FFT

Problem definition

In digital processing, Fast Fourier Transform represent one of the most important algorithms. It's used to convert the signal from Time to frequency domain so we can processing our chain on it. FFT is a way to implement DFT.

DFT complexity

- In DFT we compute N frequency domain points
- Each frequency point is calculated by performing N complex multiplications then N complex additions
- Thus we need N^2 complex multiplies and N^2 complex adds

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi jkn/N} k = 0, 1, \ldots, N-1$$

The idea of the FFT is to divide the DFT to two parts even and odd

FFT

- FFT is mathematically identical to DFT in floating point
- The FFT, however, is faster
- The complexity of FFT is O(NlogN) instead of O(N^2)
- This saving can vary between FFT algorithms and a solid bound is very hard to define, but the log limit is good

And best for implement is radix 2

We divide the equation to even and odd and will notice that the values after $\frac{N}{2}$ is the same

Means $x_n = x_{n+\frac{N}{2}}$,

FFT decimation in time – Step 1

Begin with normal FFT

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi jkn/N} k = 0,1,\ldots,N-1$$

Now divide the N-point FFT into two N/2 point FFTs

One for the even and one for the odd indices

(summation is linear, so there is no effect)

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-2\pi jk(2m)/N} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-2\pi jk(2m+1)/N} k = 0,1,\ldots,N-1$$

Using SDF “Single delay line feedback” better than using MDF” memory” has much lower latency cause in FFT we need the whole sample in 16-FFT we need to wait the 16 sample so we have latency equal to 16 sample we need not to increase that latency so we used SDF

An N-point DFT of sequence x[n] is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \;, k = 0,1,2, \dots N-1$$

$$W_N^{nk} = e^{-i2\pi nk/N} = \cos(\frac{2\pi nk}{N}) - j\ \sin(\frac{2\pi nk}{N})$$

Where k is frequency index, n is time index and the twiddle factor W_N^{nk} are the complex roots of unity, they are symmetric and equally spaced on the unity circle.

The radix 2^2 was formulated using $K = 2$ dimension linear index mapping. Radix 2^2 algorithm can be expressed as various formula using Common factor algorithm factor algorithm. The Radix 2^2 algorithm is given as follows. Applying a 3 dimensional linear index mapping.

The Radix 2^2 is derived by writing the equation

By substituting for n and k in above equation

$$k = k_1 + 2k_2 + 4k_3 \quad , \qquad n = \frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3$$

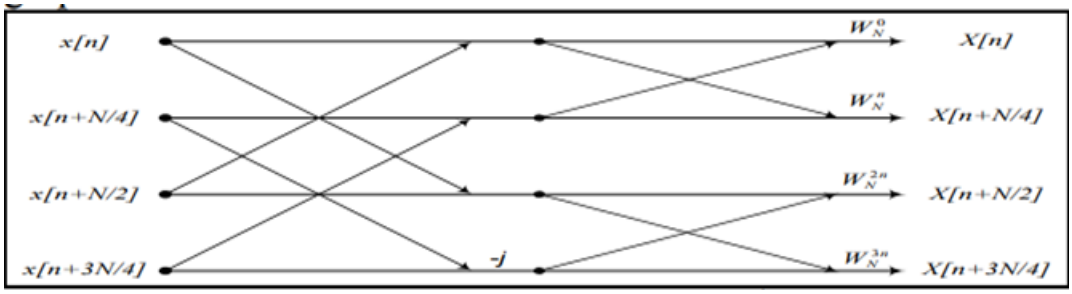
$$(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \sum_{n_1=0}^{1} x\left(\frac{N}{4}n_1 + \frac{N}{2}n_2 + n_3\right) W_N^{\left(\frac{N}{4}n_1 + \frac{N}{2}n_2 + n_3\right)(k_1 + 2k_2 + 4k_3)}\ X(k_1 + 2k_2 + 4k_3)$$

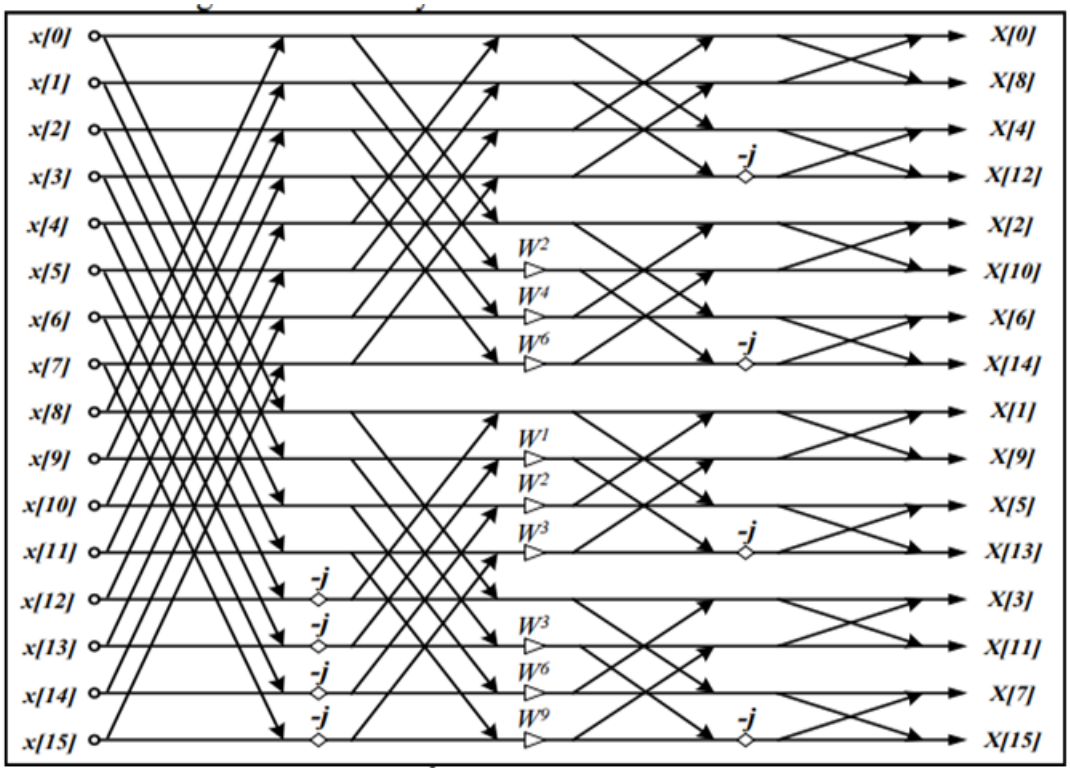
$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1} \left[H(k_1, k_2, n_3)\, W_N^{n_3 * (k_1 + 2k_2)}\right] *\ W_{\frac{N}{4}}^{n_3 * k_3}$$

$$\text{Where } 0 \le n_3 \le \frac{N}{4} - 1 \ , \ \ 0 \le k_3 \le \frac{N}{4} - 1$$

$$H(k_1, k_2, n_3) = \left[x(n_3) + (-1)^{k_1} * x\left(n_3 + \frac{N}{2}\right)\right] +\ (-i)^{k_1 + 2k_2} * \left[x\left(n_3 + \frac{N}{4}\right) + (-1)^{k_1} * x\left(n_3 + \frac{3N}{4}\right)\right]$$

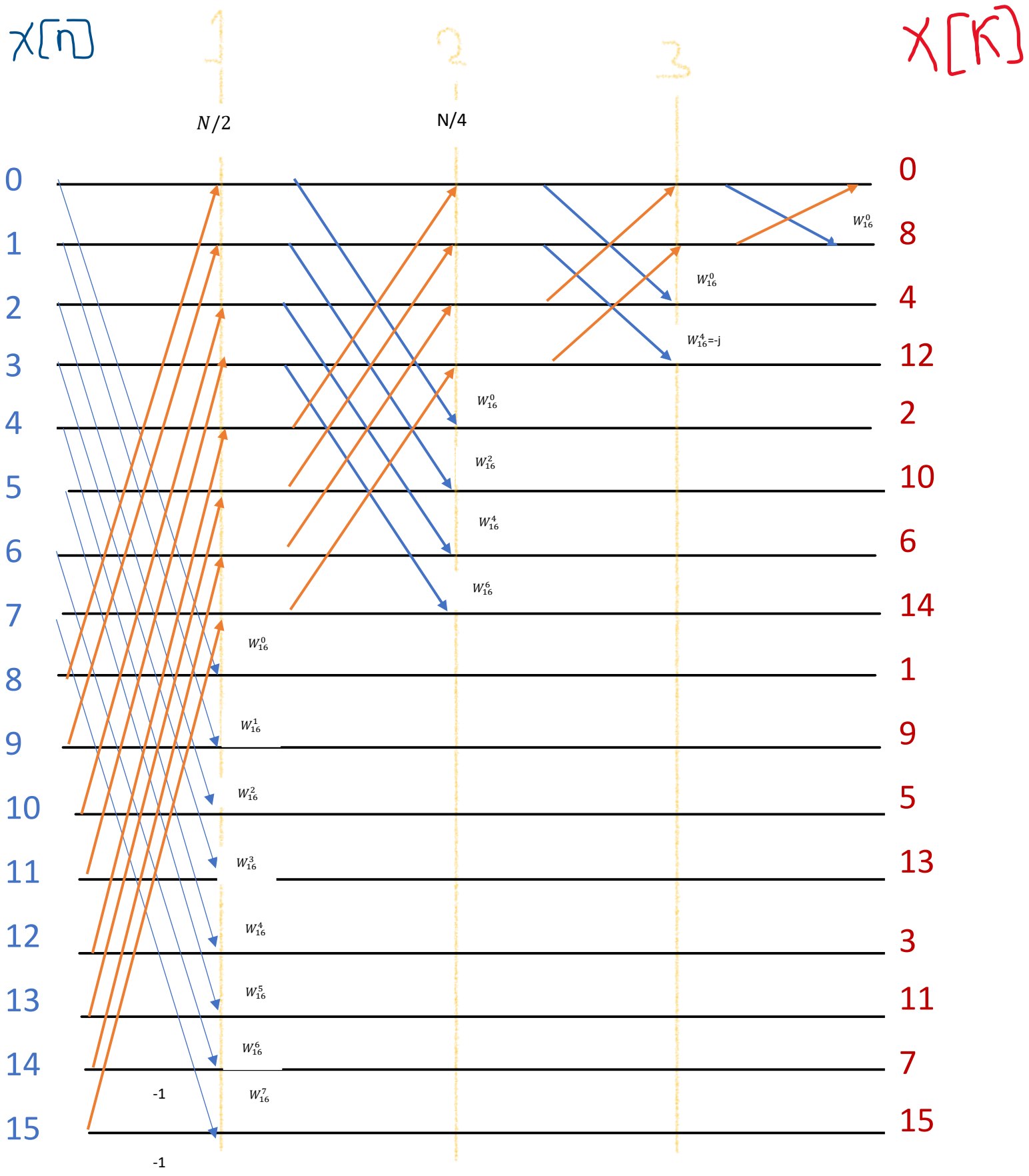
Each term in equation represents a Radix-2 butterfly (BFI) and the whole equation also represents a bigger Radix-2 butterfly (BFII) with trivial multiplication by – j separating the two BFI butterflies. RAD2 and RAD4 are not the best choice when it comes to area and power, also the split radix has irregular design which make it not suitable for digital design. Therefore, Radix 2^2 is the one which is implemented in





$$X_k = \sum_{m=0}^{N/4-1} x_{2(2m)} W_N^{2(2m)k} + W_N^{2k} \sum_{m=0}^{N/4-1} x_{2m+1} W_N^{2(2m)k}$$

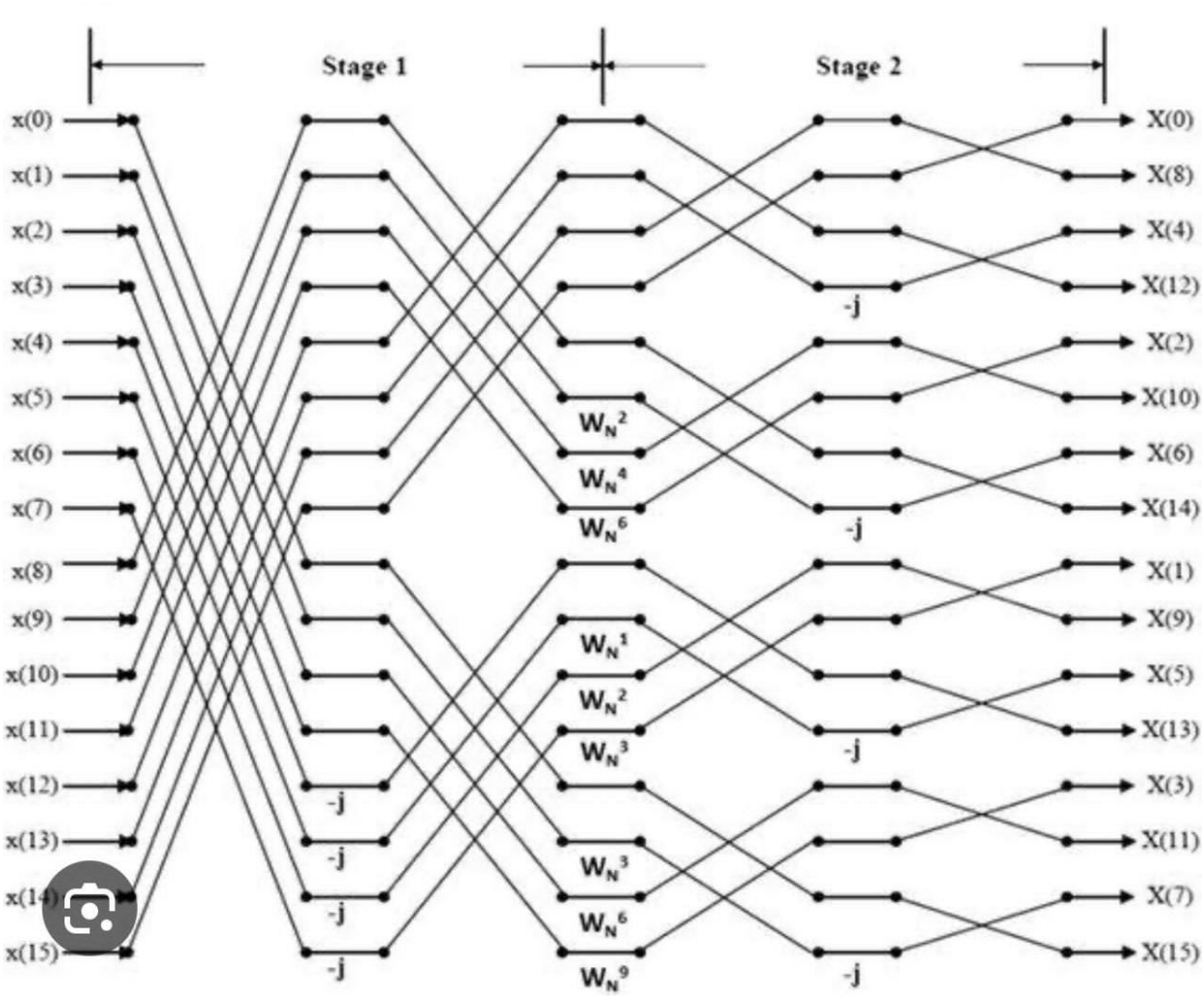
$$+W_N^k \{ \sum_{m=0}^{N/4-1} x_{2(2m+1)} W_N^{2(2m)k} + W_N^{2k} \sum_{m=0}^{N/4-1} x_{2(2m+1)+1} W_N^{2(2m)k} \}, k = 0,1,...,N-1$$



Implement of radix 2

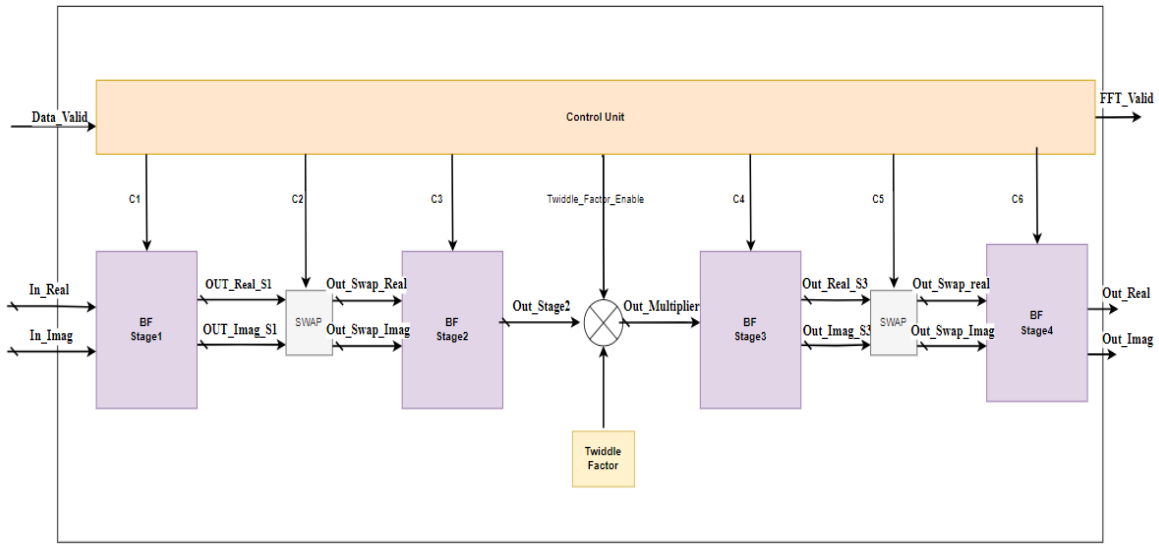
But radix 2 ^2 is better for implement as hardware because it will remove 1 multiplication block from the hardware

Implement of radix 2^2

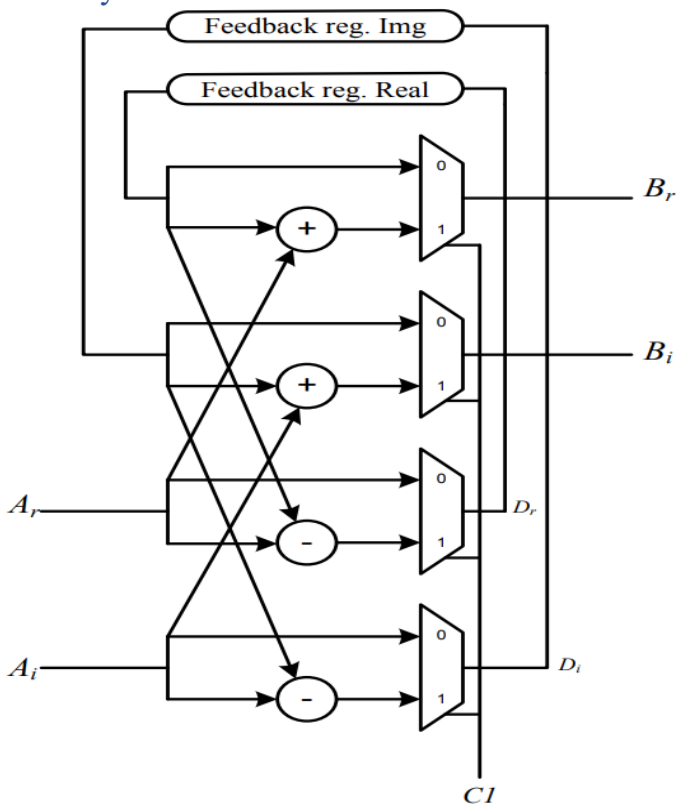


If we saw first stage we need to multiply the value from $x_{8\ldots15}$ $with\ W^{0\ldots7}$ which means 8 complex multiplier if we want to get that in parallel together, but we can reduce that number according to the clocks of our input , say our block works with 240KHZ and the input is series so we need to wait at first 16 clock to get the 16 carrier symbols. then the next input will be after latency 16 clock cycles so we just need in first stage 1 complex multiplier which the 1 complex multiplier that consists of 3 real multiplier and five real adder/subtractor

Design



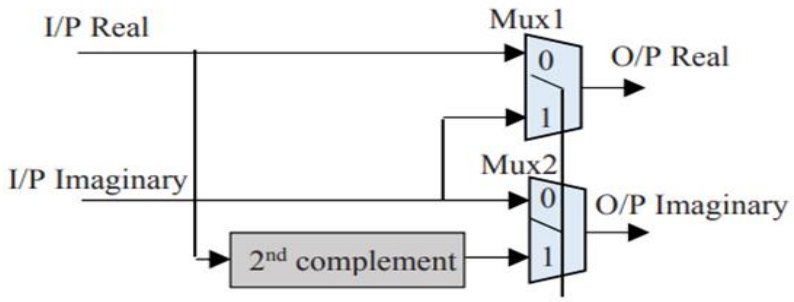
Butter fly module



- First 8 clocks will get save from $x_0 to\ x_7$ and make the c1 =0,
- then the next 8 clock cycles ($a_{re,im} + b_{re,im}$) by make C1 =1 and save ($a_{re,im} - b_{re,im}$) in the reg file in the same time
- Then the next 8 clock cycles will read the from the reg file the values of ($a_{re,im} - b_{re,im}$) from $x_{8\ldots15}$

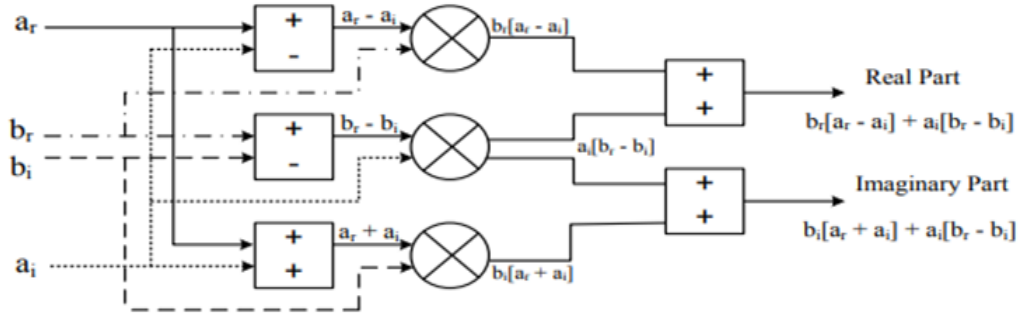
- In that 8 clock cycles we read from reg file we write the new values of the next 16 OFDM symbols

Swap Module



The real part is changed to negative by 2's complement and go to the Imaginary output and the imaginary goes to the real output with same sign

Complex Multiplier



This complex multiplier can be realized by only three real multipliers and five real adder/subtractor based on equation; this will save a lot of area in hardware implementation. A twiddle multiplication stage is implemented after every two butterfly stages. At every twiddle stage, a complex hardware multiplier is used to multiply each data sample by a corresponding complex twiddle coefficient of unit magnitude.

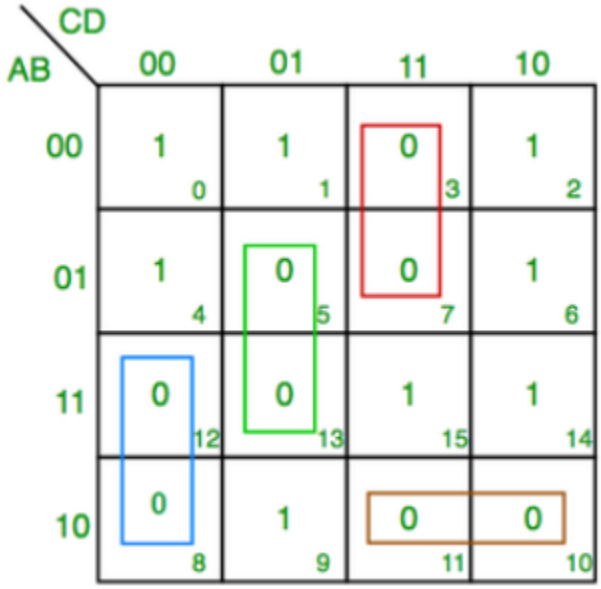
Control Unit

The control unit mainly depends on counter of 5 bits. The output signals from C1 to C6 is a combinational logic from that counter

c1 toggle every 8 cycles
c3 toggle every 4
c4 toggle every 2
c5 each 4 counter will enable for once
c6 toggle every 1
c2 from 12 to 16

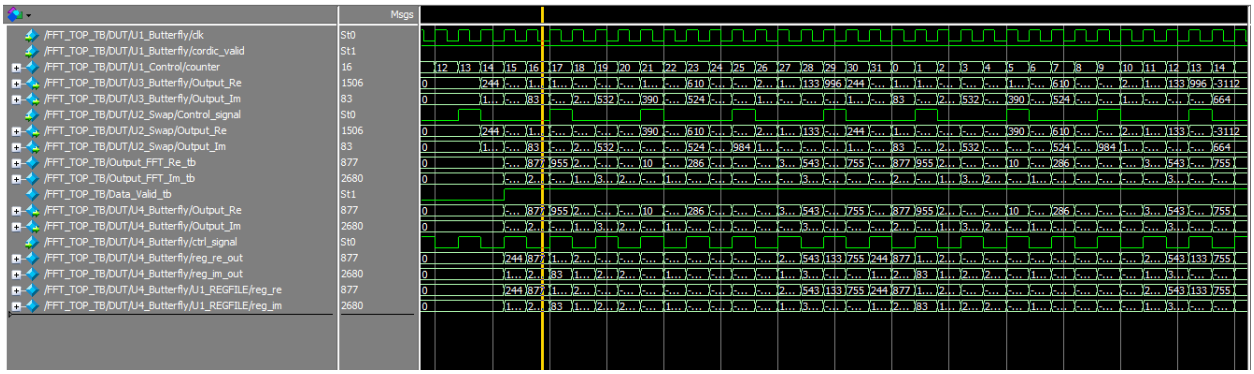
The output signals is reduced using K-map

0	1	3	2
4	5	7	6
8	9	11	10
12	13	15	14



Simulation results Vs MATLAB

1	2	3	4	5	6	7	8	9	10	11	12	13
-0.1844 - 0....	0.4248 + 1.3...	0.4653 - 0.7...	1.0035 + 0.8...	-0.8355 + 1....	-0.3187 + 1....	0.0018 - 1.1...	-0.3728 + 0....	0.1450 - 0.3...	-0.4597 - 0....	-1.7693 - 0....	-0.7304 - 1....	1.7139 - 0.



Check result change the decimal in MODELSIM to fixed

Multiply the number ($877 * 2^{-11}$) will give 0.428 as the MATLAB in sample 2