3rd November 2020

Microphone overload

alastair john underwood, GW0AJU

With the digital modes for voice transmissions, the ADC (analogue to digital convertor) does like an AM signal, have a top and tail signal limits into the encoding.

In other words, with an AM based system, be even an AM devised as a SSB signal, the amplitude of the voice signal can overload not only the final stages of the transmitter, but also the intervening stages of the radio transmitter.

A similar problem occurs with an ADC circuit, for any of the digital modes for voice translation into digital format.

To determine any microphone overload, then a microphone volume unit, “Vu” meter, would be handy. The 0dB indication point then the 100% modulation limits, and the red section into the +dB overloading of the carrier signal modulation.

However, with an FM signal, a microphone overload signal could just normally increase the carrier’s modulation index, hence the transmitters carrier signal deviation and thus RF bandwidth.

Apart from watching either the transmitters “ALC” meter or the “Tx power” meter readings, there not else that indicates the state of the microphone signal, apart from an audio signal report, or perhaps the carrier modulation monitor feedback switch into the headphones.

To help matters for my FT450d, I have constructed the Arduino project a “32 channel audio spectrum analyser” project.

https://create.arduino.cc/projecthub/shajeeb/32-band-audio-spectrum-visualizer-analyzer-902f51

As the project stands, the audio sample bandwidth of the DSP software is just short of 10KHz, sampling at 19200Hz. However, I have found that if the code line shown below:

```
 // ++ Sampling
   for(int i=0; i<SAMPLES; i++)
    {
      while(!(ADCSRA & 0x10));       // wait for ADC to complete current
conversion 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
```

If the code line “ ADCSRA = 0b11110101 ; “

and change the line to “ ADCSRA = 0b11110111 ; “

then the sampling rate is 9600Hz, equating to a sample audio bandwidth of 4·5KHz across the display.

Also, if the line “ vReal[i]= value/8; “

is altered to “ vReal[i]= value/8 * 0·5; “

then the amplitude of the display can be controlled. I divided the variable “vReal[i]” by two because the audio signal output from the FT450d data port for RTTY and Packet Radio etc, was too high in volume.

The new code version is thus:

```
// ++ Sampling
   for(int i=0; i<SAMPLES; i++)
    {
      while(!(ADCSRA & 0x10));      // wait for ADC to complete current
conversion ADIF bit set
      ADCSRA = 0b11110111 ;         // 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) * 0.5;    // Copy to bins after compressing
      vImag[i] = 0;
    }
    // -- Sampling
```

The above code correction will transform the use of the Arduino Uno audio spectrum analyser for ham radio or shortwave listener use, or even for CB operators.

Now there is a phenomena point regarding the microphone audio used for transmitters.

When using the audio analyser, you may discover after a while watching the LED dot matrix display, that bright sounding audio shows up quite well.

However, audio that is muffled by over processing the microphone audio, does show up quite so well. It is if as the processed audio has been stripped of its audio voice resonance components, and hence with a noise background to ones received radio

signal, the heavily processed voice then can disappear into the noisy background noise.

This happens even with a low level received signal, but a bright sounding audio with the voice signal resonance left with the processed audio, can still be heard to be understood by the ones hearing.

Once working, you may find the display brightness quite high. Looking around the displays command functions, I found a command to control the display brightness.

The modified “void setup() “ subroutine is shown below:

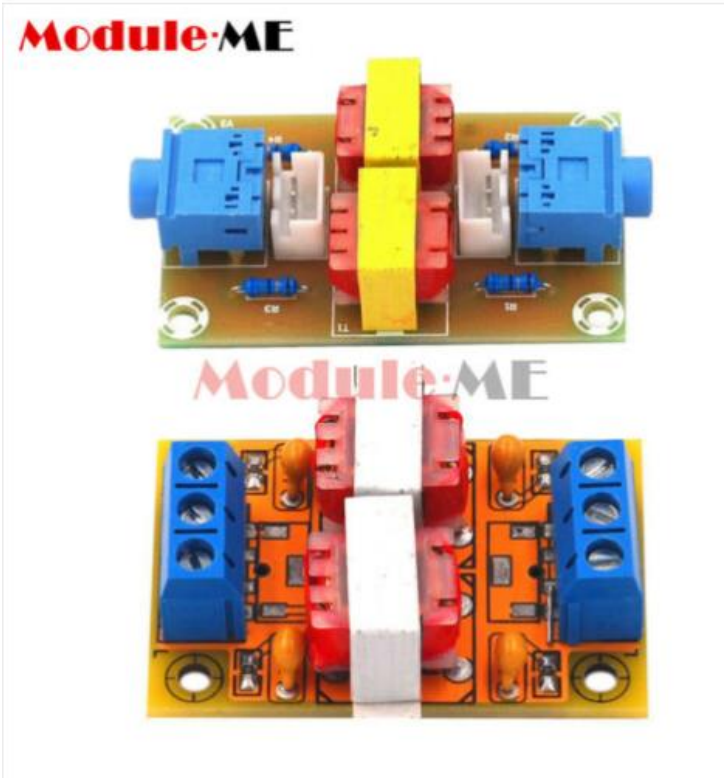
```
void setup()
{
 // ADCSRA = 0b11100101;      // set ADC to free running mode and set pre-scalar to 32 (0xe5)
 ADCSRA = 0b11100111;      // set ADC to free running mode and set pre-scalar to 32 (0xe5)
  ADMUX = 0b00000000;       // use pin A0 and external voltage reference

   pinMode(buttonPin, INPUT);
   mx.begin();          // initialize display
   delay(50);           // wait to get reference voltage stabilized
   mx.control(MD_MAX72XX::INTENSITY,0);
}
```

To stop electrical ground signal disturbance, use an audio isolation transformer, namely the link below:

https://www.ebay.co.uk/itm/XH-M372-Stereo-Audio-Isolator-Interference-Insulation-Transformer-Coupler-Module/153546492163?hash=item23c0158903:g:ikIAAOSwCwxdF1EY

> XH-M372 Stereo Audio Isolator Interference Insulation Transformer Coupler Module



The project is well worth the time to build. Below is the project link published image.