

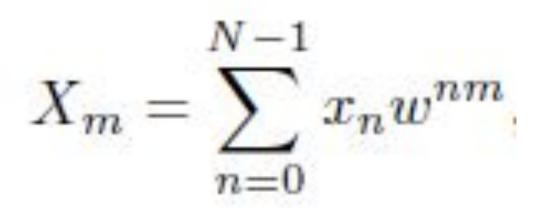


### • **Objective**:

FPGA Implementation of Fast Fourier Transform(FFT) algorithm

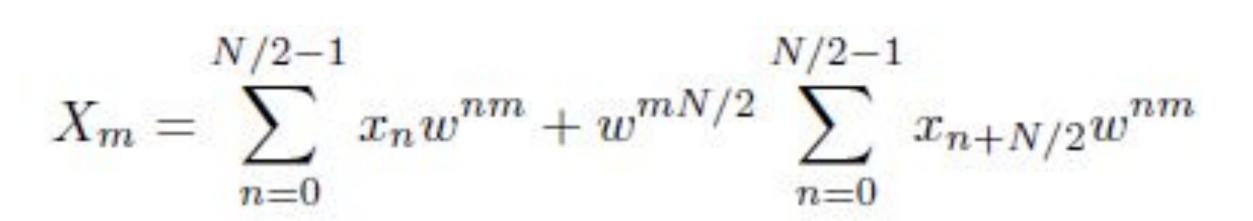
# •Theory

• The Discrete Fourier Transform(DFT) is a linear transformation of the vector x<sub>n</sub> (the time domain signal samples) to the vector Xm (the set of coefficients of component sinusoids of time domain signal). Suppose our signal is  $x_n$  for n=0...N -1, and  $x_n = x_{n+jN}$  for all n and j. The discrete Fourier transform of x is given by



where N is the size of the vector,  $\omega = e^{2\pi i/N}$  are the "roots-of-unity" (twiddle factors), and  $0 \le m \le N$ .

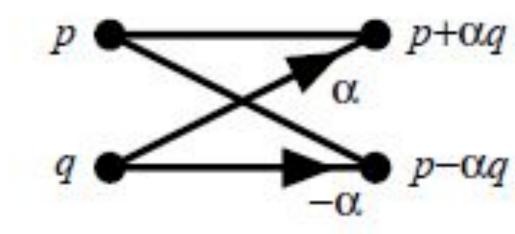
- To compute the DFT of an N-size sequence using above equation would take  $O(N^2)$  multiplies and adds.
- Precisely, every 'point' (x<sub>n</sub>) would require 4N multiplies (multiplying 2 complex numbers). These 4N multiplies for N times would give  $4N^2$  multiplies.
- The FFT is a fast algorithm for computing the DFT. If we take the 2-point DFT and 4-point DFT and generalize them to 8-point, 16-point, ..., 2<sup>r</sup>-point, we get the FFT algorithm.
- The FFT algorithm computes the DFT using O(Nlog<sub>2</sub>N) multiplies and adds. There are many variants of the FFT algorithm. We employ the Cooley-Tukey Radix-2 FFT algorithm here.
- The FFT algorithm decomposes the DFT into log<sub>2</sub>N stages, each of which consists of N=2 butterfly computations.



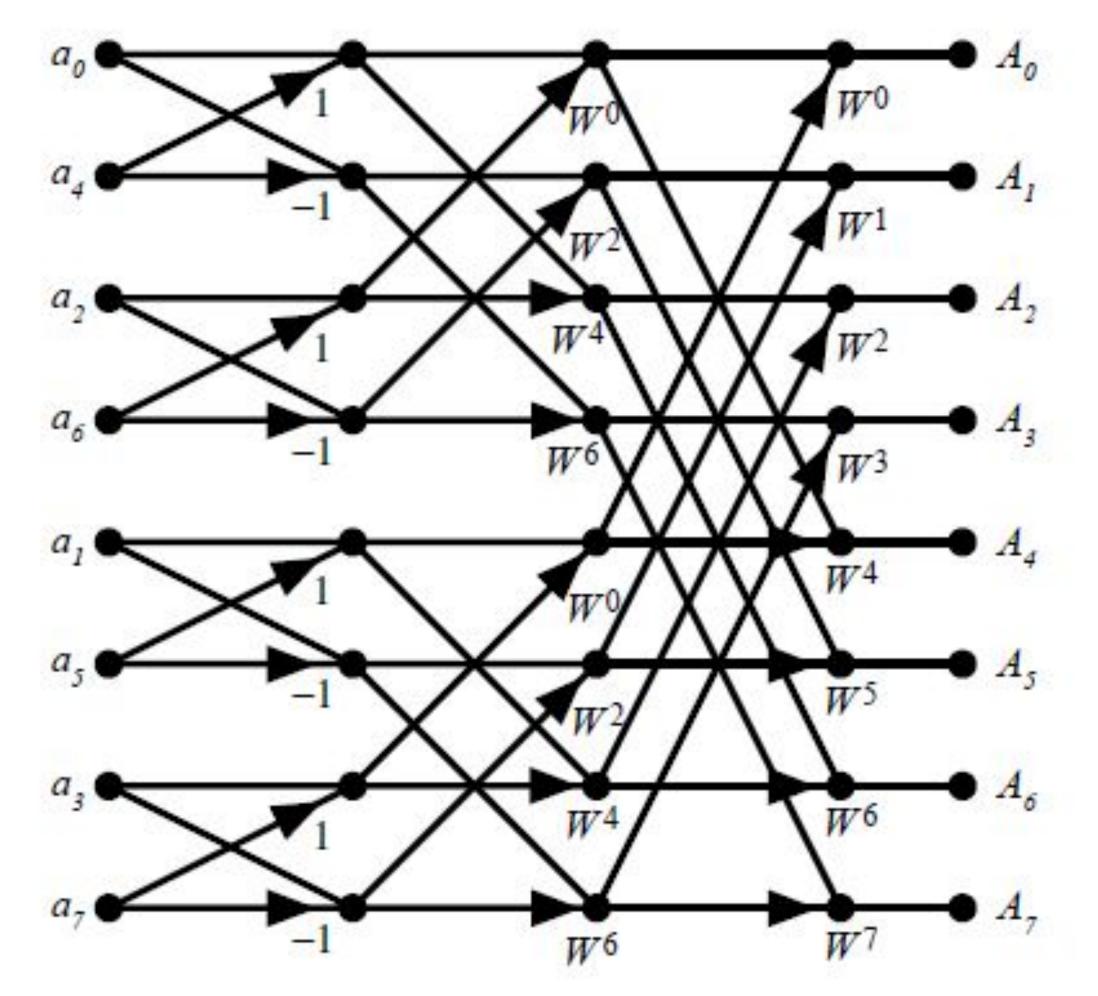
- The basic computational step of the FFT algorithm is a butterfly.
- Each butterfly computes two complex numbers of the form p  $+\alpha q$  and p  $-\alpha q$ , so it requires one complex multiply and two complex adds. This works out to 4 real multiplies and 6 real adds per butterfly.

# ES 203: DIGITAL SYSTEMS PROJECT FPGA IMPLEMENTATION OF FAST FOURIER TRANSFORM

# Jainish Chauhan(17110038) | Ojas Mithbavkar(17110083) | Mrinal Anand(17110087) | Rohit Patil(17110126)



The complete butterfly diagram is :



It is observed that the outputs are not in conventional order. Below is a table of j and the index of the jth input sample,  $n_i$ :

ILLUSTRATION OF THE BIT-REVERSED INDICES.

Index	binary	Bit reversed index	binary
0	000	0	000
1	001	4	100
2	010	2	010
3	011	6	
4	100	1	001
5	101	5	101
6	110	3	011
7	111	7	111

the bit-reversal of corresponding input index.

#### The output for a sample of inputs:

<u>File Edit Flow Tools Rep</u>	p <u>o</u> rts <u>V</u>	<u>M</u> indow Layout <u>V</u> iew	Run Help Q+ Quick Acces	Synthesis and Im	nplementation Out-of-date details
		Φ Σ % Ø	🗶 🚺 🕨 🐂 🛛 10 u	× I C	🔚 Default Layout
Flow Navigator 🗧 🚔 ?	_ SIM	IULATION Behavioral S	imulation - Functional - sim_1 - butte	$x = \frac{1}{2} + $	
P IP Catalog		dule.v x top module 2	2.v × main tb.v × main.v	× butterfly.v × Seg_7_Display.v × constraint.xdc × butterfly_tb.v ×	Untitled 29* × 4 ▶ ≡ ? ₽
	Scope		8 4 8 8 2		
IP INTEGRATOR	Sc Sc	4 4 4			
Create Block Design	e B			253.700 ns	
Open Block Design	Sources	Name	Value		280 ns
Generate Block Design	<u> </u>	> 😻 a0[31:0]	000000010000000000000000	000000010000000000000000000000000000000	
	v.	> 😻 a1[31:0]	00000001100000000000000000		
SIMULATION	Objects	> 😻 a2[31:0]	000000000000000000000000000000000000000		
Run Simulation	<u> </u>	> 😻 a3[31:0]			i
		> 😻 a4[31:0]			
RTL ANALYSIS		> V a5[31:0]			
✓ Open Elaborated Design	Ŭ	> V a6[31:0]			
🖻 Report Methodology		> V a7[31:0]			
Report DRC		> <b>b</b> b0[31:0]	000000101000000000000000000000000000000	000000101000000000000000000000000000000	
		> V b1[31:0]	00000010000000011111	00000010000111011111011110000	
🛃 Schematic		> V b2[31:0] > V b3[31:0]	11111111111000011111		
EVALUTE CLC		> <b>W</b> b4[31:0]	111111110000000000000000000000	1111111111100001111111011110000 11111111	
SYNTHESIS		> <b>W</b> b5[31:0]	111111111110000000000000000000000000000	111111111000000000000000000000000000000	
Run Synthesis		> <sup>10</sup> b6[31:0]	000000100000000000000000	000000010000000000000000000000000000000	
✓ Open Synthesized Design		> V b7[31:0]	000000100000111000000	00000011100000000100001110	
Constraints Wizard					
Edit Timing Constraints					
🐞 Set Up Debug		Tcl Console Messages			

## • Applications:

- Frequency Domain Downsampling
- Fractal Image Compression
- Phase Only Correlation
- FFT Processors for OFDM
- Three-Dimensional Face Recognition

### • **References**:

- Tutorial Based on an FPGA Implementation. I: ResearchGate.
- 2. Heckbert, P. (1995). Fourier Transforms and the Fast Fourier Transform (FFT) Algorithm. Computer Graphics, 2, 15-463.
- 3. Brigham, E. O., & Brigham, E. O. (1988). The fast Fourier transform
- Media.
- 5. https://github.com/ameyk1/Fast-Fourier-Transform

# The pattern is obvious if input and output indexes are written in binary. It is observed that each output index is

1. Slade, G. W. (2013). The Fast Fourier Transform in Hardware: A

and its applications (Vol. 448). Englewood Cliffs, NJ: prentice Hall. 4. Rao, K. R., Kim, D. N., & Hwang, J. J. (2011). Fast Fourier

transform-algorithms and applications. Springer Science & Business