International Journal of Scientific Engineering and Technology Research

www.ijsetr.com

ISSN 2319-8885
Vol.03,Issue.27
September-2014,
Pages:5486-5491

MDC FFT/IFFT Processor with 64-Point using Radix-4 Algorithm for MIMO-OFDM System

VARUN REDDY.P[1], M.RAMANA REDDY[2]

[1]PG Scholar, Dept of DSCE, RGM College of Engineering & Technology, Nandyal, AP, India,
E-mail: varunreddi.p@gmail.com.

[2]Assoc Prof, Dept of ECE, RGM College of Engineering & Technology , Nandyal, AP-India,
E-mail: Ramanareddy0106@gmail.com.

Abstract: Human needs with technical de vices are increasing rapidly. In order to meet their requirements the system should be accurate and fast. The fastness and accuracy of a system depends on its intra and inter peripherals/algorithms. In the view of t his, the proposed paper came into existence. MDC, we propose simple memory scheduling methods for input data and output bit/set-reversing, which again results in a full utilization rate in memory usage. It focuses on the development of the Fast Fourier Transform (FFT) algorithm, based on Decimation-In- Time (DIT) domain, calle d Radix-4 DIT-FFT algorithm. Verilog is used as a design entity and for simulation Xilinx ISE and modelsim. The synthesis results show that the computation for calculating the 64-point FFT is efficient in terms of speed and area is implemented on UMC 90-nm CMOS technology. This processor can be used in IEEE 802.16 WiMAX and 3G PP long term evolution applications.

Keywords: Butterfly, FFT, DITFFT, Memory Scheduling, MIMO, OFDM, Output Sorting, MDC And Radix.

I. INTRODUCTION

Fast Fourier transform (FFT) is a cruc ial block in orthog-onal frequency division m ultiplexing (OFDM) systems. OFDM has been a dopted in a wide range of applications from wired-communication modems, such as digital subscriber lines(xDSL) to wireless communication modems, such as IEEE802.11 WiFi, IEEE802.16, WiMAX or 3GPP long term evolution (LTE), to process baseband data. Inverse fast Fourier transform (IFFT) converts the modulated information from frequency domain to time domain for transmission of radio signals, while FFT gathers samples from the time domain, restoring them to th e frequency domain. With multiple input multiple output (MIMO) devices, data throughput can b e increased dramatically. Hence MIMO-OFDM systems provide promising data rate and reliability in wireless communica-tions. To handle "multiple" data streams, intuitively the functional blocks need to be duplicated for processing the concurrent inputs. Without a proper design, the complexity of FFT/IFFT processors in MIMO systems grows linearly with the number of data streams.

The fastness of the system depends on their intra and inter peripherals, the intra peripherals depend on the designers ch oice and the inter peripherals depend on the users choice. Designers choice includes components, algorithms etc, users choice includes inputs, external devices, signals etc. Currently it has been focus d on radix-2 algorithm which has more delay [1]. To overcome this problem there is a need to modify in the algorithm majorly. The paper deals with change in the algorithm called radix- 4 algorithm and focus on the design and implementation of 64-point DIT Fast Fourier Transform (DIT-FFT) for a UMC 90nm CMOS Technology kit. Verilog is used for coding; simulation and synthesis are performed by using Modelsim ISE and Xilinx ISE Design Suite respectively. For implementation, code is developed in Verilog an d dumped on UMC 90nm for verification. The proceeding section gives the existing approach and the problem hypothesis. The section III explains about the proposed methodology and architecture. The next section gives the results and synthesis report after implementation. Finally the paper is concluded in section VII.

II. EXISTING SY STEM

The existing system was based on radix-2 algorithm. The so called radix-2 is due to its base is equals to 2 and the representation is 2^M where M represents the index/stage and its value is a positive integer. The computation of radix-2 made up of butterflies called R adix-2 butterflies. Depending on the type of decimation in the different domains, it is two types; they are Decimation in Time FFT (DIT-FFT) and Decimation in Frequency FF T (DIF-FFT). For radix-2 there are two inputs and two outputs and the inputs are arranged in b it reversal order, because of saving the memory locations and outputs are in a normal order for DIT-FFT and vice versa for DIF-FFT. The given whole number decides the number of stages a s $\log_2 N$, and each stage consists of blocks it can be given as $N/2^{stage}$ and each block contains butterfllies it can be given as $2^{stage-1}$, each stage includes the twiddle factors (W). Each and every stage having complex computations or simply complex multiplications and complex additions. Generally radix-2 having N/2log$_2$ N complex multiplications and Nlog$_2$ N complex additions. Radix-2 algorithm mainly depends on these computations, as number of stages increases proportionally computations are also increases. For example: for 16-point FFT the complex multiplications and additions are 32 and 64 respectively and for 32 point FFT these are 90 and 180 respectively and so on. If number of inputs is more it became impossible to calculate by using R2 algorithm. By this as number of inputs is more, the complexity in calculations also more[1], [3].

III. PROPOSED SYSTEM

The proposed system is based on radix-4 algorithm. Radix-4 is another FFT algorithm which was surveyed to improve the speed of functioning by reducing the computation; this can be obtained by change the base to 4. For a same number if base increases the power/index will decreases. For radix-4 the number of stages are reduced to 50% since N=4^3 (N=4^M) i.e. only 3 stages. Radix-4 is having four inputs and four outputs and it follows in-place algorithm. The following will explain the functioning of radix-4 and how the computational complexity is reduced.

A. Functioning Of Radix-4 Algorithm

The radix-4 DIT-FFT recursively partitions a DFT into four quarter-length DFTs of groups of every fourth time sample. The outputs of these shorter FFTs are reused to compute many outputs, thus greatly reducing the total computational cost. The radix-4 FFTs require only 75% as many complex multiplications as the radix-2 FFTs. The radix-4 decimation-in-time and decimation-in-frequency Fast Fourier transforms (FFTs) gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs. The radix-4 decimation-in-time algorithm rearranges the DFT equation into four parts: sums over all groups of every fourth discrete-time index n = [0, 4, 8 ... N - 4], n = [1, 5, 9 ... N - 3], n = [2, 6, 10 ... N - 2] and n = [3, 7, 11 ... N - 1], (This works out only when the FFT length is a multiple of four.) Just as in the radix-2 DITFFT, further mathematical manipulation shows that the length-N DFT can be computed as the sum of the outputs of four length-N/4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors. Decimate/split the N-point input sequence into four sub sequences, x(4n), x(4n+1), x(4n+2), x(4n+3), n = 0, 1, ... , N/4-1.

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{(-j2\pi nk/N)} \qquad (1)$$

$$= \sum_{n=0}^{\frac{N}{4}-1} x(4n)e^{(-j2\pi(4n)k/N)}$$

$$\sum_{n=0}^{\frac{N}{4}-1} x(4n+1)e^{(-j2\pi(4n+1)k/N)}$$

$$\sum^{\frac{N}{4}-1} x(4n+2)e^{(-j2\pi(4n+2)k/N)}$$

$$\sum_{n=0}^{\frac{n}{4}-1} x(4n+3)e^{(-j2\pi(4n+3)k/N)}$$

$$X(k) = \{DFT[x(4n)] + W^{k}{}_{N}\, DFT_{N/4}\, [x(4n+1)] + W^{2k}{}_{N}\, DFT_{N/4N/4}\, [x(4n+2)] + W^{3k}{}_{N}\, DFT_{N/4}\, [x(4n+3)]\} \qquad (2)$$

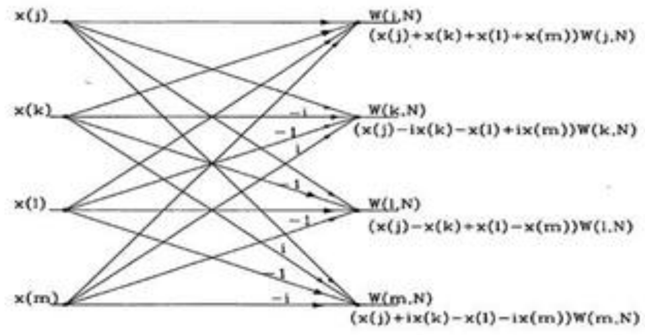

x(j)
x(k)
x(l)
x(m)
W(j,N)
(x(j)+x(k)+x(l)+x(m))W(j,N)
W(k,N)
(x(j)−ix(k)−x(l)+ix(m))W(k,N)
W(l,N)
(x(j)−x(k)+x(l)−x(m))W(l,N)
W(m,N)
(x(j)+ix(k)−x(l)−ix(m))W(m,N)
−i
−1
i
i
−1
−i
i
−1
−i

Fig.1. Basic radix-4 butterfly operation.

This is called decimation in time because the time samples are rearranged in alternating groups. In a radix-4 algorithm, the four groups are referred by "(1), (2)". The basic operation of R4 butterfly is shown in Fig.1.[8], [10].

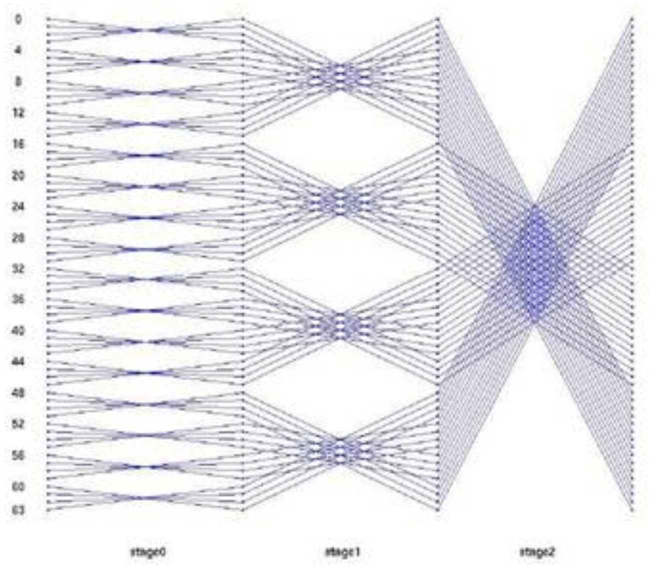


Fig.2. 64-point radix-4 DITFFT butterfly diagram.

Fig.2 depicts a 64-point radix-4 DIT-FFT using the butterfly symbol shown in Fig.1 to represent the mathematical operations. [courtesy Wikimedia].

IV.FFT ALGORITHM

In this section we present several methods for computing the DFT efficiently. In view of the importance of the DFT in various digital signal processing applications, such as linear filtering, correlation analysis, and spectrum analysis, its efficient computation is a topic that has received considerable attention by many mathematicians, engineers, and applied scientists. From this point, we change the notation that X(k), instead of y(k) in previous sections, represents the Fourier coefficients of x(n). Basically, the computational problem for the DFT is to compute the sequence {X(k)} of N complex-valued numbers given another sequence of data {x(n)} of length N, according to the formula

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad 0 \le k \le N-1 \tag{3}$$

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

In general, the data sequence x(n) is also assumed to be complex valued. Similarly, The IDFT becomes

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k) W_N^{-nk}, \qquad 0 \le n \le N-1 \tag{4}$$

Since DFT and IDFT involve basically the same type of computations, our discussion of efficient computational algorithms for the DFT applies as well to the efficient computation of the IDFT. We observe that for each value of k, direct computation of X(k) involves N complex multiplications(4N real multiplications) and N-1 complex additions (4N-2 real additions). Consequently, to compute all N values of the DFT requires N 2 complex multiplications and N 2-N complex additions. Direct computation of the DFT is basically inefficient primarily because it does not exploit the symmetry and periodicity properties of the phase factor W_N. In particular, these two properties are:

$$\text{Symmetry property}: W_N^{k+N/2} = -W_N^k \tag{9}$$

$$\text{Periodicit y property}: W_N^{k+N} = W_N^k \tag{10}$$

The computationally efficient algorithms described in this section, known collectively as fast Fourier transform (FFT) algorithms, exploit these two basic properties of the phase factor.

A. Radix-4 FFT Algorithm

When the number of data points N in the DFT is a power of 4 (i.e., N = 4^v), we can, of course, always use a radix-2 algorithm for the computation. However, for this case, it is more efficient computationally to employ a radix-r FFT algorithm. Let us begin by describing a radix-4 decimation-in-time FFT algorithm briefly. We split or decimate the N-point input sequence into four subsequences, x(4n), x(4n+1), x(4n+2), x(4n+3), n = 0, 1, ... , N/4-1.

$$X(p,q) = \sum_{l=0}^{3}\left[W_N^{lq} F(l,q)\right] W_4^{lp} \tag{11}$$

$$F(l,q) = \sum_{m=0}^{(N/4)-1} x(l,m) W_{N/4}^{mq} \tag{12}$$

$$p = 0,1,2,3; \quad l = 0,1,2,3; \quad q = 0,1,2,\ldots,\frac{N}{4}-1$$

$$x(l,m) = x(4m+1)$$

and

$$X(p,q) = X\left(\frac{N}{4}p + q\right) \tag{13}$$

Thus the four N/4-point DFTs F(l, q)obtained from the above equation are combined to yield the N-point DFT. The expression for combining the N/4-point DFTs defines a radix-4 decimation-in-time butterfly, The radix-4 butterfly is depicted in Fig.3 and in a more compact form in Fig.4. Note that each butterfly involves three complex multiplications, since $W_N^0 = 1$, and 12 complex additions.

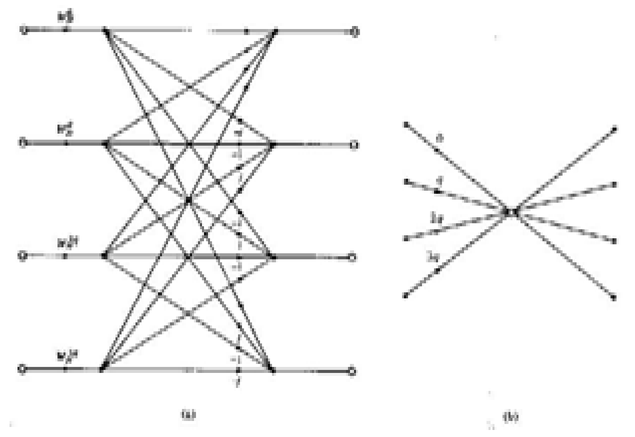


Fig.3. Basic butterfly computation in a radix-4 FFT algorithm.

By performing the additions in two steps, it is possible to reduce the number of additions per butterfly from 12 to 8. This can be accomplished t_y expressing the matrix of the linear transformation mentioned previously as a product of two matrices as follows:

$$\begin{bmatrix} X(0,q) \\ X(1,q) \\ X(2,q) \\ X(3,q) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} W_N^0 F(0,q) \\ W_N^1 F(1,q) \\ W_N^2 F(2,q) \\ W_N^3 F(3,q) \end{bmatrix}$$

B. FFT Architecture Design

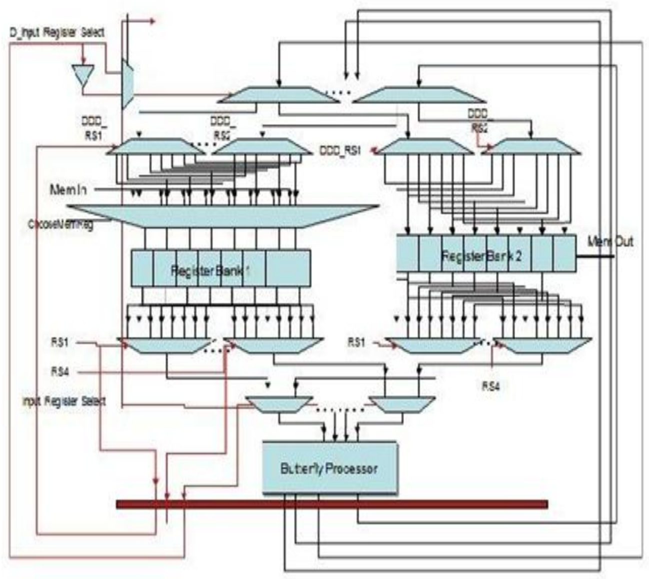


Fig.4. Radix 4 FFT processor Internal architecture.

C. Radix 4 Block Diagram

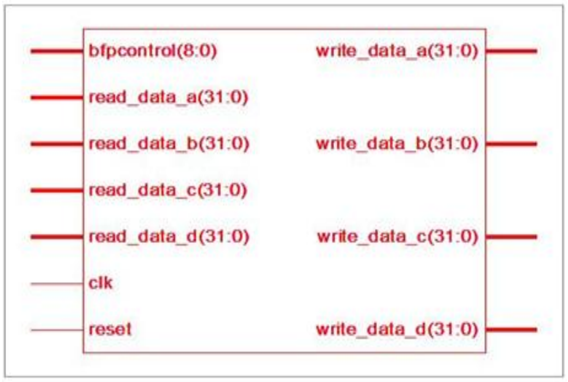


Fig.5. Black Box view of 64 point FFT using Radix 4 algorithm.

V. MDC ARCHITECTURE FOR MIMO FFT/IFFT

Storage elements dominate most of the area in conventional MDC architecture. That is, the input buffering stage for radix-4 based FFT/IFFT needs N/4 + N/2 + 3N/4 words of memory, and each computing stage needs 3N/4s words of memory, where s is the stage index. For a 64-point MDC FFT/IFFT processor is as shown in Fig.5, 160 words of memory are required. If MDC is applied in MIMO-OFDM systems, the memory size grows linearly with the number of data streams. As for the utilization rate of butterflies and multipliers, since 3/4 of the computing time is used to gather the input data, the utilization rate is only 25% in single stream radix-4 MDC FFT/IFFT. Although MDC architecture offers an intuitive and simpler data flow control, most of the previous works use SDF instead of MDC for complexity concern. However, for MIMO FFT/IFFT, we found that if the data streams are properly scheduled, the utilization rate can increase from 25% to 100%. This makes MDC very suitable for MIMO-OFDM systems.

The proposed memory scheduling mechanism reduces the size of storage elements. Moreover, the mechanism properly shuffles the four input streams such that stage one to stage five are all with the same feed-forward switch-box data flow. Therefore, the control simplicity of MDC schemes can be preserved while the memory size is greatly reduced. As for the utilization rate of butterflies and multipliers, each one of the four input symbols after memory scheduling takes 25% of one symbol-time for radix-4 butterfly computation. Consequently one radix-4 butterfly and three twiddle-factor multipliers in each pipeline stage can process four data streams without any idle period, that is, the utilization rate of butterflies and multipliers is 100%. Furthermore, the radix-8 butterfly at the last stage can be configured as a radix-4 butterfly. With such flexibility, radix-2 computation can be incorporated at the last radix-8 stage, and thus for any N in power-of-2 fashion can be computed with this proposed method. Finally, the serial blocks of output symbol format helps to reduce the memory usage for output sorting and the complexity of the modules followed by the FFT/IFFT processor.

TABLE II: Elements in Prop OS ED MDC FFT/IFFT Process OR

Components	Purpose	Number
Complex number multipliers	Twiddle factors multiplication with radix-4/8 outputs	12
FFT butterflies	Radix-4	4
	Radix-4/8	1
Memory macros	Dual-Port SRAM (words)	10224
Other control modules	Switch-box	7
	Input memory addressing	1
	Output FIFO control	1
	FIFO registers (words)	456
	Quadrant conversion of twiddle factor	12
	Twiddle factor generator	4

VI. RESULTS

A. Synthesis Results

The Fig.6 represents 64-point radix-4 DIT-FFT butterfly diagram in which there are only 3 stages. The internal stages are top radix, comutator, radix 8 and 4, basic DFT four operation, comutator.

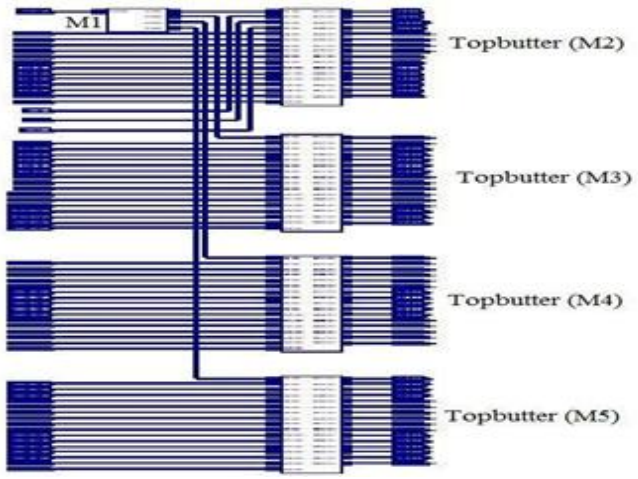


Fig.6. Internal structure of top module.

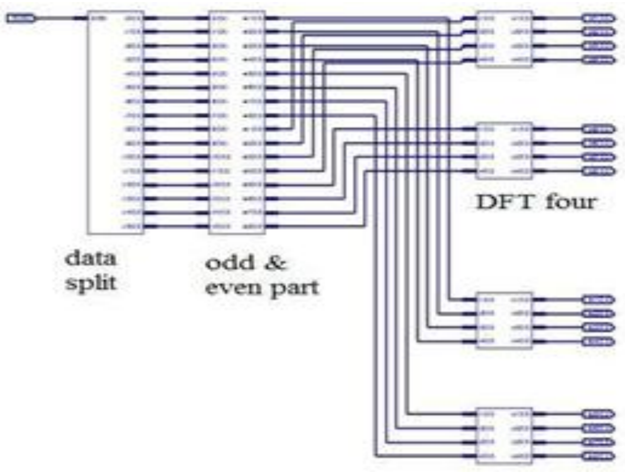


Fig.7. Internal structure of Comutator.

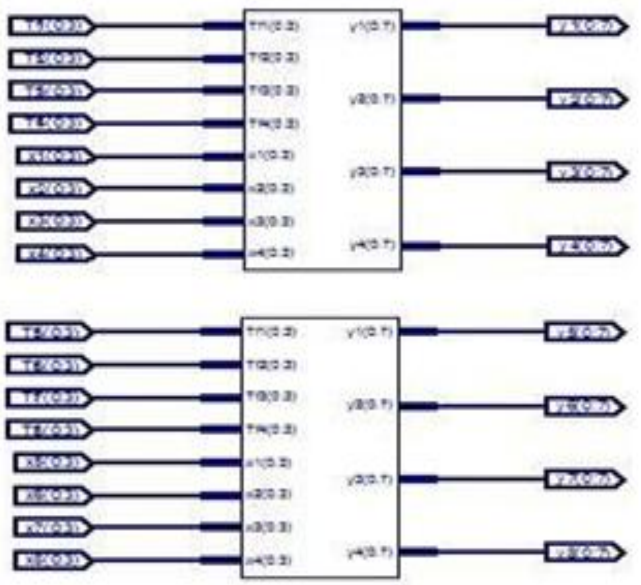
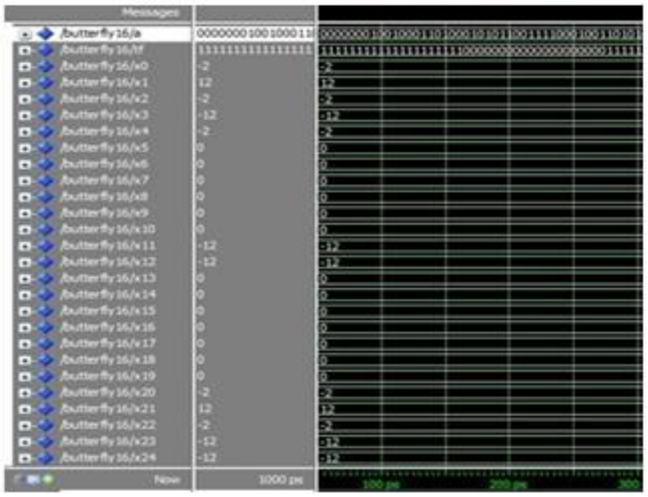


Fig.8.Even and odd parts in top radix-4.

The internal RTL view of top module is shown in fig.6 in which 256-bit input and 512-bit outputs are present. Internal structure of top module consists of four top butters which are equal in size and having same number of inputs is shown in fig.7. Internal view of top butter M_2, in this module even and odd parts are present and a comutator is used to connect all these parts inside the M_2, and this is common to other three modules M_3, M_4 and M_5 is shown in fig.8. Internal structure of comutator is shown in fig.9. Even and odd parts of top radix 8 is shown in fig.10 and internal diagram of even/odd part consists of adder, subtractor and multiplier as an internal components, is shown in fig.11.[11]

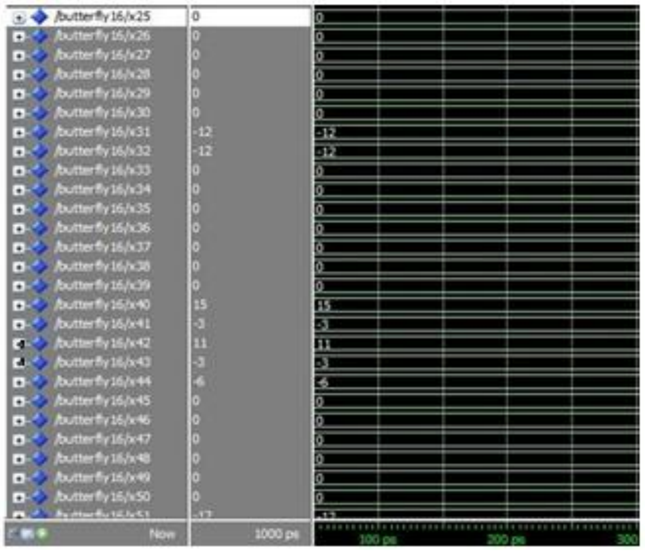
B. Simulation Results

The input A of 64-points with each point of 4-bits, totally 256-bits, twiddle factor (W) is also of 256-bits can be observed in Fig.11. The X is an output of 64-points and each point of 8-bits, (since even and odd part) resulting totally 512-bits all in binary format.[9]



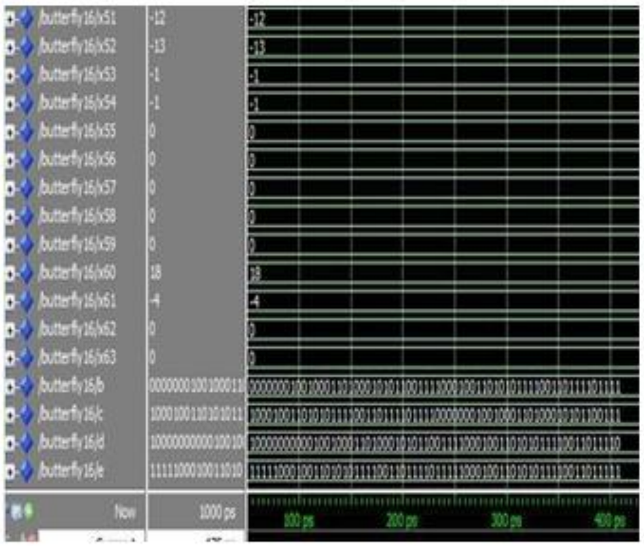
```
Messages
/butterfly16/a     000000010010001110...
/butterfly16/tf    1111111111111111...
/butterfly16/x0    -2
/butterfly16/x1    12
/butterfly16/x2    -2
/butterfly16/x3    -12
/butterfly16/x4    -2
/butterfly16/x5    0
/butterfly16/x6    0
/butterfly16/x7    0
/butterfly16/x8    0
/butterfly16/x9    0
/butterfly16/x10   0
/butterfly16/x11   -12
/butterfly16/x12   -12
/butterfly16/x13   0
/butterfly16/x14   0
/butterfly16/x15   0
/butterfly16/x16   0
/butterfly16/x17   0
/butterfly16/x18   0
/butterfly16/x19   0
/butterfly16/x20   -2
/butterfly16/x21   12
/butterfly16/x22   -2
/butterfly16/x23   -12
/butterfly16/x24   -12
Now                1000 ps
```

Fig.9. Simulation results of 256-bit, 64-point radix-4 DIT-FFT (Cont…).



```
/butterfly16/x25   0
/butterfly16/x26   0
/butterfly16/x27   0
/butterfly16/x28   0
/butterfly16/x29   0
/butterfly16/x30   0
/butterfly16/x31   -12
/butterfly16/x32   -12
/butterfly16/x33   0
/butterfly16/x34   0
/butterfly16/x35   0
/butterfly16/x36   0
/butterfly16/x37   0
/butterfly16/x38   0
/butterfly16/x39   0
/butterfly16/x40   15
/butterfly16/x41   -3
/butterfly16/x42   11
/butterfly16/x43   -3
/butterfly16/x44   -6
/butterfly16/x45   0
/butterfly16/x46   0
/butterfly16/x47   0
/butterfly16/x48   0
/butterfly16/x49   0
/butterfly16/x50   0
Now                1000 ps
```

Fig.10. Simulation results of 64-point radix-4 DIT-FFT (Cont…).



```
/butterfly16/x51   -12
/butterfly16/x52   -13
/butterfly16/x53   -1
/butterfly16/x54   -1
/butterfly16/x55   0
/butterfly16/x56   0
/butterfly16/x57   0
/butterfly16/x58   0
/butterfly16/x59   0
/butterfly16/x60   18
/butterfly16/x61   -4
/butterfly16/x62   0
/butterfly16/x63   0
/butterfly16/b     000000010010001110...
/butterfly16/c     1000100110101011...
/butterfly16/d     1000000000010010...
/butterfly16/e     1111100010011010...
Now                1000 ps
```

Fig.11. Simulation results of 256-bit, 64-point radix-4 DIT-FFT.

VII. CONCLUSION

This project presents the new high speed FFT architecture based on radix-4 algorithm. The pipelined 64-point radix-4 DIT-FFT can be implemented easily by using CMOS technologies, such portability is offered by this algorithm. The proposed design is based on an MDC architecture, which is generally not preferred, due to its low utilization rate in memory and computational elements such as adders and multipliers. However, by using the proposed memory scheduling, MDC architecture is proved suitable for FFT/IFFT processors in MIMO-OFDM systems. From the above synthesis and simulation results of radix-4 64-points it is understandable that radix-4 having less delay in processing the input when compared with radix-2. Comparing with radix-2 algorithm, 75% of time is saved in radix-4 algorithm. As the delay time is reduced the fastness of the system is increased. the proposed designs found a good balance among complexity, energy consumption, and chip area, for the MIMO-OFDM systems.

VIII. ACKNOWLEDGMENT

The author would like to thank RGM College Of Engineering & Technology, Nandyala to carry out the proposed work which with support from the organisation.

IX. REFERENCES

[1] Asmitha Haveliya, Amity University Lucknow, India "Design And Simulation Of 32-Point FFT Using Radix-2 Algorithm For FPGA Implementation" 2012 Second International Conference on Advanced Computing & Communication Technologies.
[2] J. W. Cooley and J. W. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," Math. Comput., vol. 19, pp. 297–301, Apr. 1965.
[3] Douglas L. Jones "Decimation-in-Time (DIT) Radix-2 FFT Algorithms" Connexions module: m12016
[4] J. G. Proakis and D. G. Manolakis, Digital Signal Processing Prentice-Hall, 1996.
[5] J. Rabaey,A. Chandrakasan, and B. Nikolic, Digital Integrated Circuits A Design Perspective. Prentice-Hall, 2003.
[6] K. Parhi, VLSI Digital Signal Processing Systems New York, NY, USA: John Wiley & Sons, 1999.
[7] S. Johansson, S. He, and P. Nilsson, "Wordlength Optimization of a Pipelined FFT Processor," in Proc. of 42nd Midwest Symposium on Circuits and Systems, Las Cruces, NM, USA, Aug. 8-11 1999.
[8] Douglas L. Jones "Radix-4 FFT Algorithms" Connexions module: m12027 .
[9] W. Li and L. Wanhammar, "A Pipelined FFT Processor," in IEEE Workshop on Signal Processing Systems, 1999, pp. 654–662.
[10] "FFTs on the multi-core Anemone processor", April 2011(updated on 2012), by Paralant Ltd. (www.paralant.com). Numerical Libraries for the EpiphanyTM architecture, InsightTM .
[11] G Sudhakiran, P Brundavani "Design, Simulation and Comparison of 256-bits 64-points, Radix-4 and Radix-2 Algorithms", AECE-IRAJ International Conference, 14th July 2013.
[12] P.Varun reddy, M Ramana Reddy"MDC FFT/IFFT Processor 256-bits 64-points, Radix-4 Algorithm for MIMO-OFDM systems", AECE-IRAJ International Conference, 18th July 2013.

Author's Profile:

P.Varun Reddy, born in Kadapa, A.P., India in 1990.He received B.Tech Degree in Electronics and Communication Engineering from J.N.T University Anantapur, India. Presently pursuing M.Tech (Digital Systems And Computer Electronics) from RGM College Of Engineering and Technology, Nandyal, A.P., India. His research interests include VLSI Systems, Digital Signal Processing.
Email: varunreddi.p@gmail.com.

Mr. M.Ramana Reddy, PhD, Associate Professor in ECE Department. He is the Professional memberships of the MISTE, MIE, FIETE and PhD also. He has published number of papers in conferences & journals and presently with RGM College of Engineering and Technology, Nandyal, Kurnool, A.P., INDIA. Her interest areas are Digital IC Design, VLSI Technology, & Testing and Testability of digital circuits.
Email: Ramanareddy0106@gmail.com.