© 2016 IJSRSET | Volume 2 | Issue 1 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Implementation of Fast Fourier Transform using Resource Reuse Technique on FPGA

Praveen Kumar Jhariya, Nitesh Dodkey
Department of Electronics and Communication Engineering, Surbhi Group of Institute, Madhya Pradesh, India

ABSTRACT

The utility of discrete Fourier transform (DFT) plays important role in many of digital processing including linear filtering, correlation analysis and spectrum analysis. In this work we have proposed two FFT designs, design 1 and design 2. In design 1 we have only one butterfly unit and one multiplier and this butterfly unit along with complex multiplier is used multiple times to compute the 8 point FFT. In design 2 four butterfly fly unit and two multipliers which are used 3 times. The number system used in our design is single precision (32 bit) floating point and 8 bit floating point.

Keywords : FFT, FPGA, Hardware Reuse, Butterfly

I. INTRODUCTION

The discrete Fourier transform or DFT is one of the most important problems in applied computer science, with applications in many areas of scientific computing and engineering. The first fast Fourier transform or FFT was invented by Carl Friedrich Gauss[2] in 1805 (even predating Fourier's work on harmonic analysis by two years) and reinvented by James W. Cooley and John W. Tukey in 1965[1]. The name FFT is used today to refer to any "fast" method of computing the DFT, usually O (NlogN). Sometimes the term is used more specifically to refer to some version of the Cooley-Tukey algorithm, typically for input sizes which are powers of 2.

The FFT arithmetic [3] is basically divided into two types, which is the decimation-in-time (DIT) and the decimation-infrequency (DIF). This radix-2-DIT FFT is adopted in this work. An 'N' point discrete Fourier transformation (DFT) of the input sequences x (n) is written as,

$$X(k) = \sum_{n=0}^{N-1} (x(n) W_N^{nk}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(1)}$$

Where k,n = 0,1,2……….N-1

$$WN = e^{(-j2\pi/N)}$$

x (n) could be further divided into odd part and even part using radix-2 DIT in (l), taking advantage of periodicity and symmetry we can obtain the following equations.

$$X(k) = \overbrace{\sum_{n=0}^{N/2-1}}^{\text{even terms}} (x(2n) W_N^{2nk}) + \overbrace{\sum_{n=0}^{N/2-1}}^{\text{odd terms}} (x(2n+1) W_N^{(2n+1)k}) \quad \ldots(2)$$

$$\text{Using } W^2 = (e^{-j2\pi/N})^2 = (e^{-j2\pi/(\frac{N}{2})}) \text{ in (2)}$$

We get X(k) = {(DFT of even indexed N/2 sequences) + W^k(DFT of odd indexed N/2 sequences)}

X(k) = E(k) + WkF(k) ……………………. (3)

Now for the values of k ≥ N/2, the equation (2) for X(k) can be simplified by replacing k → (k + N/2) we get

Since, WN/2 = $(e^{-j2\pi/N})^{N/2}$ *= -1 we have*

X(k + N/2) = E(k) - WkF(k) ……………….. (4)

This is where (3) & (4) the reduction in complexity comes about: one large computation is reduced to several sequential smaller computations which lead to the radix-2 butterfly as shown in figure 1.

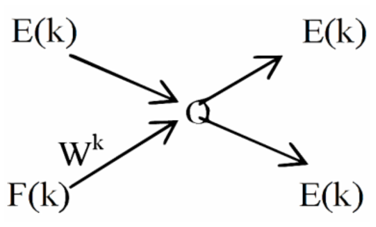


Figure 1: Basic Butterfly element

To summarize these steps for computing the DFT via decimation are

1. Shuffle input order in bit reversal form.
2. Compute N/2, two sample DFTs.
3. Compute N/ 4, four sample DFTs.
4. Continue until one, N-sample DFT is computed.
5.

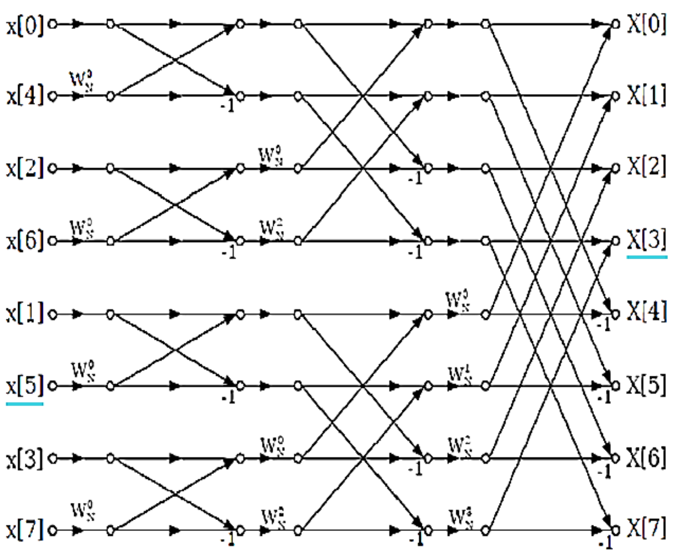


Figure 2: DIT-FFT radix-2 operation diagram

Fig.2 shows the 8 inputs decimation in time DFT operation diagram. W_N used in all the equations are also called as Twiddle Factors. Twiddle factor referred to the root of unity complex multiplicative constants in the butterfly operation. The realization of these twiddle factors can be done by using Coordinate Rotation for Digital Computers algorithm (CORDIC) or pre-computed twiddle factor values can be used. In this work pre-computed twiddle factor values are used.

II. METHODS AND MATERIAL

A. Methodology Used

The number system used in our design is 8 bit floating point. To implement the design in hardware we have divided the butterfly structure in 3 stages. Figure 3 shows the stage 1 of butterfly structure. The inputs to the stage 1 is x[n] and the outputs of stage 1 is z[n]. x[4], x[6], x[5] and x[7] gets multiplied with the twiddle factor W_N^2, the value of W_N^2 for N = 8 is 1. So no multiplication is required in stage 1. The outputs z[n] can be computed using simple addition and subtraction. In our design floating point addition and subtraction is required.

Figure 4 shows the stage 2 of butterfly unit. The inputs to the stage 2 are z[n] and the outputs of stage 1 are a[n]. z[2] and z[3] gets multiplied with the twiddle factor W_N^0, the value of W_N^0 for N = 8 is 1. So no multiplication is required. z[6] and z[7] gets multiplied by the twiddle factor value W_N^2 which is –j for N = 8. The outputs a[n] can be computed using simple addition, subtraction and two –j multipliers.

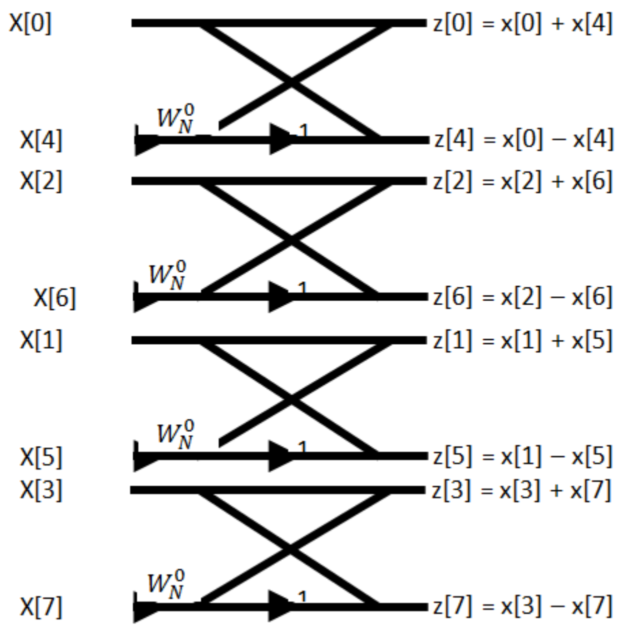


Figure 3 : Stage 1 of butterfly structure

Figure 5 shows the last stage i.e. stage 3 of butterfly structure. The inputs to this unit are a[n] and the outputs from this unit is b[n] which is assigned to X[N] which is the final output FFT transformed output. Input a[1] is multiplied by W_N^0 which is "1", input a[5] si multiplied by W_N^1, which is "0.707 – j0.707", input a[3] gets multiplied with W_N^2, which is –j and a[7] is multiplied by "-0.707 – j0.707". so complex multiplier is required to perform the multiplication with "0.707 - j0.707" and -0.707 - j0.707 along with floating point adder and subtractor.

By dividing the complete butterfly structure in 3 stages we can now determine the hardware requirements to implement the design. All the inputs are complex floating point numbers.

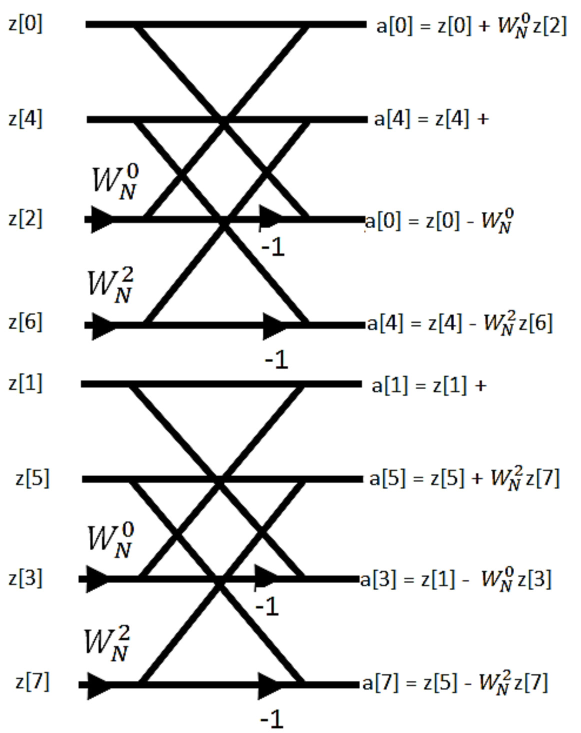


Figure 4 : Stage 2 of butterfly structure

In stage 1 only floating point adder and subtractor are required. Total 4 complex floating point adders and 4 complex floating point subtractor are required. to implement a complex floating point adder, 2 floating point adders are required and similarly to implement a floating point subtractor 2 floating point subtractor are required. So in total 8 floating point adders and 8 floating point subtractor are required.

In stage 2 apart from 8 complex floating point adders/subtractor two –j multipliers are also required. So in total 8 floating point adders and 8 floating point subtractor are required along with two –j multipliers are required.

In stage three 8 floating point adders and 8 floating point subtractor are required along with one –j multipliers and 2 complex multipliers.

Table shows the summary of hardware requirement in different stages of 8 point FFT.

Table: Hardware Requirement for 8 Point Traditional FFT

Hardware Resources	Stage 1	Stage 2	Stage 3	Total
Complex Floating Point Adders	4	4	4	12
Complex Floating Point Subtractor	4	4	4	12
-j multiplier	0	2	1	3
Complex Multiplier	0	0	2	2

In this work we have reduced the hardware resource usage of FPGA by reusing the hardware resources with small increase in delay.

B. Hardware Implementation

In this work we have proposed two designs, design 1 and design 2. In design 1 we have only one butterfly unit and one multiplier and this butterfly unit along with complex multiplier is used multiple times to compute the 8 point FFT. In design 2 four butterfly fly unit and two multipliers which are used 3 times.

a) Design – 1

In this design single butterfly unit and a single complex multiplier is used along with three –j multiplier units to compute the 8 point FFT. Figure 5 shows the high level block diagram of design 1.

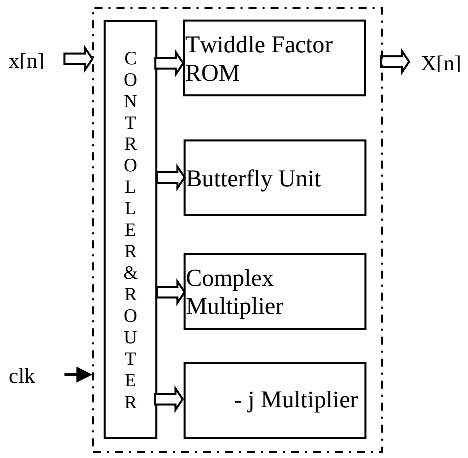


Figure 5 : High Level Block Diagram – Design 1

1. Butterfly Unit

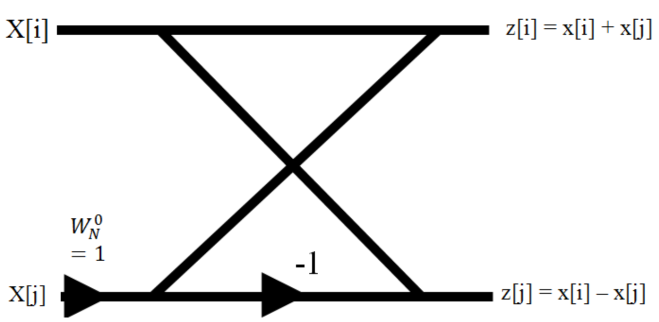
Figure 4.2 shows the internal logic diagram of butterfly unit.

x[i] = Re_x[i] + Imj_x[i]

x[j] = Re_x[j] + Imj_x[j]

z[i] = x[i] + x[j] = (Re_z[i] = Re_x[i] + Re_x[j], Imj_z[i] = Imj_x[i] + Imj_x[j])

z[j] = x[i] - x[j] = (Re_z[i] = Re_x[i] - Re_x[j], Imj_z[i] = Imj_x[i] - Imj_x[j])



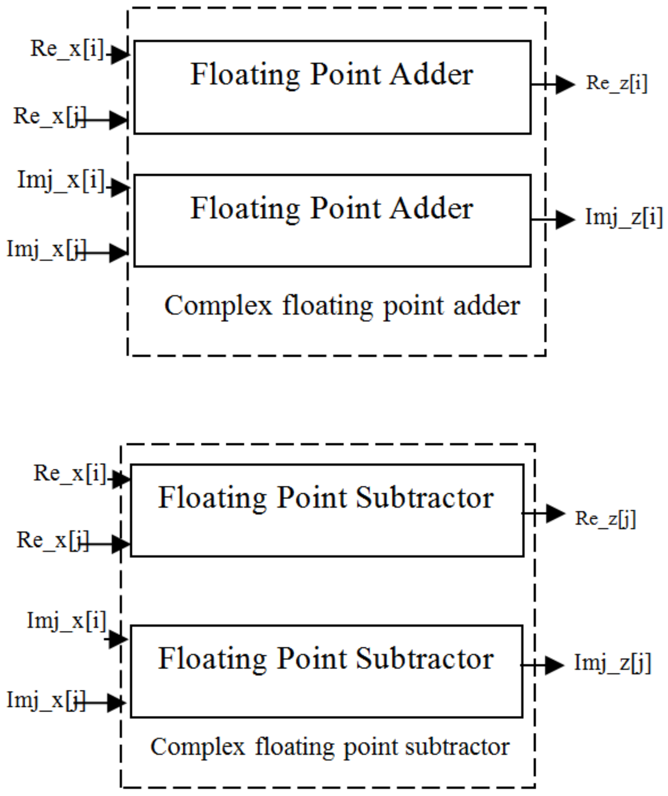


Figure 6 : Butterfly Unit

2. Complex Multiplier

Since complex multiplication is an expensive operation, to reduce the multiplicative complexity of the twiddle factor inside the butterfly processor by calculating only three real multiplications and three add/subtract operations.

The twiddle factor multiplication:

$$R + jI = (X + jY).(C + jS)$$

However the complex multiplication can be simplified:

$$R = (C - S).Y + Z$$

$$I = (C + S).X - Z$$

With: $Z = C.(X - Y)$

C and S are pre-computed and stored in a memory table. Therefore it is necessary to store the following three coefficients C, C + S, and C - S.

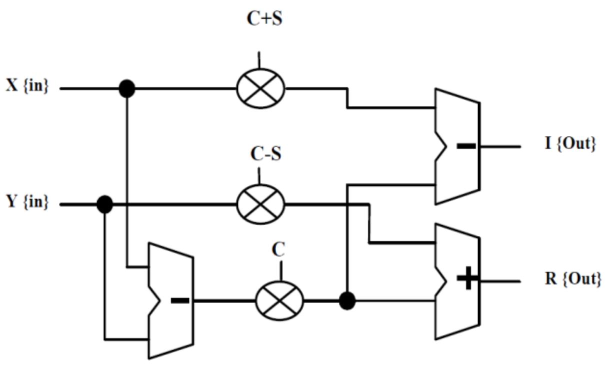


Figure 7: Complex Multiplier

The implemented algorithm of complex multiplication used in this work uses three multiplications, one addition and two subtractions as shown in Fig. 7. This is done at the cost of an additional third memory table for storing twiddle factors. In the hardware description language (VHDL) program, the twiddle factor multiplier was implemented using component instantiations of IP_CORE FPM.

3. –j Multiplier

This multiplier can be implemented using simple exchange operation and a not gate to change sign. Figure 8 shows the exchange operation

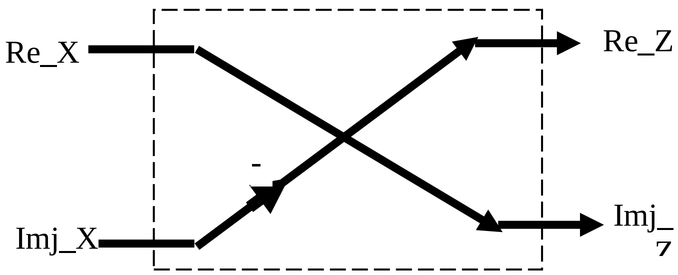


Figure 8 : j Multiplier

As shown in figure –j multiplier can be implemented using exchange operation using the following equations.

X = Re_X + jImj_X

Z = (-j)(X) = (-j)(Re_X + jImj_X) = -jRe_X + Imj_x

Z = Re_Z + imj_Z

Re_Z = imj_X

Imj_Z = - Re_X

This exchange operation requires only one NOT gate to invert the sign of Re_X signal and no other resources are used.

4. Controller & Router

Figure 9 shows the state machine diagram of controller and routing network for design 1.

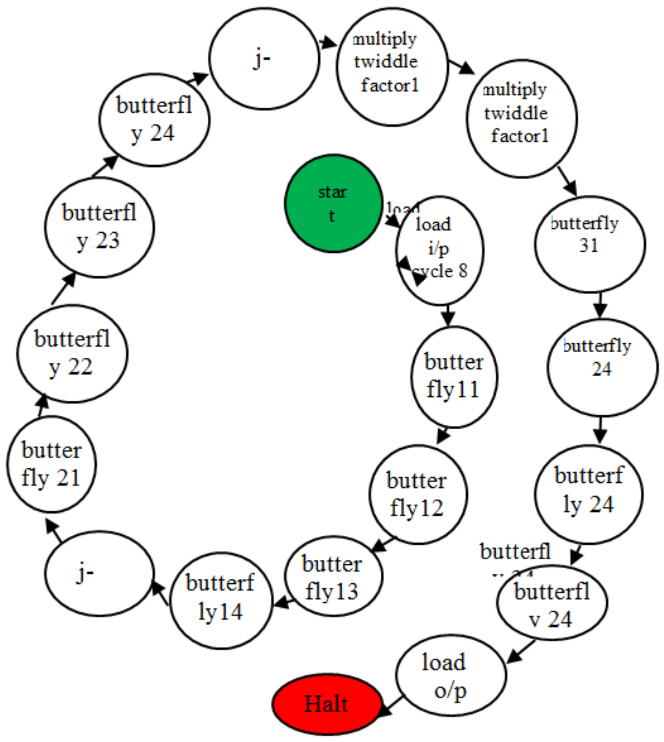


Figure 9 : State Machine controller – Design 1

The process starts by loading the inputs vectors. In design 1 serial data feed is used so it takes 8 cycles to load data, then to compute the stage 1 of FFT four butterfly cycles are used Butterfly11, Butterfly12, Butterfly13 and Butterfly14. A single butterfly unit is used four times to implement the stage 1. Then the outputs z[6] and z[7] of stage 1 are multiplied by –j. two exchange units are used to implement this process. Then stage 2 is implemented by using the previously used butterfly unit four times, the process state are called butterfly21, butterfly22, butterfly23 and butterfly24. For stage 3 implementation first the –j multiplication takes place for the stage 2 output a[3], this state is named as –j. next the multiplication of 2 twiddle factor takes place for variables a[5] and a[7], this is implemented using a single multiplier which is used twice. The states are called multiply twiddle factor 1 and multiply twiddle factor 2. Then four butterfly states butterfly31, butterfly32, butterfly33 and butterfly34 states are used to compute the final FFT which are stored in internal register named b. then these are assigned serially to the output ports p and q.

b) Design - 2

Figure 10 shows the high level diagram of design 2 of 8 point FFT. Here the difference between design 1 and design 2 is that in design 1 only 1 butterfly unit was used, This design 2 uses four butterfly unit. Design 1 uses single complex multiplier whereas design 2 uses two complex multipliers. This will increase area requirement but in turn decrease latency of the design.

Figure 11 shows the state machine controller for design 2 of FFT. Here parallel data load is used instead of serial data load used in design 1. It takes only 1 cycle to load the complete data. Then 4 butterfly units operate in parallel to calculate the stage 1 of FFT. This will only take 1 cycle then a –j multiplier is used this state is called –j. then the previously used 4 butterfly units are used in parallel to compute stage 2 of FFT. Then –j unit is again used for stage 3. Here the two complex multipliers are used in parallel in stage 3. After this 4 butterfly units are reused to compute stage 3 of FFT. After this all the eight outputs are loaded to the output port in one cycle.

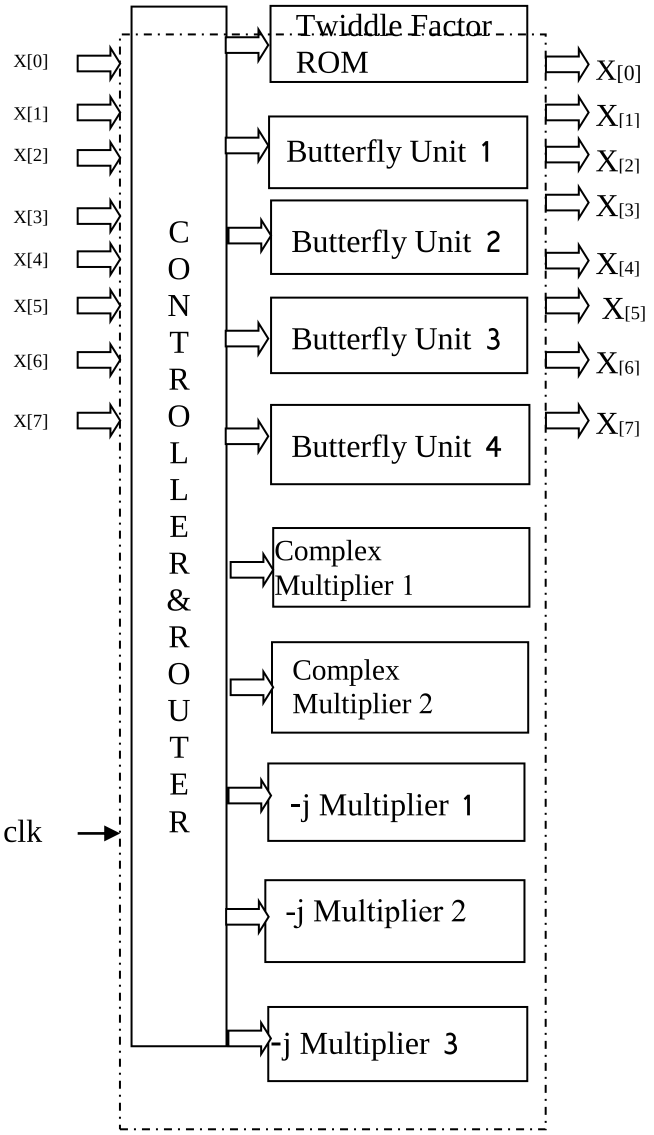


Figure 10 : High Level Block Diagram of design 2

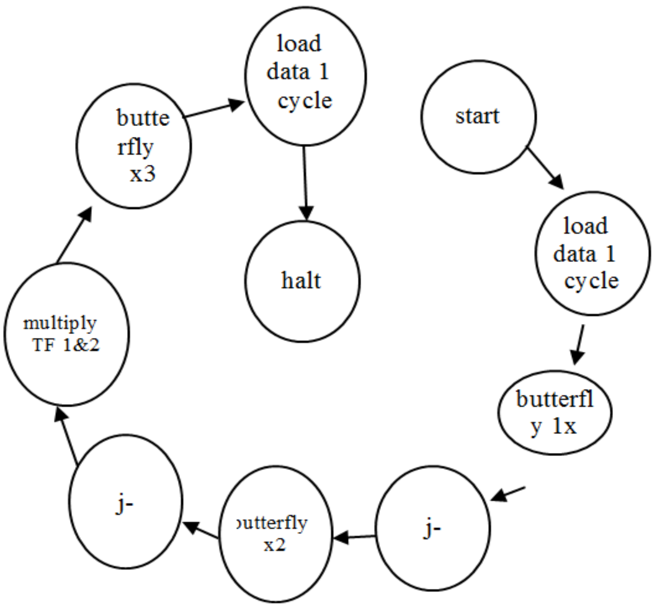


Figure 11 : High Level Block Diagram of design 2

III. RESULTS AND DISCUSSION

A. Results

This chapter shows about the synthesis results of proposed 8 point FFT designs and its behavioral simulation. In this work we have used Xilinx 14.1i for the design and implementation. Xilinx Xpower Analyzer is used for power analysis; Xilinx XST is used for the synthesis of the design and Xilinx ISIM for the behavioral simulation of the design. We have used Virtex 6 FPGA for the implementation for the design. Table 1 and 2 behavioral simulations were presented and it can be concluded that our two designs namely design 1 and design 2 are working correctly. Table depicts the synthesis report of design 1.

Table 1: Synthesis Report – Design 1

Device Utilization Summary			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	632	595200	0%
Number of Slice LUTs	1146	297600	0%
Number of fully used LUT-FF pairs	98	1680	5%
Number of bonded IOBs	34	840	4%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48E1s	6	2016	0%

Table 2 shows the device utilization summary of design 2. Design 2 takes more area than design 1 as it uses 4 butterfly units and two complex multipliers compared to 1 butterfly unit and 1 complex multiplier of design 1.

Table 2 : Synthesis Report - Design 2

Device Utilization Summary			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	596	595200	0%
Number of Slice LUTs	2643	297600	0%
Number of fully used LUT-FF pairs	187	3052	6%
Number of bonded IOBs	258	840	30%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48E1s	12	2016	0%

B. Comparison

This section compares the two proposed designs namely design1 and design 2 with other designs available in literature. Table shows the comparison summary.

Parameters	**Design 1**	**Design 2**	**Base**
Number of complex multiplier	1	2	NA
Number of Multipliers	3	6	8
Number of Adders/Subtractor	7	22	25
Number of Stages	1	3	3
Number of Butterfly Cycles	12	3	3
Number of DSP Blocks	6	12	16
Number system	Floating Point	Floating Point	Fixed Point

IV. CONCLUSION

From the behavioral simulation we can conclude that our Fast Fourier t*ransform is working properly without generating any errors. The synthesis report and the comparison table suggest that the hardware resource usage is reduced by many folds compared to the other designs available in literature. Design 1 uses only 1 complex multiplier which internally uses 3 normal multipliers, number of adder used in design 1 is only, because we have used only one butterfly unit and one complex multiplier. But the latency of this design is high it takes 12 butterfly cycles to compute FFT. Design 2 is comparatively high performance but it also takes large area. It uses 2 complex multipliers which employs 6 normal bit multipliers and a total of 22 adders/subtractor. Here the overall FFT is computed in three butterfly cycles because four butterfly units and two complex multipliers are used. When we compare these two designs with other designs in literature these two proposed designs uses less area. Design 2 is similar to the reference design found in literature but it uses less number of multiplier and adder/subtractor blocks.

V. REFERENCES

[1] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics of Computation, 19(90):297–301, 1965.

[2] M. Heideman, D.H. Johnson, and C.S. Burrus. Gauss and the history of the fast fourier transform. ASSP Magazine, IEEE, 1(4):14–21, 1984.

[3] Zahra Haddad Derafshi, Javad Frounchi, Hamed Taghipour "FPGA Implementation of a 1024-Point Complex FFT Processor" Proceeding 2010 IEEE Second International Conference on Computer and Network Technology pp312-315.

[4] J.E.Volder, "The CORDIC trigonometric compo technique" IRE Trans. Elec. Compo (1959), vol. EC-8-3, pp. 330-334.

[5] F.Angarita, A Perez-Pascual, T.Sansaloni, J.Valls, "Efficient Fpga Implementation of Cordic Algorithm for Circular and Linear Coordinates", Proceedings 2005 International Conference on Field Programmable Logic and Applications, pp535-538.

[6] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.

[7] D. Goldberg – What Every Computer Scientist Should Know About Floating Point Arithmetic – ACM Computing Surveys, Vol 32, No 1, 1991.

[8] ANSI/IEEE 754-1985 Standard for Binary Floating Point Arithmetic.

[9] More, Tushar V., and Ashish R. Panat. "FPGA implementation of FFT processor using vedic algorithm." Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on. IEEE, 2013.

[10] Ayhan, Tuba, Wim Dehaene, and Marian Verhelst. "A 128∶ 2048/1536 point FFT hardware implementation with output pruning." Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European. IEEE, 2014.

[11] Chen, Ren, and Viktor K. Prasanna. "Energy optimizations for FPGA-based 2-D FFT architecture." High Performance Extreme Computing Conference (HPEC), 2014 IEEE. IEEE, 2014.

[12] Kumar, M., A. Selvakumar, and P. M. Sobha. "Area and frequency optimized 1024 point Radix-2 FFT processor on FPGA." VLSI Systems, Architecture, Technology and Applications (VLSI-SATA), 2015 International Conference on. IEEE, 2015.

[13] Arioua, Mounir, et al. "VHDL implementation of an optimized 8-point FFT/IFFT processor in pipeline architecture for OFDM systems." Multimedia Computing and Systems (ICMCS), 2011 International Conference on. IEEE, 2011.

[14] Akshata.A, Gopika.D.K, Hameem Shanavas.I “FPGA Implementation of Decimation In Time FFT or Discrete Fourier Transform” JEST-M, Vol.1, Issue 2, 2012.

[15] Mukherjee, Atin, Amitabha Sinha, and Debesh Choudhury. "A Novel Architecture of Area Efficient FFT Algorithm for FPGA Implementation."arXiv preprint arXiv:1502.07055 (2015).

[16] Das, Ansuman Diptisankar, et al. "Efficient VLSI Architectures of Split-Radix FFT using New Distributed Arithmetic." International Journal of Soft Computing and Engineerimg (IJSCE) ISSN: 2231-2307.

[17] Wang, Zhen, et al. "A Combined SDC-SDF Architecture for Normal I/O Pipelined Radix-2 FFT." (2013).