INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2021, VOL. 67, NO. 4, PP. 631–637
Manuscript received August 27, 2021; revised November, 2021. DOI: 10.24425/ijet.2021.137856

High Performance DIF-FFT Using Dissimilar Partitioned LUT Based Distributed Arithmetic

Kusma Kumari Cheepurupalli, Muntha Charan, Jammu Bhaskara Rao, and Mahammad S Noor

***Abstract*—Real-time data processing systems utilize Digital Signal Processing (DSP) functions as the base modules. Most of the DSP functions involve the implementation of Fast Fourier Transform (FFT) to convert the signals from one domain to another domain. The major bottleneck of Decimation in frequency-Fast Fourier Transform (DIF-FFT) implementation lies in using a number of Multipliers. Distributed arithmetic (DA) is considered as one of the efficient techniques to implement DIF-FFT. In this approach, the multipliers are not used. The proposed technique exploits the very advantage of the look-up table by storing the Twiddle factors, thereby avoiding the multipliers required in the butterfly structure. DIF-FFT using Distributed Arithmetic (DIF-FFT DA) models, with different adders such as Ripple carry adder (RCA), Carry-lookahead adder (CLA), and Sklansky prefix graph adder, are proposed in this paper. The three proposed models are synthesized using Cadence 6.1 EDA tools with a 45nm CMOS technology. Compared to the traditional method, it is observed that the area is improved by 53.11%, 53.35%, and 50.15%, power is improved by 42.31%, 42.52%, and 40.39%, and delay is improved by 45.26%, 45.42%, 41.80%, respectively.**

***Keywords*—Fast Fourier Transform, Adders, Distributed Arithmetic, DSP**

I. Introduction

IN signal processing, there are many types of transforms, such as Fourier transform, Laplace transforms, and Z-Transforms to convert one form of the signal to another form of the signal. Discrete Fourier Transform (DFT) is one of the powerful mathematical tools for analyzing a discrete signal. In 1964, Kim et al. [1] used a divide and conquered approach to reduce DFT's computational complexity using FFT. This method is considered a breakthrough in the development of high speed and low complex FFT algorithms, which is still one of the majorly used algorithms in Speech Processing, Communications, and Frequency estimation. There are two types of FFT computations: Decimation in time (DIT) and Decimation in frequency (DIF), each of which has many variations such as radix-2, radix-4, split radix, etc., [2]. The radix-4 is slightly more sophisticated than the radix-2 FFT. The butterfly structure of the radix-4 algorithm consists of 4 inputs and 4 outputs, shown in Figure 1.

Radix-4 DIF-FFT requires many multipliers to multiply the twiddle factor with an integer in the butterfly stages. Every twiddle factor multiplication involves four multiplication and two addition operations. Hence radix-4 DIF-FFT requires 24 multiplications and 96 additions. Here, the number of multiplications needed is very high, making the FFT more complicated. Many multipliers in literature can optimize the DIF-FFT computations limiting them to some extent of partial product reduction. Though these multipliers are fast, their computational complexity is very high. Therefore, in this paper, multiplications are avoided by adopting the Distributed Arithmetic (DA) approach to implement a radix-4 DIF-FFT [3]. DA is a mathematical technique used to calculate the sum of products. DA-based architectures usually occupy a low area, less power, and available at less affordable prices than other architectures. Especially for FFT implementations, DA can save arithmetic calculations in butterfly structure by using look-up tables (LUTs). In the butterfly structure, twiddle factors are fixed constants, and these values are multiplied with input values that can be directly stored in the LUTs.

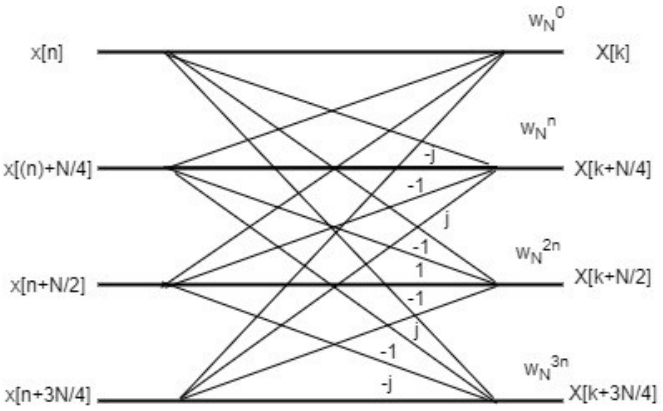

Fig. 1. Basic Radix-4 structure

The radix-4 FFT is an efficient algorithm and a mathematical tool. The advantage of radix-4 is to compute four complex multiplications at a time for one input. For an N-point DFT sequence, the twiddle factors are associated with it. This algorithm can be implemented in many ways; being radix-4 is one of the most popular [4]. The implementation of radix-4 16-point FFT requires a number of multiplier and adders. Multipliers play a critical role in FFT implementation. Fig 2 shows the Radix-4 16-point butterfly structure in a simple manner of splitting the first stage into stage-1 and stage-2. Input is given in normal order and output is taken in bit-

Authors are with Dept. of ECE, Gayatri Vidya Parishad College of Engineering, India (e-mail: chkusumasrinivas@gvpce.ac.in, charanraj-goud214rj@gmail.com, jbhaskararao@gvpce.ac.in, noor@iiitdm.ac.in).

reversal order. It consists of sixteen inputs and outputs having a real and imaginary part. For this implementation, using an 8, 10-bit array multiplier for complex multiplication for addition using different adders such as RCA, CLA, Sklansky prefix graph adder. For an N-point DFT sequence, the twiddle factors associated with it are defined by the following equations.

$$X(K) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi(\frac{nK}{N})} \tag{1}$$

$$W_N = e^{-j2\pi/N} \tag{2}$$

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{Kn} = \sum_{n=0}^{\frac{N}{4}-1} x(n) W_N^{Kn} + \sum_{n=0}^{\frac{N}{4}-1} x(n+\frac{N}{4}) W_N^{K(n+\frac{N}{4})} + \sum_{n=0}^{\frac{N}{4}-1} x(n+\frac{N}{2}) W_N^{K(n+\frac{N}{2})} + \sum_{n=0}^{\frac{N}{4}-1} x(n+\frac{3N}{4}) W_N^{K(n+\frac{3N}{4})} \tag{3}$$

$$\begin{aligned}
X(4p) &= \sum_{n=0}^{\frac{N}{4}-1} x(n) + x(n+\frac{N}{2}) + (x(n+\frac{N}{4}) + x(n+\frac{3N}{4})) W_{\frac{N}{4}}^{mn} \cap \\
X(4p+1) &= \sum_{n=0}^{\frac{N}{4}-1} \Big[x(n) - x(n+\frac{N}{2}) - j(x(n+\frac{N}{4}) - x(n+\frac{3N}{4}) \Big] W_{\frac{N}{4}}^{mn} W_N^n \cap \\
X(4p+2) &= \sum_{n=0}^{\frac{N}{4}-1} \Big[x(n) - x(n+\frac{N}{2}) - (x(n+\frac{N}{4}) + x(n+\frac{3N}{4}) \Big] W_{\frac{N}{4}}^{mn} W_N^{2n} \cap \\
X(4p+3) &= \sum_{n=0}^{\frac{N}{4}-1} \Big[x(n) - x(n+\frac{N}{2}) - j(x(n+\frac{N}{4}) - x(n+\frac{3N}{4}) \Big] W_{\frac{N}{4}}^{mn} W_N^{3n}
\end{aligned} \tag{4}$$

After evaluating the equation (3), once the summation has been split, its four terms can be equated as in equation (4). Since $X(4p)$, $X(4p+1)$, $X(4p+2)$ and $X(4p+3)$ are $\frac{N}{4}$ point FFT's this process will be repeated to obtain sixteen N/16-point FFT's.

II. Distributed arithmetic

As observed from literature, DA is considered as a suitable technique for the design of bit-level architecture to implement vector- vector multiplications [5], [6]. The whole process can be divided into two parts: one is look-up and another is computation. In DA, the computation is possible with one assumption. Either the multiplier or multiplicand has to be a fixed constant. The sum of the product representation of DA can be considered as in the equation.

$$y = \sum_{k=1}^{K} T_k X_k \tag{5}$$

The basic DA implementation shown in Figure 3, the input is X one-bit-at-a-time into ROM LUT with address values are pre-computed and stored in ROM [7]. The selector is a control signal for the adder/subtractor circuit, after that, any of two values are added via accumulator and shifter when output is ready at Y.

The pre-computed values are stored in LUT using the DA assumptions and the basic implementation of the distributed arithmetic of ROM address and contents are shown in Table I. The implementation of a finite impulse response filter on field-programmable gate arrays (FPGAs) becomes complex when multipliers are used in the design. FIR low-pass filters using modified DA based on FPGAs are implemented in which the MAC unit is replaced with DA [8].

$$y = [\sum_{K=1}^{4} T_1.b_{Kn}] = T_1 b_{1n} + T_2 b_{2n} + T_3 b_{3n} + T_4 b_{4n} \tag{6}$$

It increases the speed of architecture and reduces the computational complexity by reducing the look-up table size. Venkatachalam et al. [9] implemented an approximate sum of products design using DA in which all the look-up table contents are pre-computed. These values are retrieved from LUT with a control signal, thereby reducing the complexity. This paper aims to improve energy efficiency using the DA algorithm for Radix-4 DIF-FFT, which can perform without a multiplier. Using LUTs, the design can perform both signed and unsigned number operations depending upon the control signal.

III. DIF-FFT architecture implementation

In this work, DIF-FFT Radix-4 architecture is implemented using Distributed arithmetic algorithm and adders, which reduces the computational complexity. In this design, computational complex multipliers are replaced by LUTs, due to which the delay is reduced. DA, along with different adders, are used to implement multiplier-less DIF-FFT [10]. Therefore,

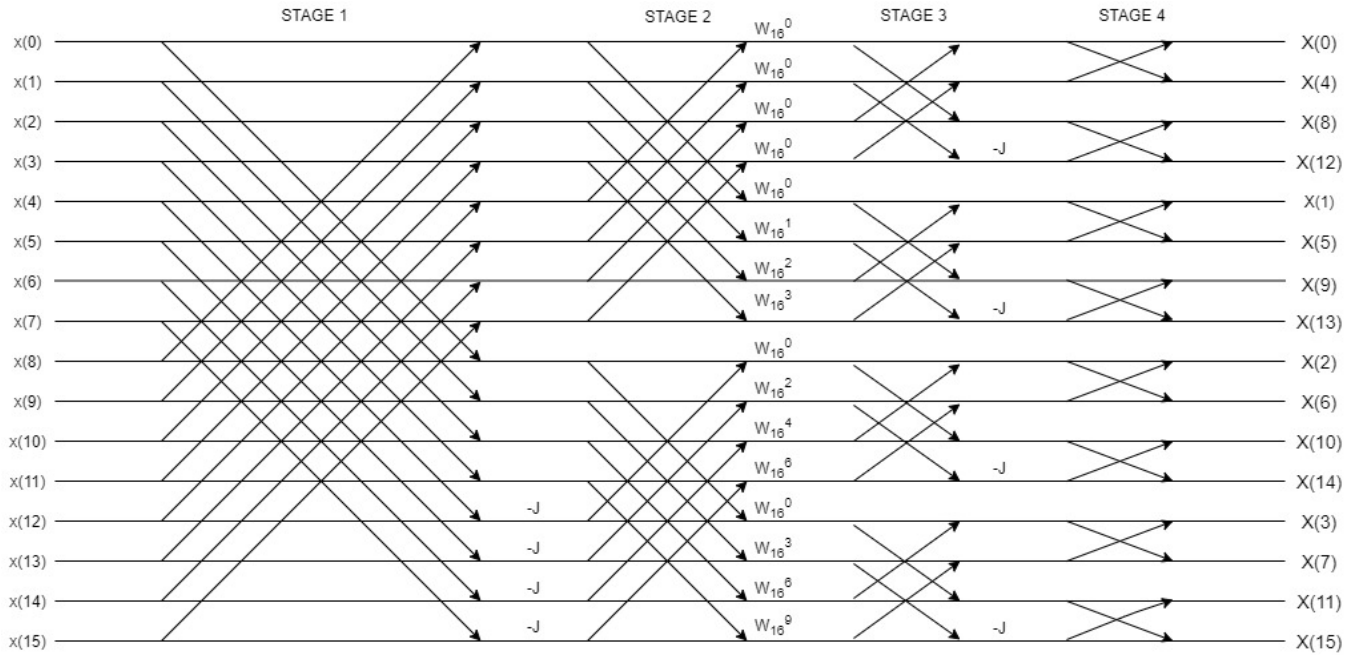

Fig. 2. Radix-4 16-point DIF-FFT structure

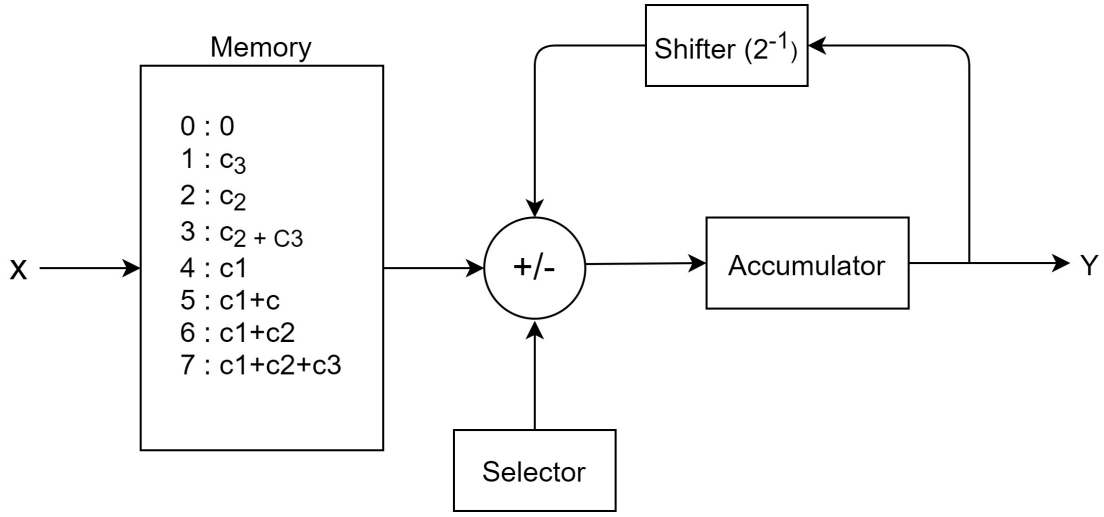

Fig. 3. DA based implementation

three models are proposed using different adders such as RCA, CLA, and Sklansky prefix graph adder. For in-depth analysis, a comparison of traditional radix-4 DIF-FFT butterfly structure using array multiplier and DIF-FFT using the DA technique is made. The implemented design is also compared with the different adders and summarized into three proposed models.

Proposed model 1 represents the implementation of Radix-4 16-point DIF-FFT DA using an RCA. RCA is a digital circuit that performs the arithmetic sum of two binary numbers, and it can be designed with full adders connected in cascade form. In RCA, the output is generated after the carry is produced by the previous stage. The final sum and carry bits will be valid after some considerable delay. Figure 4 shows the Radix-4 16-point DIF- FFT implementation using RCA (8, 9, 16, and 32-bit) and DA technique. In brief, 16-point DIF-FFT having sixteen frequency domain inputs, namely, x0, x1, x2, x3, and x15, are transformed to their time components in bit reversal order X0, X8, X4, X12, X2, and X15 via four stages.

The butterfly gets divided into halves by using the Cooley-Tukey algorithm. LUT comes into the picture at this stage, where each LUT stores the result obtained from the multiplication of twiddle factors with all possible combinations of input. For each twiddle factor, two LUTs are required to store the real and imaginary parts whose output can be treated as the LUT address, which stores the pre-computed actual production. The values stored in LUTs are extracted, which represents the

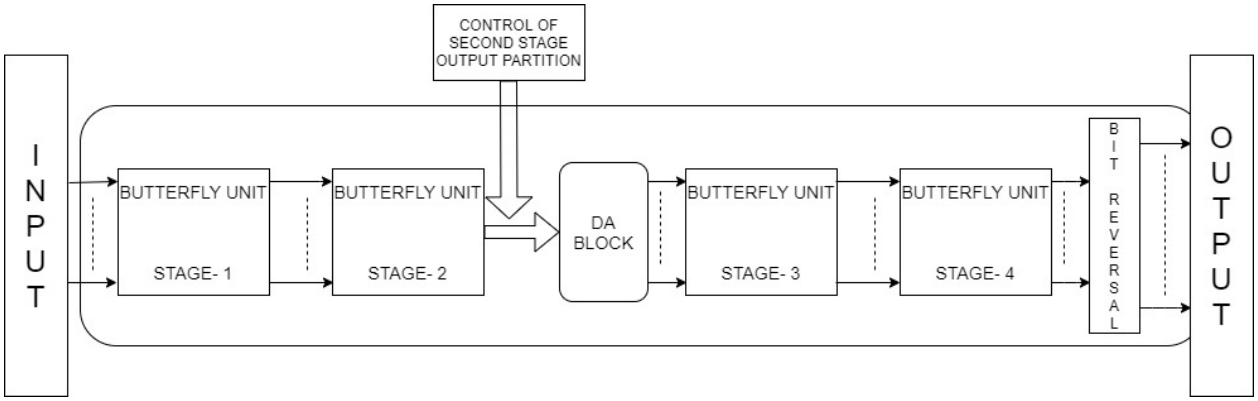

Fig. 4. Radix-4 16-point DIF-FFT using Distributed Arithmetic

TABLE I
ROM-ADDRESSES AND IT'S CONTENT.

b_{1n}	b_{2n}	b_{3n}	b_{4n}	Content
0	0	0	0	0
0	0	0	1	T_4 = 0.11
0	0	1	0	T_3 = 0.95
0	0	1	1	$T_3 + T_4$ = 1.06
0	1	0	0	T_2 = -0.30
0	1	0	1	$T_2 + T_4$ = -0.190
0	1	1	0	$T_2 + T_4$ = 0.65
0	1	1	1	$T_2 + T_3 + T_4$ = 0.75
1	0	0	0	T_1 = 0.72
1	0	0	1	$T_1 + T_4$ = 0.83
1	0	1	0	$T_1 + T_3$ = 1.67
1	0	1	1	$T_1 + T_3 + T_4$ = 1.78
1	1	0	0	$T_1 + T_2$ = 0.42
1	1	0	1	$T_1 + T_2 + T_4$ = 0.53
1	1	1	0	$T_1 + T_2 + T_3$ = 1.37
1	1	1	1	$T_1 + T_2 + T_3 + T_4$ = 1.48

partial output of the second stage. Since a single look-up table requires more memory, each twiddle factor is replaced with two LUTs. Values extracted from these two LUTs are added, representing the actual output, using 16 and 17-bit adders.

An increase in the input size of the second stage of DIF-FFT requires the large size of memory to build a LUT, which enforces the optimization. This dissimilar partitioned LUT based DA technique is capable of reducing the required memory size at the cost of computation. Since the size of LUT is 2^8, it can be partitioned into two halves: one having size 2^8 to 2^5 (32 locations) and the other with size 2^4 to 2^0 (16 locations) shown in Figure 5 and Figure 6.

In this stage, the twiddle factor values are simply -j, so LUT is not required. Finally, the output is converted using a bit reversal technique. Using the entire computations, a signal is converted from the frequency domain to the time domain. The example of c5 value 80 is the partial output of the second stage. Four LSB bits of c5 are multiplied with both real and imaginary values of a twiddle factor representing an address of LUT-1 of both the parts. Five MSB bits of c5 are multiplied with both the parts, resulting in the address bit generation of LUT-2 of each part, which is done by the control signal as shown in Figure 4. The values at these addresses from LUT-1 and LUT-2 are extracted and given as the input to the adder, which results in the final output of the second stage. For example, the binary value of c5 is 001010000. Four LSB bits 0000 are given as the input address to the LUT-1, and the five MSB bits 00101 are given as the input address to the LUT-2. The values at these addresses are extracted as explained above and given as the input to the adder which performs the addition on these values and produces the real value output shown in Figure 5. Before the addition operation, the extracted values are adjusted to ensure both the values follow the same notation. The same procedure is followed for the imaginary part as well as shown in Figure 6. The second stage output becomes the input to the third stage, and further operations will be carried out as explained in the above.

Proposed model 2 represents the implementation of Radix-4 16-point DIF-FFT DA using CLA. In this model, a CLA is used for implementation, which solves the carry delay problem by calculating the carry signals in advance, based on the input signals. No need to wait for the carry to ripple from the previous stages, thereby reducing the delay.

TABLE II
COMPARISON OF CONVENTIONAL RADIX-4 16-POINT DIF-FFT STRUCTURE USING ARRAY MULTIPLIER AND DIFFERENT ADDERS

Radix-4 16-point DIF-FFT Traditional structure using Array multiplier	Area(nm^2)	Power (μW)	Delay(nS)
Radix-4 DIF-FFT using RCA	15363.50	4015.92	1.848
Radix-4 DIF-FFT using CLA	15100.72	3907.89	1.853
Radix-4 DIF-FFT using Sklansky prefix graph adder	17574.73	4566.03	1.883

Proposed model 3 represents the implementation of Radix-4 16-point DIF-FFT DA using Sklansky prefix graph adder. Proposed implementation using Sklansky prefix graph adder achieves the minimum logic depth at the cost of large fan-out.

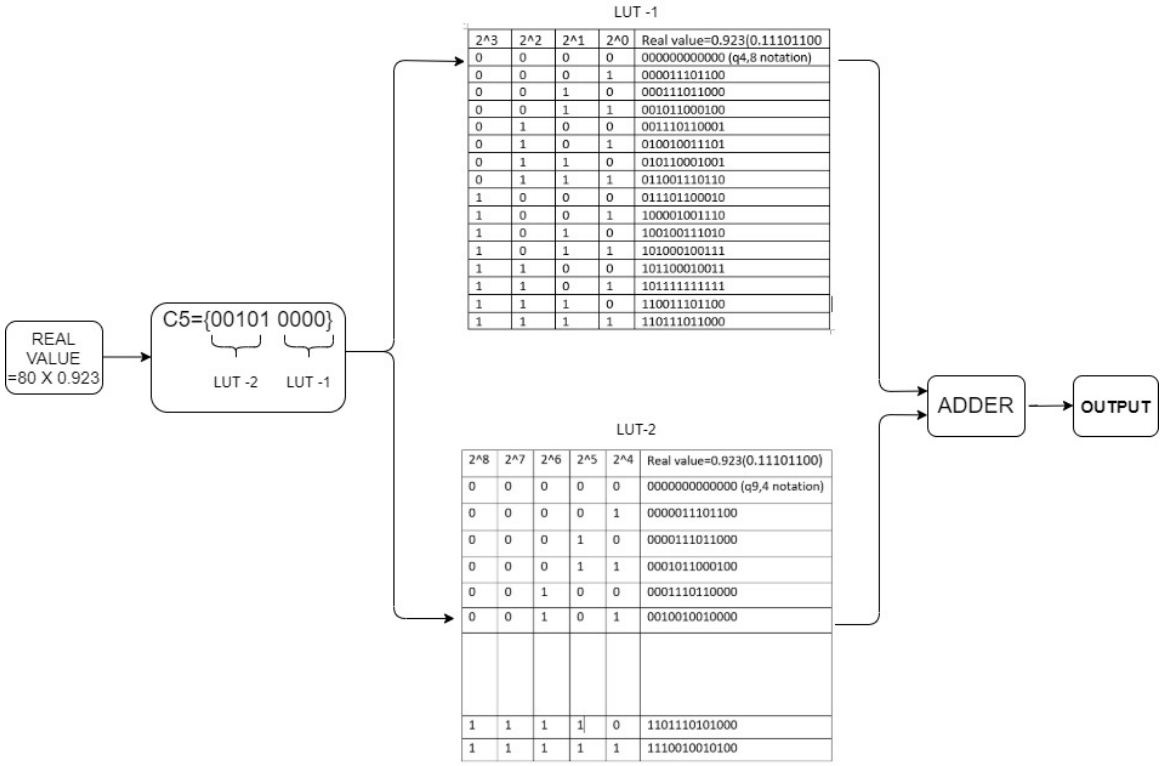

Fig. 5. Stage-2 partial output (Real)

TABLE III
COMPARISON OF PROPOSED RADIX-4 16-POINT DIF-FFT WITH DA AND ADDERS

Proposed Radix-4 16-point DIF-FFT using DA	Area(nm^2)	Power(μW)	Delay(nS)	PDP(μJ)	%improve PDP
Radix-4 DIF-FFT DA using RCA	7203.47	2317.03	1.012	2344.83	68.30
Radix-4 DIF-FFT DA using CLA	7043.30	2245.87	1.011	2270.58	68.44
Radix-4 DIF-FFT DA using Sklansky prefix graph adder	8759.49	2721.55	1.003	2982.82	65.30

This adder reduces the critical path with maximum fan-out [11], [12].

IV. RESULTS AND DISCUSSION

The Radix-4 DIF-FFT traditional structure and the proposed radix-4 DIF-FFT DA architecture design are modeled in Verilog HDL and synthesized using Cadence 6.1 EDA tools with 45nm CMOS technology, where the technology library gsclib045-translated.lef is used. The multiplier unit is used for a traditional structure for different formats such as signed, unsigned, and combined signed/unsigned partial product array has been implemented. These partial product arrays are reduced by using a CSA tree structure or other structures. In the DA algorithm, adders play a crucial role. The LUT optimization also comes with the usage of an adder. Table II lists the conventional structure comparison using array multiplier and different adders such as RCA, CLA, and Sklansky prefix graph adder. The proposed results having very much reduction comparing to the existing implementations [13]–[15].

Table III shows the results of the proposed radix-4 16-point DIF-FFT DA using different adders. While replacing multipliers, the computation is only adders. Existing adders are using to improve the PDP of the proposed architecture. The percentage of improvement in PDP compared to a conventional structure is also listed.

Figure 7 and Figure III shows the graphical representation of delay and power of three proposed models DIF-FFT with DA using different adders. The implemented layout diagram of the proposed radix-4 16point DIF-FFT using DA is presented in Figure 9.

Proposed models are very efficient are in terms of area DIF-FFT DA with CLA required less area, in terms of power DIF-FFTDA with RCA, and in terms of delay DIF-FFT DA with Sklansky prefix graph adder.

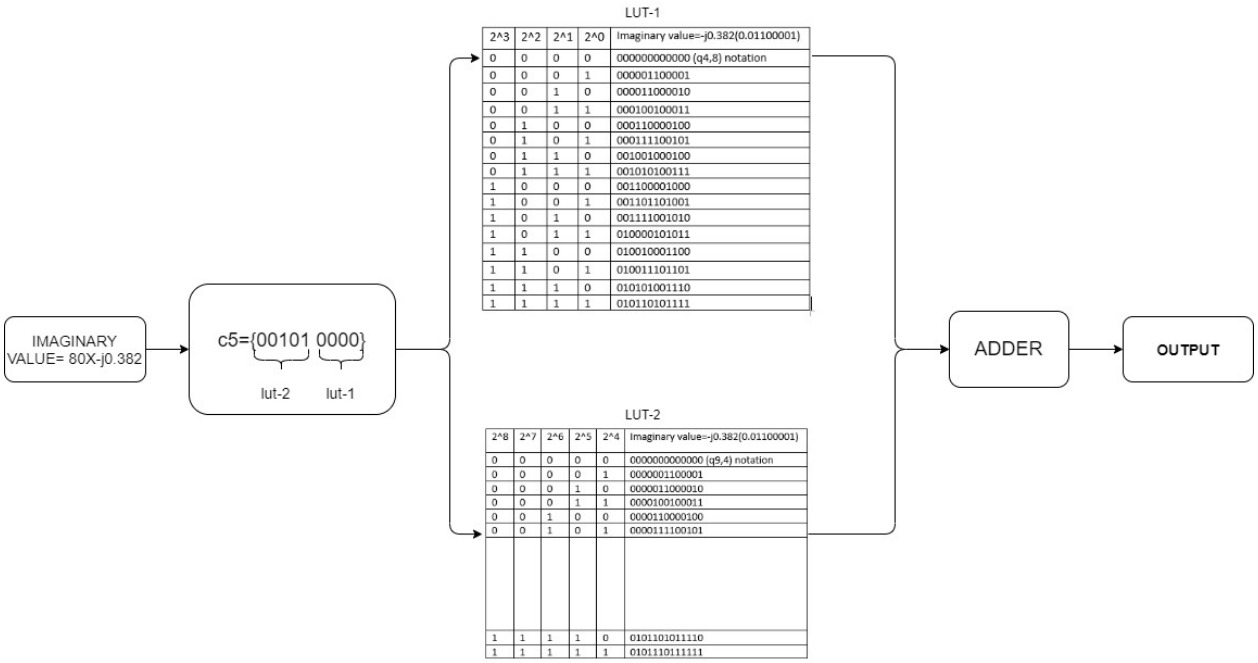

Fig. 6. Stage-2 partial output (Imaginary)

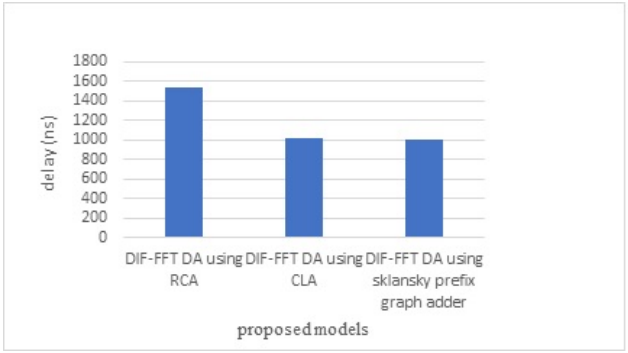

Fig. 7. Delay of the proposed DIF-FFT DA using different adders

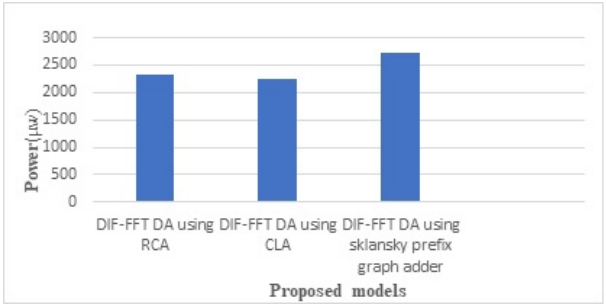

Fig. 8. Power consumption of the proposed DIF-FFT DA using different adders

V. Conclusion

Fast Fourier Transform (FFT) is the de facto standard to implement a Discrete Fourier Transform of a sequence. In general, FFT requires a more significant number of multipliers

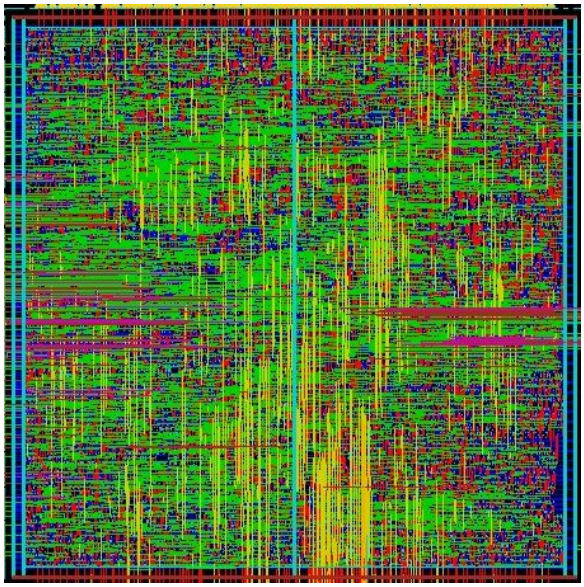

Fig. 9. Layout chip diagram for proposed radix-4 16-point DIF-FFT DA

to implement the design, which increases the delay and power consumption. This work proposes a high-performance DIF-FFT Architecture Design using Distributed Arithmetic with different adders. The computationally complex multiplier unit is replaced with a Distributed arithmetic approach in which all the pre-computed values (twiddle factors) are stored in LUTs. The obtained results of Conventional radix-4 16-point DIF-FFT with array multiplier using adders and proposed radix-4 16-point DIF-FFT with DA using adders are compared. It can be observed that the proposed method requires less hardware

to implement DIF-FFT with DA, which results in increased availability and energy savings.

VI. Future work

The FFT structure and will be further analyzed to see which technique will give the best overall performance in terms of cost, power, area size, speed, and time. This proposed DIF-FFT DA method will decrease the cost for implementing this structure compared to the traditional computing approach. Given that there is a great demand for new and improved technology for DIP and DSP applications, in calculating the SOP, DA is the most efficient and speed for SOP the method discussed in this paper will be used in implementing the new radix-4 FFT structures employing DA at different N point.

References

[1] H. Kim and S. Lekcharoen, "A cooley-tukey modified algorithm in fast fourier transform," *The Korean Journal of Mathematics*, vol. 19, no. 3, 2011.

[2] J. Watson, "Digital signal processing: Principles, devices and applications." Institution of Electrical Engineers, 1990.

[3] B. Mohindroo, A. Paliwal, and K. Suneja, "Fpga based faster implementation of mac unit in residual number system," in *2020 International Conference for Emerging Technology (INCET)*. IEEE, 2020, pp. 1–4.

[4] R. Gonzalez-Toral, P. Reviriego, J. A. Maestro, and Z. Gao, "A scheme to design concurrent error detection techniques for the fast fourier transform implemented in sram-based fpgas," *IEEE Transactions on Computers*, vol. 67, no. 7, pp. 1039–1045, 2018.

[5] K. K. Parhi, *VLSI digital signal processing systems: design and implementation*. John Wiley & Sons, 2007.

[6] D. Deepak and R. D. Kiran, "Hardware implementation of discrete cosine transform," 2002.

[7] R. Guo and L. S. DeBrunner, "A novel adaptive filter implementation scheme using distributed arithmetic," in *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. IEEE, 2011, pp. 160–164.

[8] S. Patel, "Design and implementation of 31-order fir low-pass filter using modified distributed arithmetic based on fpga," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 2, no. 10, pp. 650–656, 2013.

[9] S. Venkatachalam and S.-B. Ko, "Approximate sum-of-products designs based on distributed arithmetic," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 26, no. 8, pp. 1604–1608, 2018.

[10] K. N. Bowlyn and N. M. Botros, "A novel distributed arithmetic multiplierless approach for computing complex inner products," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer , 2015, p. 606.

[11] E. E. Swartzlander and C. E. Lemonds, *Computer Arithmetic: Volume III*. World Scientific, 2015.

[12] K. Vitoroulis and A. J. Al-Khalili, "Performance of parallel prefix adders implemented with fpga technology," in *2007 IEEE Northeast Workshop on Circuits and Systems*. IEEE, 2007, pp. 498–501.

[13] A. K. Y. Reddy and S. P. Kumar, "Performance analysis of 8-point fft using approximate radix-8 booth multiplier," in *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2018, pp. 42–45.

[14] A. Ajay and R. M. Lourde, "Vlsi implementation of an improved multiplier for fft computation in biomedical applications," in *2015 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2015, pp. 68–73.

[15] N. M. Sk *et al.*, "Multi-mode parallel and folded vlsi architectures for 1d-fast fourier transform," *Integration*, vol. 55, pp. 43–56, 2016.