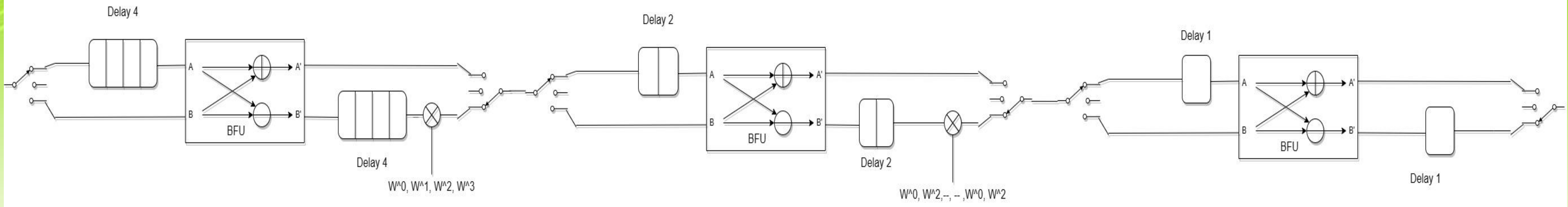
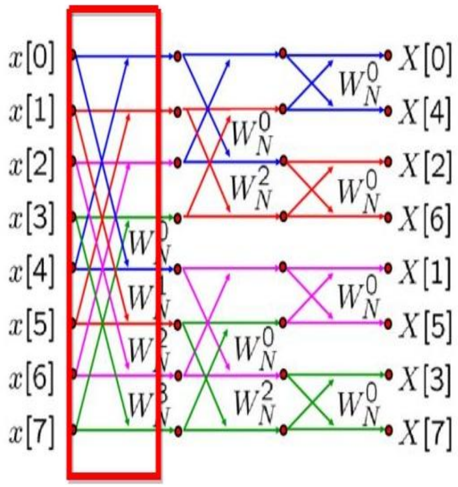
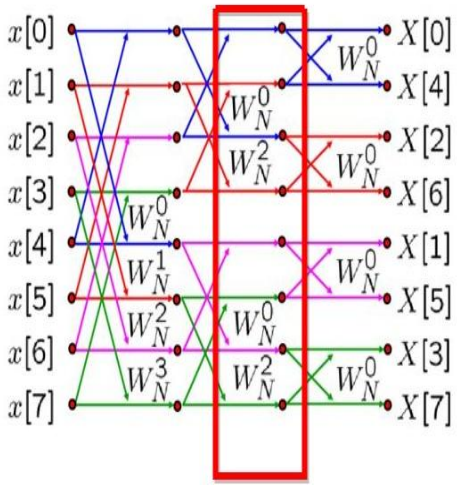
Single Delay Feedback Pipelined FFT Architecture

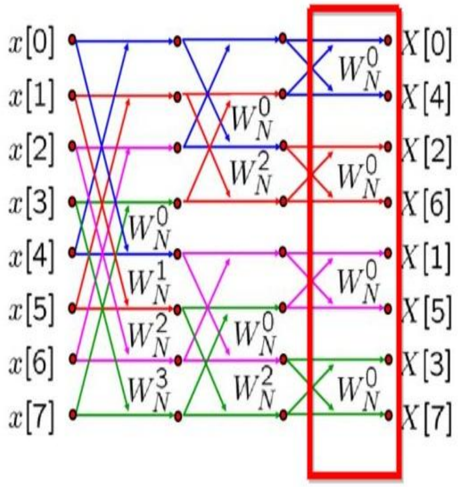
SDF Pipelined FFT Architecture



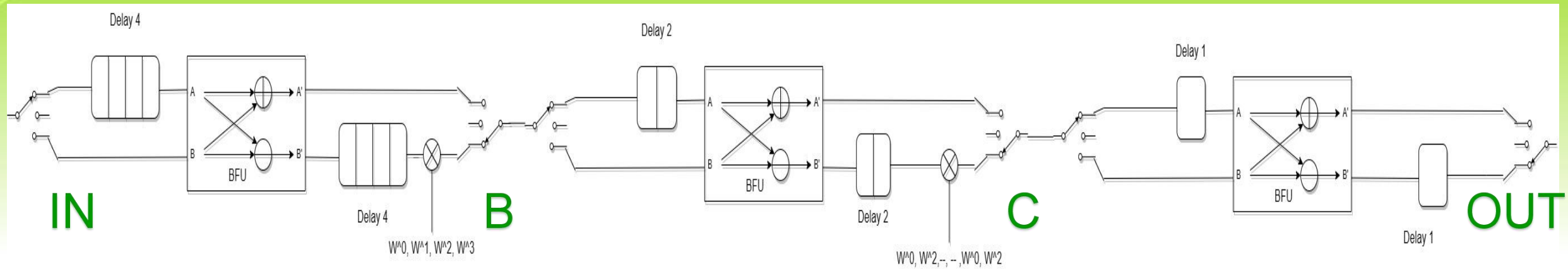
(Radix 2) (DIF)





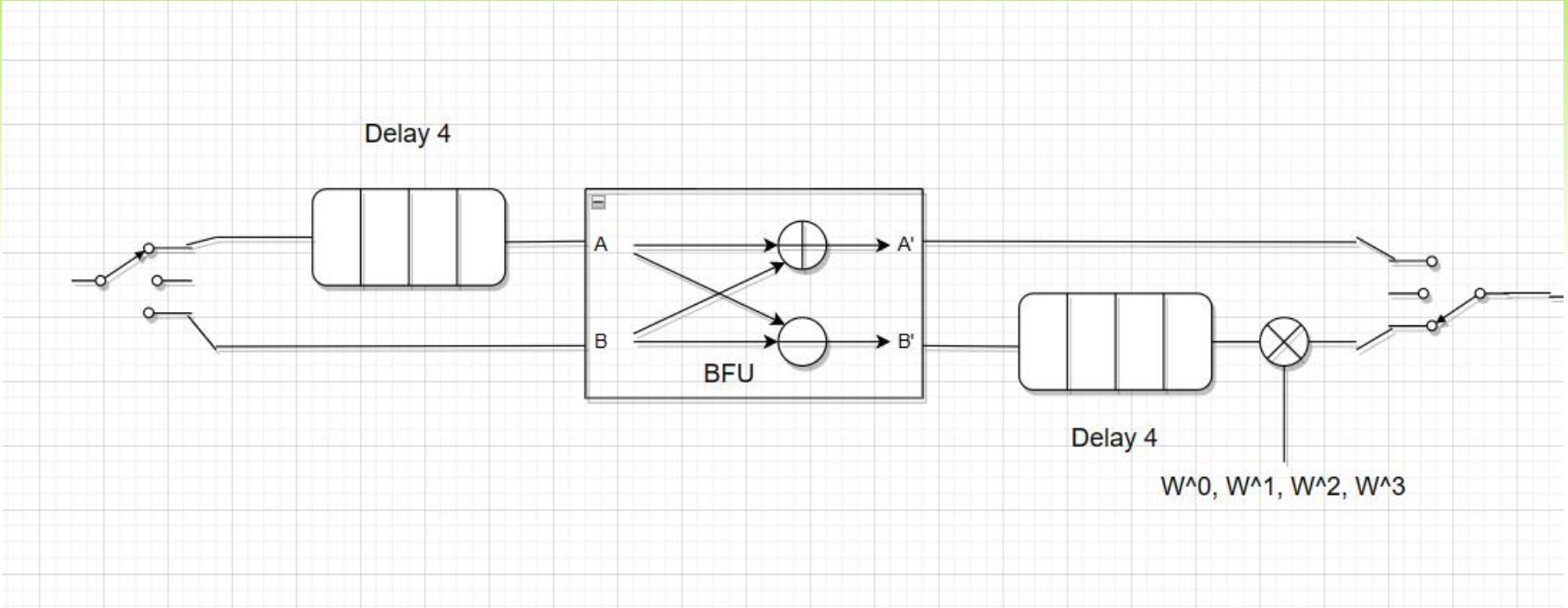


SDF Pipelined FFT Architecture

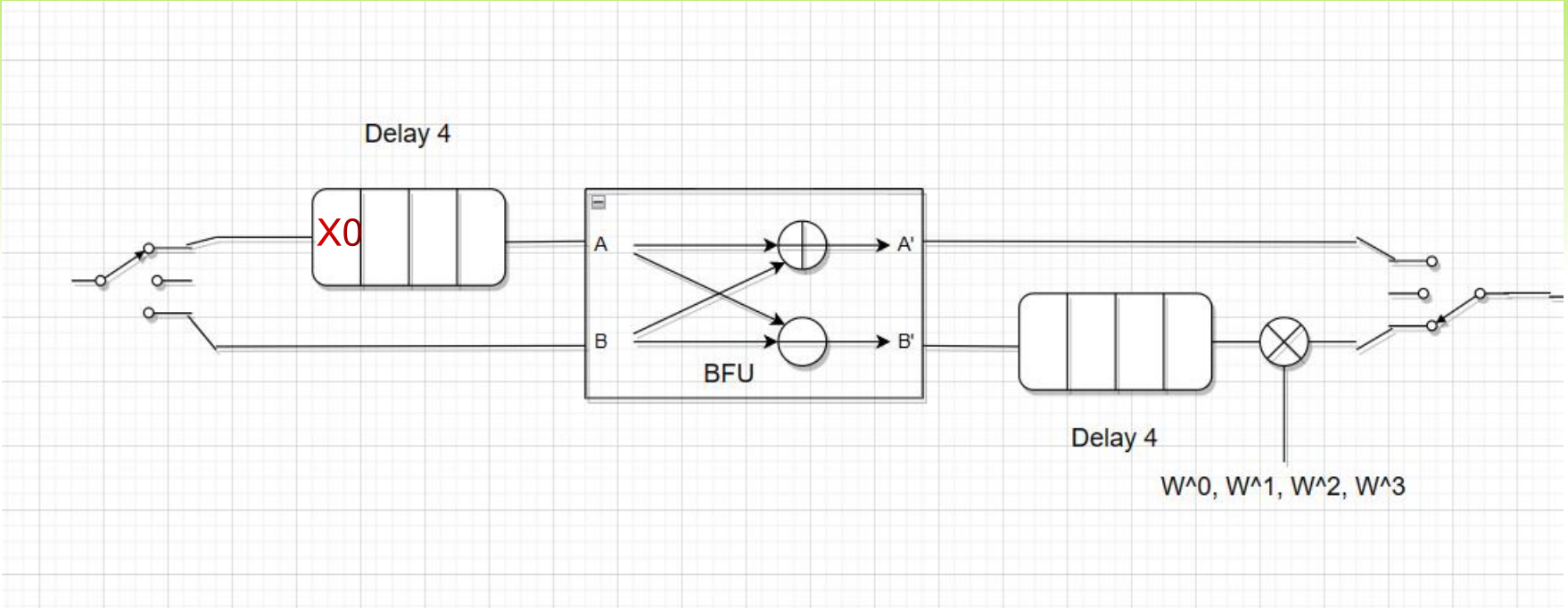


IN	0	1	2	3	4	5	6	7
B	0	1	2	3	4	5	6	7
C	0	1	2	3	4	5	6	7
OUT	0	1	2	3	4	5	6	7

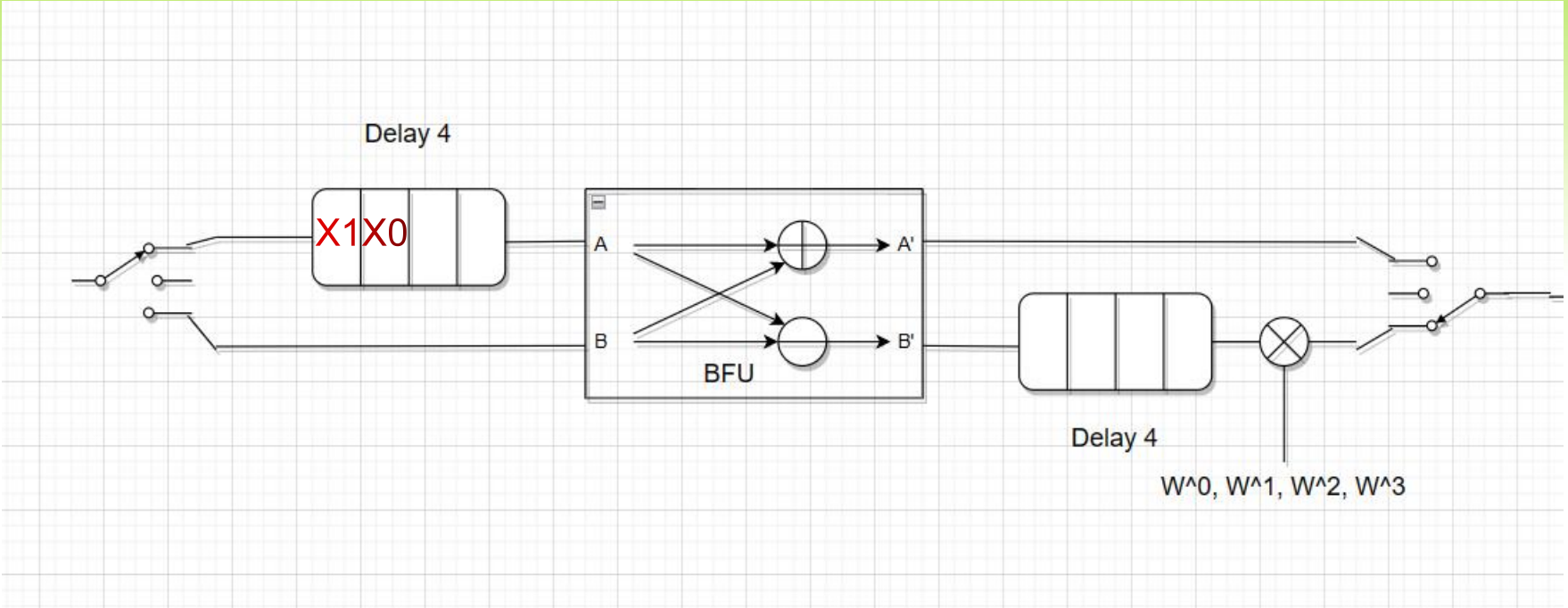
SDF Pipelined FFT Architecture



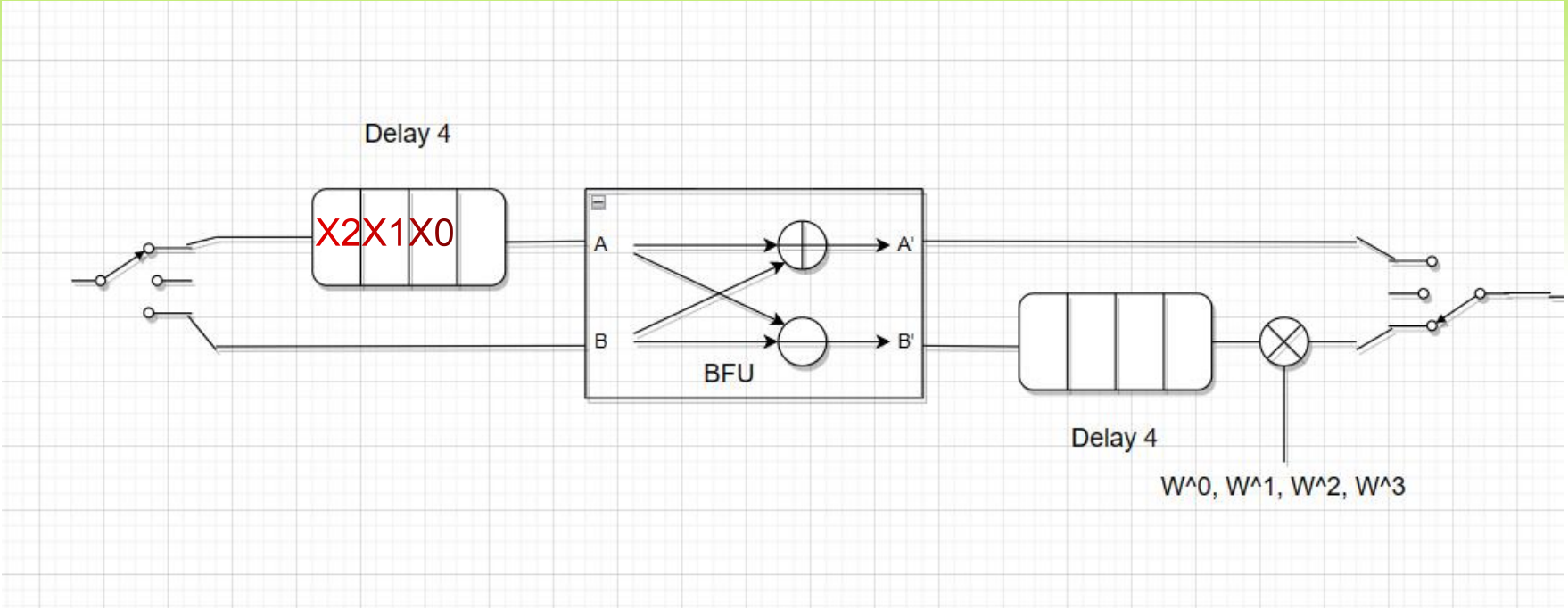
SDF Pipelined FFT Architecture



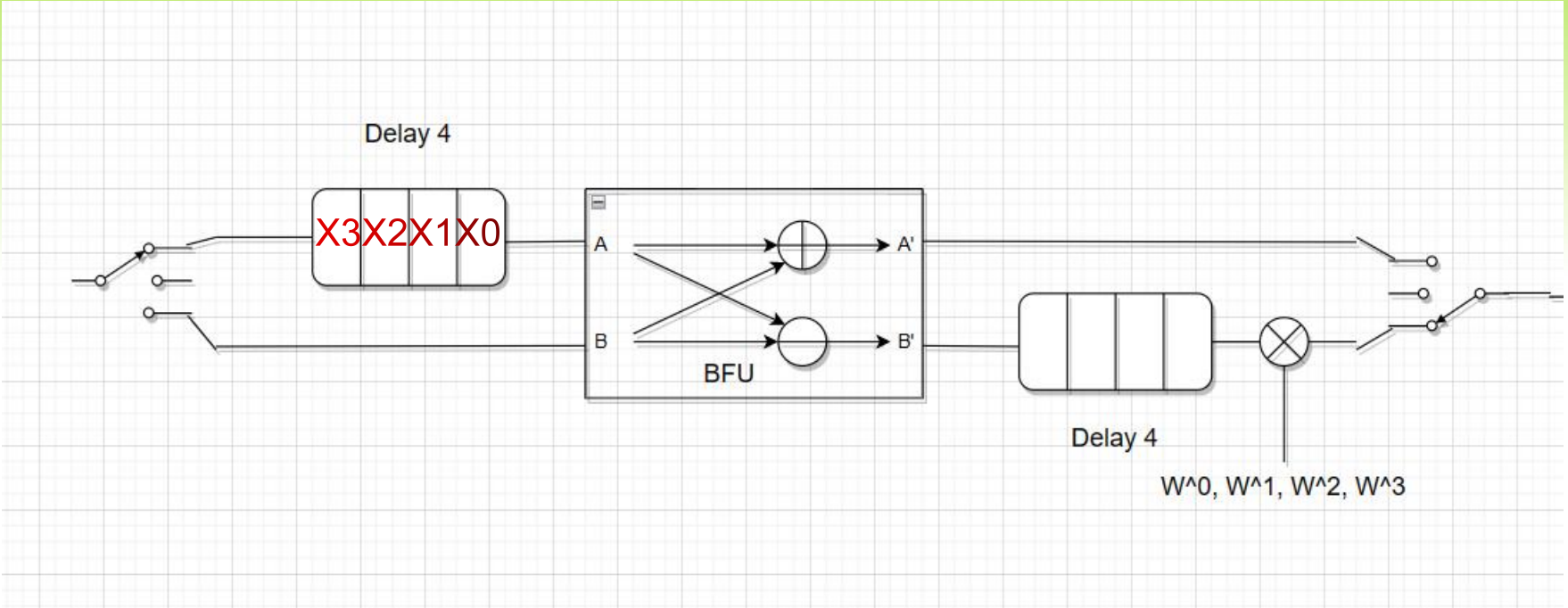
SDF Pipelined FFT Architecture



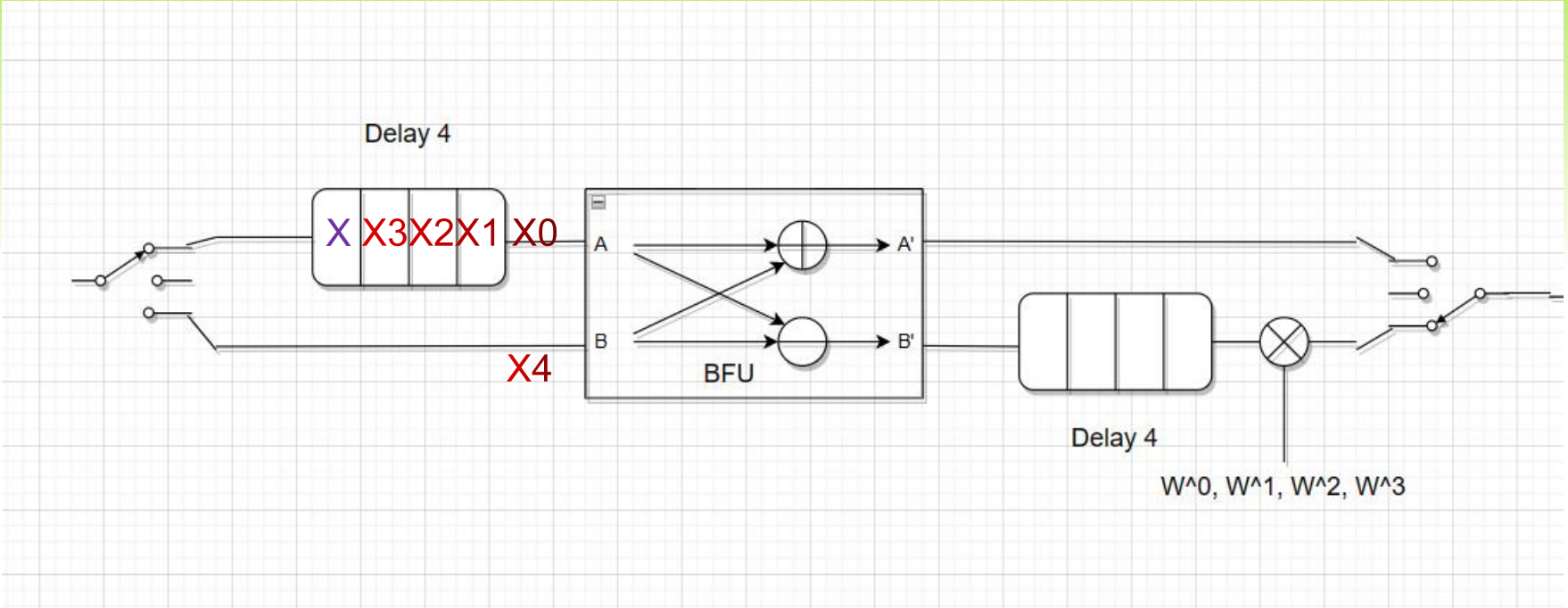
SDF Pipelined FFT Architecture



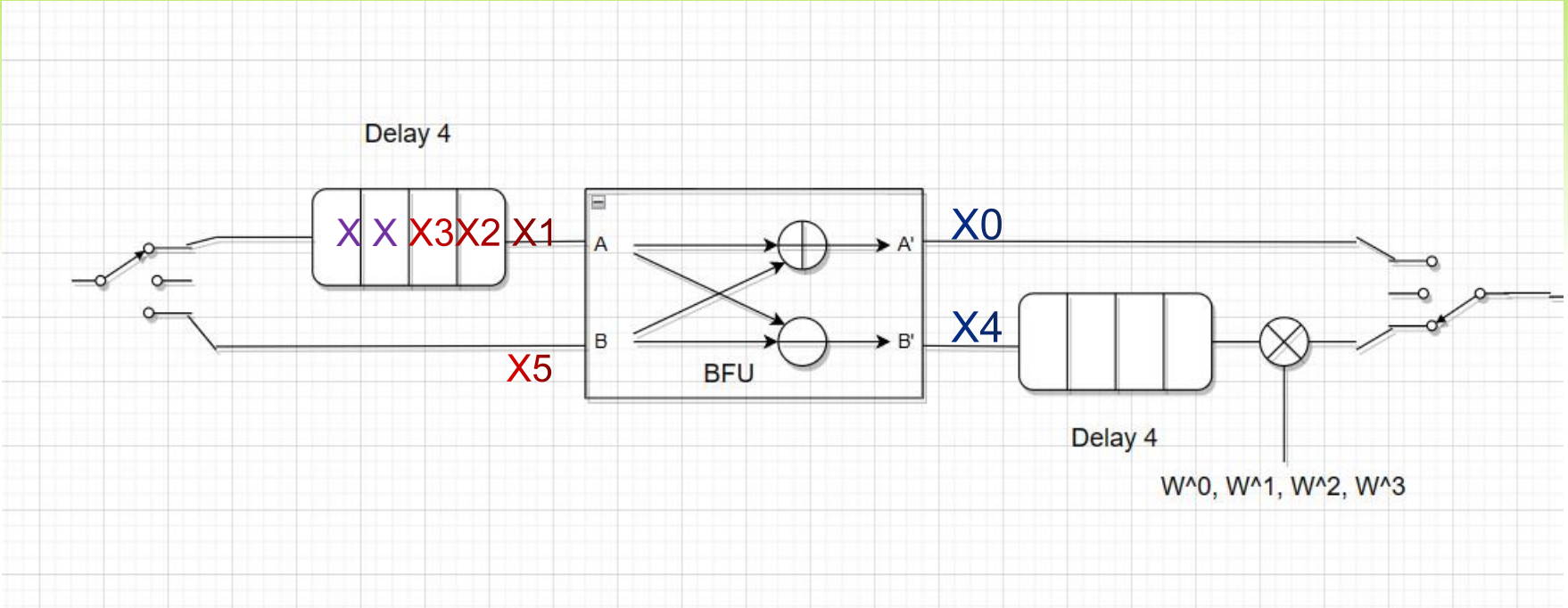
SDF Pipelined FFT Architecture



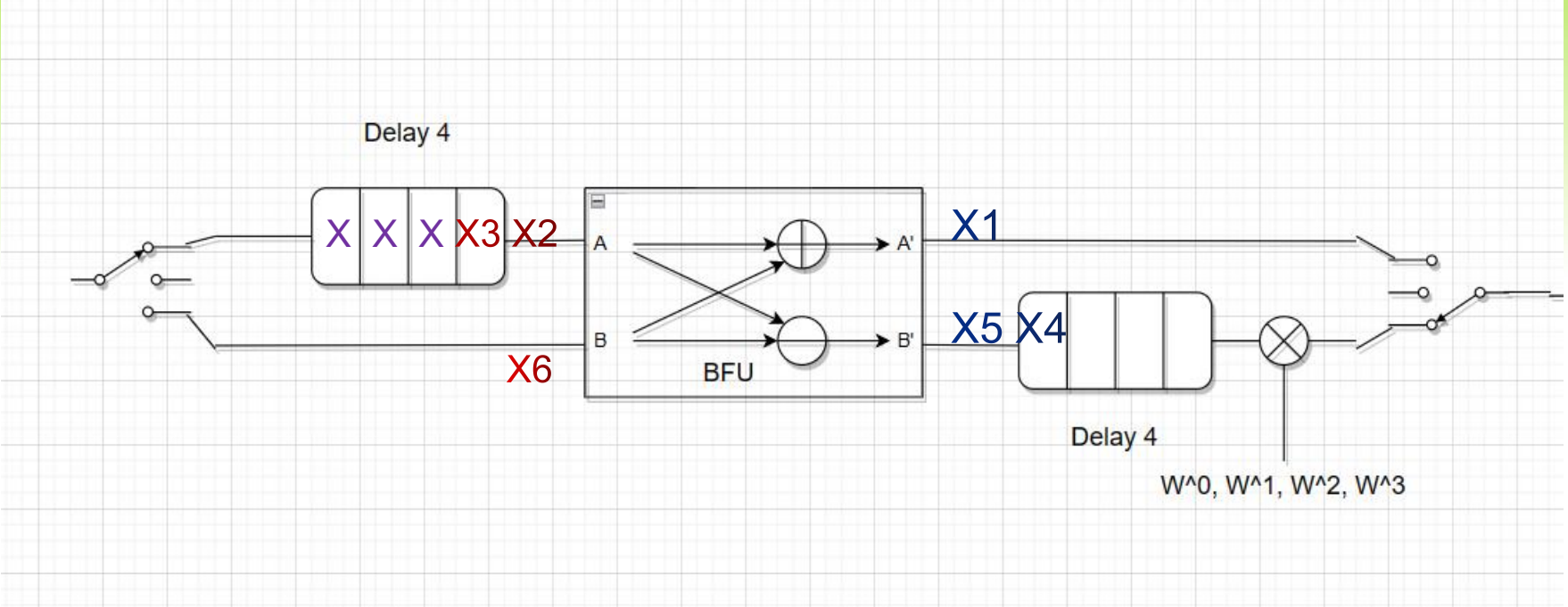
SDF Pipelined FFT Architecture



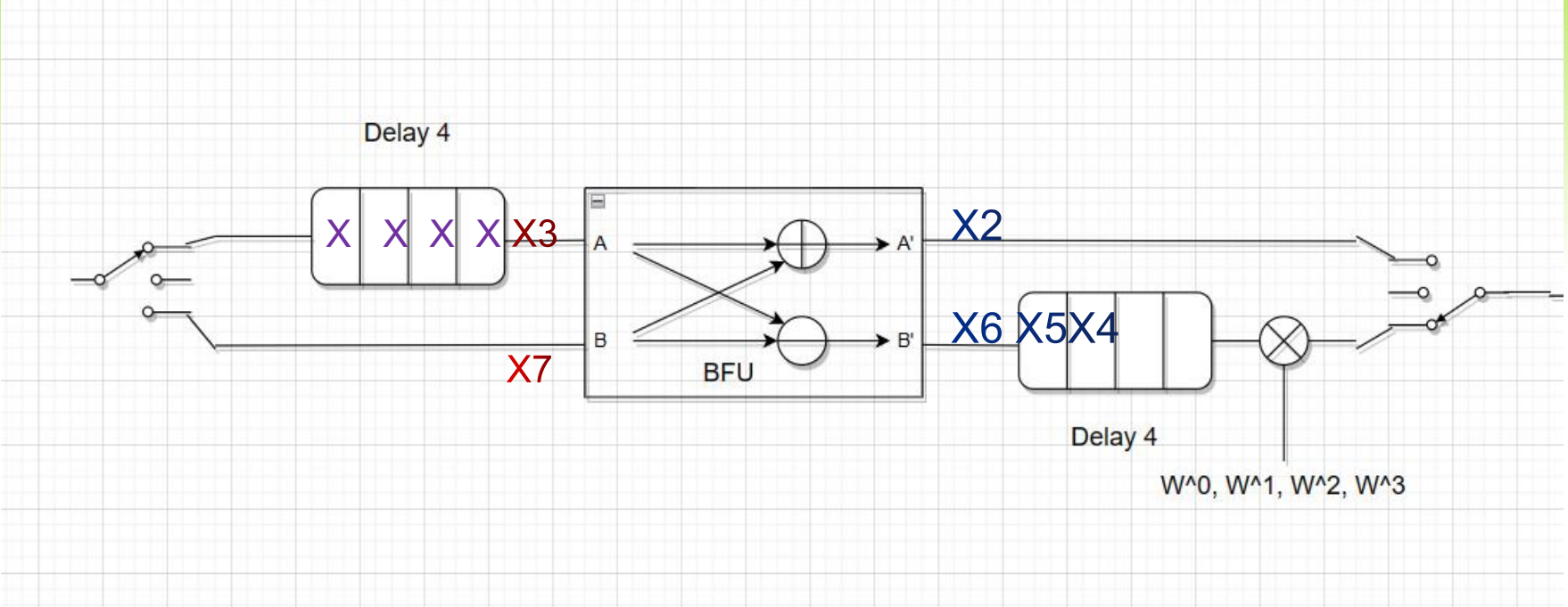
SDF Pipelined FFT Architecture



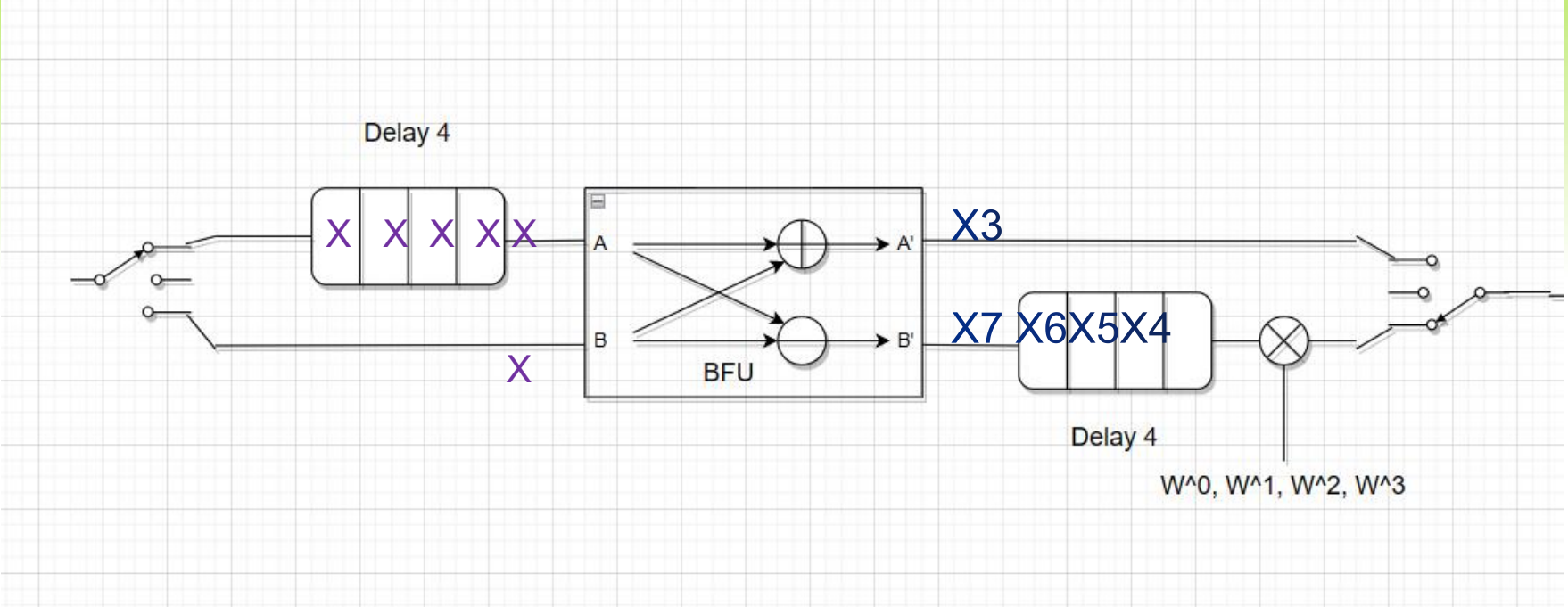
SDF Pipelined FFT Architecture



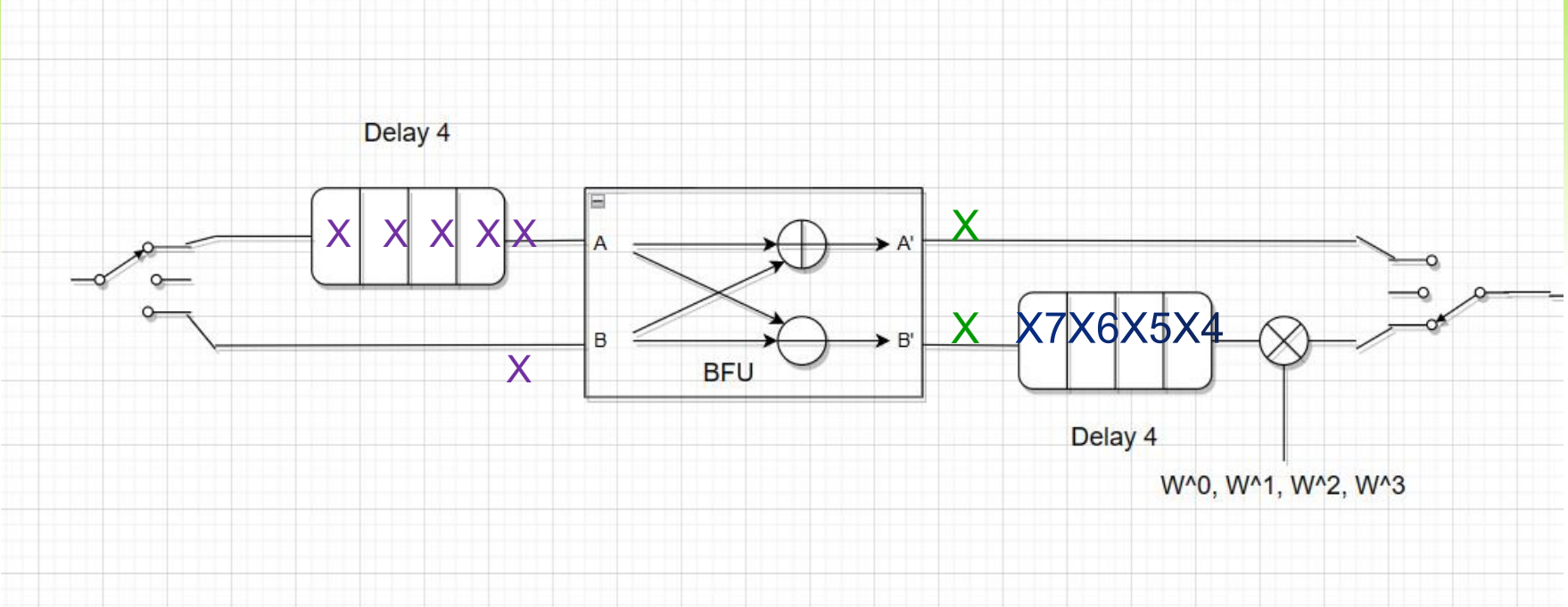
SDF Pipelined FFT Architecture



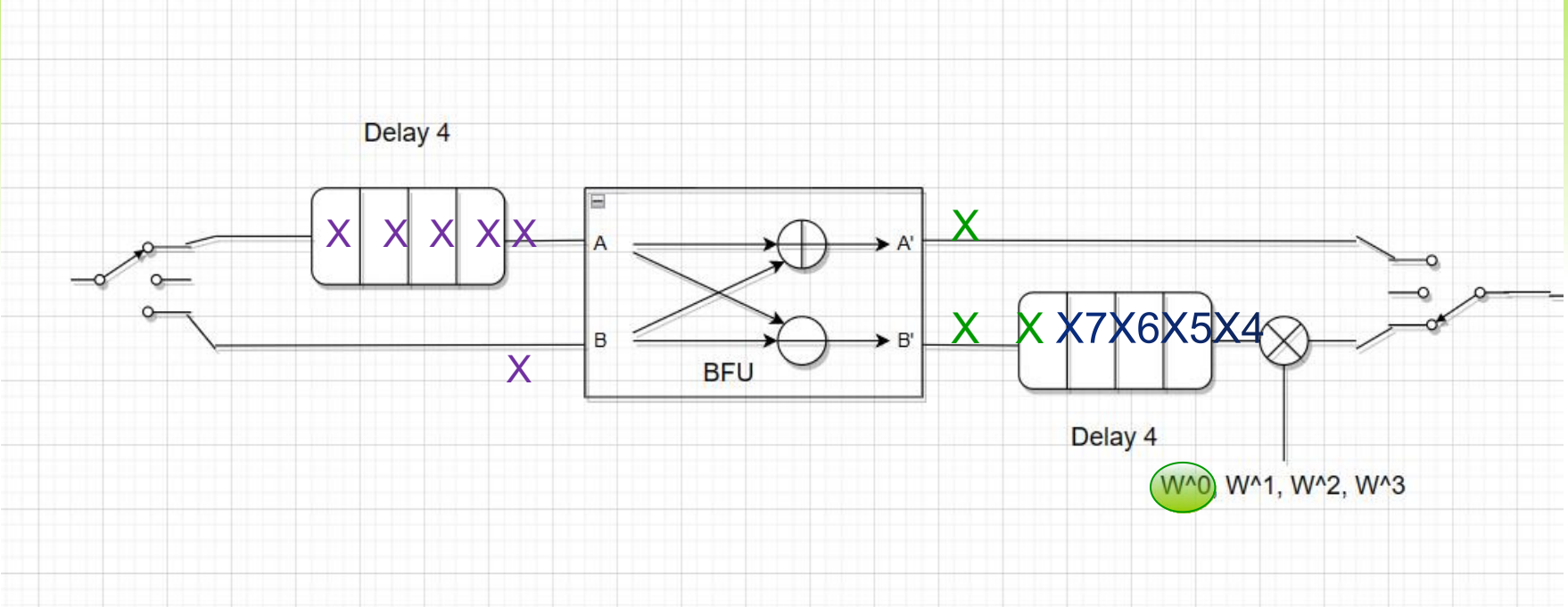
SDF Pipelined FFT Architecture



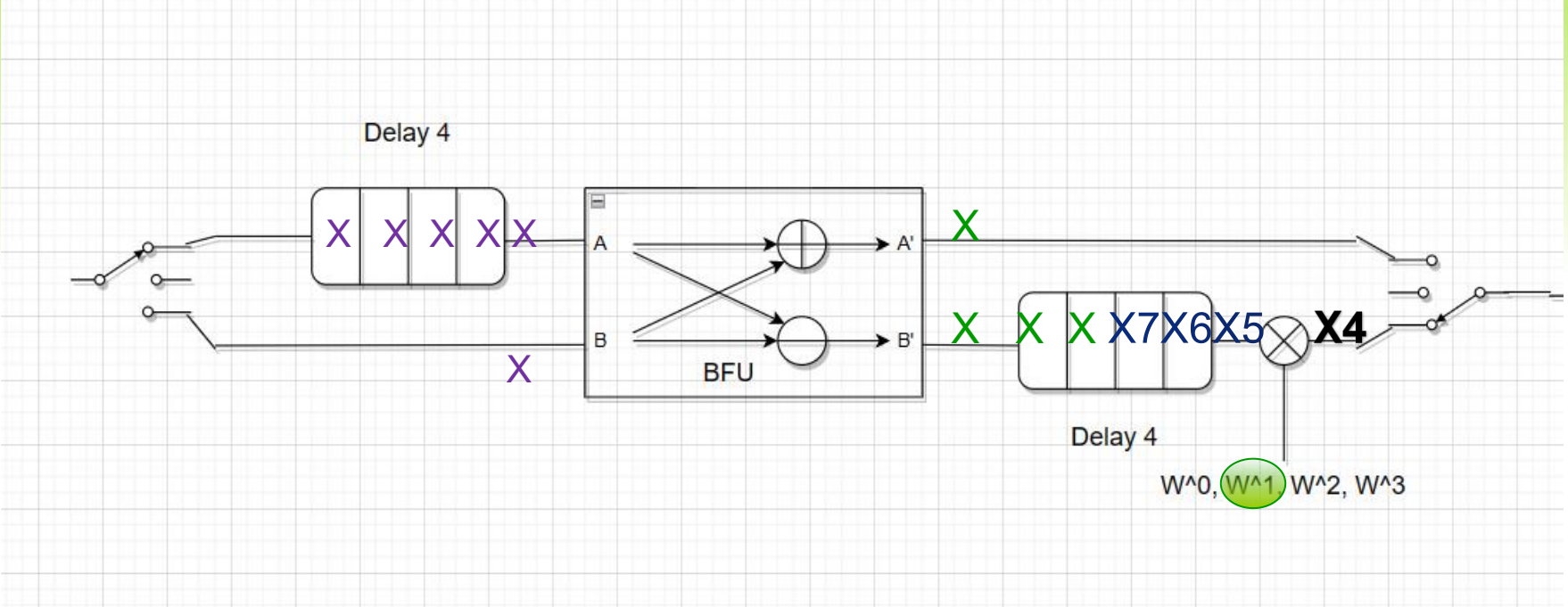
SDF Pipelined FFT Architecture



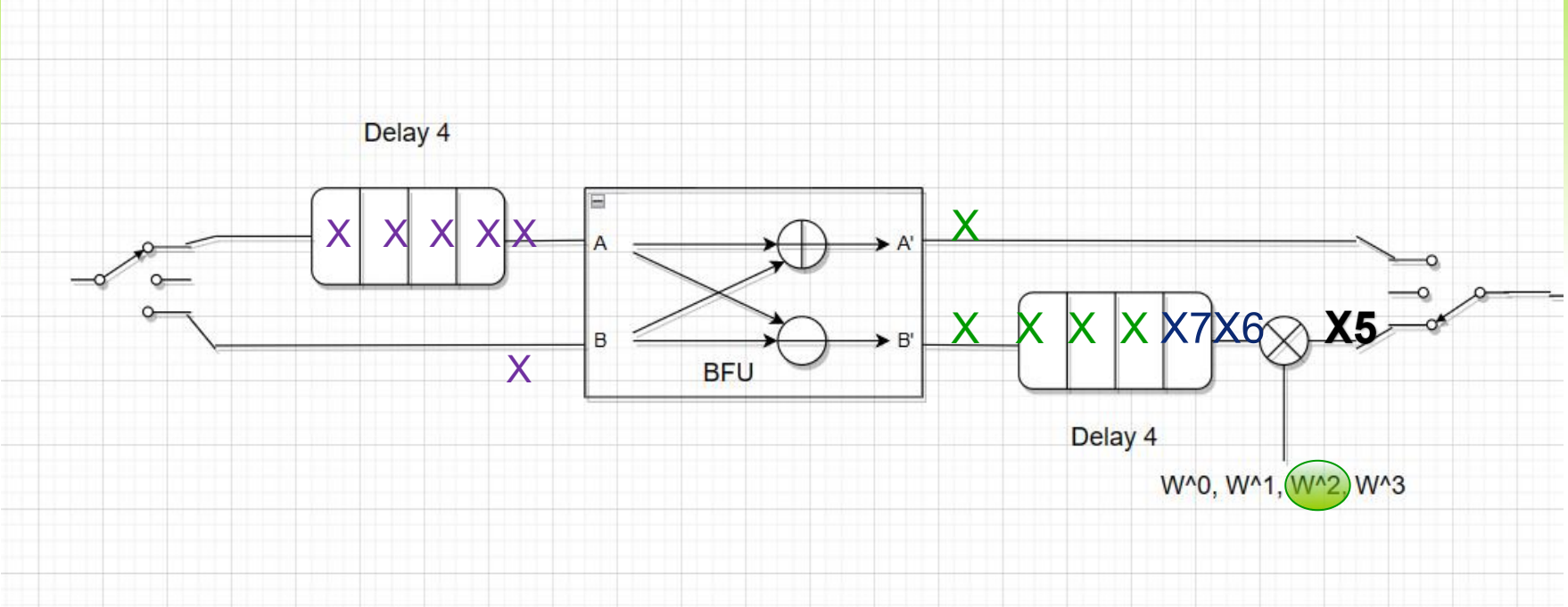
SDF Pipelined FFT Architecture



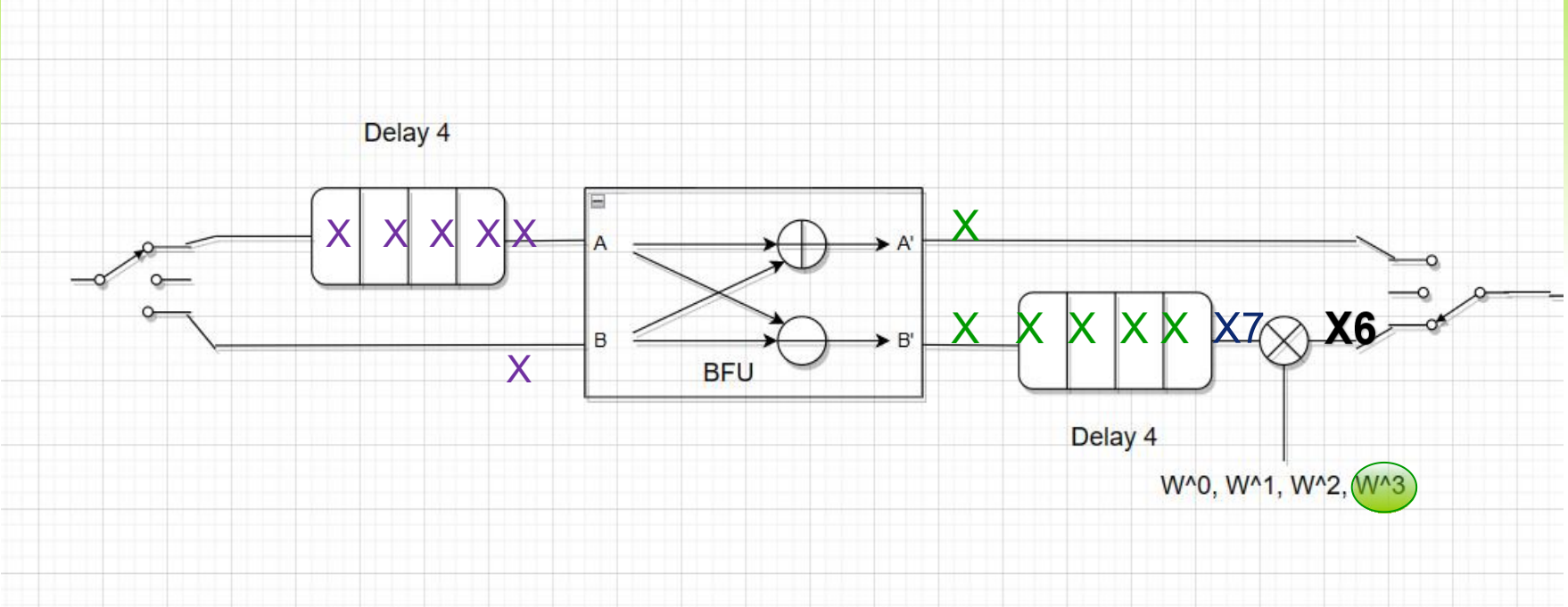
SDF Pipelined FFT Architecture



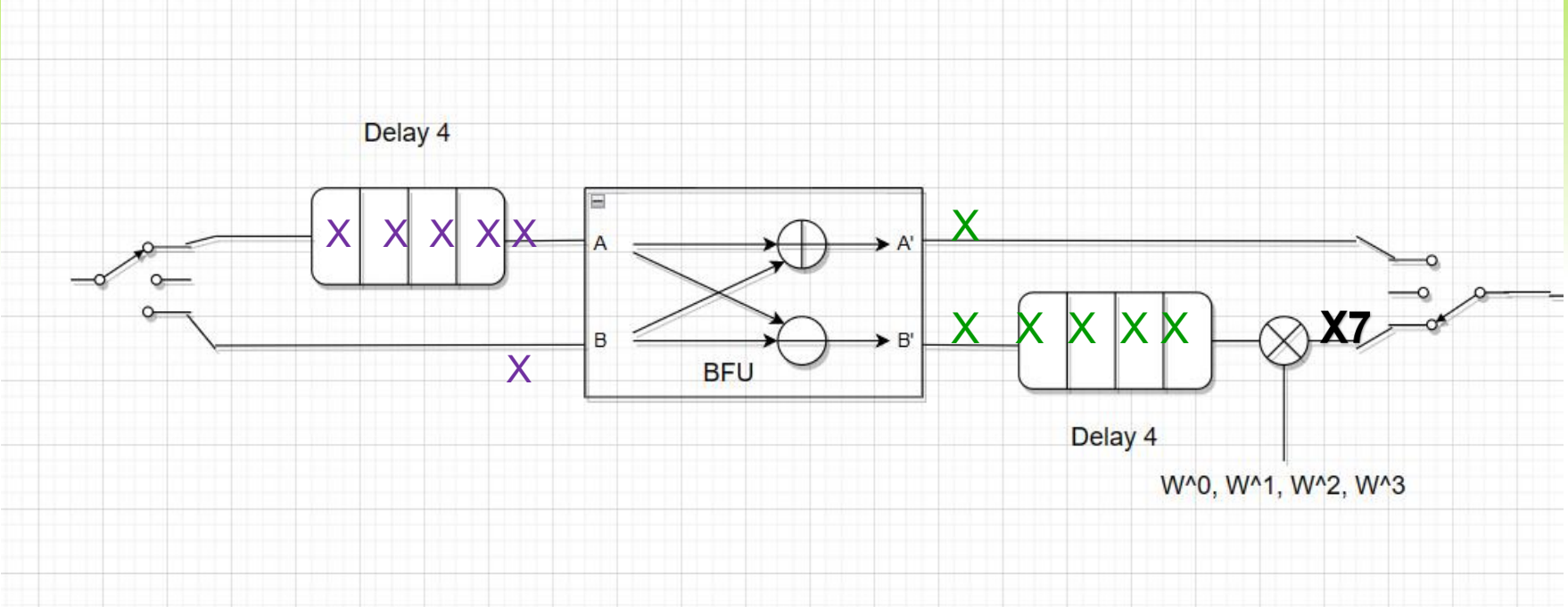
SDF Pipelined FFT Architecture



SDF Pipelined FFT Architecture



SDF Pipelined FFT Architecture



Notes about FFT design

DIT vs DIF

- No key difference in terms of HW implementation.
- However, DIT is more popular for memory based FFTs and DIF is more popular in pipelined FFT cores.
- DIT requires bit reversal at input and the twiddle factor multiplication is performed before BFU addition and subtraction.
- DIF requires bit reversal at output and the twiddle factor multiplication is performed after BFU addition and subtraction.

Fixed-point

- simply an integer where the decimal point location is in programmer's mind only.
- Q(8,7) (or) Q7
- 010111001.1101100
- 0x5cec (or) 23,788 (integer representation)
- in fixed representation = 23788 / 2^7 = 185.84375
- $sb_7b_6b_5b_4b_3b_2b_1b_0b_{-1}b_{-2}b_{-3}b_{-4}\,b_{-5}b_{-6}b_{-7}$

Fixed-point arithmetic rules

5 Fundamental Rules of Fixed-Point Arithmetic

The following are practical rules of fixed-point arithmetic. For these rules we note that when a scaling can be either signed ($A(a,b)$) or unsigned ($U(a,b)$), we use the notation $X(a,b)$.

5.1 Unsigned Wordlength

The number of bits required to represent $U(a,b)$ is $a+b$.

5.2 Signed Wordlength

The number of bits required to represent $A(a,b)$ is $a+b+1$.

5.3 Unsigned Range

The range of $U(a,b)$ is $0 \leq x \leq 2^a - 2^{-b}$.

5.4 Signed Range

The range of $A(a,b)$ is $-2^a \leq \alpha \leq 2^a - 2^{-b}$.

5.5 Addition Operands

Two binary numbers must be scaled the same in order to be added. That is, $X(c,d) + Y(e,f)$ is only valid if $X = Y$ (either both A or both U) and $c = e$ and $d = f$.

5.6 Addition Result

The scale of the sum of two binary numbers scaled $X(e,f)$ is $X(e+1,f)$, i.e., the sum of two M-bit numbers requires $M+1$ bits.

5.7 Unsigned Multiplication

$$U(a_1,b_1) \times U(a_2,b_2) = U(a_1+a_2, b_1+b_2).$$

5.8 Signed Multiplication

$$A(a_1,b_1) \times A(a_2,b_2) = A(a_1+a_2+1, b_1+b_2).$$

Fixed-point arithmetic rules

If the Q number's base is to be maintained (*n* remains constant) the Q number math operations must keep the denominator d constant. The following formulas show math operations on the general Q numbers N_1 and N_2. (If we consider the example as mentioned above, N_1 is 384 and d is 256.)

$$\frac{N_1}{d} + \frac{N_2}{d} = \frac{N_1 + N_2}{d}$$

$$\frac{N_1}{d} - \frac{N_2}{d} = \frac{N_1 - N_2}{d}$$

$$\left(\frac{N_1}{d} \times \frac{N_2}{d}\right) \times d = \frac{N_1 \times N_2}{d}$$

$$\left(\frac{N_1}{d} / \frac{N_2}{d}\right) / d = \frac{N_1 / N_2}{d}$$

- do whatever operation you want. however, to keep the decimal point location fixed the base should be “d”
- from here: after addition or subtraction no worries about the place of decimal point.
- Q8 + Q8 = Q8
- but if you do multiplication: -
- Q8 x Q8 = Q16
- the decimal point location is shifted to the left by N points where N is the number of bits on the right side of the decimal point, in this example its 8
- therefore, to take the correct result we have to right shift the result 8 bits to the right to have it in the form Q8
- *watch out for underflow (precesion loss) !!!*

Overflow

- suppose we have 2 fixed point numbers stored in 16-b registers of the form Q7 or Q(8,7) with values: -
- 0x7fff, 0x7fff
- suppose the result will be stored in 16-b reg also
- then,
- 0x7fff + 0x7fff = 0b1111 1111 1111 1110 (OR) 0xfffe
- 255.9921875 + 255.9921875 = should be 511.984375
- but interpreted as -0.015628

Avoiding overflow

- #1 saturation logic: -
- limit the 16-b register either to maximum positive value (0x7fff) or maximum negative value (0x8000)

```
int16_t q_add_sat(int16_t a, int16_t b)
{
    int16_t result;
    int32_t tmp;

    tmp = (int32_t)a + (int32_t)b;
    if (tmp > 0x7FFF)
        tmp = 0x7FFF;
    if (tmp < -1 * 0x8000)
        tmp = -1 * 0x8000;
    result = (int16_t)tmp;

    return result;
}
```

Avoiding overflow

- #2 adding guard bits: -
- despite using 16-bit registers for input values we limit the input values (implicitly) to say for example 12-bits therefore, there are 4 guard bits used only for sign extention
- maximum positive in this case: 0x07ff
- maximum negative: 0xf800
- 0x07ff + 0x07ff = 0x0ffe
- 15.9921875 + 15.9921875 = 31.984375 (for Q7)

Minimum number of twiddle factors required for storage

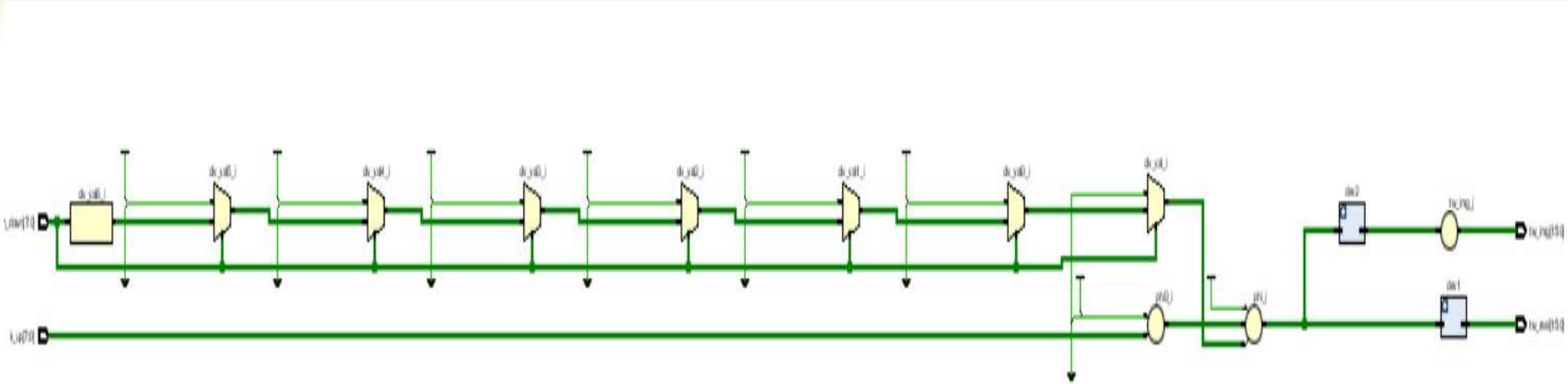
- twiddle factors with k=0 to k=N/4 are the only ones calculated.
- the remaining twiddle factors are generated by simply playing with the signs of both the real and imaginary parts.

Calculating twiddle factors on the fly

- by simply using LUT for both sine and cosine twiddle factors can be calculated cominationaly on the fly with acceptable percesion.

Calculating twiddle factors on the fly

- using sine and cosine LUT with only 64 entries of 8-bits we were able to generated twiddle factors with only 76 FPGA elements and no DSP blocks.

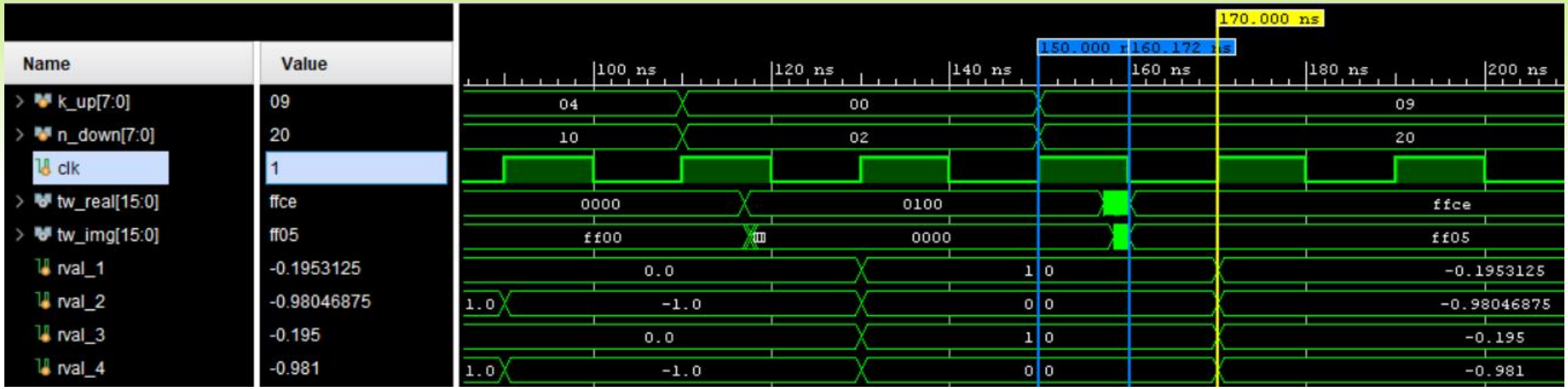


Calculating twiddle factors on the fly

Resource	Estimation	Available	Utilization %
LUT	76	17600	0.43
IO	47	100	47.00

```
simtime: 50000,   (W_(  6))^(  8) =  R: 0.000      I: j 1.000
simtime: 50000, ACTUAL C MODEL =  R: 0.000      I: j 1.000
simtime: 90000,   (W_(  4))^( 16) =  R: 0.000      I: j -1.000
simtime: 90000, ACTUAL C MODEL =  R: 0.000      I: j -1.000
simtime: 130000,   (W_(  0))^(  2) =  R: 1.000      I: j 0.000
simtime: 130000, ACTUAL C MODEL =  R: 1.000      I: j 0.000
simtime: 170000,   (W_(  9))^( 32) =  R: -0.195      I: j -0.980
simtime: 170000, ACTUAL C MODEL =  R: -0.195      I: j -0.981
$finish called at time : 270 ns : File "E:/work/vivado/tw_on_fly/tw_on_fly.srcs/sim_1/new/tb_top.v" Line 157
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_top_time_synth' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:24 ; elapsed = 00:00:25 . Memory (MB): peak = 1862.316 ; gain = 561.016
```

Calculating twiddle factors on the fly

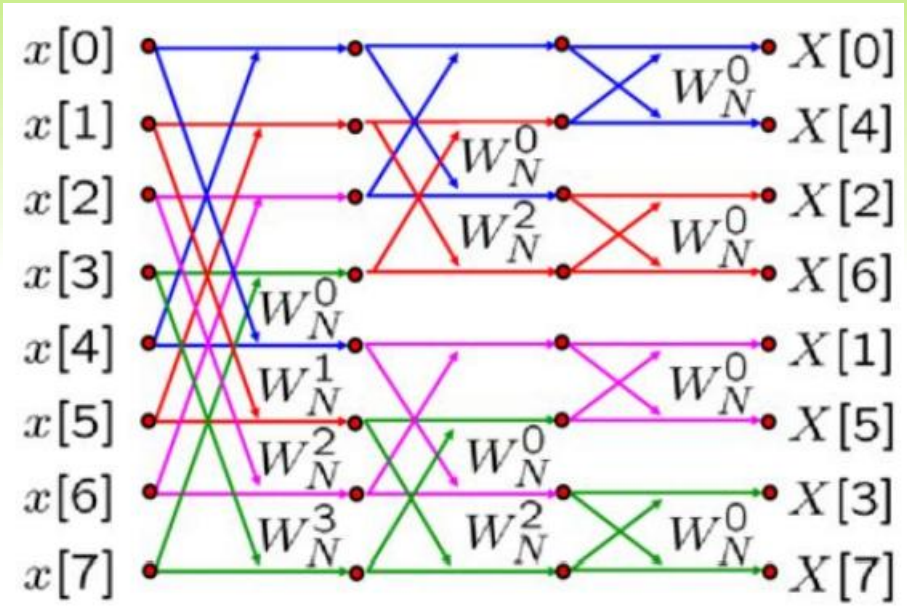


comparison between pipelined and memory based FFT architectures

-----	pipelined (sdf r-2)	memory based (r-2)
No. mult	log2 (N - 2)	1
No. adds	2 log2 (N)	1
Throughput/ number of computation cycles	at steady state ≈ 1 sample per cycle	N/2 log2 (N + 2)

8-point radix-2 DIF FFT

butterfly diagram



BFU indices

```
# STAGE 1: { 0 , 4 } {W0} ,,,{ 1 , 5 } {W1} ,,,{ 2 , 6 } {W2} ,,,{ 3 , 7 } {W3} ,,,

# STAGE 2: { 0 , 2 } {W0} ,,,{ 1 , 3 } {W2} ,,,{ 4 , 6 } {W0} ,,,{ 5 , 7 } {W2} ,,,

# STAGE 3: { 0 , 1 } {W0} ,,,{ 2 , 3 } {W0} ,,,{ 4 , 5 } {W0} ,,,{ 6 , 7 } {W0} ,,,
```

- each color represents an independent butterfly operation.
- notice how the indices in each stage are not repeated which means that no other BFU in each stage deals with other BFU input samples.
- from here we can say that computations can be done in place (the same input memory can be used for output)