Competent solution for 2D-FFT and 2D-IFFT Computation using FPGA IP-core

Shwetha*, R.Srikantaswamy, PhD
*M.Tech in Signal Processing, #Professor
(Dept of ECE)
Siddaganga Institute of Technology
Tumakuru

Jayanta Laha, Tanisha Bhatia, Arindam Mal
Scientist, LEOS (ISRO)
Peenya 1st stage, 1st cross
Bengaluru

ABSTRACT
Image processing is a recent trend that is grabbing attention in almost all areas like space, medical, defence, authentication systems etc.,. Discrete Fourier Transforms is one of the most used transforms in image processing. Discrete Fourier Transform helps to transform the signal from spatial domain to frequency domain which is often used for filtering, correlation analysis and spectrum analysis. For DFT, computational complexity is more. Among different approaches to compute DFT, Fast Fourier Transform (FFT) is the feasible method that reduces the computational complexity. FFT can be implemented using DSP or FPGA. This paper lays a path to implement image FFT on FPGA using Intellectual Property (IP) core.

Keywords
2D-FFT and IFFT, IP cores, Radix-2

1. INTRODUCTION
Fundamental steps in image processing are image restoration, compression, reconstruction, filtering and enhancement to which Fourier Transform is the key tool. Discrete Fourier Transform (DFT) is confined to, for consideration is on digital images. Two-dimensional DFT is required for images. In Digital Signal Processing and Communications, FFT is one of the most utilized operations. And in modern communication systems, FFT and IFFT are hard-core requirements. General purpose DSP implementations fail to achieve the high performance demands owing to clock rate and the number of useful operations done per clock rather well suits for extremely complex math intensive tasks, with conditional processing. Due to inherent reprogram ability feature of FPGAs, it outperforms over other approaches. Exploiting the IP core in FPGA, 2D-DFT has been computed.

A high-level implementation of an efficient pipeline FFT algorithm Radix-2^2 Single path Delay Feedback on Virtex-E FPGAs that consumes minimum required amount of multipliers and storage has been presented in paper [1]. Handel-C has been made used to realize 1D 1024-point FFT with 16-bit input and Twiddle factors word length and maximum frequency of 82 MHz Same code can be utilized for synthesizes of higher power-of-4 complex-points FFT by specifying input word length, output word length, Twiddle factors word length and processing word length parameters as design requires. The paper [2] proposes the design and implementation of 32-point FFT processing block using VHDL and Xilinx ISE Design Suite 12.1. One more application where FFT/IFFT required is Orthogonal Frequency Division Multiplexing (OFDM). In paper [3], [8], an efficient VLSI implementation of FFT has been proposed to improve performance of OFDM.

2. RADIX-2 DECIMATION-IN-TIME FFT
If number *N* of data points is highly composite, which means *N* can be factored as $N = r_1 r_2 r_3 \ldots . r_v$ where r_v are prime, then this approach stands efficient. The DFTs are of same size *r* in case of same factors i.e.,$N = r_v$, where *r* is radix of the FFT algorithm. The *N* point data sequence is partitioned into two *N/2*-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of *x (n)* respectively:

$$f_1(n) = x(2n)$$
$$f_2(n) = x(2n+1), \quad \text{..... (1)}$$

Where $n = 0,1,2, \ldots \ldots \ldots \frac{N}{2} - 1$

The resulting algorithm is known as decimation-in-time algorithm, since $f_1(n)$ and $f_2(n)$ are obtained after decimating *x (n)* by a factor of 2 [6], [7].

N-point DFT is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad k = 0,1, \ldots \ldots N-1 \quad \text{..... (2)}$$

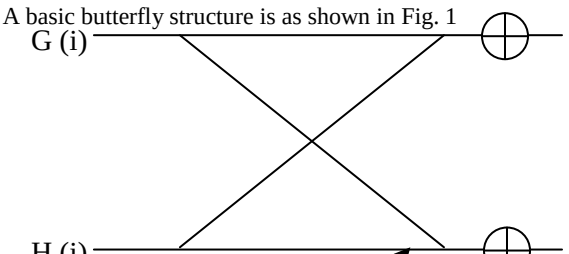
DFT expressed in terms of the DFTs of decimating sequences:

$$X(k) = \sum_{n\ even} x(n) W_N^{kn} + \sum_{n\ odd} x(n) W_N^{kn}$$
$$= \sum_{m=0}^{(N/2)-1} x(2m) W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_N^{k(2m+1)} \quad \text{..... (3)}$$

FFT exploits symmetry and periodicity properties of phase factors i.e., $W_N^{k+N/2} = -W_N^k$ and $W_N^{k+N} = W_N^k$ respectively.

Using direct DFT equation, the required number of complex multiplications is N^2 and complex additions is *N (N-1).* Four real multiplications and two real additions are required per complex multiplication. Hence, total number of real multiplications and real additions required are $4N^2$ and *N (4N-2)* respectively. In addition to these *N* complex input sequences and *N* output values has to be stored. The number of complex multiplications and complex additions will be reduced to *(N/2)* $log_2 N$ and *N* $log_2 N$ respectively to compute DFT with the use of Decimation-in-Time radix-2 FFT algorithm.

A basic butterfly structure is as shown in Fig. 1



W_N^i

$$W_N^{i+N/2} = -W_N^i$$

Fig. 1 Basic butterfly computation in the Decimation-In-Time FFT algorithm

3. FFT IMPLEMENTATION

There are various ways of implementing FFT [4-7], [10-12]. The method used is explained in detail.

3.1 IP core (CoreFFT)

Within the FPGA designs, IP core can be used as building blocks. For growing Electronic Design Automation (EDA) industry, IP cores are essential elements of design reuse. CoreFFT is designed for high- reliability applications requiring resistance to high temperature, firm- error immunity and radiation tolerance, such as radar, ground and air communications, acoustics, oil production and medical signal processing. CoreFFT produces FPGA optimized modules that perform FFTs to convert a signal from the time domain to the frequency domain in order to show the spectral content of the signal.

The Microsemi's fast Fourier transform (FFT) [1-4] core implements the efficient Cooley-Tukey algorithm for computing the discrete Fourier transform. CoreFFT is used in a broad range of applications such as digital communications, audio, measurements, control, and biomedical. CoreFFT provides highly parameterizable, area-efficient, and high performance MAC-based FFT. The core is available as a register transfer level (RTL) code of the transform in Verilog and VHDL languages. The core I/O ports are as shown in Fig. 2.

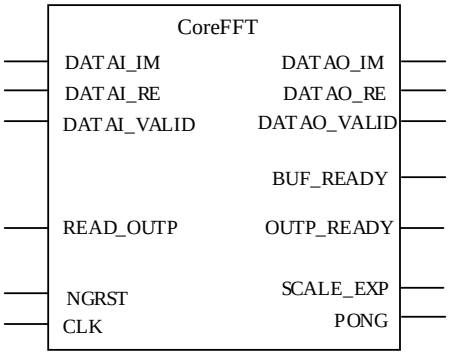


Fig. 2 Core I/O ports

Real and imaginary parts of input and output data with 'WIDTH'-bits are represented in two's compliment from each. FFT computation using CoreFFT follows frame wise processing with size of frame '*N*'. Two RAM modules constitute in-place memory with the capacity of *N /2* complex data storage doubling the bandwidth. The same in-place memory will be used for FFT computations and it begins the operation automatically after loading one frame.

The process flow is sequential i.e., stage after stage in in-place FFT computation. The number of stages required is log_2N. The data stored in memory is read in required order for FFT by the help of read switch and read address generator. The required twiddle factors for the butterfly are obtained by twiddle Look Up Table (LUT). The in-place memory also helps to store intermediate results with the help of write switch.

After the completion of last stage, the transformed results will be stored in the memory and outputs the transformed data frame as one word at a time. The twiddle factors in LUT will also be computed automatically by CoreFFT on power-on.

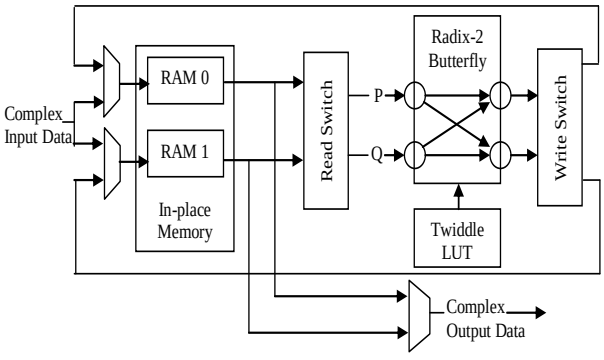
4. 2D-FFT AND 2D-IFFT COMPUTATION

Good design techniques are endorsed inherently by using state machine. It helps to prevent unnecessary development of bugs and provides flexibility in programming by modularizing the code into logical states. The most powerful aspect of state machines is its ability to make a program retort intelligently to a stimulus. Instead if looping is used to monitor repetitive tasks, in each of the loops, the functions handling these tasks must be distributed resulting in inefficiency and agitation to understand. Hence, here state machine is used to perform tasks in specified order. The State machine is shown in Fig. 4. The states used here are START, INITIALISE, ROW_FFT, COL_FFT_IM, COL_IFFT and ROW_IFFT.

4.1 2D-FFT

CoreFFT performs 1D-FFT frame-by-frame i.e., one row at a time will be processed. The row FFT results computed in this manner will be stored in SRAM row wise. After the computation of last row FFT CoreFFT is again used to compute column FFT by inputting the row results column wise. The results of FFT will be stored back to same SRAM.

The step wise flow of computation of 2D-FFT is as shown in Fig. 5.



Radix-2 FFT Functional Block Diagram (Minimal Configuration)

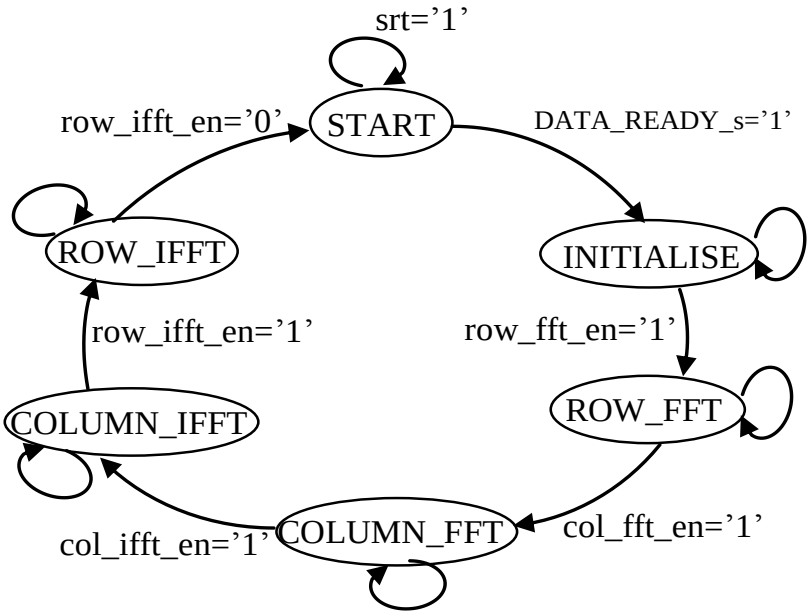
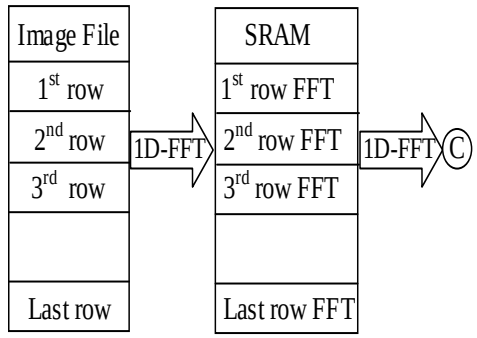


Fig. 4 State Machine



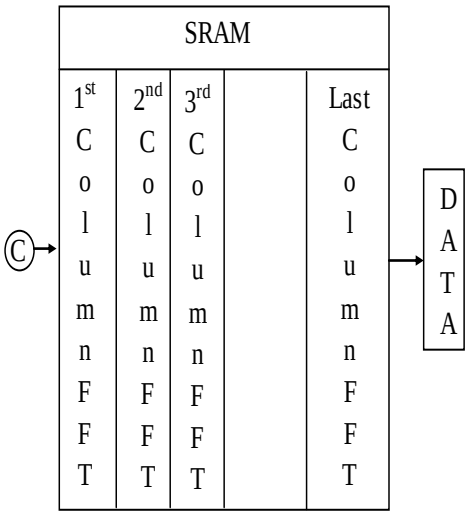


Fig. 5 Computational flow of 2D-FFT

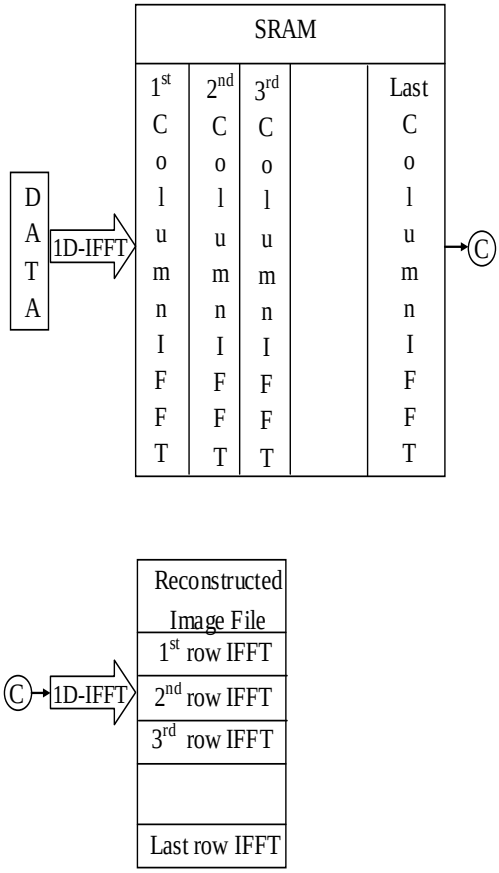


Fig. 6 Computational flow of 2D-IFFT

4.2 2D-IFFT

The transformed data stored will be given as input in column direction to CoreFFT with type specified as inverse. The 1D-IFFT results will be stored back to SRAM column wise. Then again the column IFFT results are fed as input to CoreFFT in row direction. The final transformed data will be stored in SRAM as well as written to a text file from which comparison with input image file can be made for error computation.

5. TEST RESULTS

The images are taken from LRO open source. The input image 1 and reconstructed image is as shown in Fig. 10 (a) and (b) respectively. The error measures used are Maximum Absolute Error (MAE), Relative Root Mean Square (RRMS) and Root Mean Square (RMS) [9].

RMS is given by:

$$RMS = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}|I^{ref}(i,j) - I^{rec}(i,j)|^2}{NXN}} \quad \text{..... (4)}$$

RRMS is given by

$$RRMS = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}|I^{ref}(i,j) - I^{rec}(i,j)|^2}{\sum_{i=1}^{N}\sum_{j=1}^{N}|I^{ref}(i,j)|^2}} \quad \text{.....(5)}$$

MAE is given by:

$$MAE = max\left|I^{ref}(i,j) - I^{rec}(i,j)\right| \quad \text{.....(6)}$$

Where, $I^{ref}(i, j)$ is input image

$I^{rec}(i, j)$ is reconstructed image

i, j are row and column values respectively

Fig. 7 shows the simulation results of the whole operation i.e., 2D-FFT and 2D-IFFT. First row FFT input and output are shown in Fig. 8. The results obtained are for the image of size 32x32. The 32-first row FFT values are shown in Fig. 9

The estimated RMS, RRMS and MAE values for images shown in Fig. 10 and 11 are tabulated as in Table I.

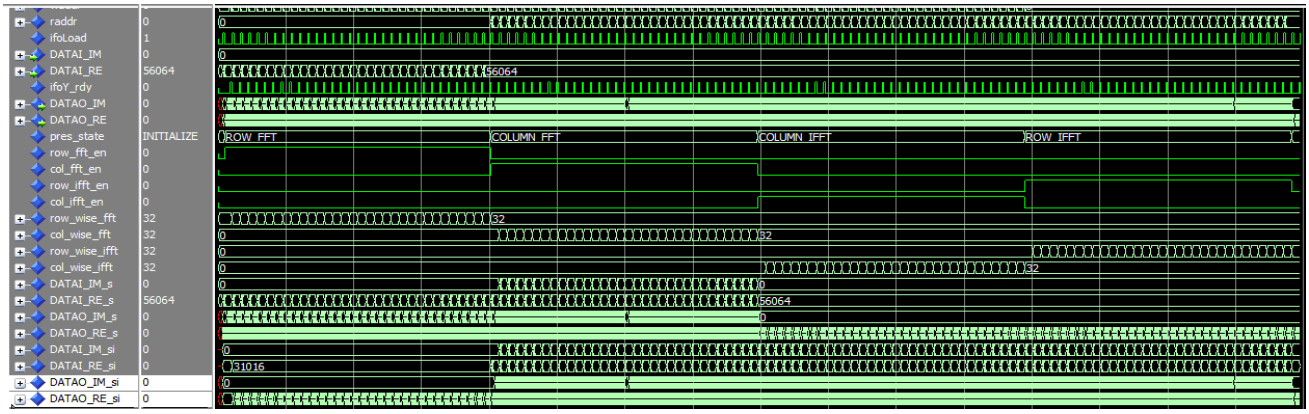


Fig. 7 Simulation result of 2D-FFT and 2D-IFFT operation on FPGA

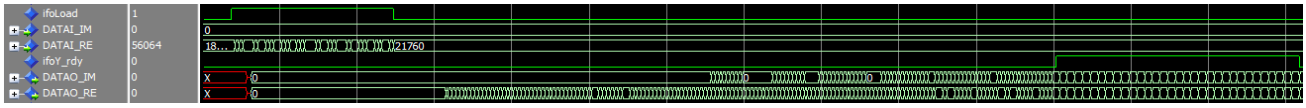


Fig. 8 Simulation result displaying first row FFT input and output set

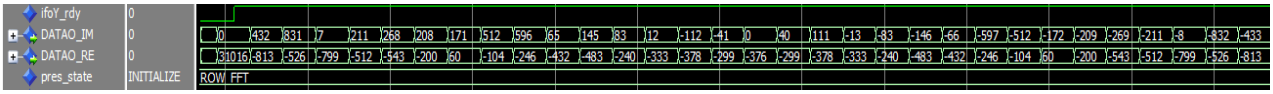


Fig. 9 Simulation result displaying first row FFT output

MAE values are scaled up by respective scaling factor. When scaling is removed, MAE values are 0.2109375 (scaling factor used for computation is 128) and 0.05078125 (scaling factor used for computation is 256) for image 10 and 0.1640625 (scaling factor used for computation is 128) and 0.046875 (scaling factor used for computation is 256) for image 11. Along with data width, scaling factor which is used to retain the accuracy will also affect the performance. Larger scaling factor leads to more accurate results which in turn require larger data width [9]. This is shown in Fig. 12 and 13 for both the images shown in Fig.10 and 11 respectively. From the graphs the inference drawn is, as scaling factor increases, the errors will be reduced. The time taken for 16-bit data width is less compared to 32- bit data width processing which is shown in Fig. 14. Depending on the application, the selection of scaling factor and data-width has to be done.

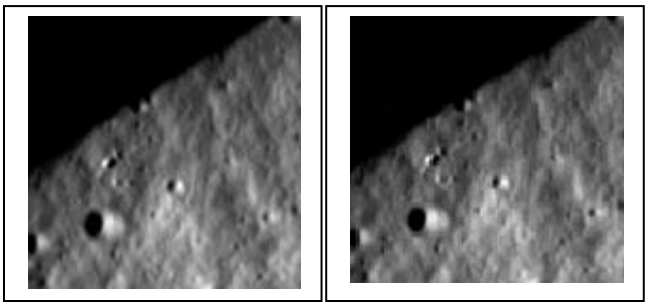


Fig. 10 (a) Input image 1 and (b) Reconstructed image 1

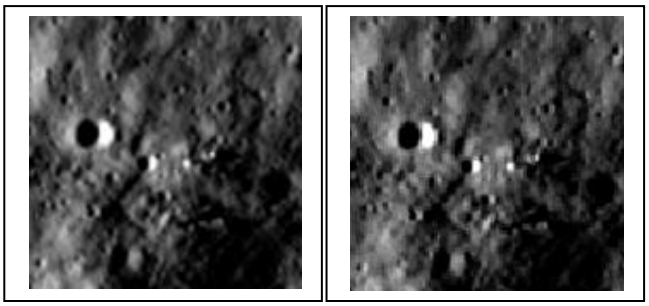


Fig. 11 (a) Input image 2 and (b) Reconstructed image 2

Table 1. Quantitative Analysis

	Scale factor	32-bit Data width (32-point)				16-bit Data width (32-point)			
		RMS	RRMS	MAE (Scaled version)	Computation time(µs)	RMS	RRMS	MAE (Scaled version)	Computation time(µs)
Image 10(a)	128	6.8645	0.6009	27	316	6.8997	0.6064	27	278
	256	3.5350	0.3107	13	316	NA	NA	NA	NA
Image 11(a)	128	6.6716	1.6879	21	316	6.8902	1.7601	21	278
	256	3.4127	0.8718	12	316	NA	NA	NA	NA

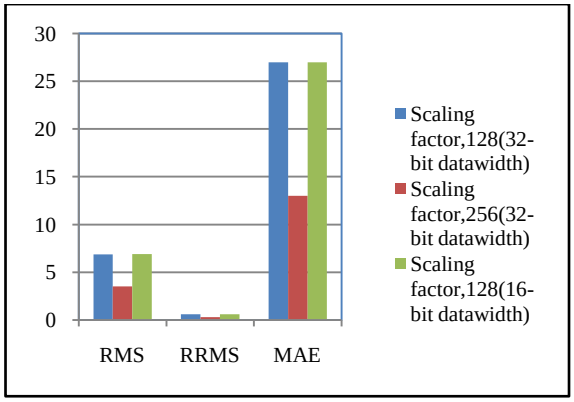


Fig. 12 Graph representing errors for images shown in Fig. 10

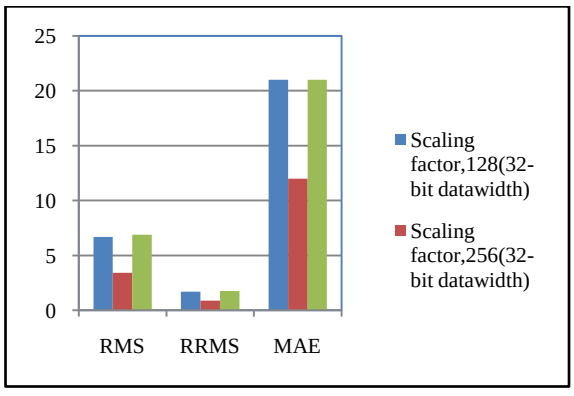


Fig. 13 Graph representing errors for images shown in Fig. 11

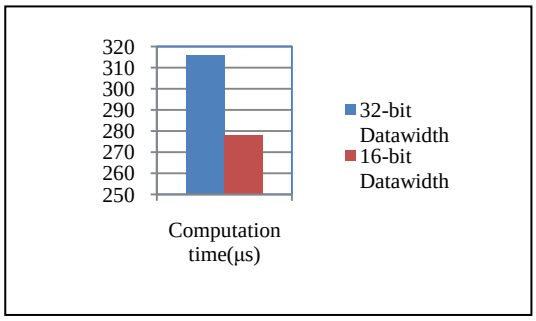


Fig. 14 Graph representing time taken for computation

6. CONCLUSION

This paper gives an elegant way of computing 2D-FFT and 2D-IFFT using IP core on FPGA. The FFT results obtained henceforth can be utilised for spectrum analysis. Image processing techniques like noise removal, filtering can be applied. Then image is obtained back by IFFT. 2D-FFT and 2D-IFFT computed using IP core is presented in simulation results and the effect of scaling factor and data width is analysed. 32-bit data width yields better results at the cost of computation time. The precision and accuracy of FFT and IFFT results can be improved by floating-point computation.

7. ACKNOWLEDGMENTS

Our thanks to the director Dr. G Nagendra Rao and group head Subhalakshmi Krishnamoorthy of LEOS (ISRO), Bengaluru and Dr. R. Kumaraswamy, head of the department of Electronics and Communication Engineering, Siddaganga Institute of Technology, Tumakuru.

8. REFERENCES

[1] S Sukhsawas, K Benkrid,"A High-level Implementation of a High Performance Pipeline FFT on Virtex-E FPGAs", Proceedings of the IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design (ISVLSI'04), 2004.

[2] Asmita Haveliya,"Design and Simulation of 32-Point FFT Using Radix-2 Algorithm for FPGA Implementation", Second International Conference on Advanced Computing & Communication Technologies, 2012.

[3] V. Arunachalam, Alex Noel Joseph Raj, "Efficient VLSI multiplexing applications", IET Circuits, Devices & Systems, 2014.

[4] Aleksei Kharin, Sergey Vityazev, Vladimir Vityazev and Naim Dahnoun, "Parallel FFT implementation on TMS320C66X Multicore DSP", IEEE proceedings of the 6th European Embedded design in education and research, 2014.

[5] Victor Montano and Manuel Jimenez, "Design and Implementation of a Scalable Floating-point FFT IP Core for Xilinx FPGAs", Page no. 533 - 536 ,Circuits and Systems (MWSCAS), 53rd IEEE International Midwest Symposium, 2010.

[6] Bhawesh Sahu , Anil Sahu, "FPGA Implementation of IP-core of FFT Block for DSP Applications", International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 10, December 2014.

[7] Ahmed Saeed, M. Elbably, G. Abdelfadeel, and M. I. Eladawy, "Efficient FPGA implementation of FFT/IFFT Processor", International Journal of Circuits, Systems and Signal processing, Issue 3, Volume 3, 2009.

[8] Aboelaze, M, "An FPGA based low power multiplier for FFT in OFDM systems using precomputations", ICT Convergence (ICTC) IEEE International Conference, 14-16 Oct. 2013.

[9] Mohammadnia, M.R, Shannon, L., "Minimizing the error: A study of the implementation of an Integer Split-Radix FFT on an FPGA for medical imaging", Page no. 360 – 367, Field-Programmable Technology (FPT), IEEE International Conference, 10-12 Dec. 2012.

[10] Mankar. A., Das.A.D., Prasad. N., "FPGA implementation of 16-point radix-4 complex FFT core using NEDA", Page no. 1-5, Engineering and Systems (SCES), IEEE Students Conference on 12-14 April 2013.

[11] Ren Chen, Prasanna, V.K., "Energy optimizations for FPGA-based 2-D FFT architecture", Page no. 1 – 6, High Performance Extreme Computing (HPEC) IEEE International Conference, 9-11 September 2014.

[12] Ranganathan, S., Krishnan, R., Sriharsha, H.S., "Efficient hardware implementation of scalable FFT using configurable Radix-4/2", Page no. 1-5, Devices, Circuits and Systems (ICDCS), 2nd International Conference, 6-8 March 2014.