**C++ Code for Busy Detector For Phil**

**This Subroutine averages the bins of the FFT. The value 43.0664 below is the Hz per bin computed by taking the FFT sampling rate and dividing it by the size of the FFT. E.g. If the sampling rate were 12000 samples/sec from the sound card and the FFT size was 256 the value would be 12000/256 or 46.875.**

// \*\*\*\* Sub AvgFFTBins \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Status: Confirmed OK 3/25/2020 Rev 7.3.2 \*\*\*\*\*\*\*\*\*\*\*\*\*

void AvgFFTBins(boolean blnInit, float fltKavg, int intLoFHz, int intHiFHz)

// 3/25/2020 Good results for fltKavg = .1, averaging every 20 bins (~220ms)

// 3/26/2020 Improved with using 2x fltKavag on attack or .5x fltKavg on release

{

float fltTemp = 0; int intHiBin = intHiFHz/43.0664; int intLoBin = intLoFHz /43.0664;

float fltAvgBinSum = 0.0;

if (blnInit)//Initialize the average bins

{

for (int i = 1;i <151; i++) //Skip bin 0 (DC component)

{

fltFFT[i] = 4000.0 \* fft1024.read(i);

if ((i >= intLoBin)&&(i <=intHiBin)){fltAvgBinSum += fltFFT[i];}

}

fltAvgBinsOfInt = fltAvgBinSum/(1 + intHiBin - intLoBin);//Initialize fltAvgBinsOfInt on first call with blnInit = true

return;

}

else//Average in new values with fast attack, slow release Rolling Avg Filter

{

for (int i = 1; i < 151; i++)// Modified 3/26/2020 to use fast attack slow release Rolling Avg Filter

// This definitely helps by "holding" the busy condition for a while following an indication.

{

fltTemp = 4000.0 \* fft1024.read(i);

if ((i > intHiBin) || (i < intLoBin))

{//Normal = attack and release for bins outside the band of interest

fltFFT[i] = ((1.0 -( fltKavg)) \* fltFFT[i]) + ( fltKavg \* fltTemp);

}

else if (fltTemp > fltFFT[i])//Faster attack using 2x fltKavg

{

fltFFT[i] = ((1.0 -(2\* fltKavg)) \* fltFFT[i]) + (2 \* fltKavg \* fltTemp);

}

else // Slower release using .5x fltKavavg

{

fltFFT[i] = ((1.0 - (.2 \*fltKavg)) \* fltFFT[i]) + (.2 \* fltKavg \* fltTemp);

}

}

return;

}

}/// End Sub AvgFFTBins \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Subroutine SearchRatioDetect This Function searches for the largest and smalles group of FFT bins of intSearchWidthHz bins looking for bins above intThreshdB**

//\*\*\*\*\*\*\*\* Sub SearchRatioDetect\*\*\*\*\*\*\*\*\*\*\*\*\*Status: Works well for both narrow and wide band signal detection...Still optimizing\*\*\*\*\*\*\*\*

int SearchRatioDetect(int intLowFHz, int intHiFHz, int intSearchWidthHz, int intThreshdB)

{

int intLowBin = round(intLowFHz/43.0664);int intHiBin= round(intHiFHz/43.0664);

int intSearchWidthBins = round(intSearchWidthHz/43.0664); int intMaxS\_NdB;int intIatMax =0; int intIatMin = 0; int intMinS\_NdB;

float fltMaxDetect = 0.0; float fltSearchSum = 0.0; float fltSearchSumMax =0.0; float fltSearchSumMin = 1000000;

float fltHi\_LowSum = 0.0; float fltDenom = 0.0; float fltNum = 0.0; float fltMaxSN =0;

// Search for largest AND smallest "intSearchWidthHz" signal (between intLowBin and intHighBin inclusive)

for (int i=intLowBin; i<= (intHiBin); i++)

{

fltHi\_LowSum = fltHi\_LowSum + fltFFT[i];//Sums all the bins between intLowBin and intHighBin inclusive

if (i <= (intHiBin - intSearchWidthBins)) //This bounds the search to the region of interest

{

fltSearchSum = 0;// Search for the max and min contiguous bins of width intSearchWidthBins

for (int j= 0;j <intSearchWidthBins; j++)

{

fltSearchSum += fltFFT[i + j];// Sum all the bins within intSearchWidthBins

}

if (fltSearchSum > fltSearchSumMax)

{

fltSearchSumMax = fltSearchSum;//The max sum found in a range of intSearchWidth

intIatMax = i;

}

if (fltSearchSum < fltSearchSumMin)

{

fltSearchSumMin = fltSearchSum; //The minimum sum found in a range of intSearchWidth

intIatMin = i;

}

}

}

// Generate Numerator and Denominator normalized to per bin for the Highest S:N

fltDenom =(fltHi\_LowSum -fltSearchSumMax)/((intHiBin- intLowBin)-intSearchWidthBins);//Avg energy/bin outside region of max

fltNum = fltSearchSumMax/intSearchWidthBins;//Maximum Energy per bin over intSearchWidthHz between intLowBin and intHighBin

intMaxS\_NdB = round(20\*log10(fltNum/fltDenom));

if (intMaxS\_NdB >= intThreshdB)//This trips detector when a narrow band peak is found

{

blnPlotBusyRed = true;

return intMaxS\_NdB;

}

// Generate the Numerator and Denominator normalized to per bin for the Lowest S:N

fltDenom = (fltHi\_LowSum - fltSearchSumMin)/((intHiBin- intLowBin)-intSearchWidthBins);//Avg energy/bin outside region of max

fltNum = fltSearchSumMin/intSearchWidthBins;//Minimum Energy per bin over intSearchWidthHz between intLowBin and intHighBin

intMinS\_NdB = round(20\*log10(fltNum/fltDenom));//Typically -3 to -10

if ((intMaxS\_NdB - intMinS\_NdB) >= (2 \* intThreshdB))//This trips the detector when a narrow band minimum is found

{

blnPlotBusyRed = true;

//Serial.print("MaxS\_N=");Serial.print(intMaxS\_NdB);Serial.print(" MinS\_N=");Serial.println(intMinS\_NdB);

return (intMaxS\_NdB - intMinS\_NdB);

}

else if (intMinS\_NdB <= (intThreshdB -2))//Only clear the busy condition when intMinS\_NdB is 2 dB or more BELOW threshold

{

blnPlotBusyRed = false;

return 0;

}

else return intMaxS\_NdB;

}//End\*\*\*\*\*\*\*\*\*\*\*\*\* SearchRatioDetect \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Calling code for the prior two subroutines:**

if (fft1024.available()) This is Teensy code that says the FFT (in this case 1024 samples of 44.1KHz sample rate or ~43 Hz/bin has been computed and is available

{

AvgFFTBins((intBinsAveraged == 0), .2 ,intBusyBWLoHz,intBusyBWHiHz);//Average the fltFFT[] bins, initialize if intBinsAveraged == 0

intBinsAveraged +=1;

if (intBinsAveraged > intAvg)//If enough averages completed

{

intDetectSN = SearchRatioDetect(intBusyBWLoHz, intBusyBWHiHz, 258, intThresh);//Search and detect Region of Interest for 258 Hz BW)

if ((intDetectSN >= intThresh ) && (!blnChanBusySent) && blnEnbBusyDetect)

{

Serial.print("DETECT:"); Serial.println(intDetectSN);

blnChanBusySent = true;

blnChanClearSent = false;

}

else if ((intDetectSN == 0) && (!blnChanClearSent) && blnEnbBusyDetect)

{

Serial.println("DETECT:0");

blnChanClearSent = true;

blnChanBusySent = false;

}

}

if (0 == (intBinsAveraged % intAvg))// Only plot every intAvg averages (about 11ms /average)

{

if (blnPlotSpectrum)

{

PlotSpectrum(fltFFT,intBusyBWLoHz,intBusyBWHiHz,blnInitSpectrum,intBandwidth);

blnInitSpectrum = false;// only init on first call to print headers.

//Typical time for 20 bin averages ~ 220 ms

//Serial.print("Time to avg 20 bins: ");Serial.println(millis() - ulngLastSpectrumUpdate);

ulngLastSpectrumUpdate = millis();

}

if (intBinsAveraged > 100000){intBinsAveraged = 0;}//To prevent overflow of intBinsAveraged

}

}

}