Design and Simulation of a Low Power and High Speed Fast Fourier Transform for Medical Image Compression

Ernest Ravindran R. S* **Ngangbam Phalguni Singh and Sudhakiran Gunda**

[1]Department of Electronics and Communication Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, AP, India.

Abstract

For front-end wireless application in small battery-powered devices, the discrete Fourier (DFT) transform is a critical processing method for discrete time signals. Advanced radix structures are created to reduce the impact of transistor malfunction. To develop DFT, with radix sizes 4, 8, etc. is complex and tricky issue for algorithm designers. The main reason for this is that the butterfly algorithm's lower radix level equations were manually estimated. This requires the selection of new design process. As a result of fewer calculations and smaller memory requirements for computationally intensive scientific applications, this research focuses on Radix-4 Fast Fourier Transform (FFT) technique. A new 64-point DFT method based on Radix-4 FFT and multi-stage strategy to solving DFT-related issues is presented in this paper. Based on the results of simulations with Xilinx ISE, it can be concluded that the algorithm developed is faster than conventional approaches, with18.963 ns delay and 12.68 mW of power consumption. It was found that the computed picture compression drops ratios of 0.10, 0.31, 0.61 and 0.83 had direct relationship to the varied tolerances tested 0.0007625, 0.003246, 0.013075 and 0.03924. Fast reconstruction techniques, wireless medical devices, and other applications benefit from this FFT's low power consumption, little storage requirements, and high processing speed.

Keywords: Discrete Fourier Transform, Inverse Discrete Fourier Transform, Fast Fourier Transform, Inverse Fast Fourier Transform, Radix-4, Image Compression, Drop Ratio.

1. Introduction

One of the most widely utilized mathematical operations is Fast Fourier Transform. Several medical applications use Fast Fourier Transform for image reconstruction and frequency domain analysis. Image processing applications such as filtering, compression, and de-noising all rely on FFT to certain extent. Figure 1 shows the usage of FFT, an improved version of the traditional discrete signal processing tool (Discrete Fourier Transform), for medical image compression with various drop ratios. FFT is widely used in medical imaging, engineering, communication, and other fields because it transitions quickly from the T-domain to the F-domain and vice versa [1-6].

The medical imaging method provides images of the human body and its components for clinical application. Computer tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound and Optical Imaging Technology are the most prevalent modes of medical imaging that produce a prohibitive amount of data. The images produced by these instruments are pixels representing the operations of human organs in terms of their visual depiction. They are also the patient's most vital information and demand high storage and transmission width [7, 8].

FFT-based compression is a compression algorithm which can process the image quickly coupled with the transformed domain compression. The modified domain includes coefficients of both low and high frequencies that are measured. Various quantified coefficient values of high frequency are unimportant and almost equivalent to zero and remove them from the modified image. This pre-processing step leads to the compression platform. By supplying different symbols, the FFT method accomplishes compression. The majority of appearing symbols is assigned to be shorter while the less are assigned longer size symbols. The variable-length compressed data subsequently is stored on transmission media.

As hospitals are progressing into digitization, filmless imaging and telemedicine, the medical imagery becomes significantly important in the health sector. This has led to the major difficulty of developing compression algorithms that prevent diagnostic errors and have a high compression ratio for lower bandwidth and storages. In the medical area particularly, quick diagnosis is only achievable when the required diagnostic information is maintained by the compression approach. These images help physicians to easily diagnose the inner parts of the body. It also helps to perform keyhole procedures without too many incisions to reach the inner sections of the body. They can be processed fast, analysed objectively, and made available in numerous places simultaneously by means of communication protocols and networks likes Digital Imaging and Communications in Medicine (DICOM) protocol and Picture Archiving and Communication Systems (PACS) networks respectively. The X-ray, CT, MRI or Ultrasound images contain huge amounts of data that demand vast channels or storage capacities. The implementation costs limit storage capacity even with the progress in storage capacity and connectivity [8-10]. There are certain approaches that create imperceptible variations and acceptable fidelity that can lead to medical picture low loss compression. In this article an FFT based compression is proposed.

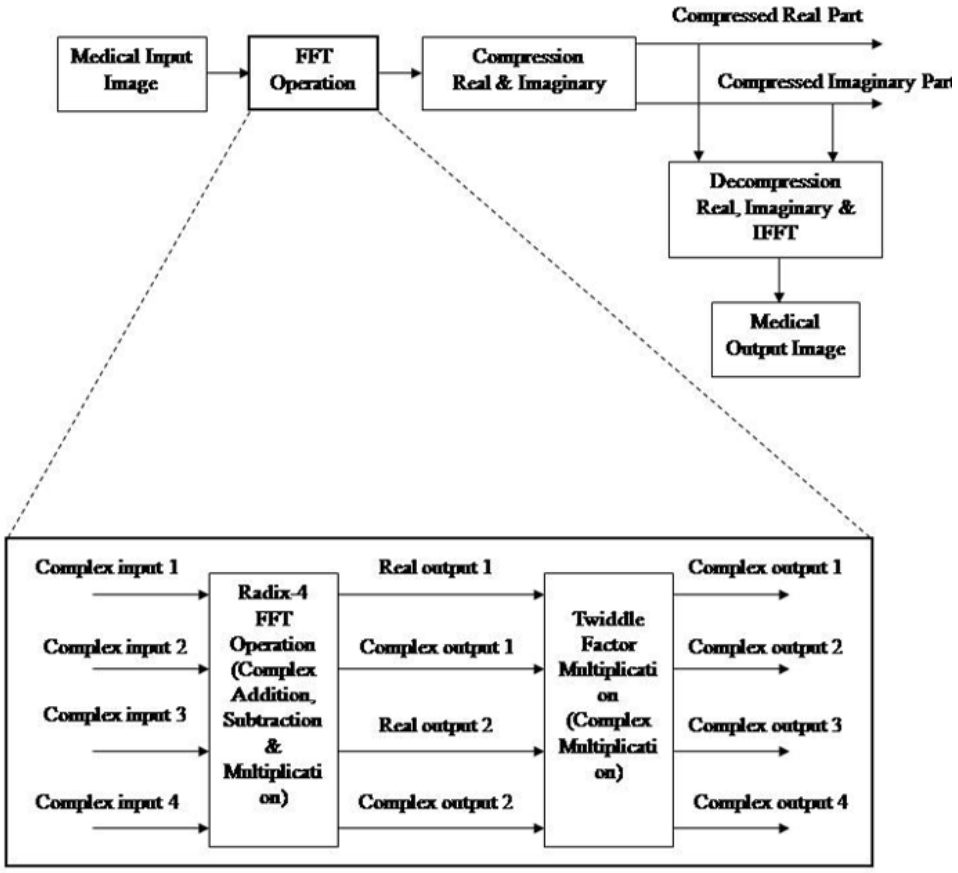


Figure 1: Block representation of medical image compression using proposed FFT.

Many different mathematical FFT algorithms vary from easy theory of complex numbers arithmetically to group and numerical theory; this paper provides an available technical outline and few characteristics while explaining the algorithms in the subsidiary sections. The DFT is obtained via the decomposition of a series of values into various frequency components as given in Eq. (1) & (2). This operation is useful in several fields, but it is always too slow to be practical for computing it directly from given description.

$$\boldsymbol{X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\left(\frac{2\pi}{N}\right)kn},\ 0 \le k \le N-1} \textbf{ (1)}$$

$$\boldsymbol{x(n) = 1/N \sum_{k=0}^{N-1} X(K) e^{j\left(\frac{2\pi}{N}\right)kn}; 0 \le n \le N-1} \textbf{(2)}$$

The FFT is one of the new ways to calculate the similar results faster: DFT takes N^2 arithmetical multiplications and N^2-N addition operations ($O(4N^2)$ real multiplications and O(N(4N-2) real additions) to naively compute the DFT of N-points the speed difference may be huge, especially in long data sets where N may be higher and higher. FFT can compute the DFT for $\frac{N}{2}\log_r N$

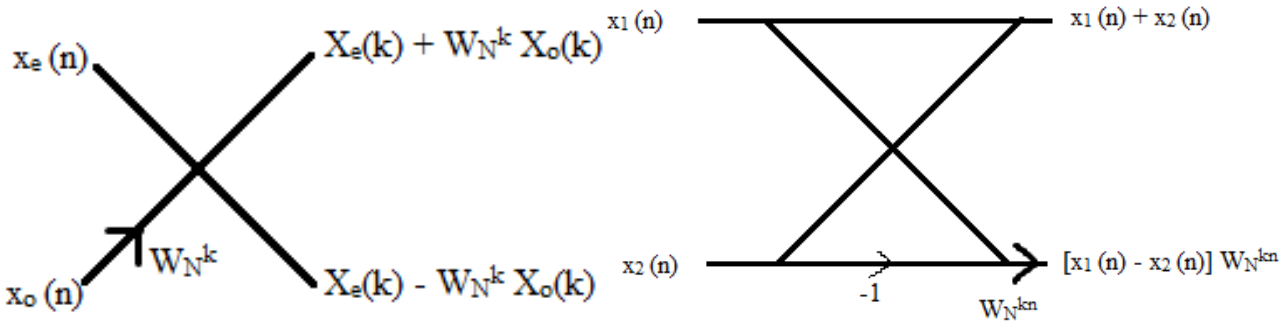
multiplications and $N \log_r N$additions corresponding operation alone using twiddle factor $W_N = e^{-j2\pi/N}$ [4, 6] since it is using butterfly operation and computes p ± αq (results 6 real adds and 4 real multiplications). As FFTs are staged algorithms, there are log_2^N stages and each stage have N/2 butterflies, so there should be 4.

N/2. log_2^N = 2N log_2^N real multiplications & 3N log_2^N real additions for N-point DFT through FFT (3a) (an optimization is still possible, but these are basic equations) And N^2 & N(N-1) multiplications and additions in a normal DFT for N-point (3b). DFT estimation was practical due to these huge changes. For a broad range of applications, FFTs are of great importance – from DSP to algorithms for quick multiplication of high integer range. [10, 11] From Eq. 3(a) & 3(b), the cost estimation is given in the following table 1.

Table 1: Cost estimation of DFT and FFT

N	DFT		FFT (radix 2)(optimized)		Speed factor improvement	
	$4N^2$	N(4N-2)	$2N log_2^N$	$3N log_2^N$	Multiplications	Additions
2	16	12	4	6	4	2
32	4,096	4,032	320	480	12.8	8.3
64	16,384	16,256	768	1,152	21.333	14.11
.	.	.	.	.	.	.
1024	41,94,304	41,92,256	20,480	30,720	204.8	136.466

From Table 1, it's evident that FFT is less computationally intensive than DFT. When comparing the two methods, FFT is faster. It is important to note that, because FFTs are radix algorithms, this work shows that making little changes to the algorithm in the order 2 results in faster FFT times. A DIT-FFT algorithm decomposes a signal based on the time sequence 'x(n)'. Another categorization is the Decimation-in-Frequency FFT (DIF-FFT) algorithm, which decomposes using the frequency sequence 'X(k)'. Radices are the foundation of these algorithms. Many intermediate results and memory locations are re-used in these algorithms, which makes them more efficient in the long run. These computational approaches is happen with the help of butterflies called Radix-2 butterflies as shown in Fig.2 (a) & (b).



(a) Butterfly computation in DIT-FFT (b) Butterfly computation in DIF-FFT

Figure.2 DIT & DIF FFT Radix-2 butterfly computation diagrams (radix 2 nodes – butterfly nodes)

From fig. 2 the radix-2 butterflies have equal number of wings in input and output section. In DIT-FFT the inputs are arranged in bit reversal/normal order and outputs are obtained in a normal order/bit reversal. Radix-2 DIT-FFT algorithm is a staged algorithm. The effective functioning of radix-2 depends on stages, butterflies etc. Each stage has the block(s) and each block has butterflies. These can be defined as follows: Radix-2 algorithm consists of $\log_2 N$ stages, and each stage consists of $N/2^{stage}$ blocks, and each block consists of $2^{stage-1}$ butterflies. As in signal processing (digital) most required arithmetic computations are additions and multiplications, radix-2 offers $N/2\log_2 N$ complex multiplications and $N\log_2 N$ complex additions [9-14].

The Radix-4 is an additional fast Fourier Transform Algorithm (FFT) that can be obtained by moving the base from 2 to 4. The power/index diminishes in direct proportion to the size of the base. There are 50% fewer stages in radix-4 than in radix-2 since N=4M, indicating that stages have decreased by 50%. It is explained in more detail in the later sections on how radix-4 simplifies difficult calculations [15].

For computing sequences, the radix-4 algorithm is comparable to the radix-2 technique in terms of type and speed management. It's taken place as follows; the given sequence divides into four parts based on 'n': The given sequence layout in radix-4 is as follows:

n = [0, 4, 8, ... N - 4] results x(4n),

n = [1, 5, 9, ... N - 3] results x(4n+1),

n = [2, 6, 10, ... N - 2] results x(4n+2), and

n = [3, 7, 11, ... N - 1] results x(4n+3). [16-18]

After the division of N-point DFT it can be computed as the sum of the outputs of 4 N/4-point DFTs, and these sub-sequences are interconnected with so-called twiddle factors $W^{lk}{}_N = e^{-(j2l\pi k/N)}$, l=0, 1, 2, 3, as shown in (4).

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

$$= \sum_{n=0}^{\frac{N}{4}-1} \boldsymbol{x(n)W_N^{nk}} + \sum_{n=\frac{N}{4}}^{\frac{N}{2}-1} \boldsymbol{x(n)\,W_N^{nk}} + \sum_{n=\frac{N}{2}}^{\frac{3N}{4}-1} \boldsymbol{x(n)W_N^{nk}} + \sum_{n=\frac{3N}{4}-1}^{N-1} \boldsymbol{x(n)W_N^{nk}} \qquad (4)$$

Thus,

$$\boldsymbol{X(k)} = \sum_{n=0}^{\frac{N}{4}-1} \boldsymbol{x(n)W_N^{nk}} + \boldsymbol{W_N^{\frac{Nk}{4}}} \sum_{n=\frac{N}{4}}^{\frac{N}{2}-1} \boldsymbol{x(n+\frac{N}{4})\,W_N^{nk}} + \boldsymbol{W_N^{\frac{Nk}{2}}} \sum_{n=\frac{N}{2}}^{\frac{3N}{4}-1} \boldsymbol{x\left(n+\frac{N}{2}\right)W_N^{nk}} +$$

$$\boldsymbol{W_N^{\frac{3Nk}{4}}} \sum_{n=\frac{3N}{4}-1}^{N-1} \boldsymbol{x(n)W_N^{nk}} \qquad (5)$$

Final representation of X(k) is,

$$\boldsymbol{X(k)} = \sum_{n=0}^{\frac{N}{4}-1} \left[\boldsymbol{x(n) + (-j)^k x\left(n+\frac{N}{4}\right) + (-1)^k x\left(n+\frac{N}{2}\right) + (j)^k x(n+\frac{3N}{4})}\right] \boldsymbol{W_N^{nk}} \qquad (6)$$

According to (4)-(6) the process is called decimation in time because of the samples of time are arranged into groups. The basic operation of R4 butterfly is shown in fig. 3 [17]. The decimation-in-time process consolidates the inputs at each stage of decomposition, resulting in "input order that is bit-reversed" at the end. This set-up allows for the intermediate outputs to be stored in the same memory regions as the inputs (in-place algorithm). Radix-4 FFT's slight reorganization allows the inputs to be redirected from digit to bit [18] as shown in table 2.

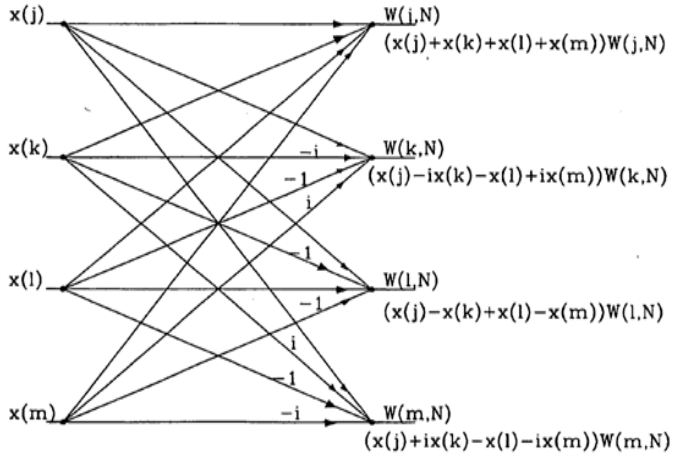


Figure.3: Operation of Radix-4 butterfly. (Radix 4 nodes – Dragon nodes)

Table 2: Numbering according to base-4 for bit reverse

Normal sequence order	Normal sequence Bit Order	Reversed Bit Order	Reversed sequence Number
0	000	000	0
1	001	100	16
2	002	200	32
3	003	300	48
4	010	010	4
5	011	110	20
14	032	230	44

Figure 3 depicts the computation of the Radix-4 project's flow chart. The input sequence might be in bit reversal order or normal order. The updated sequence can be operated on in the stage after being arranged. Each stage has a group of butterflies, and each butterfly group is made up of other butterflies. After that, the butterfly (radix-4) algorithm is used. For each further stage or group, the operation repeats until all butterflies in a group and stages have been completed by scaling with the required twiddle factor. Here, the radix-4 operation is completed with the output in either a normal or bit-reversed order depending on the input sequence [15-18]. Radix-4 operations are completed.

Radix-2 adds twiddle integer factors in 0° and 180° angles, whereas radi-x4 adds twiddle integer factors in 0°, 90°, 180°, and 270° angles, all while accounting for the computational cost of multiplication. There is no need to multiply the sine and cosine counterparts of the above-mentioned angles within a unit circle. The Radix-8 is not preferred because of its factors of fractional twiddle (2) at 45°, 135°, 225°, and 315° in a unit circle, despite the fact that the number of radix minimizes the number of computation steps [19, 20].

2.Development of a lossless medical image compression using FFT algorithm

Medical image compression using FFT is developed as shown in the flow chart given in the figure 4. Load/Read any medical input image and convert it into 2D array of doubles image. Compress the loaded image with different tolerance values. While compressing the image it takes as inputs the original image X and the drop tolerance parameter and outputs a compressed image Y. It also returns the drop ratio given in (7) which is defined to be as the ratio of “Total number of nonzero Fourier coefficients dropped to the Total number of initially nonzero Fourier coefficients”.

For every drop count for a compressed image apply FFT to each sub block.

Drop ratio = (Total number of nonzero Fourier coefficients dropped/Total number of initially nonzero Fourier coefficients) (7)

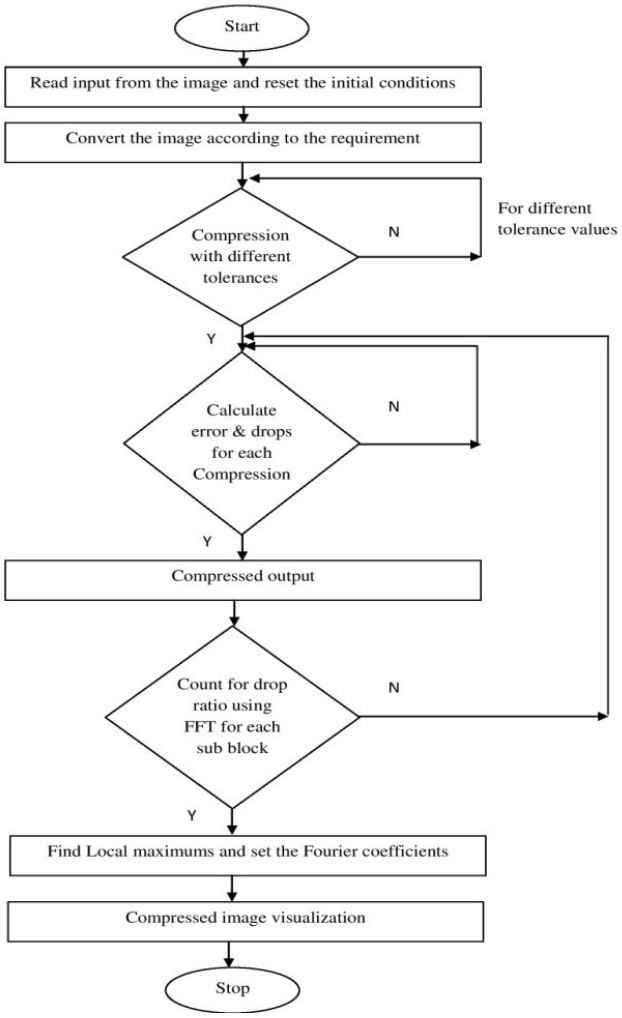
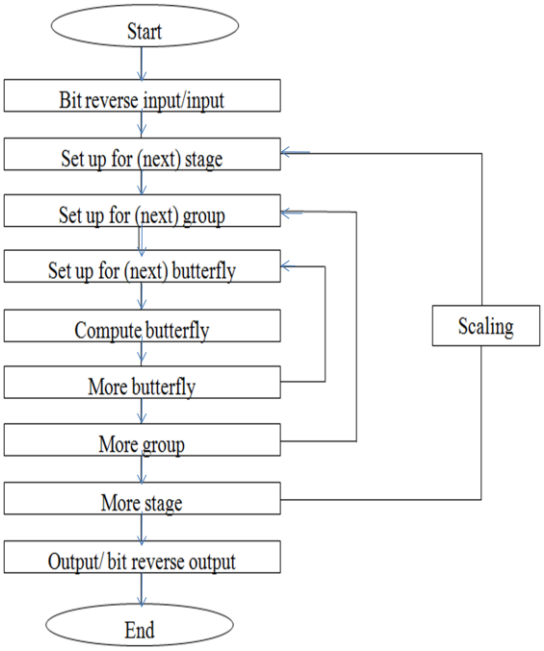


Figure.4: Medical image compression using FFT algorithm flowchart.

3. Results and Discussion

In 64-Point DFT using Radix-4 DIT-FFT algorithm has 3 stages, in first stage 16 blocks are present, and each block consists of only one butterfly. At first stage the inputs are applied in a bit reversal order to save the memory space. In the second stage, 4 blocks and each block consist of four butterflies as a set totally 4 sets are present, in the third stage and in the final stage only one block is present in that 16 butterflies are present as a one set and finally obtained output from the final stage which is in a normal order. The structural view of 64-point radix-4 DIT-FFT is as shown in Fig. 5. The RTL view of the proposed algorithm consists of data splitting of 256-bits into four 64-bits and

each 64-bits are further divide into 16 4-bit points and separate into even and odd sequences. All are communicated with a communicator called Comutator.



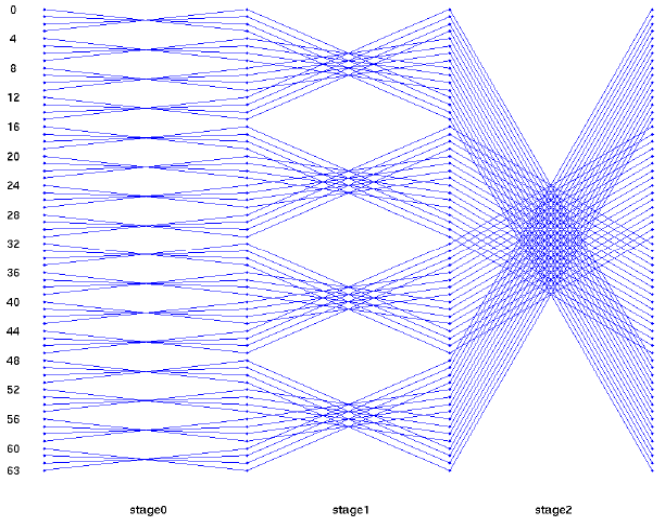


Figure.5: Flow chart of Radix-4 64-point DIT-FFT operation **Figure.6:** Butterfly diagram of 64-point DFT using radix-4 DITFFT.

The target device xc3s500e-5fg320 is used for the execution. The device contains the 9,312 LUTs and 4,656 slices for functionality of the input sequence. Slices used for the related logic are 3,286 and for unrelated logic are 3,286. Device contains 9,312 number of 4-input LUTs and works under the speed grade of '-5'. The chronological view of proposed work is shown in Fig. 6.

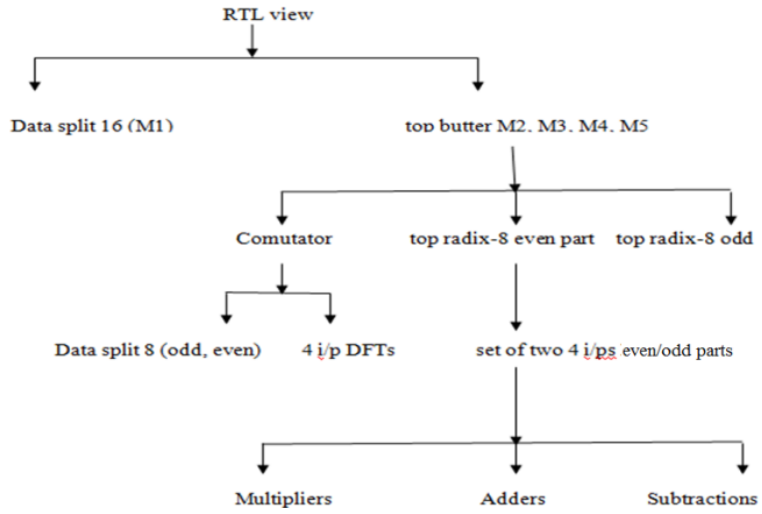


Figure.7 Chronological view of proposed work

Generally, the 64-point radix-4 DIT-FFT butterfly unit (butterfly 16) consists of butterflies, set of butterflies, stages. Entire code was developed in a structural made using HDL language. The internal modules designed based on the behavioral model or dataflow model. The different internal modules are, Splitting the entire sequence into equal parts to save the memory, 4-bit butterfly unit, Odd and even parts, Butter for 8-point & 4-point and Comutator for connecting all the stages and sub modules. All these modules called as sub modules in top butter. The RTL view and simulation result of 64-point radix-4 DIT-FFT are as shown in Fig. 8.

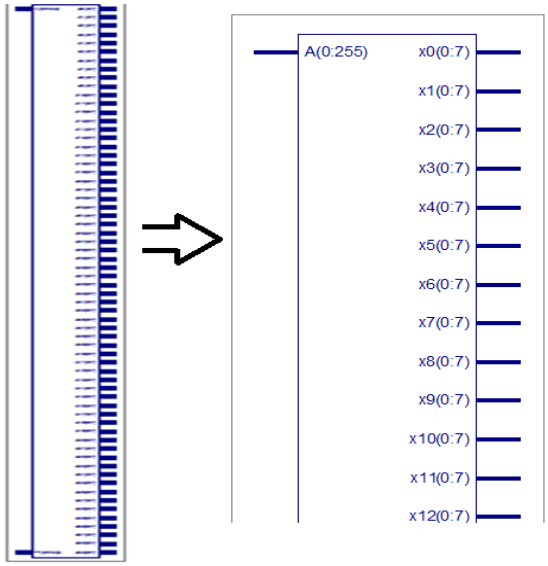


Figure.8 RTL view of 256-bits 64-point radix-4 DITFFT

3.1. Data split of 64-points

The entire 64-point radix-4 DIT-FFT can be split into 4 equal parts, each of 16-points and each point is the combination of 4-bits, totally 64-bits in each equal part. As A is an input sequence of 256-bits (0 to 255), divide into 4 equal parts of each 16-point, 64-bit (0 to 63, 64 to 127, 128 to 191, 192 t0 255). The RTL view of data split 64-points into 4 equal 16-points and its simulation results are shown in Fig. 9, 10 & 11 respectively.

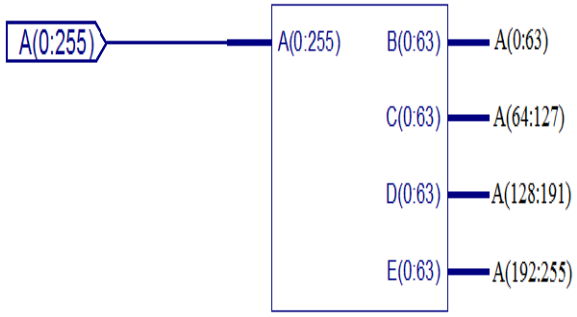


Figure.9 RTL view of data split of 64-points

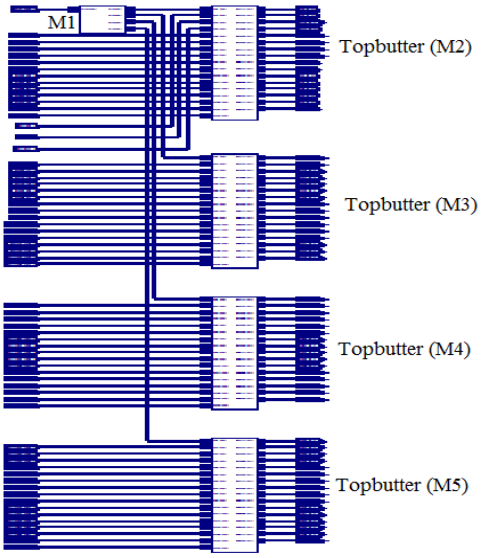


Figure. 10: Internal structure of 4.3(a)

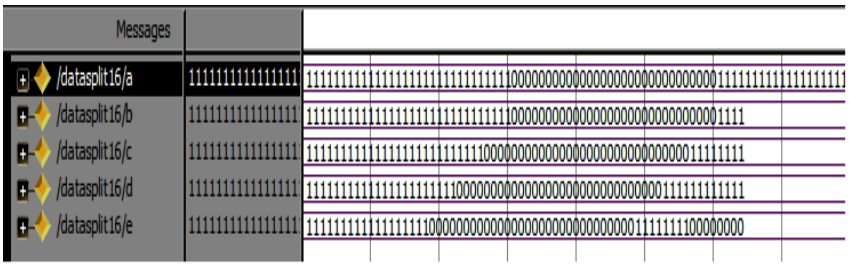


Figure.11: Simulation results of data split of 64-points

3.1.1. Top Butter

Top butter is the divided part of entire sequence. This has 16-point, 64-bits input and 64 twiddle totally 128-bits output in which 64-bits are even and remaining bits are odd. Top butter contains comutator, even and odd parts. For the entire sequence total 4-top butters (M2, M3, M4, and M5) are

present. The RTL view of top butter (M2) and simulation results are shown in the Fig. 12 & 13 respectively:

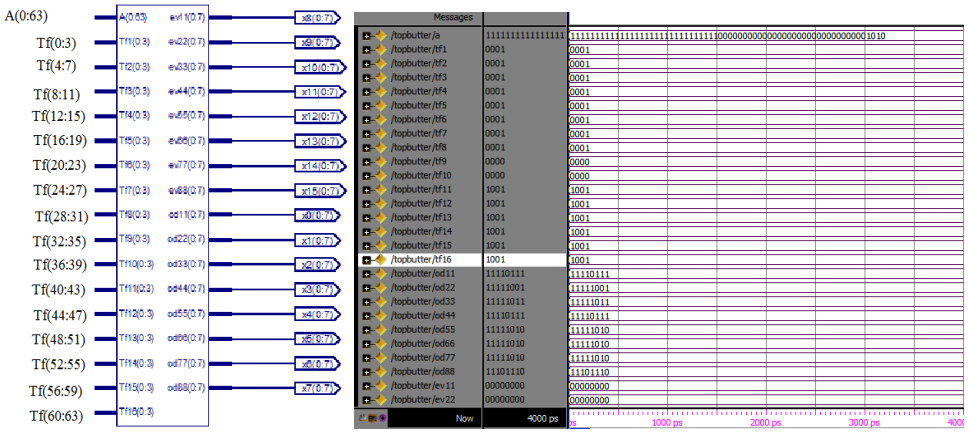


Figure.12: Overview of top butter (M2) **Figure.13:** Simulation results of top butter

3.1.2. Internal structure of Top butter

The equally divided 16-point (64-bit) sequence again undergo for further division to increase the speed of execution. The 64-bit (0 to 63) sequence divides as A (0 to 3), A (4 to 7) and so on to A (59 to 63) like 16-points are divided in the remaining 4 equal parts. The RTL view of this unit having comutator, even and odd parts and simulation results of data split of 16 point are shown in Fig. 14 & 15 respectively:

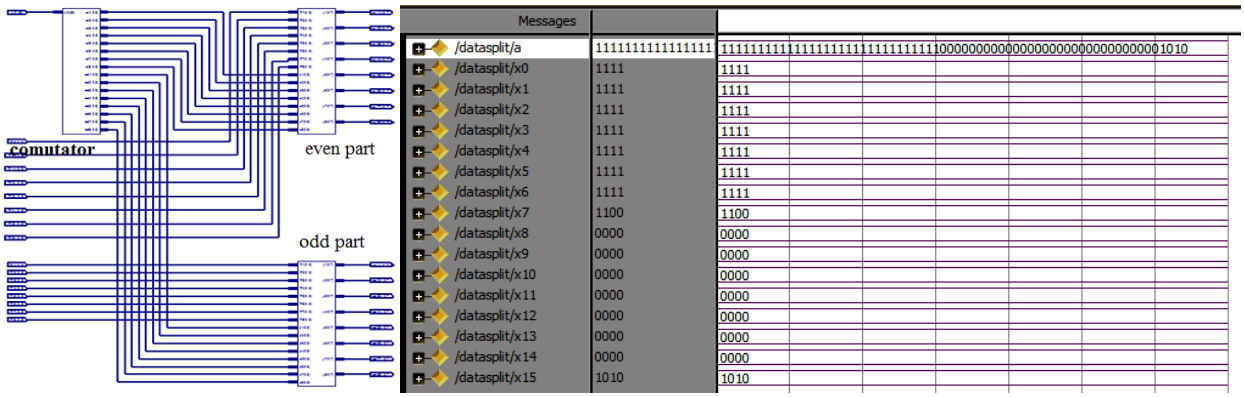


Figure.14. Internal view of top butter (M2) **Figure.15.** Simulation results of top butter

3.1.3. Comutator

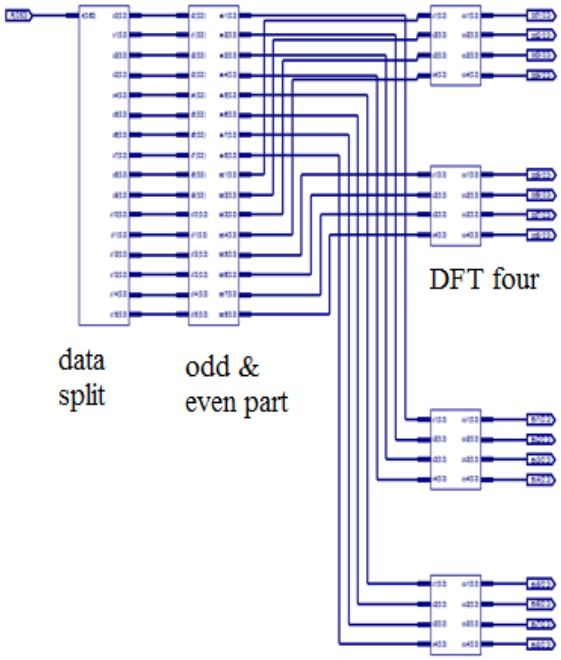


Figure.16. Internal structure of comutator

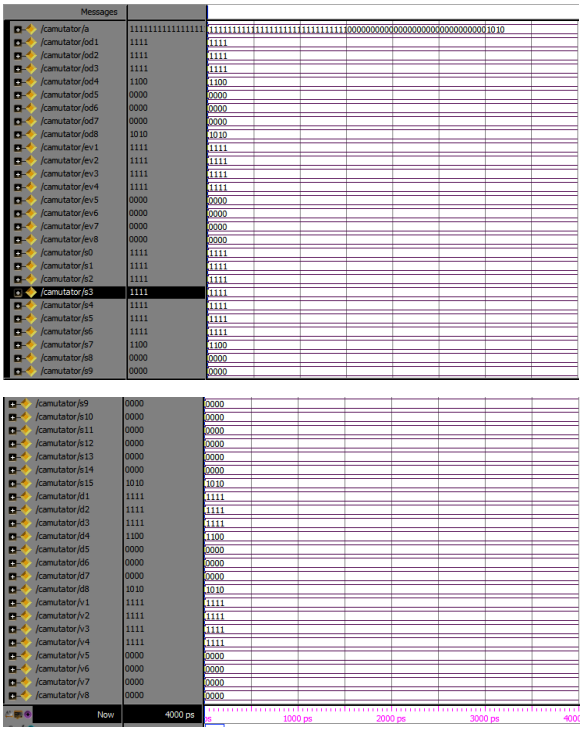


Figure.17. Simulation results of Comutator

3.1.4. DFT Four

DFT four is the basic unit in radix-4 structure, because it transfers the input value to output. Each stage having this unit, four inputs and four outputs are present in this unit. The RTL view and simulation results of DFT four are shown in Fig. 18 & 19 respectively.

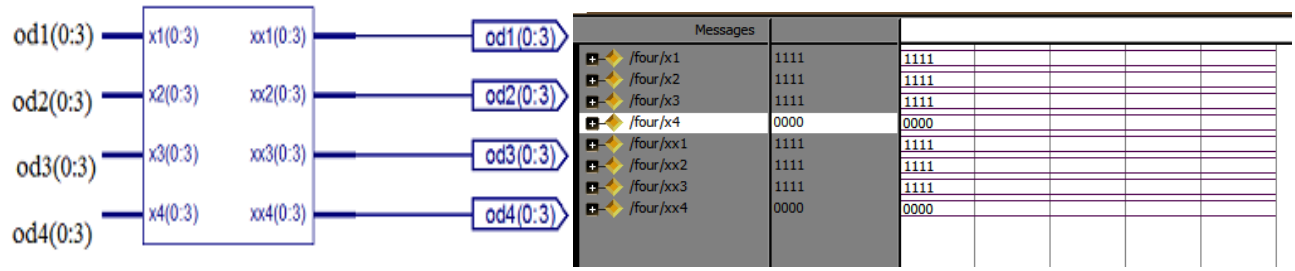


Figure.18. DFT four block **Figure.19.** Simulation results of DFT four

3.1.5. Butter R8

Butter R8 is the internal unit in the butterfly diagram. It is the combination of both even and odd parts. And each even and odd part is the combination of two butter R4 blocks so total 2 R4 blocks for each butter R8 block. This butter R8 block exists from 16- point block means division of 16-point into smaller parts for easy execution. The RTL view and simulation results of butter R8 are shown in Fig. 20 & 21 respectively.

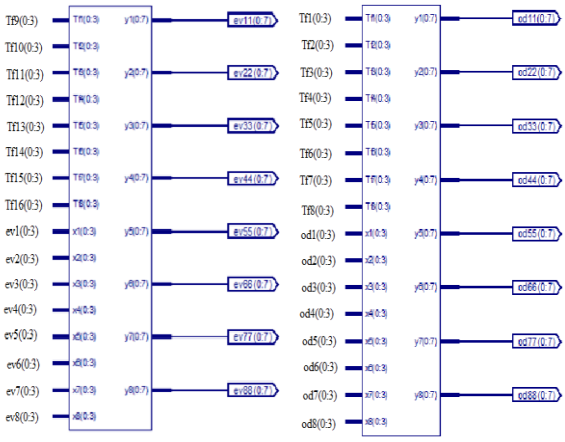


Figure.20. Even and odd part of butter R8

3.1.6. Even and Odd Parts

Even and odd parts are the two different functioning in the entire butterfly unit. The combination of even and odd part unit presents in all modules (M2, M3, M4 and M5). In four modules, four even and odd part combinations are present. In the different parts, the divided sequence can be ordered into even and odd parts/places to save the memory requirements. The even and odd part has two

instances internally to make easy execution. The RTL view and their simulation results are shown in Fig. 22 & 23 respectively.

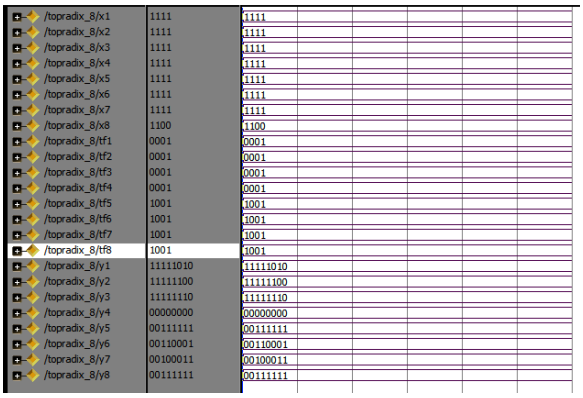


Figure.21. Simulation results of butter R8

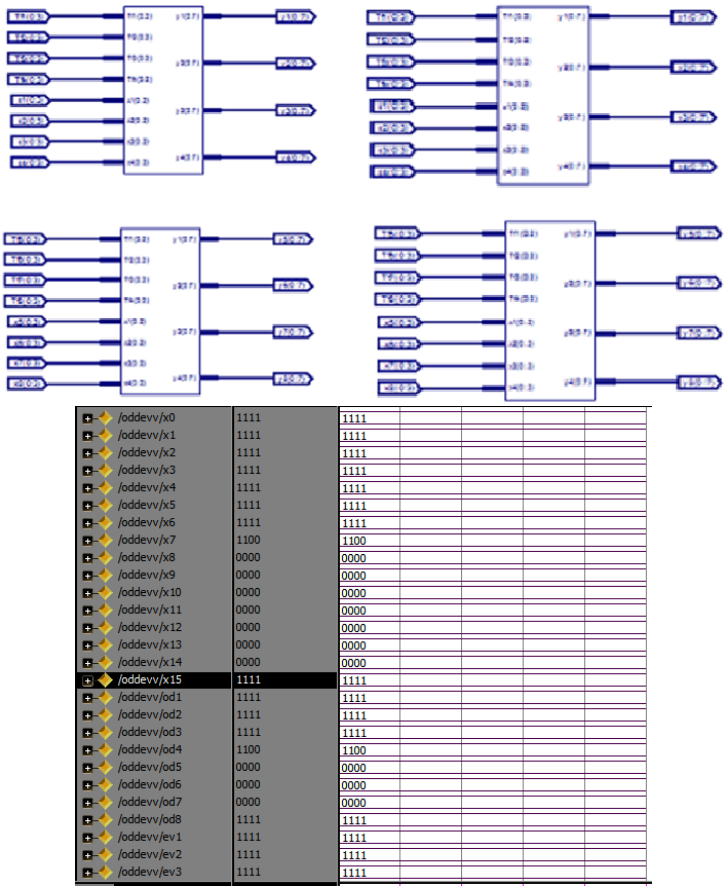


Figure.22. Internal view of even and odd part of butter R8 **Figure.23.** Simulation results of even and odd part of butter R8

3.1.7. Butter R4

Butter R4 is the basic unit in this structure because it represents the radix-4 function. Each stage is having this unit, either as a single unit or as a set/group. In this unit, four inputs and four outputs are present which is as shown in the Fig. 14. Each input is multiplied with twiddle factor and gives the output. Implies that for 256-bit input there are 256 twiddle factors are present. The RTL view and simulation results of butter R4 are shown in Fig. 24 & 25 respectively.

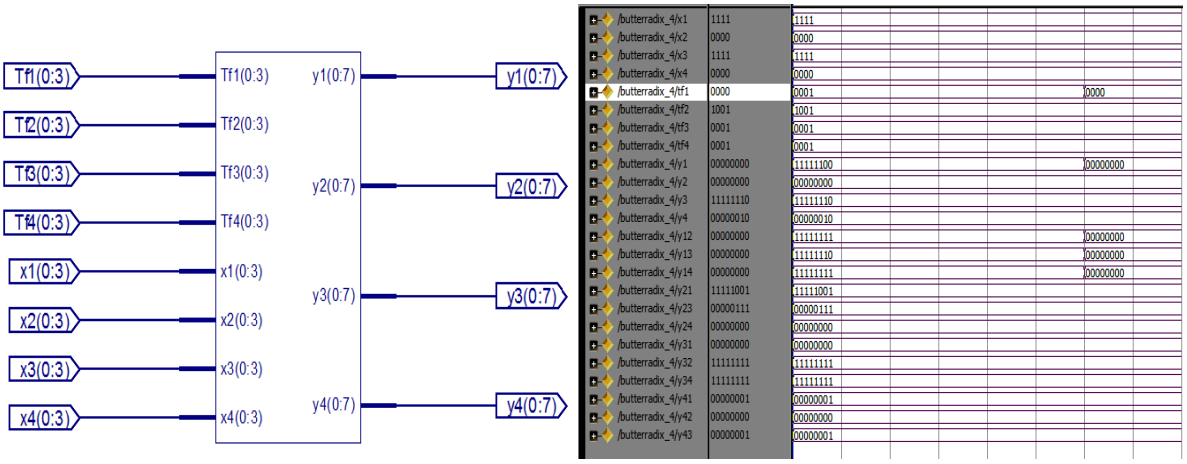


Figure.24. Butterfly R4 with four twiddle factors **Figure.25**. Simulation results of radix-4

3.1.8. Internal structure of butter R4

The internal structure of butter R4 consists of different units. These units are responsible to the entire functionality of the butterfly unit. The different units are adders, subtractors and multipliers and these units are called as basic building blocks for butter R4. The RTL view is as shown in Fig. 26.

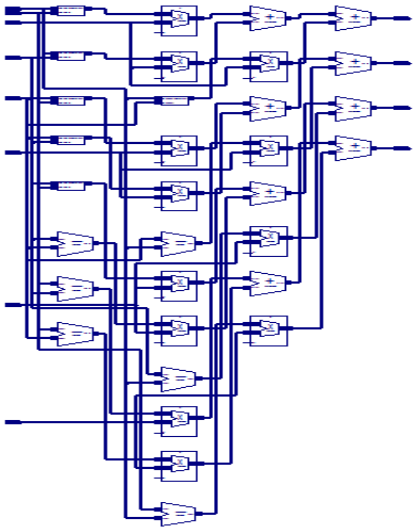


Figure.26. Gate level structure of R4

3.2. Simulation results of radix-4 algorithm:

Fig. 27 has shown the simulation results of the 64-points radix-4 DITFFT butterfly block. The A of 64-points each point of 4-bits totally 256-bits, W of 256 points represents the inputs, twiddle factors respectively. The X is a output of 64-points each point of 8-bits since even and odd part results totally 512 points all are in binary formats.

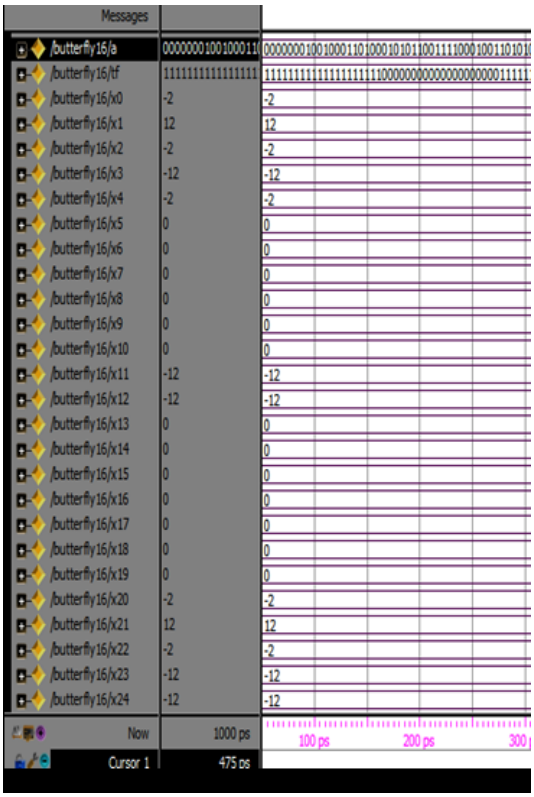
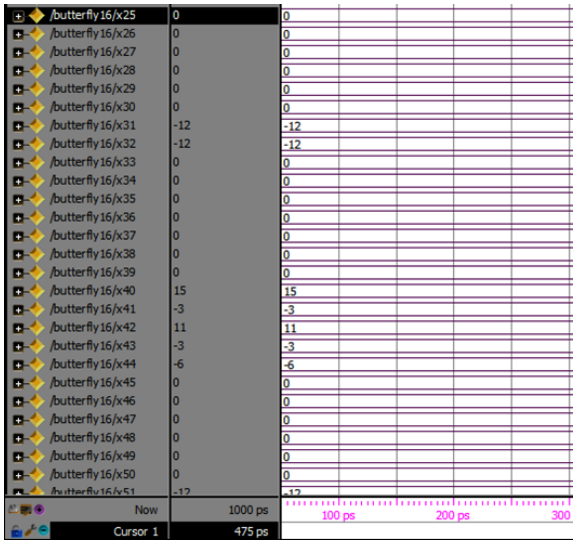


Figure.27. Simulation results of 64-points radix-4 DITFFT



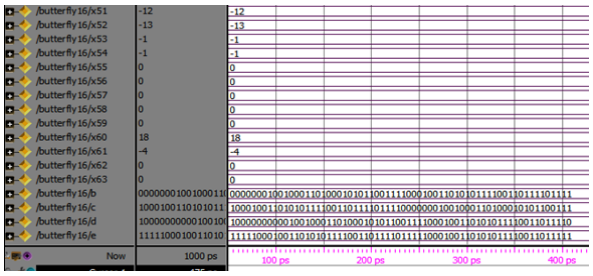


Figure.27. Simulation results of 64-points radix-4 DITFFT (cont…)

3.3 Design summary

The design summary of 64-point Radix-4 DIT-FFT algorithm is given in Table 3. Total equivalent gate count for the proposed FFT is 54,224 which includes –input LUT's, Logic Slices, IOB's etc. Since IOB's and BUF's are over mapped, the target device used additional 49,152 JTAG IOB's. Excluding IOB's proposed FFT utilized 62.67% of devices in the target device.

Table 3: Design summary of 64-point Radix-4 DITFFT algorithm

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	5,792	9,312	62%	
Logic Distribution				
Number of occupied Slices	3,286	4,656	70%	
Number of Slices containing only related logic	3,286	3,286	100%	
Number of Slices containing unrelated logic	0	3,286	0%	
Total Number 4 input LUTs	5,968	9,312	64%	
Number used as logic	5,792			
Number used as a route-thru	176			
Number of bonded IOBs	1,024	232	441%	OVERMAPPED
Number of bonded Out/Bidir IOBs	512	176	290%	OVERMAPPED
Number of bonded Input IBUFs	512	56	1600%	
Number of MULT18X18SIOs	16	20	80%	
Total equivalent gate count for design	54,224			
Additional JTAG gate count for IOBs	49,152			

3.4 Timing report

The timing report is generated under speed grade -5. This report includes all the input and output cells and their fan-out. Each gate delay and net delay is also considered, and the summation of gate delay gives the timing delay of the project as 19ns. The comparison between the performance of radix-2 and radix-4 algorithm based on different aspects like number of slices used, LUTS, bonded IOBs, flip flops, global clocks for their operations is given in Table 4.

Table 4: Performance comparison between the radix-2 and radix-4 algorithms

Device utilization	**Radix-2**			**Radix-4**		
	used	**available**	**Utilization (%)**	**used**	**available**	**Utilization (%)**

Slices	2388	4656	51	3332	4656	71
4 input LUTs	4282	9312	45	5936	9312	63
IOBs	135	232	58	1024	232	441
Delay	75.050 ns			18.963 ns		
Memory	0.206720 GB			0.228 GB		
Power	56.78 mW			12.68 mW		

From the Table 4, it is observed that the minimum delay for functioning of inputs and outputs for radix-4 is very less when compared with radix-2 and memory usage is almost same with radix-2 even radix-4 using a greater number of inputs. And observed that 75% computations were saved in Radix-4 even though device utilization is more.

Table 5: Performance comparison among the Proposed FFT with existing works

Parameter	**This work**	**[2]**	**[8]**	**[10]**	**[14]**
FFT Size	64-4	32-8	64-4	16-4	16-4
Delay (ns)	18.96	419	8.10	2.2	2.67
Power(mW)	12.68	739.5	33.5	3.5	4

3.6 Medical image Compression

The proposed algorithm for image compression is simulated using the same targeted device given in starting of section 5 used for Radix-4. The considered medical image has compressed using different tolerance values like 0.0007625, 0.003246, 0.013075 and 0.03924 and it also returns the drop values calculated using the formula given in (7) as 0.10, 0.31, 0.61 and 0.83 respectively which is shown in figures 28, 29, 30, 31 and 32.

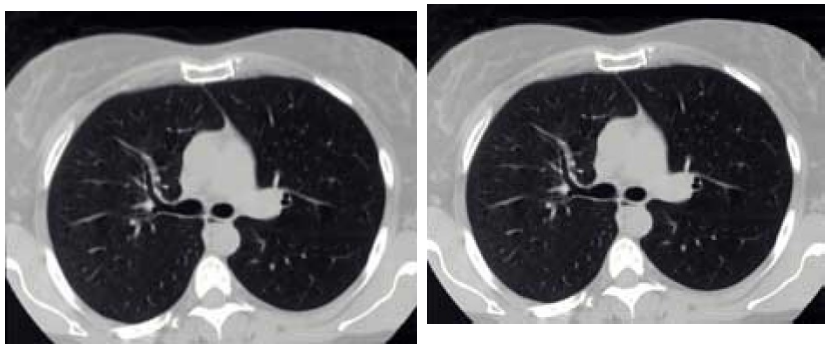


Figure.28: Original image and Compressed image with tolerenece=0.0007625 resulting drop ratio of 0.10

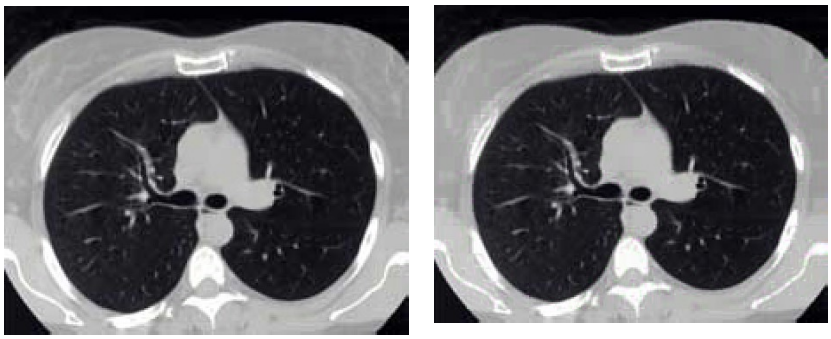


Figure.30: Compressed image with tolerenece=0.003246 resulting drop ratio of 0.31and Compressed image with tolerenece=0.013075 resulting drop ratio of 0.61

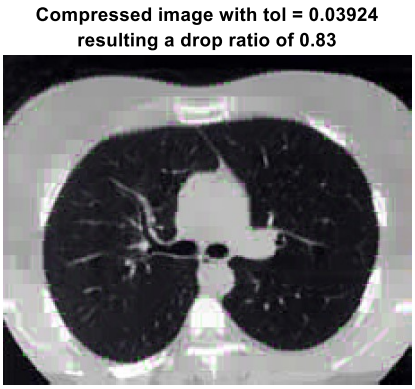


Figure.32: Compressed image with tolerenece=0.03924 resulting drop ratio of 0.83

4.Conclusion

In this article the new high-speed DIT-FFT algorithm based on radix-4 algorithm for medical image compression was proposed and simulated on a target device xc3s500e-5fg320. The simulation results show that radix-4 processes the input with less delay. From the time delay table, it is very clear that approximately 75% of processing time is saved with less memory usage. Proposed radix algorithm also shown low power consumption than the existing radix2 this makes the use of the present algorithm in medical field where low power devices are preferable. This is another milestone for this article. Due to these advantages the proposed algorithm used in the medical image compression. Results observed for different tolerances and their drop ratios. It is observed that the scaling factor for image compression depends on tolerance as directly proportional. According to the obtained results for different tolerances like 0.0007625, 0.003246, 0.013075 and 0.03924 the drop values are 0.10, 0.31, 0.61 and 0.83 respectively. The level of compression and tolerance values for medical

images can be chosen based on the drop ratio and application. Before being taken it for ASIC, it can be tested on FPGA for speed and further embedding of required components.

References

[1] Shashikala, B. N., Sudha, B. S., & Sarkar, S. (2020, November). Efficient Implementation of Radix-2 FFT Architecture using CORDIC for Signal Processing Applications. In 2020 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT) (pp. 137-142). IEEE.

[2] Y. Jyotsna, N. Nithiyameenatchi, E. Konguvel and M. Kannan, Performance analysis of radix-2/3/5 decompositions in fixed point DIT FFT algorithms. International Conference on Computer Communication and Informatics (ICCCI), 2020, (pp. 1-7). IEEE.

[3] A. Ganguly, A. Chakraborty & A. Banerjee A novel VLSI design of radix-4 DFT in current mode, International Journal of Electronics, 2019; 106(12):1845-63.

[4] Sonali D. Patil, Manish Sharma. A 2048-point Split-Radix Fast Fourier Transform Computed using Radix-4 Butterfly Units, International Journal of Recent Technology and Engineering (IJRTE) 2019; 8(3):2043-46.

[5] G. Ramprabu, V. Rajmohan, V. R. Prakash, and N. Shankar. Analysis of Feed forward Radix-2^2 FFT 4-Parallel Architecture. International Conference on Smart Systems and Inventive Technology (ICSSIT) 2019, (pp. 168-172). IEEE.

[6] S. M. Noor, E. John and M. Panday. Design and Implementation of an Ultralow-Energy FFT ASIC for Processing ECG in Cardiac Pacemakers, in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems 2019; 27(4):983-7,

[7] Z. A. Abbas, N. B. Sulaiman, N. A. M. Yunus, W. Z. Wan Hasan and M. K. Ahmed, An FPGA implementation and performance analysis between Radix-2 and Radix-4 of 4096-point FFT. IEEE 5th International Conference on Smart Instrumentation, Measurement and Application (ICSIMA) 2018, (pp. 1-4). IEEE.

[8] Anitha T G, K Vijayalakshmi, FFT Based Compression Approach for Medical Images. International Journal of Applied Engineering Research 2018; 13(6):3550-67.

[9] Mario Garrido Gálvez, Miguel Angel Sanchez, Maria Luisa Lopez-Vallejo and Jesus Grajal. A 4096-Point Radix-4 Memory-Based FFT Using DSP Slices. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems 2017; 25(1):375-9.

[10] B. N. Mohapatra and R. K. Mohapatra. FFT and sparse FFT techniques and applications. Fourteenth International Conference on Wireless and Optical Communications Networks (WOCN), 2017, (pp. 1-5). IEEE.

[11] R. H. Neuenfeld, M. B. Fonseca, E. A. C. da Costa and J. P. Oses. Exploiting addition schemes for the improvement of optimized radix-2 and radix-4 FFT butterflies. IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS), 2017 (pp. 1-4). IEEE.

[12] Anitha T.G, K Vijayalakshmi. Design of Novel FFT Based Image Compression Algorithms and Architectures. International Journal of Progressive Science and Technology (IJPSAT) 2017; 5(1):24-42.

[13] SumeetWalia, Sachin Majithia, Adaptive Gaussian Filter Based Image Recovery Using Local Segmentation, International Journal of Technology And Computing (IJTC) 2016; 2(1).

[14] R. Neuenfeld, M. Fonseca and E. Costa, Design of optimized radix-2 and radix-4 butterflies from FFT with decimation in time, IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS), 2016, (pp. 171-174). IEEE.

[15] Z. Qian and M. Margala, "Low-Power Split-Radix FFT Processors Using Radix-2 Butterfly Units," in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 2016; 24(9):3008-12.

[16] Z.-G. Ma, X.-B. Yin, and F. Yu, A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm, *IEEE Trans. Circuits Syst. II, Exp. Briefs*, 2015; 62(9):876–80.

[17] K. Jayaram and C. Arun. Survey report for Radix-2, Radix-4 and Radix-8 FFT Algorithms, in International Journal of Innovative Research in Science, Engineering and Technology 2015; 4(7):5149-54.

[18] Brundavani P. FPGA Implementation of 256-Bit, 64-Point DIT-FFT Using Radix-4 Algorithm, in International Journal of Advanced Research in Computer Science and Software Engineering, 2015; 3(9):126- 33.

[19] Zhuo Qian, Nasibeh Nasiri, Oren Segal, and Martin Margala. FPGA implementation of low-power split-radix fast fourier transform processors, in Proc. 24th International Conference. Field Program. Logic Applications. Munich, Germany, 2014, (pp. 1–2). IEEE.

[20] Amarnath Reddy and Venkata Suman, Design and Simulation of FFT Processor Using Radix-4 Algorithm Using FPGA, International Journal of Advanced Science and Technology, 2013; 61:53-62.