Abstract

This project aims at software implementation of signal spectrum analyzer algorithm on Xilinx ZynqTM-7000 with minimal execution time. OLED display is used to display the frequency spectrum of the input audio samples. There are five stages involved in this project such as audio samples capture, FFT computation, magnitude calculation, moving window averaging and the noise cancellation. Input audio signal is sampled by ADAU1761 chip (Audi codec) at 96 KHz and the sampled values are sent to Zynq FPGA through I2S interface for further processing. Execution time achieved for this project is **2707us.**

FFT

This module performs 128 point radix 2 decimation in time Fast Fourier transform on incoming audio samples. There are 7 stages involved in 128 point FFT computation. Imaginary input to this module is always driven zero since the input sequence is real. To minimize the execution time redundant multiplications and memory read/write operations are optimized by ignoring imaginary parts at first and second stages of the FFT because of real nature of the input samples. Output of the FFT module is sent to the magnitude module followed by the average and noise cancellation modules.

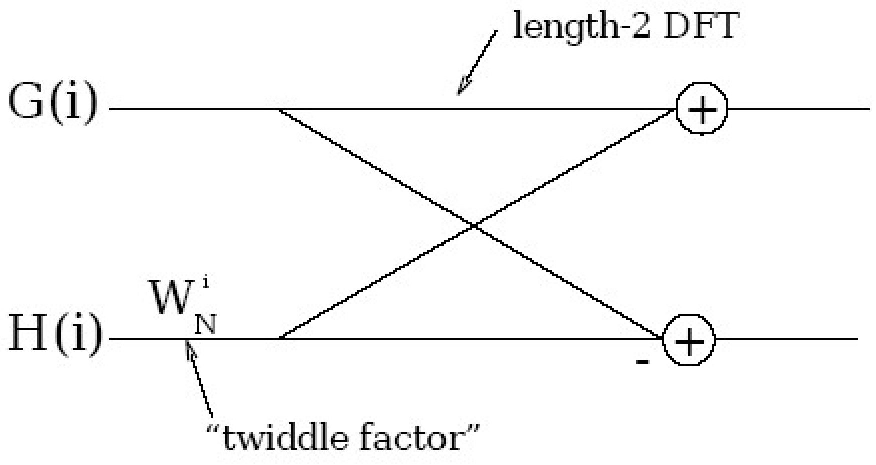


Figure 1: Radix 2 butterfly

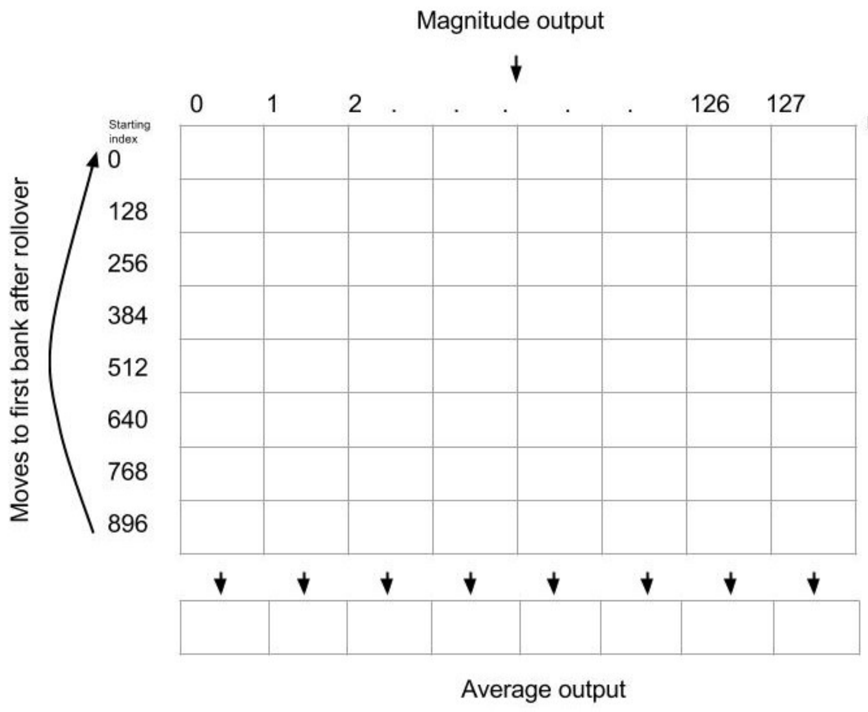
Magnitude

$$\text{Magnitude} = Re^2 + Im^2$$

This module operates on output from FFT module. Magnitude is calculated as sum of square of real and imaginary parts. Magnitude values are stored into a window buffer of 128x8 in size for moving average computation, here 128 is the number of sample in one window and totally there are 8 windows. This buffer is kind of ring buffer, Whenever it receives a new window of samples after calculating magnitude it is stored to the next available bank as shown in **Fig 2**.

Averaging

Window averaging is implemented by taking average of eight samples (one sample from each of 8 windows) with the help of ring buffer as depicted below. 128 locations in window buffer is called bank, samples are stored starting from bank 0 when the buffer is filled with samples of 8 windows, then the next incoming window is stored in bank 0 which is called rollover. Output from Averaging module is sent to Noise cancellation module.



Each 128 location is called as bank

Pseudocode - ring buffer

```
index = bank * 128

for i = 0 to 128
        write (index++, data
[i])

/*Rollover to start*/
if bank is 7 then
   bank = 0
else bank++
```

Figure2: Window buffer structure

Noise Cancellation

Adaptive noise cancellation technique is used to cancel the ambient noise prevailing in the samples. This technique is quite advanced than just subtracting constant noise value, since the system can adapt to any situation according to noise in the place. Ambient noise is calculated for the first one second to measure the noise in the vicinity which can be subtracted from samples to effectively remove the noise.

Optimizations

As mentioned already the execution time achieved is 2707us, this was possible by optimizing the code at each and every module effectively. **Initially without optimizations the latency was close to 190 us excluding *get_audio* function, after applying the below mentioned optimizations latency is reduced drastically by 60us to 130us.** Most of the optimizations are based on symmetric property of FFT.
According to the symmetric property of FFT, if the input sequence is real then the FFT spectrum of that sequence is symmetric in nature. In mathematical form this property is expressed as X[N-k] =X[k] where N implies it is N point FFT.

Followings are the optimizations performed for our design.

1. Magnitude module is restricted to perform only 64 magnitude computations for each window instead of 128. This technique saves 128 memory read, 64 additions and 128 square operations per each 128 samples window.
2. As we are performing decimation in time FFT, input samples are needed to be arranged in bit reversal order. Instead of performing bit reversal at the time of FFT calculation, it is done while storing input samples to the FFT buffer. This optimizes 128 memory read/writes per each window.
3. Redundant multiplications and memory read/write operations are optimized by ignoring imaginary parts at first and second stages of the FFT because of real nature of the input samples.
4. Similar optimization technique (Mentioned in point 1) is been adopted for Average module.