Arduino FFT Library

FFT Functions

There are multiple functions you can call to perform the FFT. The reason they are broken up into these sections, is so you can tailor the FFT to your needs. If you don't neet particular parts, you can eliminate them and save time. There are 2 pre-processor functions fft_reorder() and fft_window(), the main function fft_run(), and 4 output options: linear 16b, linear 8b, logarithmic 8b, and octave 8b.

fft_run() - This is the main FFT function call. It takes no variables and returns no variables. It assumes there is a block of data already in SRAM, and that it is already re-ordered. The data is stored in array called fft_input[], which contains 2 16b values per FFT data point - one for real, the other for imaginary. If you are filling the array yourself, place the real values in the even bins, and the imaginary values in the odd bins. For example:

- fft_input[0] = real1, fft_input[1] = imaginary1
- fft_input[2] = real2, fft_input[3] = imaginary2

Therefore, there are 2 times as many points in the array as there are FFT bins. If you are using only real data (i.e. values sampled from the ADC), then put those values into the even numbered bins, and be sure to write 0 to the odd valued bins.
The final output is kept in fft_input[], with the even bins being the real magnitudes, and the odd bins being the imaginary magnitudes. The bins are in sequence of increasing frequncy. For example:

- fft_input[0] & fft_input[1] = bin0 magnitudes (0hz -> Fs/N)
- fft_input[2] & fft_input[3] = bin1 magnitudes (Fs/N -> 2Fs/N)

You will have to run one of the magnitude functions to get useful data from these bins.

fft_reorder() - This reorders the FFT inputs to get them ready for the special way in which the FFT algorithm processes data. Unless you do this yourself with another piece of code, you have to call this before running fft_run(). It takes no variables and returns no variables. This runs on the array fft_input[], so the data must be first filled into that array before this function is called.

fft_window() - This function multiplies the input data by a window function to help increase the frequency resolution of the FFT data. It takes no variables, and returns no variables. This processes the data in fft_input[], so that data must first be placed in that array before it is called. It must also be called before fft_reorder() or fft_run().

fft_mag_lin8() - This gives the magnitude of each bin in the FFT. It sums the squares of the imaginary and real parts, and then takes the square root, rounding the answer to 8b precision (it uses a lookup table, and scales the values to fit the full 8b range. It takes no variables, and returns no variables. It operates on fft_input[], and returns the data in an array called fft_lin_out8[]. You can then use the data in fft_lin_out8[]. The magnitude is only calculated for the first N/2 bins, as the second half of an FFT is identical to the first half for all real inputs. Therefore, fft_lin_out8[] has N/2 8b values, with each index representing the bin order. For example:

- fft_lin_out8[0] = first bin magnitude (0hz -> Fs/N)
- fft_lin_out8[1] = second bin magnitude (Fs/N -> 2Fs/N)

The output can be scaled to maximize the resolution using the SCALE factor. See the

#define section for more detials.

fft_mag_lin() - This gives the magnitude of each bin in the FFT. It sums the squares of the imaginary and real, and then takes the square root. It uses a lookup table to calculate the square root, so it has limited precision. You can think of it as an 8b value times a 4b exponent. It covers the full 16b range, but only has 8b of precision at any point in that range. The data is taken in from fft_input[] and returned in fft_lin_out[]. The values are in sequential order, and there are only N/2 values total, as the FFT of a real signal is symetric about the center frequency.

fft_mag_log() - This gives the magnitude of each bin in the FFT. It sums the squares of the imaginary and real, and then takes the square root, and then takes the log base 2 of that value. Therefore, the output is compressed in a logarithmic fashion, and is essentially in decibels (times a scaling factor). It takes no variables, and returns no variables. It uses a lookup table to calculate the log of the square root, and scales the output over the full 8b range {the equation is $16*(\log_2((img^2 + real^2)^{1/2}))$}. It is only an 8b value, and the values are taken from fft_input[], and returned in fft_log_out[]. The output values are in sequential order of FFT frequency bins, and there are only N/2 total bins, as the second half of the FFT result is redundant for real inputs.

fft_mag_octave() - This outputs the RMS value of the bins in an octave (doubling of frequencies) format. This is more useful in some ways, as it is closer to how humans perceive sound. It doesn't take any variables, and doesn't return any variables. The input is taken from fft_output[] and returned in fft_oct_out[]. The data is represented as an 8b value of 16*log2(sqrt(mag)). There are LOG_N bins, and they are given as follows:

- FFT_N = 256 : bins = [0, 1, 2:4, 5:8, 9:16, 17:32, 3:64, 65:128]
- FFT_N = 128 : bins = [0, 1, 2:4, 5:8, 9:16, 17:32, 3:64]

Where, for example, (5:8) is a summation of all bins, 5 through 8. The data for each bin is squared, imaginary and real parts, and then added with all the squared magnitudes for the range. It is then divided down by the number of bins (which can be turned off - see #defines section), and the square root is taken, followed by the log being computed.

#Define options

These values allow you to modify the FFT code to fit your needs. For the most part, they just turn off stuff you aren't using. By default most functions are off, so you will have to turn them on to use them. You must also declare these #defines before the #include statements in your sketch.

FFT_N - Sets the FFT size. Possible options are 16, 32, 64, 128, 256. 256 is the defualt

SCALE - Sets the scaling factor for fft_mag_lin8(). Since 8b resolution is pretty poor, you will want to scale the values to max out the full range. Setting SCALE multiplies the output by a constant before doing the square root, so you have maximum resolution. It does consume slightly more resources, but is pretty minimal. SCALE can be any number from 1 -> 255. By default it is 1. 1, 2, 4, 128, and 256 consume the least resources.

WINDOW - Turns on or off the window function resources. If you are not using fft_window(),

then you should set WINDOW 0 (off). By default its 1 (on).

REORDER - Turns on or off the reorder function resources. If you are not using fft_reorder(), then you should set REORDER 0 (off). By default its 1 (on).

LOG_OUT - Turns on or off the log function resources. If you are using fft_mag_log(), then you should set LOG_OUT 1 (on). By default its 0 (off).

LIN_OUT - Turns on or off the lin output function resources. If you are using fft_mag_lin(), then you should set LIN_OUT 1 (on). By default its 0 (off).

LIN_OUT8 - Turns on or off the lin8 output function resources. If you are using fft_mag_lin8(), then you should set LIN_OUT8 1 (on). By default its 0 (off).

OCTAVE - This turns on or off the octave output function resources. If you are using fft_mag_octave(), then you should set OCTAVE 1 (on). by default it is 0 (off).

OCT_NORM - This turns on or off the octave normilisation feature. This is the part of fft_mat_octave() that divides each bin grouping by the number of bins averaged. Since a lot of sound sources are pink noise (they drop off in amplitude as the frequency increases), the scale tends to drop off rather quickly. This artificially boosts the higher frequencies when off (OCT_NORM 0). By default, the normilisation is on (OCT_NORM 1).