Frequency Identifier

Nikhil Prakash 09026015
Avinash Ashok 09026010
Parth Jinger 09026014

November 14, 2011

Abstract

We've build a Frequency Identifier which can Identify the Frequency of a Unknown single Frequency Signal using the principle of Discrete Fourier Tranform which convert a Time Domain Signal to a Frequency Domain. We'll use a Microphone to get the Analog signal and fed to Arduino board and using fourier transform we'll analyze the different frequency content in the signal .This project can be extended to Learning more about Music Signals.

0.1 Principle Of Discrete Fourier Transform

Discrete Fourier transform (DFT) is a specific kind of discrete transform, used in Fourier analysis. It transforms one function into another, which is called the Frequency Domain Representation, or Simply the DFT, of the original function (which is often a function in the time domain). But the DFT requires an input function that is discrete and whose non-zero values have a limited (finite) duration. Such inputs are often created by sampling a continuous function, like a person's voice. Unlike the discrete-time Fourier transform (DTFT), it only evaluates enough frequency components to reconstruct the finite segment that was analyzed.
The Basic Equation for the evaluation of DFT :

$$X[k] = \sum_{n=0}^{N-1} x[n] * exp((-i2\pi kn)/N)$$

0.1.1 Description of Radix-4 FFT Algorithm

The radix-4 decimation-in-time and decimation-in-frequency fast Fourier transforms (FFTs) gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs. The radix-4 decimation-in-time algorithm rearranges the discrete Fourier transform (DFT) equation into four parts: sums over all groups of every fourth discrete-time index $n = [0, 4, 8, ..., N-4], n = [1, 5, 9, ..., N-3], n = [2, 6, 10, ..., N-2], n = [3, 7, 11, ..., N-1]$
Mathematically, Length N DFT is calculated as the sum of output of four length $N/4$ DFTs
Working of Radix 4 Algorithm:

$$X[k] = \sum_{n=0}^{N} -1x[n] * exp((-i2\pi kn)/N)$$

$$X[k] = \sum_{n=0}^{N/4-1} x[4n]*exp((-i2\pi k(4n))/N) + \sum_{n=0}^{N/4-1} x[4n+1]*exp((-i2\pi k(4n+1))/N) + \sum_{n=0}^{N/4-1} x[4n+2]*exp((-i2\pi k(4n+2))/N) + \sum_{n=0}^{N/4-1} x[4n+3]*exp((-i2\pi k(4n+3))/N)$$

$X[k] = DFT_{N/4}[x(4n)] + W_N^k DFT_{N/4}[x(4n+1)] + W_N^{2k} DFT_{N/4}[x(4n+2)] + W_N^{3k} DFT_{N/4}[x(4n+3)]$

0.1.2 Frequency Content in the Signal

As our signal will be from $100Hz - 2000Hz$ and our sampling frequency is $50Hz$ therefore there will be aliasing thus although the signal have a single frequency but it'll pose as if having a mixture of frequency thats why we need to examine all coefficients of DFT we got.

0.1.3 Windowing

After taking the samples a window is applied to the input signal. Window is choosen depending upon the nature of the signal .

0.2 Working

A Electret Microphone is used as Transducer to convert sound signal to Electric Signal . As the Electric Signal from the Mircophone is of the order of millivolts(mV) , therefore we needed it to be amplified . For this purpose we used Operational Amplifier (OpAmp) **UA 741** with a amplification of 200x . We gave $+5V$ as $+V$ Power Suppply and $0V$ as $-V$ Power Supply and provided a $+2.5V$ bias to get a reference of 512 as zero in Discrete Time Domain . Then the amplified Electric Signal was sent to Analog Input of Arduino and Analog to Digital Conversion we got discrete values of the Electric signal with reference of 512 and range of $0 - 1024$. Then Code Tranforms the Time Domain Discrete Signal to Frequency Domain Discrete Signal using DFT algorithm of Radix-4 and then using a test based on Experimental Values we can get the Identification of the Frequency in the Signal.

0.3 Data Testing

We implemented our code on 6 of Different signals of different frequency and got the data which will get better after testing it again and again . And using the data we posed some conditions to identify the frequency . Frequency of our setup is $200Hz, 400Hz, 600Hz, 800Hz, 1000Hz and 1200Hz$.

0.4 Work Distribution

Nikhil Prakash	Coding and PCB Design and Electronics Part
Avinash Ashok Aher	PCB Design and Radix 4 FFT Algorithm Design
Parth Jinger	Report and Radix 4 FFT Algorithm Design

0.5 Components Used

i) Electret Microphone

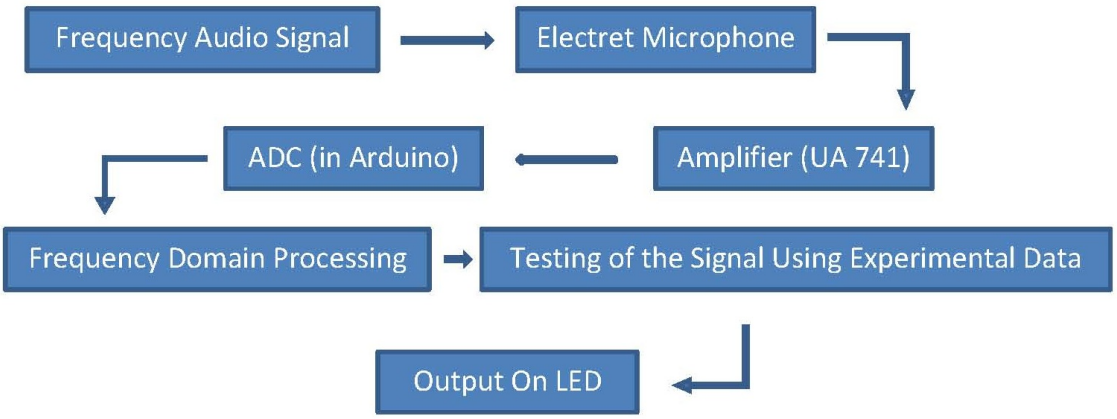
ii) Opamp UA 741

iii) Arduino

iv) LEDs

v) PCB

Figure 1: Block Diagram



0.6 Extension Of Project

Project can be extended to other projects in which frequency identification is needed. Our aim initially was to create a Guitar Chord Identifier but due to limitations of Arduino it was not possible to built that.

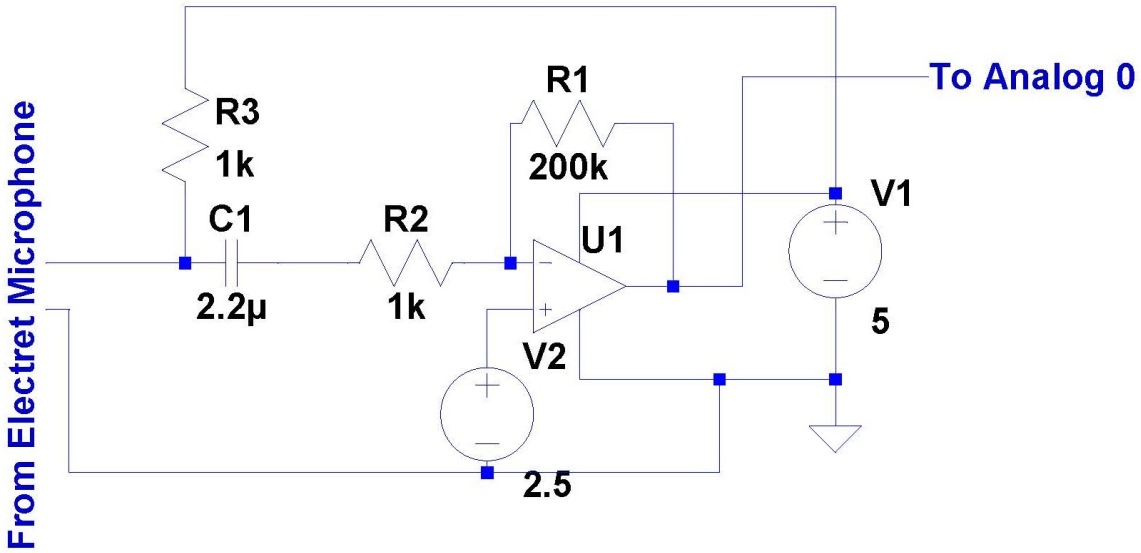
0.7 Refernces

i) http://www.arduino.cc/playground/Interfacing/Processing

ii) http://didier.longueville.free.fr/arduinoos

iii) Digital Signal Processing by Proakis and Manaklov

Figure 2: Circuit Diagram of the Input Circuit



```
// fft.pde
// Custom constants
#define pi 3.14
// Windowing type
#define WIN_TYP_RECTANGLE 0      // Rectangle
#define WIN_TYP_HAMMING 1        // Hamming
#define WIN_TYP_HANN 2           // Hann
#define WIN_TYP_TRIANGLE 3       // Triangle (Bartlett)
#define WIN_TYP_BLACKMAN 4       // Blackmann
#define WIN_TYP_FLT_TOP 5        // Flat top
#define WIN_TYP_WELCH 6          // Welch


#define check 2

int LedPins[]={2,3,4,5,6,7};      //Output LEDs

// Defining Weighting Factors
#define SIN_2PI_16 0.38268343236508978
#define SIN_4PI_16 0.707106781186547460
#define SIN_6PI_16 0.923879532511286740
#define COS_2PI_16 0.923879532511286740
#define COS_4PI_16 0.707106781186547460
#define COS_6PI_16 0.38268343236508978


#include<math.h>
const uint8_t samples = 16;    //No. Of Samples taken
float Data_input[samples];
float Data_output[samples];


// Code For Windowing
void windowing(float *Data_input, uint8_t sample, uint8_t windowType) {


  // The weighing function is symetric; half the weighs are recorded
    float samplesMinusOne = (float(samples) - 1.0);
    for (uint8_t i = 0; i < samples/2; i++) {
        float indexMinusOne = float(i);
        float ratio = (indexMinusOne / samplesMinusOne);
        float weighingFactor = 1.0;
        // compute and record weighting factor
        switch (windowType) {
        case WIN_TYP_RECTANGLE: // rectangle (
            weighingFactor = 1.0;
            break;
        case WIN_TYP_HAMMING: // hamming
            weighingFactor = 0.54 - (0.46 * cos(2.0 * pi * ratio));
```

```
            break;
        case WIN_TYP_HANN: // hann
            weighingFactor = 0.54 * (1.0 - cos(2.0 * pi * ratio));
            break;
        case WIN_TYP_TRIANGLE: // triangle (Bartlett)
            weighingFactor = 1.0 - ((2.0 * abs(indexMinusOne - (samplesMinusOne / 2.0))) /
samplesMinusOne);
            break;
        case WIN_TYP_BLACKMAN: // blackmann
            weighingFactor = 0.42323 - (0.49755 * (cos(2.0 * pi * ratio))) + (0.07922 * (cos(4.0 * pi *
ratio)));
            break;
        case WIN_TYP_FLT_TOP: // flat top
            weighingFactor = 0.2810639 - (0.5208972 * cos(2.0 * pi * ratio)) + (0.1980399 * cos(4.0 * pi *
ratio));
            break;
        case WIN_TYP_WELCH: // welch
            weighingFactor = 1.0 - sq((indexMinusOne - samplesMinusOne / 2.0) / (samplesMinusOne /
2.0));
            break;
        }

            Data_input[i] *= weighingFactor;
            Data_input[samples - (i + 1)] *= weighingFactor;



    }
}

void fft_compute(float *input,float *output){    //Code for Computing DFT using Radix 4 algorithm
  float output_r[16],output_i[16];
  output_r[0]=input[0]+input[8]+input[1]+input[9]+input[10]+input[2]+input[11]+input[3]+input[12]+i
nput[4]+input[13]+input[5]+input[6]+input[14]+input[7]+input[15];

  output_i[0]=0.0;

  output_r[1]=input[0]+COS_2PI_16*(input[1]+input[15]-input[7]-input[9])
+COS_4PI_16*(input[2]+input[14]-input[6]-input[10])+COS_6PI_16*(input[3]+input[13]-input[5]-
input[11])-input[8];

  output_i[1]=-1*(SIN_2PI_16*(input[1]+input[7]-input[9]-input[15])
+SIN_4PI_16*(input[2]+input[6]-input[10]-input[14])+SIN_6PI_16*(input[3]+input[5]-input[11]-
input[13])+input[4]-input[12]);

  output_r[2]=input[0]+COS_4PI_16*(input[1]+input[7]-input[5]-input[3])-
input[4]+input[8]+COS_4PI_16*(input[9]+input[15]-input[11]-input[13])-input[12];

  output_i[2]=-1*(SIN_4PI_16*(input[1]+input[3]-input[5]-input[7])+input[2]-
```

```
input[6]+input[10]+SIN_4PI_16*(input[9]+input[11]-input[13]-input[15])-input[14]);

  output_r[3]=input[0]+COS_6PI_16*(input[1]-input[7]-input[9]+input[15])+COS_4PI_16*(input[6]-
input[2]+input[10]-input[14])-COS_2PI_16*(input[3]-input[5]-input[11]+input[13])-input[8];

  output_i[3]=-1*(SIN_6PI_16*(input[1]+input[7]-input[9]-input[15])
+SIN_4PI_16*(input[2]+input[6]-input[10]-input[14])-SIN_2PI_16*(input[3]+input[5]-input[11]-
input[13])+input[12]-input[4]);

  output_r[4]=input[0]+input[4]+input[8]+input[12]-input[2]-input[6]-input[10]-input[14];

  output_i[4]=-1*(input[1]+input[5]+input[9]+input[13]-input[3]-input[7]-input[11]-input[15]);

  output_r[5]=input[0]-COS_6PI_16*(input[1]-input[7]-input[9]+input[15])
+COS_4PI_16*(input[6]+input[10]-input[2]-input[14])+COS_2PI_16*(input[3]-input[5]-
input[11]+input[13])-input[8];

  output_i[5]=-1*(SIN_6PI_16*(input[1]+input[7]-input[9]-input[15])+SIN_4PI_16*(input[10]-
input[2]+input[14]-input[6])+input[4]-input[12]-SIN_2PI_16*(input[3]-input[11]-input[13]+input[5]));

  output_r[6]=input[0]+COS_4PI_16*(input[3]+input[5]-input[7]-input[1]-
input[9]+input[11]+input[13]-input[15])-input[4]+input[8]-input[12];

  output_i[6]=-1*(0*input[0]+SIN_4PI_16*(input[1]+input[3]-input[5]-input[7]+input[9]+input[11]-
input[13]-input[15])-input[2]+input[6]-input[10]+input[14]);

  output_r[7]=input[0]+COS_2PI_16*(input[7]+input[9]-input[15]-input[1])+COS_4PI_16*(input[2]-
input[6]+input[14]-input[10])+COS_6PI_16*(input[5]-input[3]+input[11]-input[13])-input[8];

  output_i[7]=-1*(0*input[0]+SIN_2PI_16*(input[1]+input[7]-input[15]-input[9])
+SIN_4PI_16*(input[14]+input[10]-input[6]-input[2])-input[4]+SIN_6PI_16*(input[5]+input[3]-
input[11]-input[13])+input[12]);

  output_r[8]=input[0]+input[2]+input[4]+input[6]+input[8]+input[10]+input[12]+input[14]-input[1]-
input[3]-input[5]-input[7]-input[9]-input[11]-input[13]-input[15];

  output_i[8]=0.0;

  output_r[9]=input[0]+COS_2PI_16*(input[7]+input[9]-input[1]-input[15])
+COS_4PI_16*(input[2]+input[14]-input[6]-input[10])+COS_6PI_16*(input[5]-input[3]+input[11]-
input[13])-input[8];

  output_i[9]=-1*(SIN_2PI_16*(input[15]+input[9]-input[7]-input[1])
+SIN_4PI_16*(input[6]+input[2]-input[10]-input[14])+SIN_6PI_16*(input[11]+input[13]-input[3]-
input[5])-input[12]+input[4]);

  output_r[10]=input[0]+COS_4PI_16*(input[13]-input[15]-input[1]+input[3]+input[5]-input[7]-
input[9]+input[11])-input[12]-input[4]+input[8];
```

```
  output_i[10]=-1*(SIN_4PI_16*(input[15]-input[1]-input[3]+input[5]+input[7]-input[9]-
input[11]+input[13])-input[14]+input[2]+input[10]-input[6]);

  output_r[11]=input[0]+COS_6PI_16*(input[9]-input[1]+input[7]-input[15])+COS_4PI_16*(input[6]-
input[14]-input[2]+input[10])-input[8]+COS_2PI_16*(input[3]-input[5]-input[11]+input[13]);

  output_i[11]=-1*(SIN_6PI_16*(input[15]+input[9]-input[1]-input[7])
+SIN_2PI_16*(input[3]+input[5]-input[11]-input[13])+SIN_4PI_16*(input[2]+input[6]-input[10]-
input[14])+input[12]-input[4]);

  output_r[12]=input[0]+input[4]+input[8]+input[12]-input[2]-input[6]-input[10]-input[14];

  output_i[12]=-1*(-input[1]+input[3]-input[5]+input[7]-input[9]+input[11]-input[13]+input[15]);

  output_r[13]= input[0]+COS_6PI_16*(input[1]-input[7]+input[15]-input[9])
+COS_4PI_16*(input[10]-input[2]+input[6]-input[14])+COS_2PI_16*(input[11]-input[3]+input[5]-
input[13])-input[8];

  output_i[13]=SIN_6PI_16*(input[15]-input[7]-input[1]+input[9])+SIN_4PI_16*(input[14]-input[2]-
input[6]+input[10])+SIN_2PI_16*(input[3]+input[5]-input[11]-input[13])-input[12]+input[4];

  output_r[14]=input[0]-input[12]+input[8]-input[4]+COS_4PI_16*(input[1]-input[3]-
input[5]+input[7]+input[9]-input[11]-input[13]+input[15]);

  output_i[14]=-1*(input[6]-input[10]+input[14]-input[2]+SIN_4PI_16*(-input[1]-
input[3]+input[5]+input[7]-input[9]-input[11]+input[13]+input[15]));

  output_r[15]=input[0]+COS_2PI_16*(input[1]+input[15]-input[7]-input[9])
+COS_4PI_16*(input[2]+input[14]-input[6]-input[10])+COS_6PI_16*(input[3]-input[5]-
input[11]+input[13])-input[8];

  output_r[15]=-1*(SIN_2PI_16*(input[15]-input[7]+input[9]-input[1])+SIN_4PI_16*(input[14]-
input[2]-input[6]+input[10])+SIN_6PI_16*(input[11]+input[13]-input[3]-input[5])+input[12]-
input[4]);









  for(int i=0 ;i<16;i++){   // Magnitude Spectrum
    output[i]=sqrt(pow(output_r[i],2)+pow(output_i[i],2));

  }


  free(output_r);      //Saving some space
  free(output_i);
```

```
}
```

```
int input=0,i;

void setup(){
  pinMode(input,INPUT);

  Serial.begin(9600);


  pinMode(SWITCH,INPUT);

  for( int i=0;i<6;i++)
  pinMode(LedPins[i],OUTPUT);


}

void loop(){
  if(digitalRead(SWITCH)==HIGH)
  {test_signal();}


}




void test_signal(){



  Serial.println("****************************************");
  Serial.println("The Input Samples : ");
  for(i=0;i<16;i++){         //taking from the signal
    Data_input[i]=analogRead(input);
```

```
  delay(20);   // Delay to give time analogread to stablize
  Serial.println(Data_input[i]);
 }
 Serial.println("Samples After Windowing");

 windowing(Data_input,samples,0);   // Multiplying the coefficients with Window Coefficients
 for(i=0;i<16;i++){
  Serial.println(Data_input[i]);
 }
 delay(5000);
 fft_compute(Data_input,Data_output);
 Serial.println("Coefficients of The 16 point DFT");
 for(i=0;i<16;i++){
  Serial.println(Data_output[i]);        // Magnitude spectrum of the Input
 }

 Serial.println("****************************************");

    // Testing the Input Signal using Training Data

   if(Data_output[0]>8190.0 && Data_output[0]<8270.0 && Data_output[4]<30.0)    //For 200Hz
   glow_led(0);

   if(Data_output[0]>8210.0 && Data_output[0]<8300.0 && Data_output[4]>15.0 &&
Data_output[4]<60 && Data_output[10]>60 && Data_output[10]<200)     //For 400Hz
   glow_led(1);

   if(Data_output[0]>7800.0 && Data_output[0]<8600.0 && Data_output[2]>50.0 &&
Data_output[2]<180.0 && Data_output[1]>50.0 && Data_output[1]<180.0 )  //For 600Hz
   glow_led(2);

   if(Data_output[0]>8200.0 && Data_output[0]<8300.0 && Data_output[4]>75.0 &&
Data_output[4]<150.0 && Data_output[6]>100.0 && Data_output[6]<250.0 )  //For 800Hz
   glow_led(3);

   if(Data_output[0]>8000.0 && Data_output[0]<8400.0 && Data_output[5]>500.0 &&
Data_output[5]<900.0 && Data_output[4]>200.0 && Data_output[4]<500.0 )  //For 1000Hz
   glow_led(4);

   if(Data_output[0]>7700.0 && Data_output[0]<8800.0 && Data_output[2]>50.0 &&
Data_output[2]<400.0 && Data_output[1]>300.0 && Data_output[1]<900.0 )  //For 1200Hz
   glow_led(5);

   if(Data_output[1]<20.0 && Data_output[2]<20.0 && Data_output[3]<20.0 &&
Data_output[4]<20.0 && Data_output[5]<20.0)   // if nothing is there
   noled();
```

```
}

void glow_led(int k){        // Function For Glowing the LEDs

  for(int i=0;i<6;i++){
  digitalWrite(LedPins[i],LOW);
  }

  digitalWrite(LedPins[k],HIGH);


}

void noled(){              // Function for Switching off all LEDs
  for(int i=0;i<6;i++){
  digitalWrite(LedPins[i],LOW);
  }
}
```