International Journal For Technological Research In Engineering
Volume 1, Issue 10, June-2014 *ISSN (Online): 2347 - 4718*

EFFICIENT VLSI ARCHITECTURE USING DIT-FFT RADIX-2 AND SPLIT RADIX FFT ALGORITHM

Rashmi M J[1], G S Biradar[2], Meenakshi Patil[3]
[1]VLSI Design & Embedded Systems, Department of PG Studies, VTU RC, Gulbarga, India.
[2,3]Department of Electronics & Communication Engg., PDA College Of Engineering, Gulbarga, India.

Abstract: FFT has wide use in communication for processing the data being exchanged. Hence it is important to develop high-performance FFT architecture to meet the requirements of real time and low cost in many different systems. Efficient VLSI architecture based on Field Programmable Gate Array (FPGA) for Wireless Local Area Networks (WLAN) is presented in this paper. This paper concentrates on the development of the Fast Fourier Transform (FFT), based on Decimation-In-Time (DIT) domain, Radix-2 FFT algorithm and Split Radix FFT Algorithm and finally architectures by two different algorithms are compared for speed and device utilization. This paper concerns about design of DIT-FFT for different sized inputs using Verilog HDL as a design entity, and their Synthesis by Xilinx Synthesis Tool on Spartan kit. Among the different proposed algorithms, split-radix FFT has shown considerable improvement in terms of reducing hardware complexity of the architecture compared to radix-2 and radix-4 FFT algorithms. The synthesis results show that the computation for calculating the 32-point Fast Fourier transform is efficient in terms of speed.
Keywords: Butterfly, DFT, Split Radix, FFT, VLSI.

I. INTRODUCTION

Fast Fourier Transform (FFT) has become ubiquitous in many engineering applications [1]. High-speed FFT architectures are necessary to implement several communication systems, signal processing systems, etc. [2] – [4]. The FFT blocks are also used in mechanical engineering and civil engineering applications [5] – [6]. FFT has been considered as the most efficient way of implementing the discrete Fourier transform (DFT) and it was first implemented in 1965 [7]. The efficiency of the FFT algorithm lies in its reduced number of arithmetic operations. DFT has the order O (N * N) of arithmetic operations whereas FFT has the order of O (N log N) arithmetic operations. If the architecture is designed for complex inputs, the number of arithmetic operations becomes approximately double when compared to those which are designed for real inputs. One of the disadvantages of conventional FFT architectures is the presence of multiplier blocks, which has increased hardware, increased power consumption and reduced operating frequency. The basic FFT design is based on radix-2 butterfly block, which was proposed by Cooley-Tukey [7]. Recent advances in the algorithm include FFT architectures based on higher and split-radix such as radix-4, radix-8, radix-2k, etc. [8] – [12]. Split-radix FFT is one of the FFT algorithms that use combination of different radix FFT. Split-radix FFT algorithm combines simplicity of radix-2 FFT with less computational complexity radix-4 FFT. The advantage of split-radix FFT is that it has considerably fewer number of arithmetic computations compared to that of radix-4 and radix-2 FFT. Split-radix also has several other advantages such as regular structure, no reordering of internal signals except for outputs, etc. Since it mostly uses radix-2 block in its architecture, it is possible to implement split-radix FFT for inputs of kind 2k, k being an integer. In the following sections, first section presents a Radix-2 Cooley-Tukey FFT Algorithm. Then, next section presents efficient VLSI architectures of DIT-FFT using split-radix algorithm. Next, proposed architectures with the existing ones are compared. Final section concludes the paper with mentioning possible further improvements.

II. VLSI ARCHITECTURES USING DIT-FFT RADIX-2 ALGORITHM

Butterfly is a portion of the computation that combines the results of smaller discrete Fourier transforms (DFTs) into a larger DFT, or vice versa (breaking a larger DFT up into sub-transforms). The name "butterfly" comes from the shape of the data-flow diagram in the radix-2 case, as described below.[A] The same structure can also be found in the Viterbi algorithm, used for finding the most likely sequence of hidden states. Most commonly, the term "butterfly" appears in the context of the Cooley–Tukey FFT algorithm, which recursively breaks down a DFT of composite size n = rm into r smaller transforms of size m where r is the "radix" of the transform. These smaller DFTs are then combined via size-r butterflies, which themselves are DFTs of size r (performed m times on corresponding outputs of the sub-transforms) pre-multiplied by roots of unity (known as twiddle factors). This is the "decimation in time (DIT)".

A. Block diagram of Radix-2 FFT Algorithm

In the case of the radix-2 Cooley–Tukey algorithm, butterfly is basic block which is shown in Fig. 1.

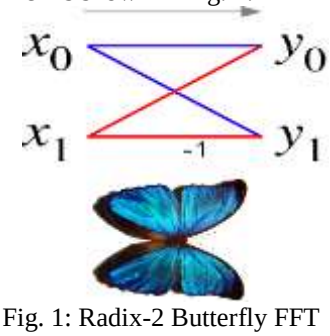

Fig. 1: Radix-2 Butterfly FFT

Butterfly is simply a DFT of size-2 that takes two inputs (x_0, x_1) (corresponding outputs of the two sub-transforms) and gives two outputs (y_0, y_1) by the formula (not including twiddle factors):

$$y_0 = x_0 + x_1$$
$$y1 = x_0 - x_1 \quad (1)$$

If one draws the data-flow diagram for this pair of operations, the (x0, x1) to (y0, y1) lines cross and resemble the wings of a butterfly, hence the name (see also the illustration at right).
More specifically, a decimation-in-time FFT algorithm on n = 2 p inputs with respect to a primitive n^{th} root of unity w_n^k relies on O (n log n) butterflies of the form:

$$y_0 = x_0 + x_1 w_n^k$$
$$y_1 = x_0 - x_1 w_n^k \quad (2)$$

where k is an integer depending on the part of the transform being computed.
In general N-point DFT of a sequence x(n) is given by

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad (3)$$

$$k = 0,1,\ldots\ldots, N-1$$

The radix-2 algorithms are the simplest FFT algorithms. The decimation-in-time (DIT) radix-2 FFT recursively partitions a DFT into two half-length DFTs [13] of the even-indexed and odd-indexed time samples. The outputs of these shorter FFTs are reused to compute many outputs, thus greatly reducing the total computational cost. The radix-2 decimation-in-time and decimation-in-frequency fast Fourier transforms (FFTs) are the simplest FFT algorithms. Like all FFTs, they gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs.

1) 4-point Radix-2 FFT: 4-point transform can be reduced to two 2-point transforms: one for even elements, one for odd elements. The odd one will be multiplied by W4k. Diagrammatically; this can be represented as two levels of butterflies. Notice that using the identity WN/2n = WN2n, we can always express all the multipliers as powers of the same WN (in this case we choose N=4).

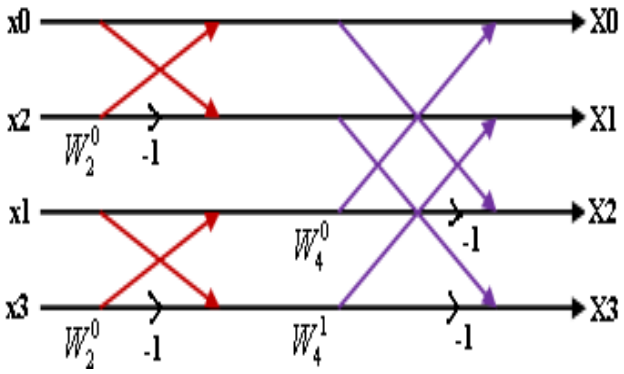

Fig. 2: 4-point Radix-2 FFT

2) 8-point Radix-2 FFT: An 8 input butterfly diagram has 12 2-input butterflies and thus 12*2 = 24 multiplies.
N Log N = 8 Log (8) = 24. A straight DFT has N*N multiplies, or 8*8 = 64 multiplies. That's a pretty good savings for a small sample. The savings are over 100 times for N = 1024 and this increases as the number of samples increases.

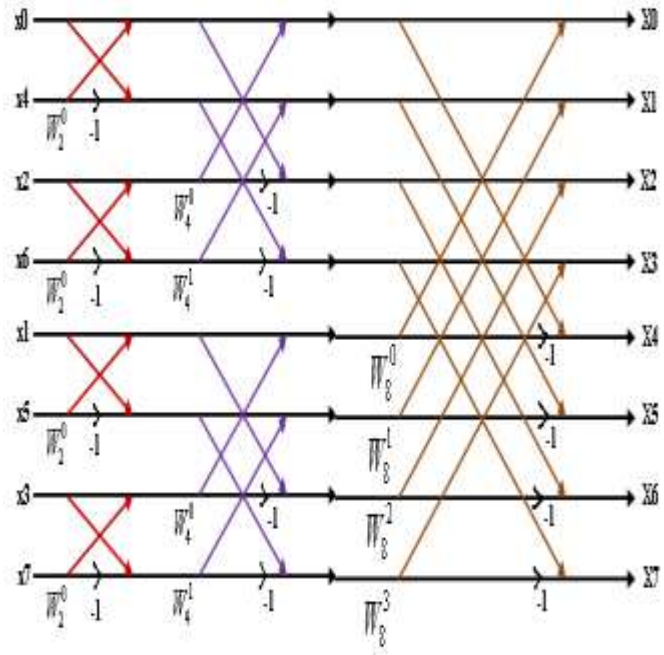

Fig. 3: 8-point Radix-2 FFT

3)16-point Radix-2 FFT: 16 input butterfly diagram has 32 2-input butterflies and thus 32*2 = 64 multiplies. N Log N = 16 Log (16) = 64. A straight DFT has N*N multiplies, or 16*16 = 256 multiplies.

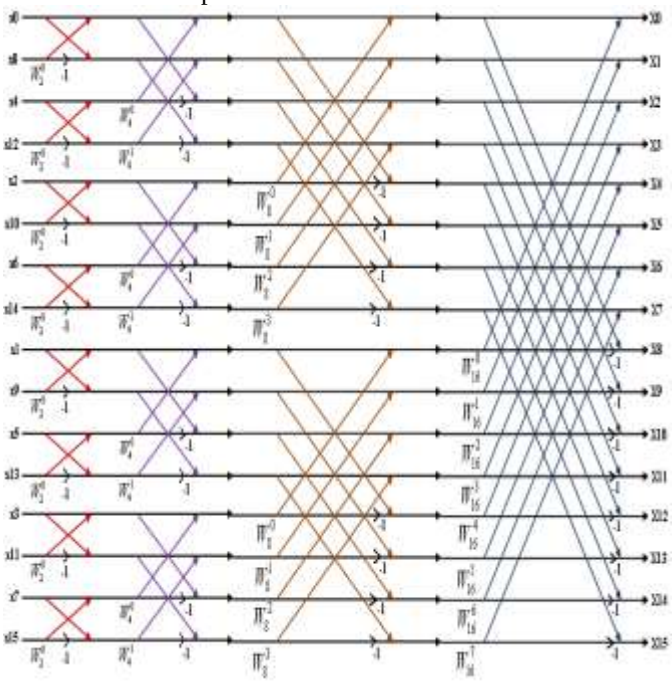

Fig. 4: 16-point Radix-2 FFT

4) 32-point Radix-2 FFT: 32 input butterfly diagram has 64 2-input butterflies and thus 64*2 = 128 multiplies. N Log N = 32 Log (32) = 128. A straight DFT has N*N multiplies, or 32*32 = 1024 multiplies.

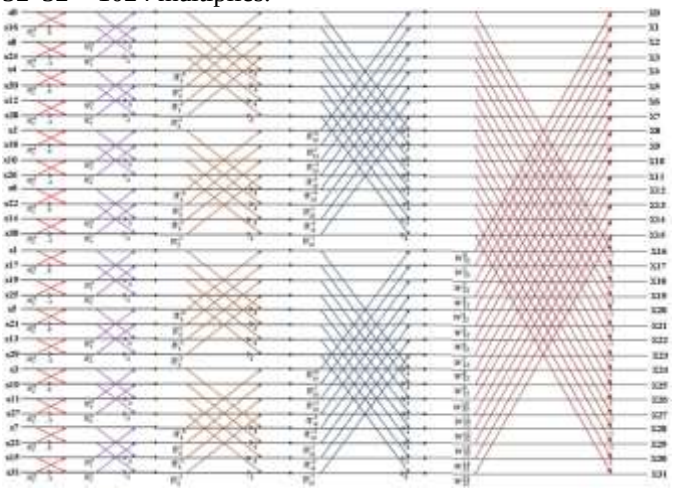

Fig. 5: 32-point Radix-2 FFT

III. VLSI ARCHITECTURES USING SPLIT-RADIX FFT ALGORITHM

The split-radix FFT is a fast Fourier transform (SRFFT) algorithm for computing the discrete Fourier transform (DFT). split radix is a variant of the Cooley-Tukey FFT algorithm that uses a blend of radices 2 and 4: it recursively expresses a DFT of length N in terms of one smaller DFT of length N/2 and two smaller DFTs of length N/4. The split-radix algorithm can only be applied when N is a multiple of 4, but since it breaks a DFT into smaller DFTs it can be combined with any other FFT algorithm as desired. While calculating FFT using Radix-2 method, it can be concluded that the even-numbered points and the odd-numbered points are computed independently. This leads to the possibility of using different computational methods for different independent parts of the algorithm which will reduce computational complexity. Split-radix algorithm uses the above method by combining the simplicity of radix-2 algorithm and lesser computational complexity of radix-4 algorithm, achieving the lowest number of arithmetic operation count to compute DFT of power-of-two sizes N. Split-radix method recursively expresses DFT of length N in terms of one smaller DFT of length N/2 and two smaller DFTs of length N/4. Split-radix is only applicable when N is a multiple of 4, but we can combine this with other FFT algorithms. The algorithm for the fast and less complexity computation of the DFT by Split-radix (SRFFT) was developed by Duhamel and Hollmann [16], [17] for data sequences having a length N that is an integer power of 2. According to them, the even-numbered samples of the N-point DFT can be calculated by

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1}[x(n) + x(n+\frac{N}{2})]W_{N/2}^{nk} \qquad (4)$$

$$k = 0,1,.....\frac{N}{2}-1$$

Those even-numbered DFT points can be calculated without any additional multiplications. So, radix-2 algorithm is sufficient for the above calculation. The odd-numbered samples X(2k+1) requires an additional multiplication of twiddle factor W_N^n.

To implement this, radix-4 algorithm is used for its lesser computational complexity. Using radix-4 algorithm for the odd –numbered samples of the N-point DFT, the following N/4-point DFTs are obtained.

$$X(4k+1) = \sum_{n=0}^{\frac{N}{4}-1}[\{x(n) - x(n+\frac{N}{2})\} - j\{x(n+\frac{N}{2}) - x(n+\frac{3N}{4})\}]W_N^n W_{N/4}^{nk}$$

$$k = 0,1,.....,\frac{N}{4}-1 \qquad (5)$$

And

$$X(4k+3) = \sum_{n=0}^{\frac{N}{4}-1}[\{x(n) - x(n+\frac{N}{2})\} + j\{x(n+\frac{N}{2}) - x(n+\frac{3N}{4})\}]W_N^{3n} W_{N/4}^{nk}$$

$$k = 0,1,.....,\frac{N}{4}-1 \qquad (6)$$

Hence, the N-point DFT now has been decomposed into one N/2-point DFT without phase factor and another two N/4-point DFTs with phase factor. Fig. 6 shows the split-radix butterfly unit and Fig. 7 shows its equivalent.

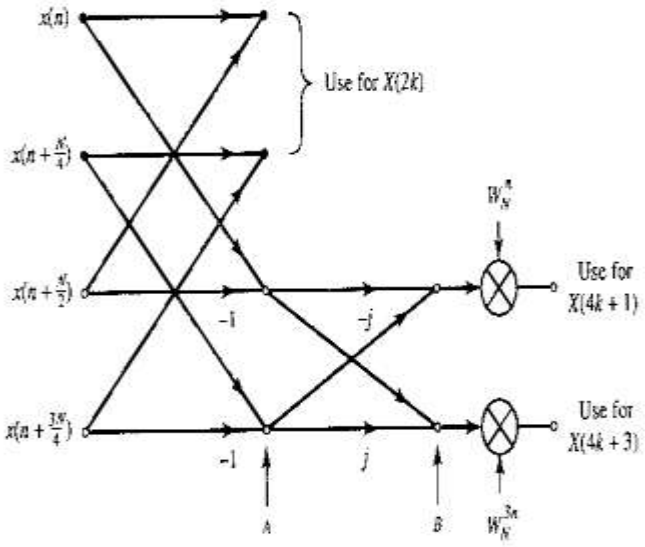

Fig. 6: Split-radix butterfly unit

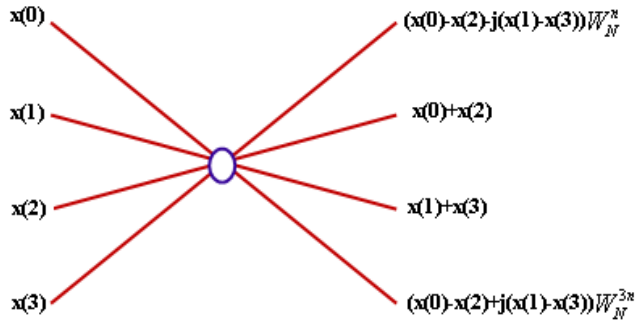

Fig. 7: Equivalent representation of Split-radix butterfly unit

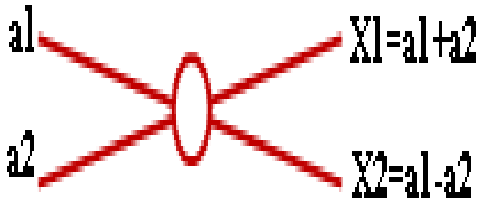

Fig. 8: Radix-2 butterfly used in SRFF architectures

A. 4-point SRFFT

The 4-point SRFFT uses one split radix butterfly unit and one radix-2 butterfly for computation of final output.

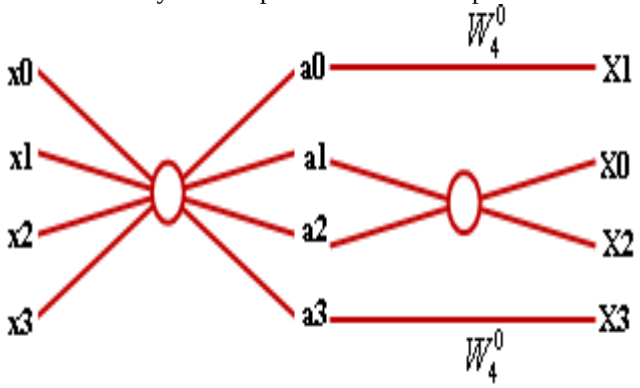

Fig. 9: 4-point SRFFT

B. 8-point SRFFT

The 8-point SRFFT uses 3 split radix butterfly unit and 3 radix-2 butterfly unit for computation of final output.

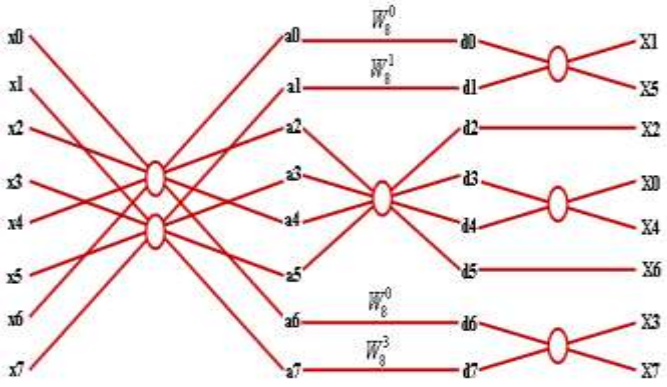

Fig. 10: 8-point SRFFT

C. 16-point SRFFT

The 16-point SRFFT uses 9 split radix butterfly unit and 5 radix-2 butterfly for computation of final output.

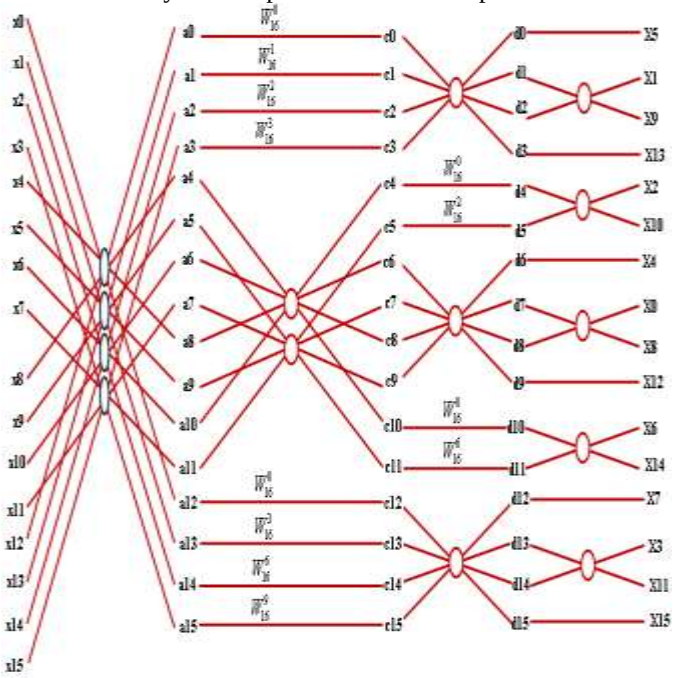

Fig. 11:16-point SRFFT

D. 32-point SRFFT

The 32-point SRFFT uses 23 split radix butterfly unit and 11 radix-2 butterfly for computation of final output.

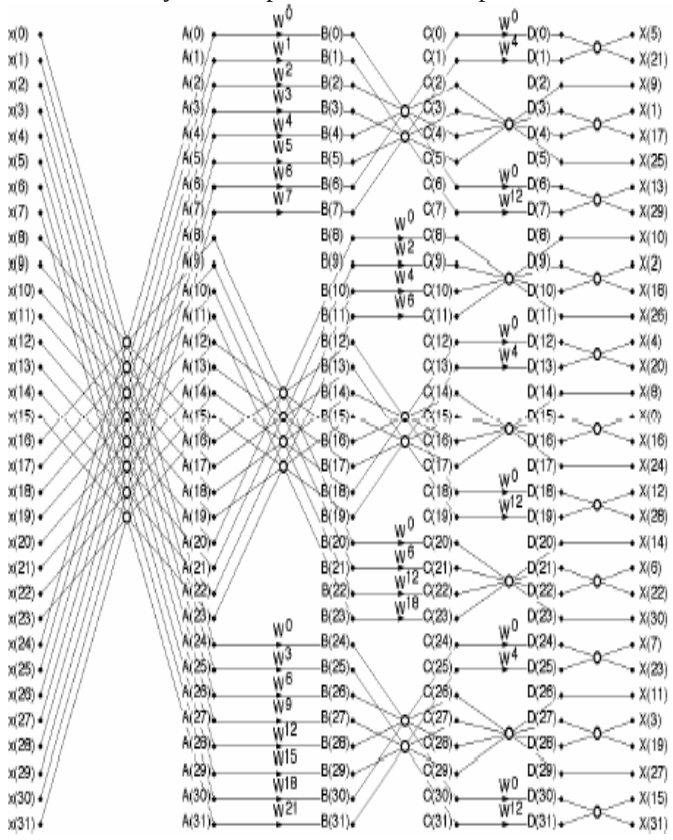

Fig. 12: 32-point SRFFT

IV. SOFTWARE SIMULATION RESULTS

FFT block of signal length 4 is simulated and synthesized using the Xilinx Design Suite 12.1. The RTL block thus obtained for the decimation in time domain radix -2 Fast Fourier transform algorithm is shown

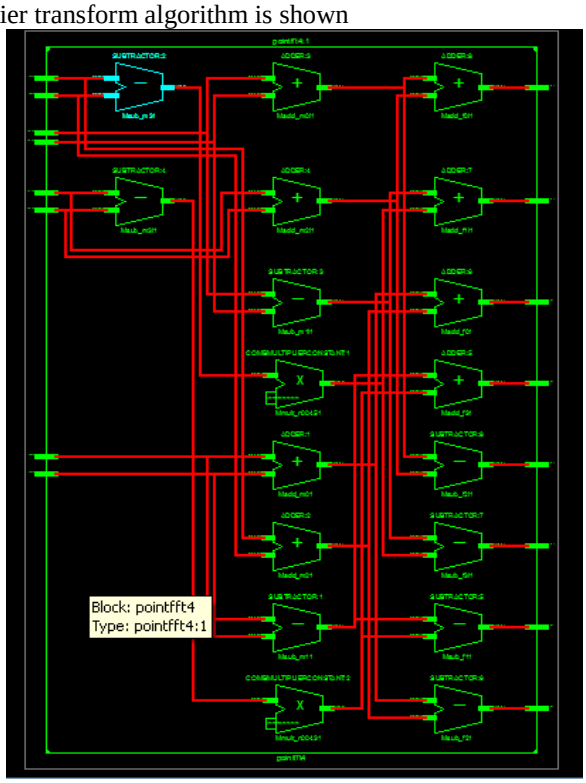

Fig. 13: RTL Schematic of 4-point Radix-2 FFT

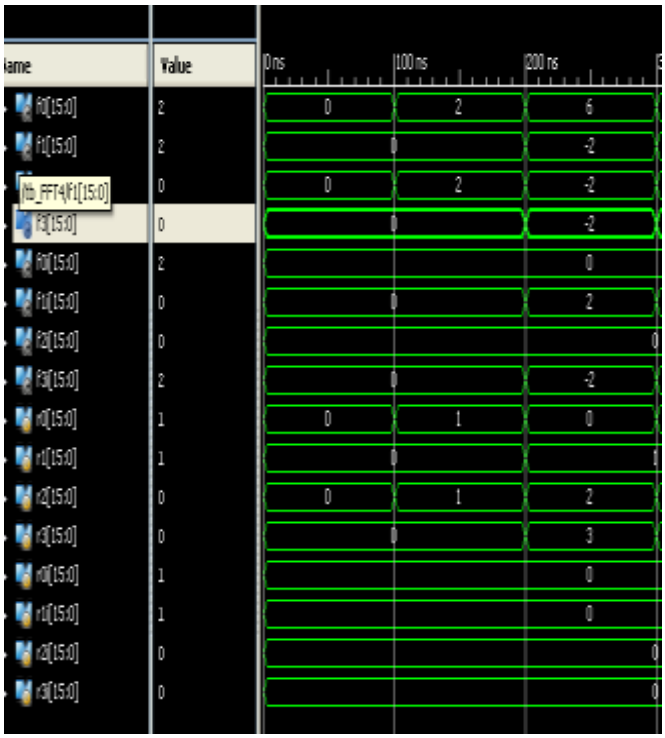

Fig. 14: Simulation result of 4-point SRFFT

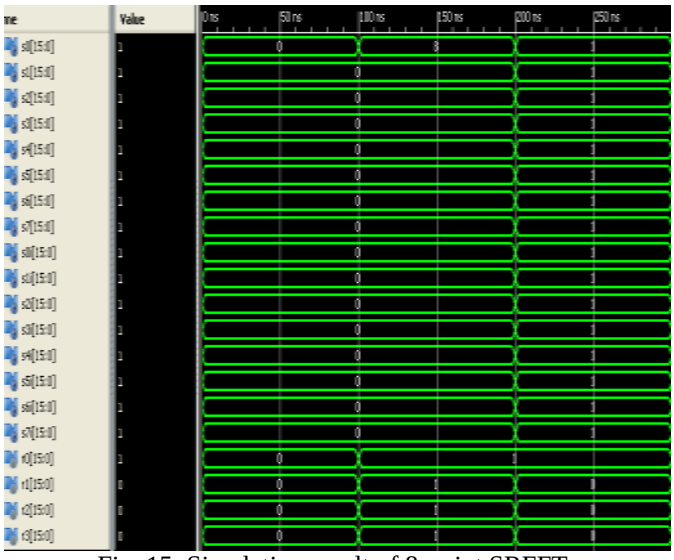

Fig. 15: Simulation result of 8-point SRFFT

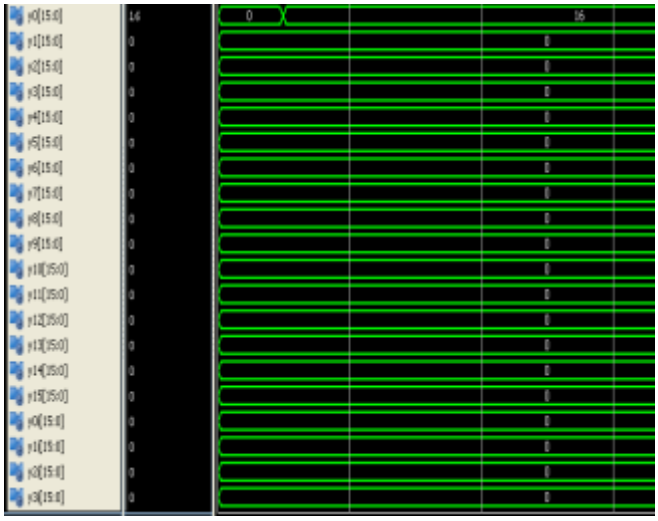

Fig. 16: Simulation result of 16-point SRFFT

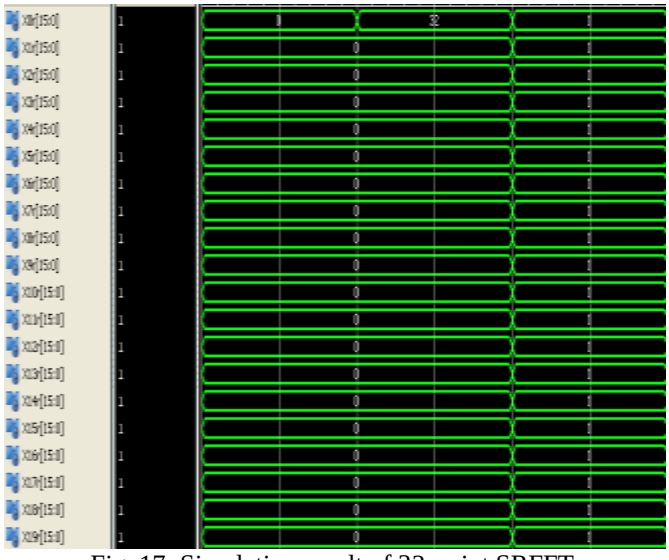

Fig. 17: Simulation result of 32-point SRFFT

TABLE I
Memory utilization and Minimum Delay of Radix-2 FFT

	Total memory usage	**Minimum Delay**
4 point FFT	159000 kb	2.780 ns
8 point FFT	166488 kb	7.663 ns
16 point FFT	195800 kb	12.588 ns
32 point FFT	324568 kb	17.514 ns

TABLE II
Memory utilization and Minimum Delay of Split Radix FFT

	Total memory usage	**Minimum Delay**
4 point FFT	159000 kb	2.107 ns
8 point FFT	165656 kb	7.000 ns
16 point FFT	189272 kb	8.444 ns
32 point FFT	266776 kb	13.362 ns

Table I and Table II shows the memory utilization and the minimum delay of Radix-2 FFT nd Split Radix FFT for different signal length respectively. From the tables it can be observed that the there is almost same memory usage for the both architectures when the signal length is very small. However the memory usage is improved for Split Radix FFT architectures by an amount of 6.5 megabytes as the signal length is increased. And finally there is a great improvement in the memory usage required by an amount of 58 megabytes for Split Radix FFT compared to that of Radix-2 FFT for the very high signal length. It can be clearly seen in the below 2D graph in Fig. 18.

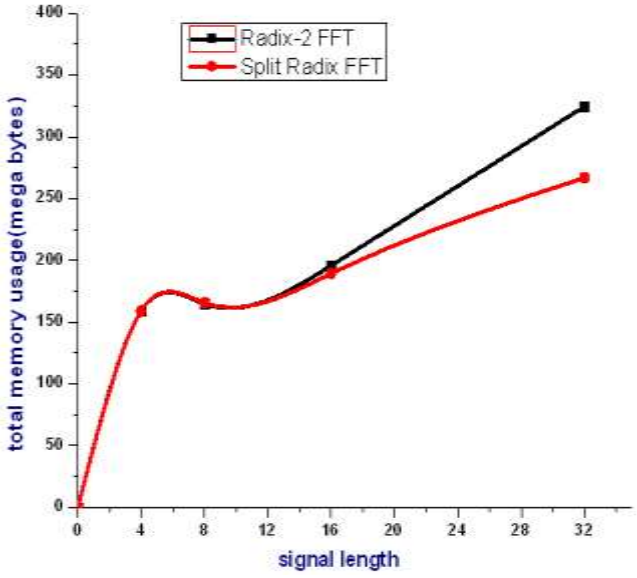

Fig. 18: Device utilization with variable input signal length

It can also be observed from the tables that the minimum delay of Split Radix is reduced compared to that of Radix-2 FFT architecture. Split Radix FFT architecture has minimum delay which is almost 0.6 ns less than the minimum delay of Radix-2 FFT when the signal length is very small. This can be clearly observed in the below 2D graph in Fig. 19.

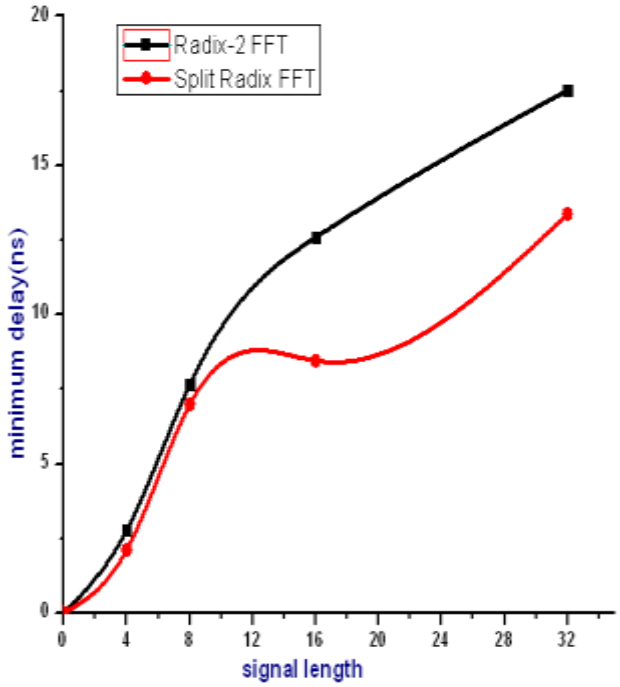

Fig. 19: Minimum delay with variable input signal length

Below Fig. 20 and Fig. 21 shows the same comparison graphs in 3D view of the graph in Fig. 18 and Fig. 19.

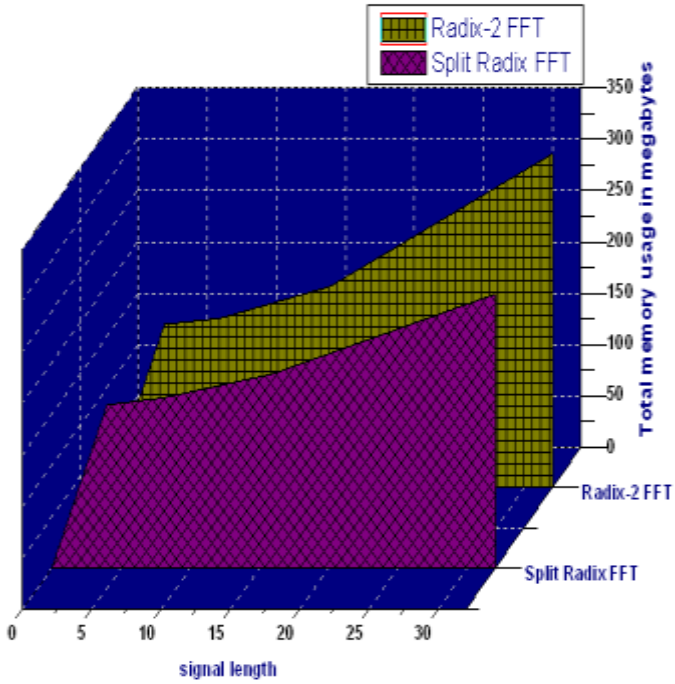

Fig. 20: 3D view of Device utilization with input signal length

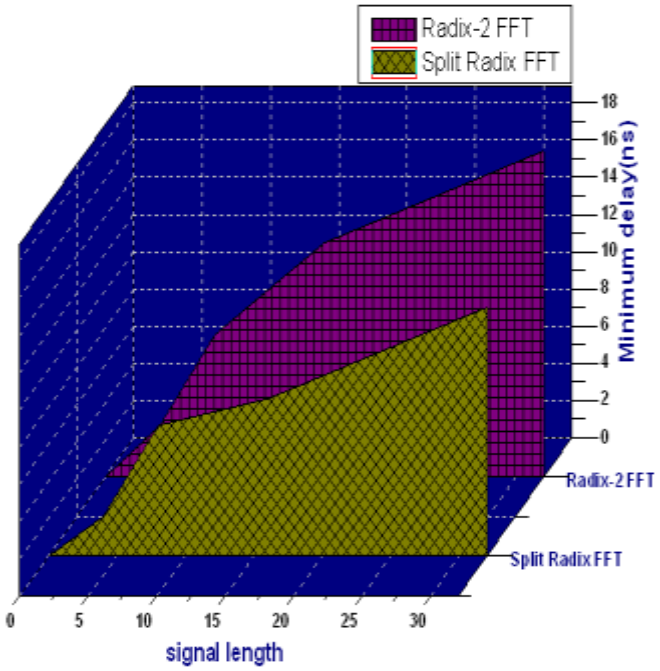

Fig. 21: 3D view of Device utilization with input signal length

V. CONCLUSIONS

This paper has reported two efficient VLSI architectures of DIT-FFT. Both proposed architectures are designed for complex inputs with a data width of 16 bits, maintained constant all along. The simulation outputs of proposed architectures have not shown much deviation from numerical values. But, proposed architecture using Radix-2 algorithm has high memory usage and also has huge number of arithmetic operations which causes the delay to be maximum as the signal length increases; this drawback has been overcome in the later VLSI architecture using Split Radix FFT Algorithm. Performance parameters of both architectures shown in table and graph clearly show the improvement of SRFFT VLSI architecture. The Delay of SRFFT less than that of Radix-2 FFT as the signal length increases.

REFERENCES

[1] P. Duhamel and M. Vetterli, "Fast Fourier Transforms: A Tutorial Review and A State of The Art," IEEE Signal Processing Society, vol. 4, no. 19, 1990, pp. 259 – 299.
[2] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, "A 1-GS/s FFT/IFFT processor for UWB applications," IEEE Journal of Solid-State Circuits, vol. 40, no. 8, Aug. 2005, pp. 1726 – 1735
[3] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-GS/s FFT Processor for OFDM-Based WPAN Applications," IEEE Trans. Circuits Syst. II: Exp. Briefs, vol. 57, no. 6, Jun. 2010, pp. 451 – 455.
[4] J ohn G. Proakis, Dimitris G. Manolakis, "Digital Signal Processing: Principles, Algorithms, and Applications", Prentice- Hall, 1998.
[5] Z. Ismail, N. H. Ramli, Z. Ibrahim, T. A. Majid, G. Sundaraj, and W. H. W. Badaruzzaman, "Design Wind Speeds using Fast Fourier Transform: A Case Study," Computational Intelligence in Control, Idea Group Publishing, 2012, ch. XVII.
[6] Robert Frey, "The FFT Analyzer in Mechanical Engineering Education," Sound and Vibration: Instrumentation Reference Issue, Feb. 1999, pp. 1 – 3.
[7] James W. Cooley and John W. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol. 19, 1965, pp. 297 – 301.
[8] Mario Garrido, J. Grajal, M. A. Sánchez, and Oscar Gustafsson, "Pipelined Radix-2k Feedforward FFT Architectures," IEEE Trans. VLSI Syst., vol. 21, no. 1, Jan. 2013, pp. 23 – 32.
[9] Y. Chen, Y. Tsao, Y. Wei, C. Lin, and C. Lee, "An indexed- scaling pipelined FFT processor for OFDM-based WPAN applications," IEEE Trans. Circuits Syst. II: Exp. Briefs, vol. 55, no. 2, Feb. 2008, pp. 146–150.
[10] M. Shin and H. Lee, "A high-speed four-parallel radix-24 FFT processor for UWB applications," Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), 2008, pp. 960–963.
[11] F. Arguello and E. Zapata, "Constant geometry split-radix algorithms," Journal of VLSI Signal Processing, 1995.
[12] Steven G. Johnson and Matteo Frigo, "A Modified Split-Radix FFT with Fewer Arithmetic Operations," IEEE Trans. Signal Processing, vol. 55, no. 1, Jan. 2007, pp. 111 – 119.
[13] Stanley A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," IEEE ASSP Magazine, vol. 6, no. 3, Jul. 1989, pp. 4 – 19.

