DEPARTMENT OF ELECTRICAL ENGINEERING, IIT BOMBAY

EE 344: Electronic Design Lab

Final Project Report

Digital Equalizer

Group Number: BT11

CHANDAN KUMAR(16D070061)

SAARTHAK KAPSE(16D070041)

By prof Siddharth Tallur

05.05.2019(Sunday)

Contents

1 Abstract 2

2 Introduction 2

2.1 Background and Motivation 2
2.2 Project Goals 2
2.3 Design : Overview 2

3. Concept 3

3.1 basics 3
3.2 Theory.. 3
3.3 Table for frequency ranges......................................5
3.3 Circuit 5
3.4 Code 8

4. Conclusion 11

4.1 Results 11
4.2 Problem we faced 11
4.3 Suggestions for Future work 11

5. Appendix 12

1 Abstract

Recent history has seen significant trend towards digitalization.It has become necessary for engineers accustomed to perform the digital tasks that have traditionally been accomplished in analog. Digital control audio equalizer has been one of the tasks. Our project begins by discussing the needs for equalization and sound quality advantages offers for multi -media systems. Digital equalizer can be designed, tested and modified. The report is to show the advantage of digital equalization over analog like easy verification, it's flexibility, it's reliability and superior sound quality.

2 Introduction

2.1 Background and Motivation:

1. To change the sound profiling and genre of music in favor of listener taste.
2. It is practically useful to adjust the sound of speaker(person).
3. Differentiate between female and male voice: The speech of adult male will have a fundamental frequency from 85 to 180 Hz, and that of a adult female from 165 to 255 Hz

2.2 Project Goals:

Goal of our project is to make digital equalizer using analog filters and digital potentiometer with frequency range from 0 Hz to 20,000 Hz.

2.3 Design : Overview:

For Designing digital equalizer we proceed by making 6 band audio equalizer (ie, filters) having 1 low pass filter, 4 band pass filter of their respective frequencies range and 1 high pass filter to filter out our required sound. Used inverting amplifier to control the gain of each components(frequency bands).Feedback of this inverting amplifier being the digital potentiometer. Basically, Digital potentiometer was used to control the gain of each filter separately from 0 to 10 (the range of the gain). Digital pot will be controlled by Tiva-c launchpad (from texas instrument) and this will be connected to Pc/Laptop via usb. We have chosen band of six filters to be approximately equidistant on the logarithmic scale of frequencies from 20Hz to 20 KHz.

3. Concept

3.1 basics:

In low frequency applications (frequency up to 50-kHz), filters are generally constructed using simple Resistor-Capacitor networks, These filters are function of frequency.

A Low Pass Filter is a circuit that reject all unwanted high frequencies of an electrical signal and pass only those signals wanted by the circuits designer. It only allows low frequency signals from 0Hz to its cut-off frequency fc to pass while blocking any higher frequency. Cut off frequency in our case is 50Hz.

High Pass Filter – The high pass filter only allows high frequency signals from its cut-off frequency(fc) and higher to infinity to pass through while blocking any lower frequency.

Band Pass Filter – The band pass filter allows signals within a certain frequency band to pass through while blocking both the lower and higher frequencies on either side of this frequency band.

3.2 Theory;

Equation for low pass filter circuit transfer function is $\frac{1}{1+jR\omega C}$

Cut off frequency(fc) can and it's phase shift(Φ) is given as

$$^{*}f_c = \frac{1}{2\pi RC} \qquad \Phi = -\arctan(2\pi fRC)$$

Equation for high pass filter circuit transfer function is $\frac{R}{R+j\omega C}$

Cut off frequency(fc) can and it's phase shift(Φ) is given as

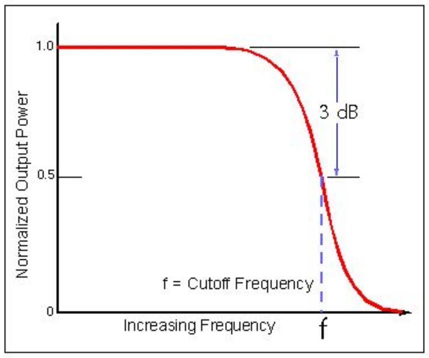
$$^{*}f_c = \frac{1}{2\pi RC} \qquad \Phi = \arctan\left(\frac{1}{2\pi fRC}\right)$$

Equation for band pass filter circuit transfer function is

$$f_{cl} = \frac{1}{2\pi R1C1} \qquad f_{ch} = \frac{1}{2\pi R2C2}$$

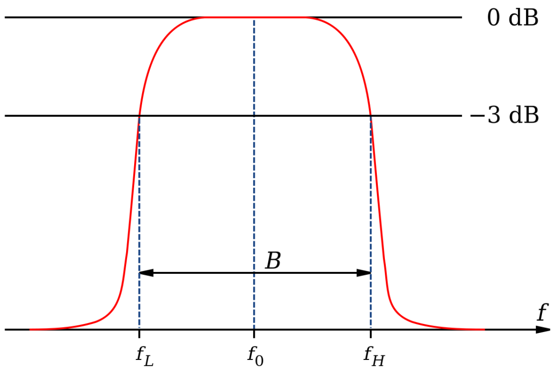
Center frequency can be calculated as $f_0 = \sqrt{fcl * fch}$

Bandwidth of any band pass filter can be calculated as BW = $(f_{ch} - f_{cl})$

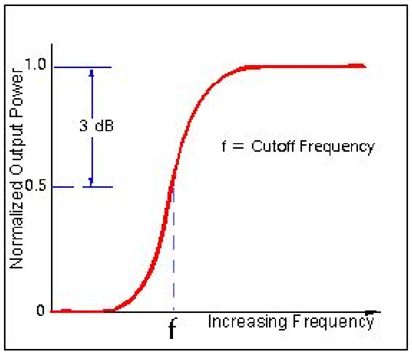


low pass filter

Low pass filter gain vs frequency plot



Band pass filter gain vs frequency plot



high-pass filter

High pass filter gain vs frequency plot

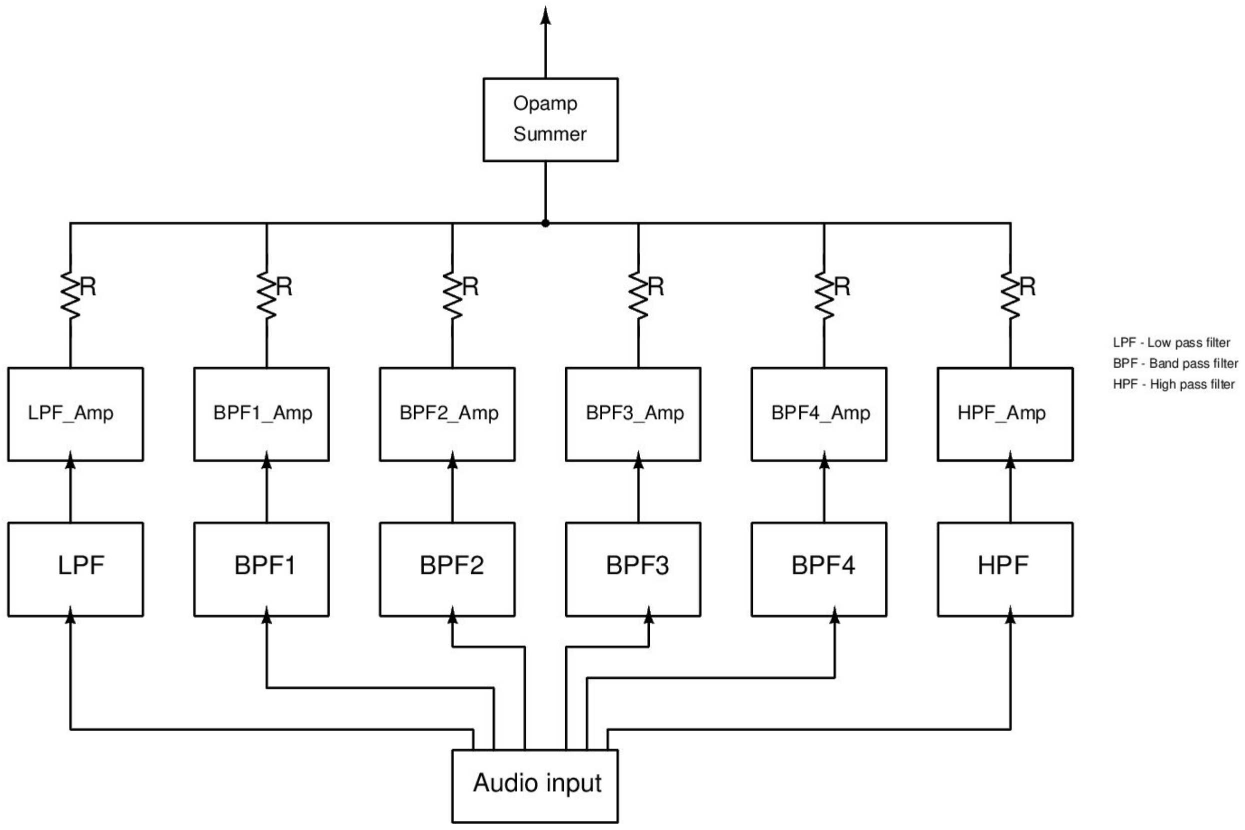
* Note-- RC values can be calculated from fc equation.

3.3 Table for frequency ranges:

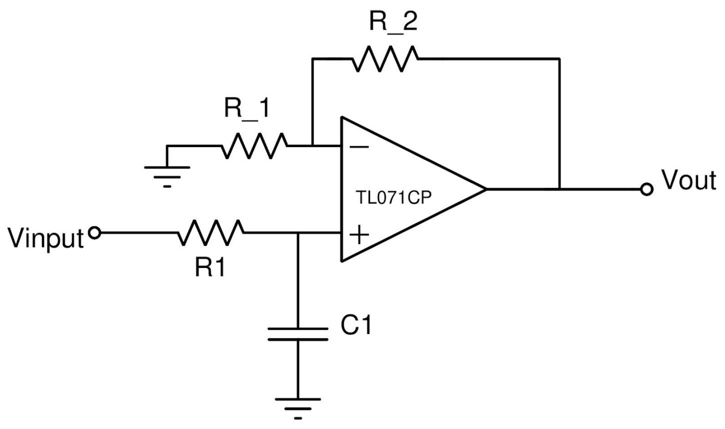
Filters	3db Frequency (HZ)	Center frequency (Hz)	Bandwidth (Hz)
HPF	10400	--	-
BPF4	1500-10400	3950	8900
BPF3	385-2600	1000	2215
BPF2	120-800	310	680
BPF1	50-320	126.5	270
LPF	50	--	50

3.4 Circuit:

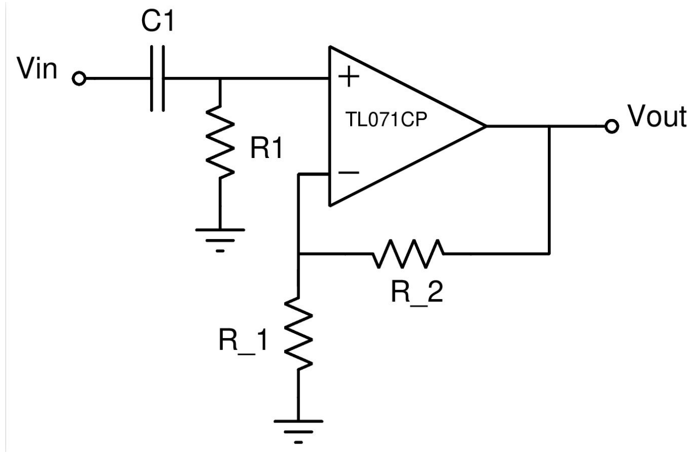
Block Diagram of circuit:- Here R=10k


Opamp Summer
R
LPF_Amp
BPF1_Amp
BPF2_Amp
BPF3_Amp
BPF4_Amp
HPF_Amp
LPF
BPF1
BPF2
BPF3
BPF4
HPF
Audio input
LPF - Low pass filter
BPF - Band pass filter
HPF - High pass filter

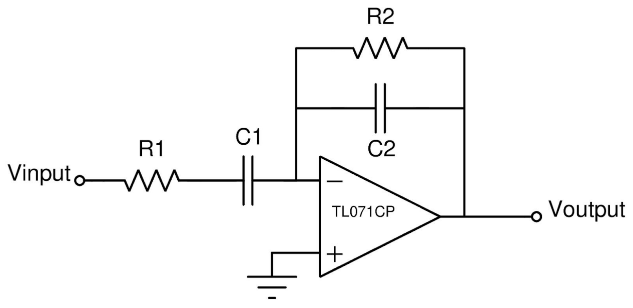
LPF:-


R_2
R_1
TL071CP
Vout
Vinput
R1
C1

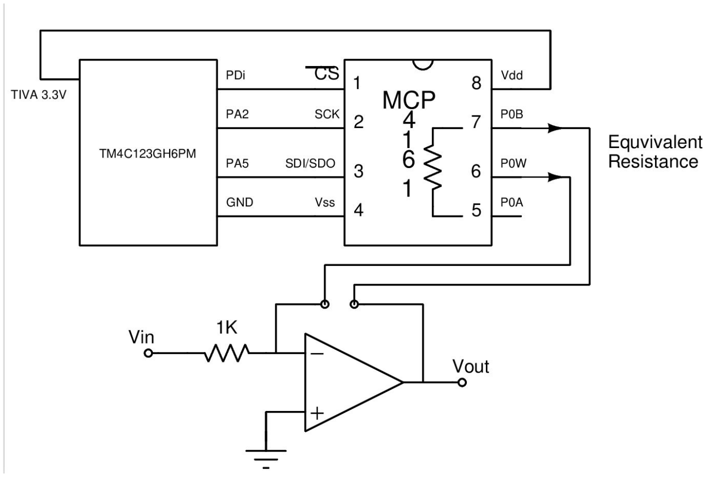
HPF:-



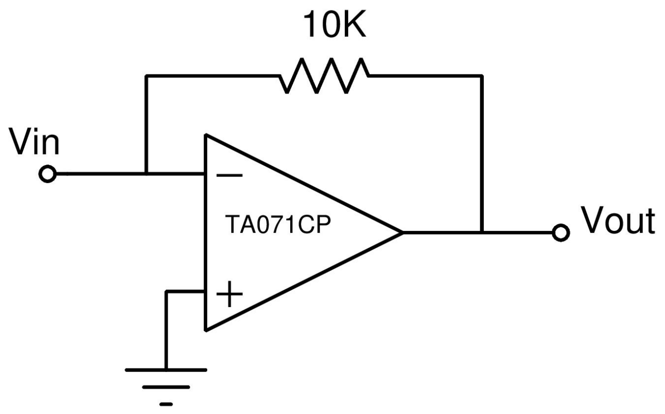
BPF:-



Amplifier:-



Opamp Summer:-



Filter	R1	C1	R2	C2
BPF1	678k	0.0047uF	821k	560pF
BPF2	594k	0.0022uF	713k	270pF
BPF3	185k	0.0022uF	213k	270pF
BPF4	48k	0.0022uF	57k	270pF

Filter	R1	C1	R_1	R_2
LPF	33k	0.01uF	1.2	8.2
HPF	15.9k	0.001u	5.6k	1.2k

3.5 Code:

Code for digital potentiometer control is

Acknowledgment-Shashank

```
#include <stdint.h>
#include <stdlib.h>
#include "TM4C123GH6PM.h"
#define SSI0_SR_R             (*((volatile unsigned long *)0x4000800C))
```

```
#define SSI0_CPSR_R            (*((volatile unsigned long *)0x40008010))
#define SSI0_CC_R              (*((volatile unsigned long *)0x40008FC8))
#define SSI_CR0_SCR_M          0x0000FF00  // SSI Serial Clock Rate
#define SSI_CR0_SPH            0x00000080  // SSI Serial Clock Phase
#define SSI_CR0_SPO            0x00000040  // SSI Serial Clock Polarity
#define SSI_CR0_FRF_M          0x00000030  // SSI Frame Format Select
#define SSI_CR0_FRF_MOTO       0x00000000  // Freescale SPI Frame Format
#define SSI_CR0_DSS_M          0x0000000F  // SSI Data Size Select
#define SSI_CR0_DSS_8          0x00000007  // 8-bit data
#define SSI_CR1_MS             0x00000004  // SSI Master/Slave Select
#define SSI_CR1_SSE            0x00000002  // SSI Synchronous Serial Port Enable
#define SSI_SR_RNE             0x00000004  // SSI Receive FIFO Not Empty
#define SSI_SR_TNF             0x00000002  // SSI Transmit FIFO Not Full
#define SSI_SR_TFE             0x00000001  // SSI Transmit FIFO Empty
#define SSI_CPSR_CPSDVSR_M     0x000000FF  // SSI Clock Prescale Divisor
#define SSI_CC_CS_M            0x0000000F  // SSI Baud Clock Source
#define SSI_CC_CS_SYSPLL       0x00000000  // Either the system clock (if the
PLL bypass is in effect) or the
                                           // PLL output (default)
#define SYSCTL_RCGC1_R         (*((volatile unsigned long *)0x400FE104))
#define SYSCTL_RCGC2_R         (*((volatile unsigned long *)0x400FE108))
#define SYSCTL_RCGC1_SSI0      0x00000010  // SSI0 Clock Gating Control
#define SYSCTL_RCGC2_GPIOA     0x00000001  // port A Clock Gating Control
#define APINT_R                (*((volatile unsigned long *)0xE000ED0C))

void ssi_Init(void);
void timer0A_delayMs(int ttime);
void ss_pd0(int c0);
void ss_pd1(int c1);
void ss_pd2(int c2);
void ss_pd3(int c3);
void ss_pd6(int c6);
void ss_pd7(int c7);

int i=0,r0_pd0,r1_pd1,r2_pd2,r3_pd3,r6_pd6,r7_pd7;

int main(void){
   ssi_Init();
   volatile unsigned long delay;
   /********* PD 0, 1, 2, 3, 6, 7********************/
   SYSCTL_RCGC2_R |= 0x00000008;     //enable clock
   delay = SYSCTL_RCGC2_R;           //delay needed for above to finish
   GPIO_PORTD_CR_R = 0xCF;           //enable writing in it
   GPIO_PORTD_AMSEL_R &= 0x30;       //disable analog
   GPIO_PORTD_PCTL_R = 0x00;         //regular digital
```

```
    GPIO_PORTD_DIR_R = 0xCF;          // output
    GPIO_PORTD_AFSEL_R &= 0x30;         //disable alt function
    GPIO_PORTD_DEN_R = 0xCF;          //enable digital function
    GPIO_PORTD_DATA_R = 0xCF;

    while(1)
    {
      ss_pd0(1); //HPF
      ss_pd1(1); //BPF4
      ss_pd2(1); //BPF3
      ss_pd3(255); //BPF2
      ss_pd6(255); //BPF1
      ss_pd7(1); //LPF

    }
}
//////////////////////////////////////////////////////
void ssi_Init(void){
  volatile unsigned long delay2;
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_SSI0;  // activate SSI0
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
  GPIO_PORTA_AFSEL_R |= 0x3C;           // enable alt funct on PA2,3,5
                                  // configure PA2,3,5 as SSI
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R & 0xFF0000FF)+0x00222200;
  GPIO_PORTA_DEN_R |= 0x3C;             // enable digital I/O on PA2,3,4,5
  GPIO_PORTA_AMSEL_R &= ~0x3C;          // disable analog functionality on
PA2,3,5,6,7
  SSI0_CR1_R &= ~SSI_CR1_SSE;           // disable SSI
  SSI0_CR1_R &= ~SSI_CR1_MS;            // master mode
                                  // configure for system clock/PLL baud clock source
  SSI0_CC_R = (SSI0_CC_R&~SSI_CC_CS_M)+SSI_CC_CS_SYSPLL;
                                  // clock divider for 2 MHz SSIClk (16 MHz PIOSC/2)
  SSI0_CPSR_R = (SSI0_CPSR_R&~SSI_CPSR_CPSDVSR_M)+2;
  SSI0_CR0_R &= ~(SSI_CR0_SCR_M |       // SCR = 0 (2 Mbps data rate)
                SSI_CR0_SPH |         // SPH = 0
                SSI_CR0_SPO);         // SPO = 0
                                  // FRF = Freescale format
  SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_FRF_M)+SSI_CR0_FRF_MOTO;
                                  // DSS = 16-bit data
  SSI0_CR0_R = (SSI0_CR0_R&~SSI_CR0_DSS_M)+SSI_CR0_DSS_16;
  SSI0_CR1_R |= SSI_CR1_SSE;            // enable SSI
}

void ss_pd0(int c0)
{
    GPIO_PORTD_DATA_R &= ~(0x01);
    SSI0_DR_R = c0;
    while((SSI0_SR_R & 0x01)==0x00){};   // wait until data has finished
transmitting
    timer0A_delayMs(10);
    GPIO_PORTD_DATA_R |= 0x01;
}

void ss_pd1(int c1)
{
    GPIO_PORTD_DATA_R &= ~(0x02);
    SSI0_DR_R = c1;
    while((SSI0_SR_R & 0x01)==0x00){};   // wait until data has finished
transmitting
    timer0A_delayMs(10);
    GPIO_PORTD_DATA_R |= 0x02;
}

void ss_pd2(int c2)
{
    GPIO_PORTD_DATA_R &= ~(0x04);
    SSI0_DR_R = c2;
    while((SSI0_SR_R & 0x01)==0x00){};   // wait until data has finished
transmitting
    timer0A_delayMs(10);
    GPIO_PORTD_DATA_R |= 0x04;
}

void ss_pd3(int c3)
{
    GPIO_PORTD_DATA_R &= ~(0x08);
    SSI0_DR_R = c3;
    while((SSI0_SR_R & 0x01)==0x00){};   // wait until data has finished
transmitting
    timer0A_delayMs(10);
    GPIO_PORTD_DATA_R |= 0x08;
}
void ss_pd6(int c6)
{
    GPIO_PORTD_DATA_R &= ~(0x40);
    SSI0_DR_R = c6;
    while((SSI0_SR_R & 0x01)==0x00){};   // wait until data has finished
transmitting
    timer0A_delayMs(10);
    GPIO_PORTD_DATA_R |= 0x40;
```

```
}
void ss_pd7(int c7)
{
   GPIO_PORTD_DATA_R &= ~(0x80);
   SSI0_DR_R = c7;
   while((SSI0_SR_R & 0x01)==0x00){};   // wait until data has finished
transmitting
   timer0A_delayMs(10);
   GPIO_PORTD_DATA_R |= 0x80;
}

void timer0A_delayMs(int ttime)
{
   int i;
   SYSCTL_RCGCTIMER_R |= 1;     /* enable clock to Timer Block 0 */

   TIMER0_CTL_R = 0;            /* disable Timer before initialization */
   TIMER0_CFG_R = 0x04;         /* 16-bit option */
   TIMER0_TAMR_R = 0x02;        /* periodic mode and down-counter */
   TIMER0_TAILR_R = 16000 - 1;  /* Timer A interval load value register */
   TIMER0_ICR_R = 0x1;          /* clear the TimerA timeout flag*/
   TIMER0_CTL_R |= 0x01;        /* enable Timer A after initialization */

   for(i = 0; i < ttime; i++) { while ((TIMER0_RIS_R & 0x1) == 0) ;      /* wait for
TimerA timeout flag */
      TIMER0_ICR_R = 0x1;      /* clear the TimerA timeout flag */      }        }
```

4 Conclusion:

4.1 Results :

1. Bass: We got bass at our output (Speaker/earphone) When we amplified the 1st and 2nd band pass filter of frequencies ranges 50 Hz - 120Hz and 12 Hz-385 Hz respectively only.
2. Only instrumental sound: Since, Fundamental frequency of most of the instruments lies between 400 Hz to 4 KHz. We did hear the sound of instruments when 3rd and 4th band pass filter were amplified only.
   For more detail:

https://docs.google.com/spreadsheets/d/1JDkWlz62iOTgQhlCyl07U2Bhliu1yrkgU3AIqtTI_qI/edit?usp=sharing

3. Only voice sound: ` Normal vocal sound of human are of frequency range 85 Hz to 255 Hz. So, we were able to hear the sound of the person singing.

4.2 Problems faced:

1. During making the inverting amplifier it was somehow showing the offset on Ua741 op-amp. so, we moved to TL071.
2. LPF and HPF gain was not equal to 1. So, we added an extra resistor at feedback.
3. On completion of analog it was giving some noise because of wire looping and stereo input jack.
4. Since, Output of filters was not coming correctly at our initial resistor and capacitor value calculated. We had to adjust accordingly. So, that it can give our desired output value at different frequencies.

4.3 Suggestions for Future work:

1. Gui:

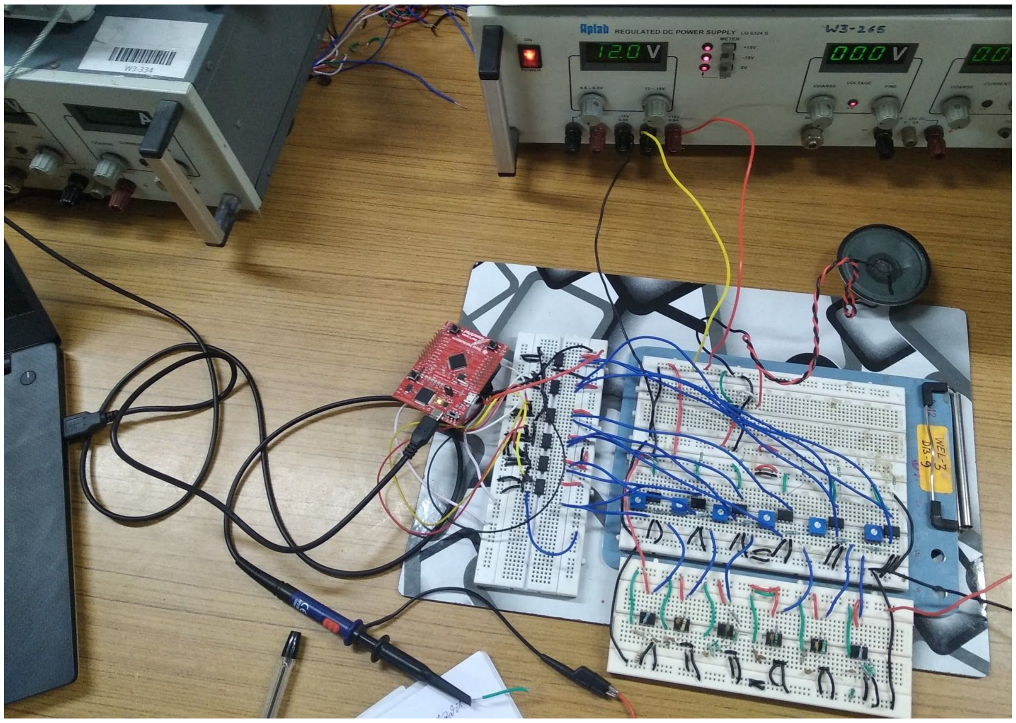
   We can implement gui based digital equalizer for easy control of the digital pots instead of changing the value of resistor of digital pots in laptop.

2. Higher order filter:

   We can implement higher order filters for more feathering of the Sound Like reverse sound, rotating sound, high tone, low tone and 3-D sounds genres.

5 Appendix:

Our Final circuit on breadboard looked something like this :



End report