    send_VT('[');
    send_VT(1);
    send_VT('D');
    send_VT('|');
}

send_VT(27);
send_VT('[');
send_VT('2');
send_VT(';');
send_VT('1');
send_VT('8');
send_VT('f');

send_VT('T');
send_VT('U');
send_VT('N');
```

```
    send_VT('E');
    send_VT('R');

    send_VT(27); // loading new cursor location
    send_VT('[');
    send_VT('4');
    send_VT(';');
    send_VT('1');
    send_VT('2');
    send_VT('f');

    send_VT('P');
    send_VT('o');
    send_VT('t');
    send_VT('t');
    send_VT('s');
    send_VT(',');
    send_VT(' ');
    send_VT('F');
    send_VT('o');
    send_VT('r');
    send_VT('d');
    send_VT(',');
    send_VT(' ');
    send_VT('W');
    send_VT('o');
    send_VT('n');
    send_VT('g');

    send_VT(27); // loading new cursor location
    send_VT('[');
    send_VT('6');
    send_VT(';');
    send_VT('6');
    send_VT('f');

    send_VT('f');
    send_VT('l');
    send_VT('a');
    send_VT('t');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT('g');
    send_VT('o');
    send_VT('o');
    send_VT('d');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT(' ');
    send_VT('s');
    send_VT('h');
    send_VT('a');
    send_VT('r');
    send_VT('p');
```

```
send_VT(27); // loading new cursor location
send_VT('[');
send_VT('1');
send_VT('0');
send_VT(';');
send_VT('6');
send_VT('f');

send_VT('y');
send_VT('o');
send_VT('u');
send_VT('r');
send_VT(' ');
send_VT('f');
send_VT('r');
send_VT('e');
send_VT('q');
send_VT('u');
send_VT('e');
send_VT('n');
send_VT('c');
send_VT('y');
send_VT(':');
send_VT(' ');

send_VT(27); // loading new cursor location
send_VT('[');
send_VT('1');
send_VT('2');
send_VT(';');
send_VT('6');
send_VT('f');

send_VT('d'); // loading new cursor location
send_VT('e');
send_VT('s');
send_VT('i');
send_VT('r');
send_VT('e');
send_VT('d');
send_VT(' ');
send_VT('f');
send_VT('r');
send_VT('e');
send_VT('q');
send_VT('u');
send_VT('e');
send_VT('n');
send_VT('c');
send_VT('y');
send_VT(':');
send_VT(' ');

send_VT(27); // loading new cursor location
send_VT('[');
send_VT('1');
send_VT('4');
send_VT(';');
send_VT('6');
send_VT('f');

send_VT('d'); // loading new cursor location
send_VT('e');
send_VT('s');
send_VT('i');
send_VT('r');
send_VT('e');
send_VT('d');
```

```
    send_VT(' ');
    send_VT('n');
    send_VT('o');
    send_VT('t');
    send_VT('e');
    send_VT(':');
    send_VT(' ');

}

void send_VT(int msg){

    while(!(EUSCI_A0->IFG & 0x02)) { }  /* wait for transmit buffer empty */

    EUSCI_A0->TXBUF = (msg);

}
```