Team Veriloggers

Hilbert Transform Implemantation on FPGA

Anadi Chaman
Ankit Raj
Saurav Kumar
Vishnu Lokhande

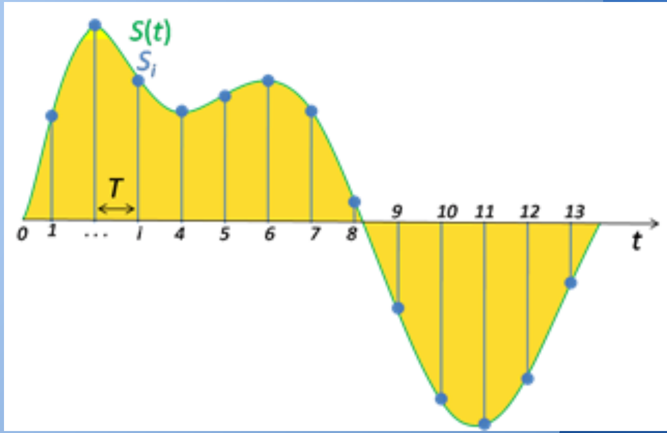
Fourier Transform

Tranformation of a signal from time domain to frequency domain.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x \xi}\, dx,$$ for any real number ξ.

Discrete Fourier Transform: X(k)

We use samples of the continuous signal sampled at a particular period satisfying Nyquist Criteria for reconstruction.



T is the sampling period. 2*pi/T > Bandwidth of the input. Our choice of T is pi/1600.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}.$$

Discrte Fourier Transform: X(k)

The X(k) represents the envelope of fourier transform at different frequencies.
We can obtain the original signal by simply applying Inverse Fourier Transform.
It can easily be observed that X(k)/N represent the fourier series coefficients.

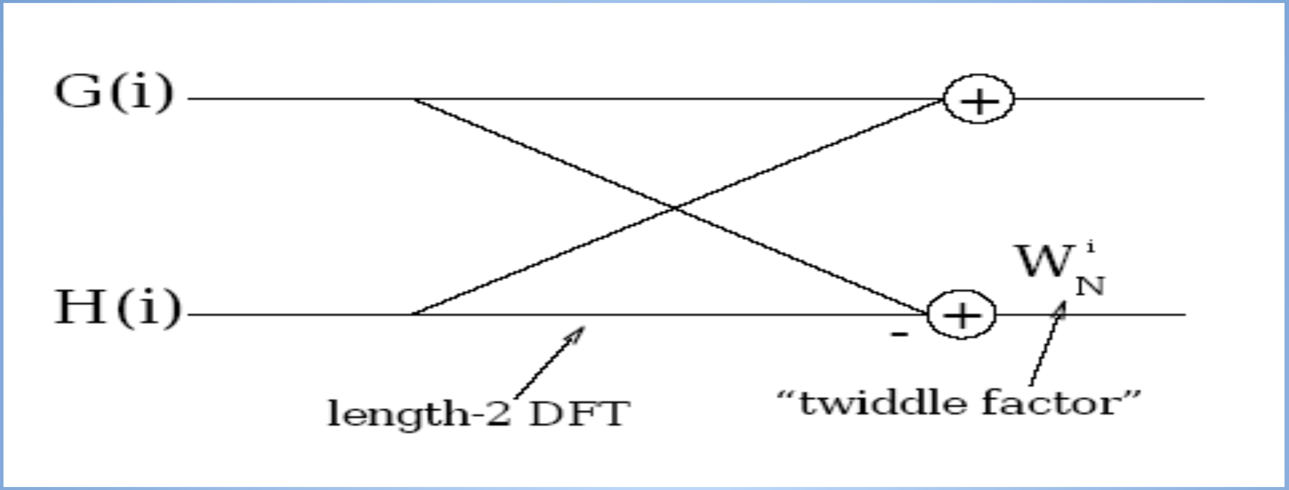
$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1} X(k)\, e^{\frac{i2\pi nk}{N}}, n=0\ldots N-1$$

Fast Fourier Transform

A fast Fourier Transform is an algorithm to compute the discrete fourier transform(DFT) and its inverse. Evaluating DFT by its definition requires O(n^2). However, FFT reduces the number of operations to O(N logN).

To find the FFT of the input, we have used the Cooley-Tukey Algorithm. It involves dividing the transform into two pieces of size N/2 at each stage, and is ,therefore, limited to power of two samples(also called Radix 2).

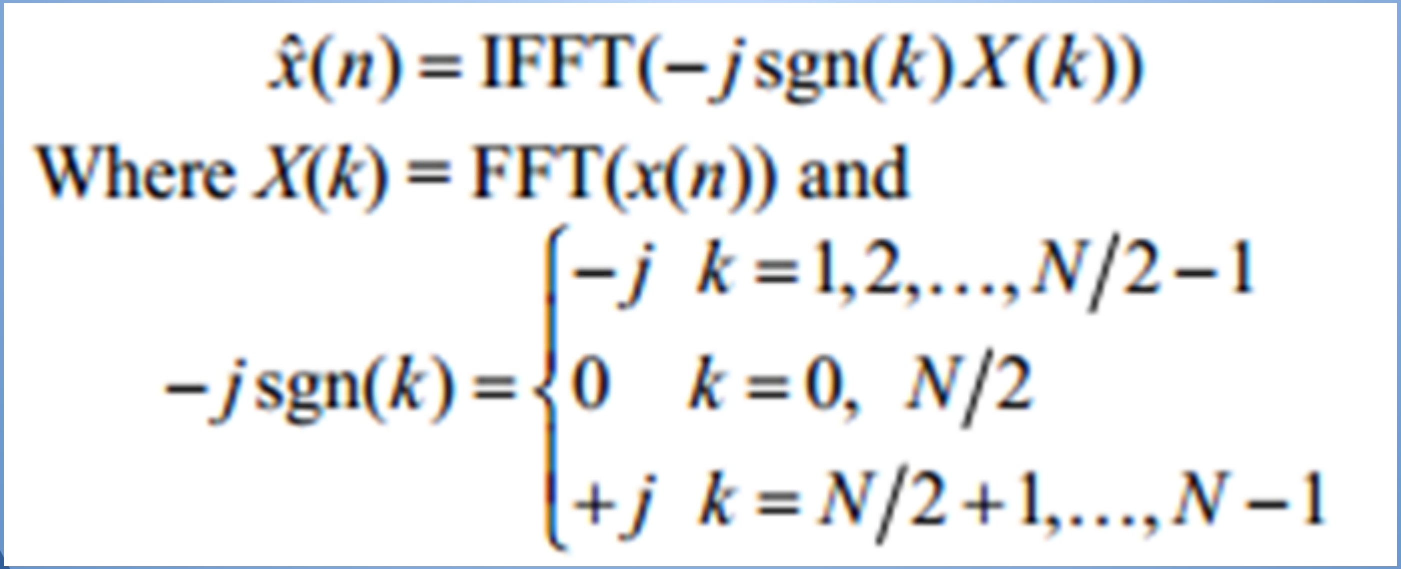
Butterfly diagram and Twiddle factors



The value of the twiddle factor Wn depends on the stage of the Cooley-Tukey Algorithm being worked upon, and the value of the terms acting as the input for the 2 point DFT of that stage.

Implementation of Hilbert Transform

To obtain the hilbert transform of an input, we find an FFT of the input sequence and then multiply the sequence by the fourier transform of the hilbert function. Then we take the inverse fast fourier transform to get the final hilbert transform.



$$\hat{x}(n) = \mathrm{IFFT}(-j\,\mathrm{sgn}(k)X(k))$$

Where $X(k) = \mathrm{FFT}(x(n))$ and

$$-j\,\mathrm{sgn}(k) = \begin{cases} -j & k = 1, 2, \ldots, N/2 - 1 \\ 0 & k = 0,\ N/2 \\ +j & k = N/2 + 1, \ldots, N - 1 \end{cases}$$

IMPLEMENTATION IN VERILOG

1. Storing Numbers:

- Using a 16 bit representation
- First bit is sign bit, rest 15 used for storing absolute value
- Storing just the integer part of any number anytime.

Number System

- 16 bits will let us store values from -65535 to 65535
- During the intermediate calculations, values ~1500 were seen, still we are using a number system which can safely handle numbers of "Third Case".

How to store sine and cos?

Values of sine and cosine for twiddle factor. Since sines and cosines are <=1, we have stored sines and cosines by multiplying it with 2**15.

How to handle multiplications?

All the multiplications involved in whole process are either with sine or with cosine, so after the multiplication (which generated 30+1 bit number), we are dividing it by 2**15, giving us back 16 bit number which can be safely stored in our number system.

File Descriptions

adder.v : module adder(a,b,c)

- Adds a and b, and puts in c
- Sign sensitive
- No worry for overflow : extra bits in number system

File Description

module multiplier(a,b,c)

- Since all multiplications are with sines and cosines, and our number system allows us to drop 15 least significant bits from the product.

File Description

module butterfly (argument list)

- Let us go through the code.

Top level verilog code: fft.v

Finding Inverse Transform using same Twiddle Factors

We have derived the relation of the fourier transform with its original signal without taking conjugate of the previous twiddle factor.If we interchange the real and imaginary parts of each fourier transform and then apply inverse fourier transform with previous twiddle factors , we will get a result, swapping whose real and imaginary parts gives us the desired answer.