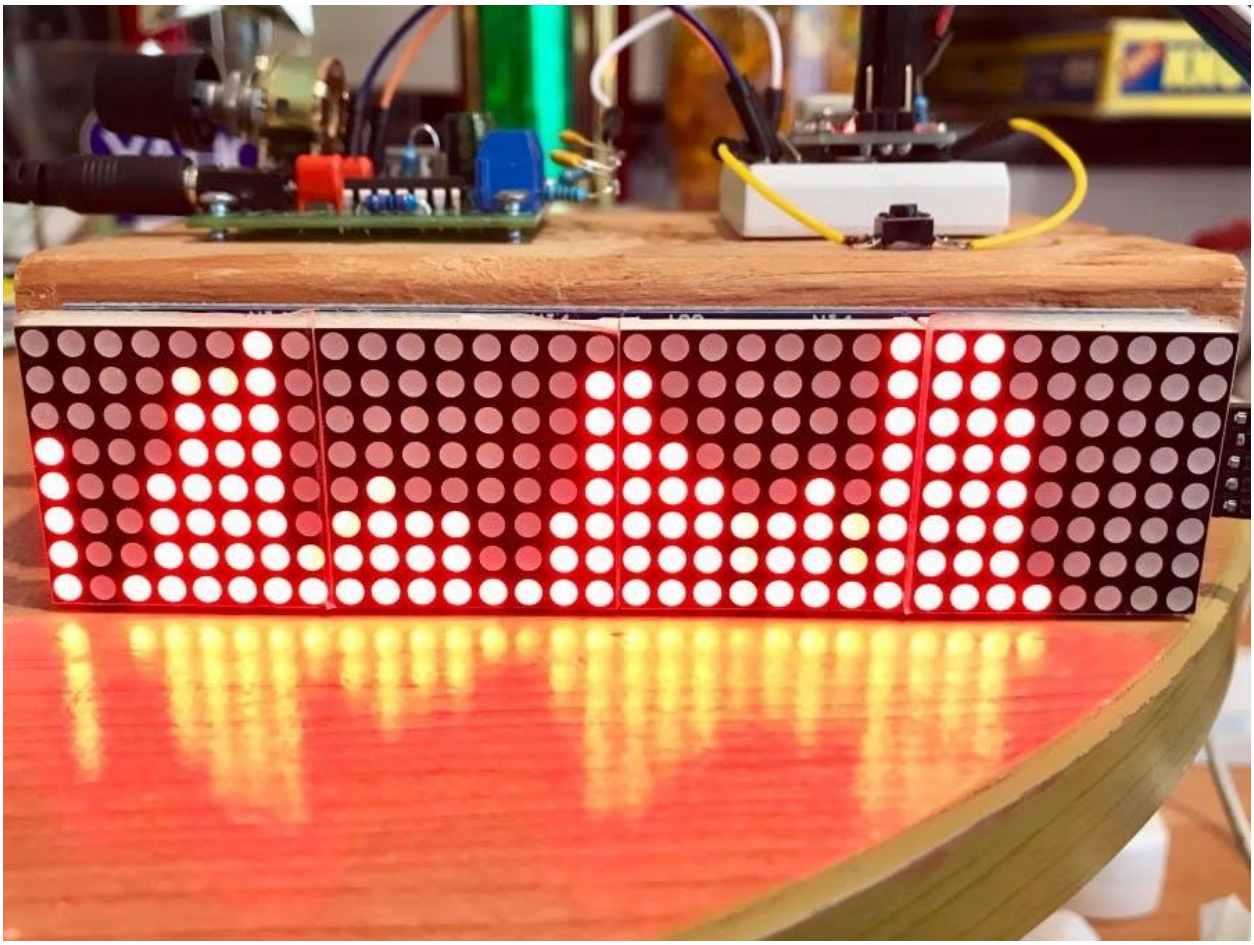


32-Band Audio Spectrum Visualizer Analyzer

This project is for making a 32-band audio (music) frequency spectrum analyzer / visualizer using Arduino.

About this project

This project is for making a 32-band audio (music) frequency spectrum analyzer / visualizer using Arduino. Expected audience of this project is any audio enthusiast, student or a beginner who has basic understanding of electronic components, Arduino and C programming. Components used in this project are low cost items and are easy to assemble.

Main features of this frequency spectrum analyzer

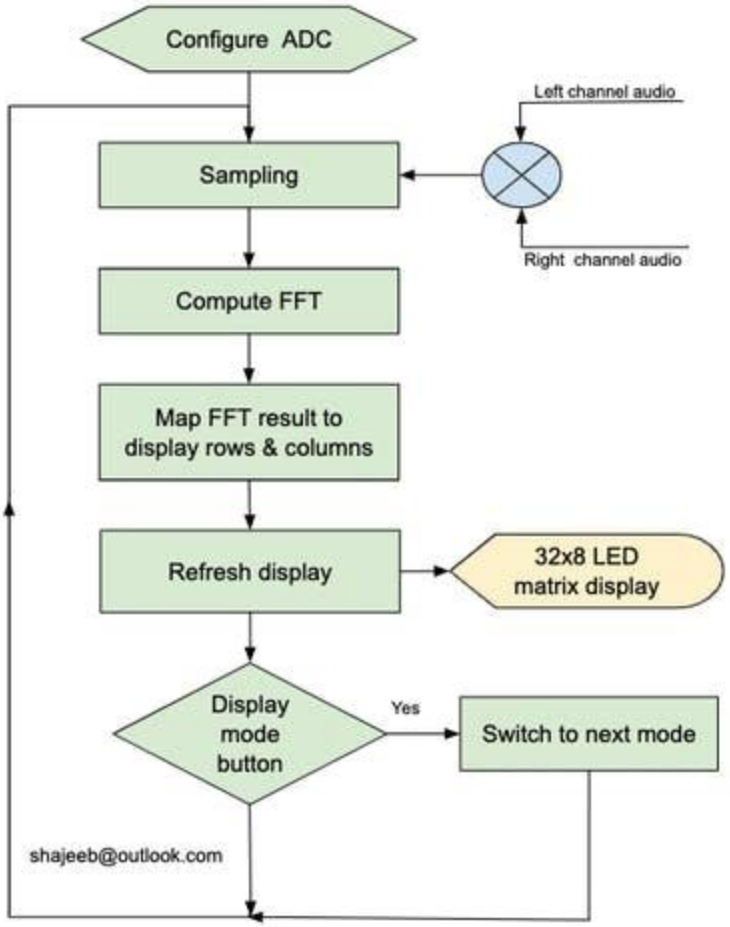
- Uses easily installable libraries “arduinoFFT” and “MD_MAX72xx”
- Five different display modes are supported which can be switched with the push button
- Both left and right channels of audio signal are mixed so that you don't miss any beat
- Prototype use 32x8 LED matrix display, this can be changed and easily modified
- Audio can be fed from headphone output or Line-out of music system / amplifier

Components required

- Arduino Nano or Uno (I tried with Nano and Uno, must work with other models as well)
- 32 x 8 LED matrix display - 1 no
- Push button switch - 1 no (normally comes with Arduino kit)
- 100nf capacitor - 2 nos
- 5 kilo ohms resistor - 3 nos
- 10 kilo ohms resistor - 1 no
- 100 kilo ohms resistor - 2 nons
- 5 volt power supply (usb supply will do)

Resistor values are not very strict, you may choose any closest value. Please make sure R1 & R2 (refer schematic) are of same value.

Program flow chart



Description of the system

Arduino board (ATmega328P) has built in Analog To Digital converter (ADC) which is being used here for converting input audio signal into digital samples. ADC is configured to sample input signal with the clock frequency of 38.46khz. This is achieved by configuring ADC prescaler to 32. Sampling frequency of 38.64Khz means that digital samples can reproduce input frequency of upto 19.32Kz (Nyquist's theorem) which is good enough for audio signals.

As I mentioned in the beginning, the intended purpose of this project is to display frequency spectrum of audio music signal. Hence left and right audio channels are mixed together and fed into the A0 analog input of the ADC. You may use a audio splitter cable so that you can feed same music simultaneously into spectrum analyzer and into another amplifier (if needed).

ADC is configured to use external reference voltage. In this project reference voltage for is derived from the 3.3v stabilized voltage source on the Arduino board. As analog signal oscillates above and below zero voltage level we need a DC bias at the analog input of the ADC. This ensure that ADC output doesn’t clip on the negative cycles of the input signal. Same 3.3v stabilized voltage is divided with two resistors R1 & R2 and then fed into the analog input for

DC bias. With this DC bias ADC will produce 512 in the output even if the input signal is disconnected. Later in the code this 512 which is caused by DC bias is being subtracted so that the reading represent the actual input signal variation.

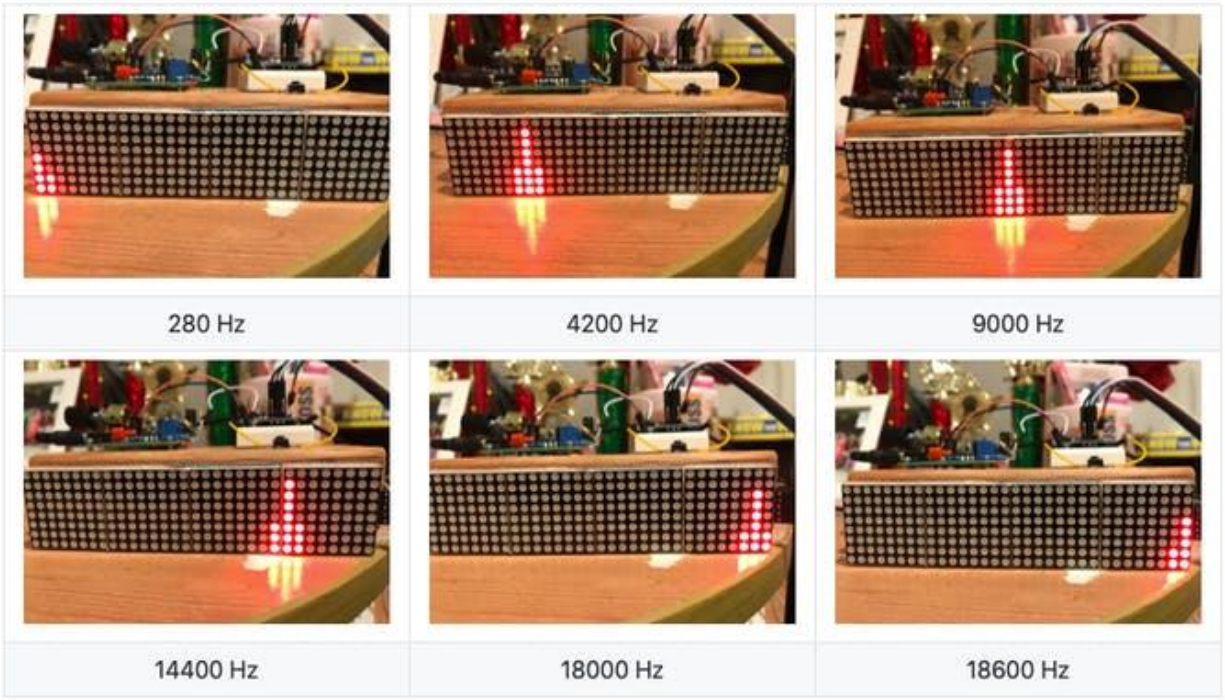
ArduinoFFT library is the heart of the code which does translation of input analog signal into frequency spectrum. I found this library is easy to use and produced best accurate output for this project. Prototype is configured to make 64 samples and does FFT with those samples. ArduinoFFT library can do FFT of samples between 16 to 128, this can be configured in the program. But arduinoFFT library is slow for calculation with 128 samples hence I stick to the best highest of 64 samples.

Display used in this project is 32 columns x 8 rows LED matrix. MD_MAX72xx library made the display controlling part very easy. This library provides function to turn on/off any number of LEDs in a column which is being used in this program. Amplitude of every frequency band is mapped between 0 to 8, depending upon the amplitude corresponding number of LEDs in each column get turned ON.

Five display modes are available in this program which is basically achieved by turning on/off LEDs at different positions in every column. You can modify / create different pattern easily. A push button is used here for changing the display mode. With every press display pattern moved to the next one and finally resets back to default mode. Push button is connected to one of the digital input and that input is scanned after every one round of display refresh.

Frequency response

Frequency response of the system was tested by feeding sine wave generated by one of the online signal generator website. It is verified that system is able to respond for frequencies up to 18.6Khz.



Additional details

Please visit my github for extra information on this project :
https://github.com/shajeebtm/Arduino-audio-spectrum-visualizer-analyzer/

Spectrum analyzer in action

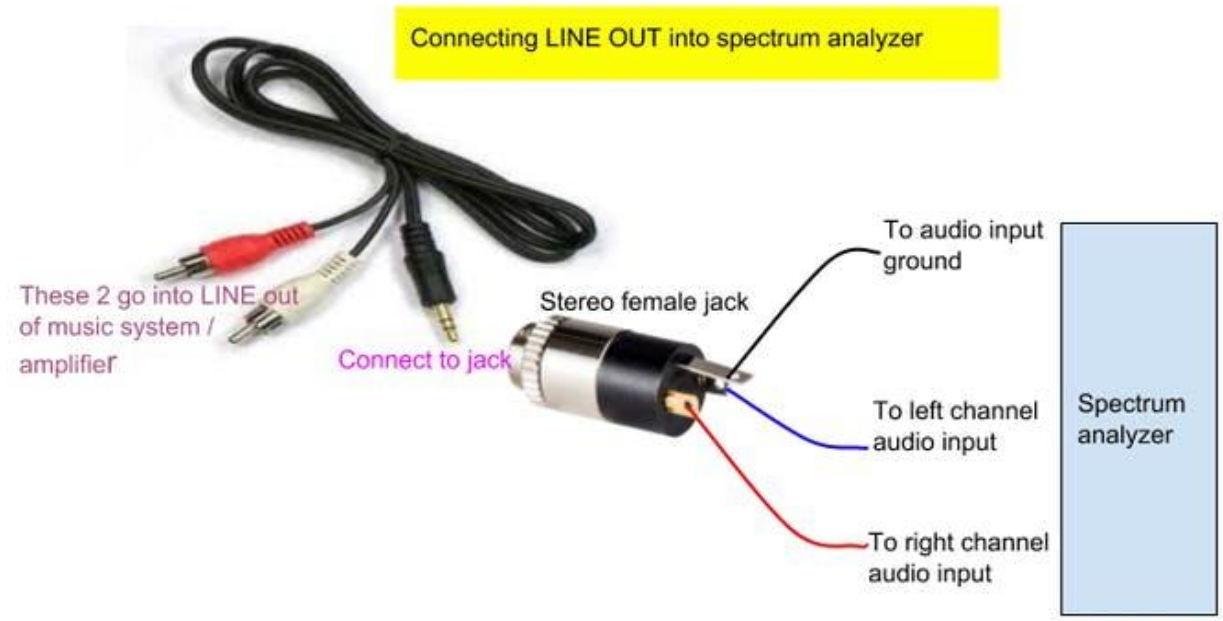
Look at this demo video.

https://youtu.be/GF_i0EnUEro

Connecting input

There are multiple ways you can feed audio input into this Spectrum analyzer. You may take audio output from the LINE out of the music system / amplifier. Other option is to get audio from the headphone output of mobile / music system. I dont suggest to to use another mic for receiving audio as the signal level and frequency response will depend on many factors.

Here is a sample diagram for connecting LINE out of amplifier / music system into Spectrum analyzer.



Below is a sample diagram for connecting headphone output of a mobile / music system into the Spectrum analyzer. When you connect a cable into headphone output no sound will come out of the mobile / music system. Hence you may have to split the audio and use another amplifier if you would like to hear audio along with visualizing.

Code

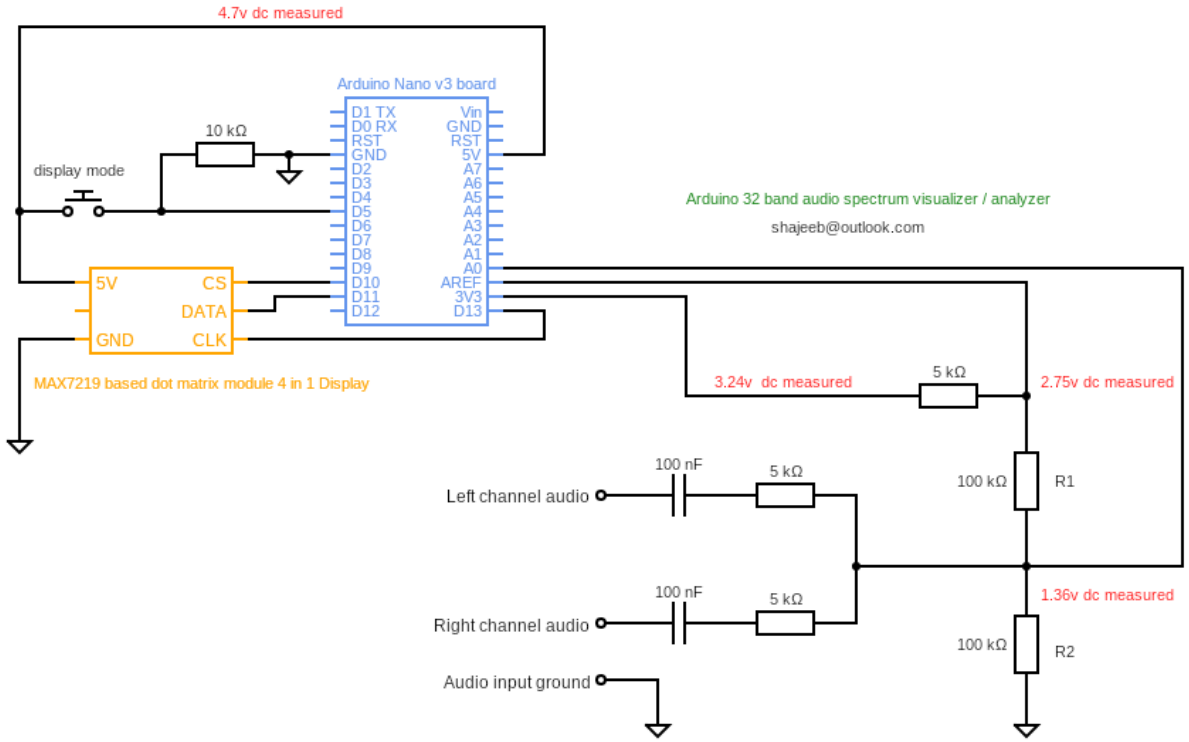
```
/*
Copyright (c) 2019 Shajeeb TM

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in
all
copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/

#include <arduinoFFT.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

#define SAMPLES 64            //Must be a power of 2
#define HARDWARE_TYPE MD_MAX72XX::FC16_HW   // Set display type  so that
MD_MAX72xx library treets it properly
#define MAX_DEVICES  4   // Total number display modules
#define CLK_PIN   13  // Clock pin to communicate with display
#define DATA_PIN  11  // Data pin to communicate with display
#define CS_PIN    10  // Control pin to communicate with display
#define  xres 32      // Total number of  columns in the display, must be <=
SAMPLES/2
#define  yres 8       // Total number of  rows in the display


int MY_ARRAY[]={0, 128, 192, 224, 240, 248, 252, 254, 255}; // default =
standard pattern
int MY_MODE_1[]={0, 128, 192, 224, 240, 248, 252, 254, 255}; // standard
pattern
int MY_MODE_2[]={0, 128, 64, 32, 16, 8, 4, 2, 1}; // only peak pattern
int MY_MODE_3[]={0, 128, 192, 160, 144, 136, 132, 130, 129}; // only peak +
bottom point
int MY_MODE_4[]={0, 128, 192, 160, 208, 232, 244, 250, 253}; // one gap in
the top , 3rd light onwards
int MY_MODE_5[]={0, 1, 3, 7, 15, 31, 63, 127, 255}; // standard pattern,
mirrored vertically


double vReal[SAMPLES];
double vImag[SAMPLES];
char data_avgs[xres];
```

```
int yvalue;
int displaycolumn , displayvalue;
int peaks[xres];
const int buttonPin = 5;    // the number of the pushbutton pin
int state = HIGH;             // the current reading from the input pin
int previousState = LOW;   // the previous reading from the input pin
int displaymode = 1;
unsigned long lastDebounceTime = 0;  // the last time the output pin was
toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the
output flickers


MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);   // display
object
arduinoFFT FFT = arduinoFFT();                                    // FFT
object



void setup() {

    ADCSRA = 0b11100101;      // set ADC to free running mode and set pre-
scalar to 32 (0xe5)
    ADMUX = 0b00000000;       // use pin A0 and external voltage reference
    pinMode(buttonPin, INPUT);
    mx.begin();           // initialize display
    delay(50);            // wait to get reference voltage stabilized
}

void loop() {
   // ++ Sampling
   for(int i=0; i<SAMPLES; i++)
    {
      while(!(ADCSRA & 0x10));        // wait for ADC to complete current
conversion ie ADIF bit set
      ADCSRA = 0b11110101 ;               // clear ADIF bit so that ADC can
do next operation (0xf5)
      int value = ADC - 512 ;                 // Read from ADC and subtract
DC offset caused value
      vReal[i]= value/8;                      // Copy to bins after
compressing
      vImag[i] = 0;
    }
    // -- Sampling


    // ++ FFT
    FFT.Windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
    FFT.Compute(vReal, vImag, SAMPLES, FFT_FORWARD);
    FFT.ComplexToMagnitude(vReal, vImag, SAMPLES);
    // -- FFT


    // ++ re-arrange FFT result to match with no. of columns on display (
xres )
    int step = (SAMPLES/2)/xres;
```

```
    int c=0;
    for(int i=0; i<(SAMPLES/2); i+=step)
    {
      data_avgs[c] = 0;
      for (int k=0 ; k< step ; k++) {
          data_avgs[c] = data_avgs[c] + vReal[i+k];
      }
      data_avgs[c] = data_avgs[c]/step;
      c++;
    }
    // -- re-arrange FFT result to match with no. of columns on display (
xres )


    // ++ send to display according measured value
    for(int i=0; i<xres; i++)
    {
      data_avgs[i] = constrain(data_avgs[i],0,80);            // set max &
min values for buckets
      data_avgs[i] = map(data_avgs[i], 0, 80, 0, yres);        // remap
averaged values to yres
      yvalue=data_avgs[i];

      peaks[i] = peaks[i]-1;    // decay by one light
      if (yvalue > peaks[i])
          peaks[i] = yvalue ;
      yvalue = peaks[i];
      displayvalue=MY_ARRAY[yvalue];
      displaycolumn=31-i;
      mx.setColumn(displaycolumn, displayvalue);              // for left to
right
     }
     // -- send to display according measured value

    displayModeChange ();         // check if button pressed to change
display mode
}

void displayModeChange() {
  int reading = digitalRead(buttonPin);
  if (reading == HIGH && previousState == LOW && millis() - lastDebounceTime
> debounceDelay) // works only when pressed

  {

   switch (displaymode) {
    case 1:    //       move from mode 1 to 2
      displaymode = 2;
      for (int i=0 ; i<=8 ; i++ ) {
        MY_ARRAY[i]=MY_MODE_2[i];
      }
      break;
    case 2:    //       move from mode 2 to 3
      displaymode = 3;
      for (int i=0 ; i<=8 ; i++ ) {
        MY_ARRAY[i]=MY_MODE_3[i];
      }
```

```
      break;
    case 3:    //     move from mode 3 to 4
      displaymode = 4;
      for (int i=0 ; i<=8 ; i++ ) {
        MY_ARRAY[i]=MY_MODE_4[i];
      }
      break;
    case 4:    //     move from mode 4 to 5
      displaymode = 5;
      for (int i=0 ; i<=8 ; i++ ) {
        MY_ARRAY[i]=MY_MODE_5[i];
      }
      break;
    case 5:    //      move from mode 5 to 1
      displaymode = 1;
      for (int i=0 ; i<=8 ; i++ ) {
        MY_ARRAY[i]=MY_MODE_1[i];
      }
      break;
  }

    lastDebounceTime = millis();
  }
  previousState = reading;
}
```

Schematics


4.7v dc measured
Arduino Nano v3 board
D1 TX
D0 RX
RST
GND
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
Vin
GND
RST
5V
A7
A6
A5
A4
A3
A2
A1
A0
AREF
3V3
D13
10 kΩ
display mode
Arduino 32 band audio spectrum visualizer / analyzer
shajeeb@outlook.com
5V
CS
DATA
GND
CLK
MAX7219 based dot matrix module 4 in 1 Display
3.24v dc measured
5 kΩ
2.75v dc measured
100 nF
5 kΩ
Left channel audio
100 kΩ
R1
100 nF
5 kΩ
Right channel audio
1.36v dc measured
100 kΩ
R2
Audio input ground