Area and Frequency optimized 1024 point Radix-2 FFT Processor on FPGA

Vinay Kumar M, David Selvakumar A, Sobha P M,
{vinaym, david, sobhapm}@cdac.in,
Secure Hardware and VLSI Design group,
CDAC, Bangalore.

Abstract: This paper presents a Fast Fourier Transform (FFT) processor optimized for both 'area' and 'frequency'. The processor architecture is deeply pipelined Radix-2 butterfly unit, 1024 point, 64bit Fixed Point input with 32bit real and 32bit imaginary, Decimation In Time (DIT) FFT processor on Field Programmable Gate Array (FPGA). The proposed architecture is based on Dual RAM Ping-Pong Burst I/O with efficient addressing techniques which clocks at 385.804MHz on Xilinx Virtex-6 xc6vlx550t-2ff1759 taking 16.376µs to calculate one set of 1024 point FFT.

Key words— FFT processor, FPGA, 1024-point FFT, Radix-2 Butterfly, CORDIC, 64bit Fixed Point Arithmetic, Ping-Pong operation, Virtex-6, Verilog HDL.

I. INTRODUCTION

Discrete Fourier Transform (DFT) [1] is the basis to perform Fourier analysis in many practical applications. Computing the DFT of 'N' points in the naive way takes $O(N^2)$ arithmetical operations. FFT is an algorithm [2] to calculate DFT with reduced number of arithmetical operations from [1]$O(N^2)$ to [2]$N/2(\log_2^N)$ by taking full advantage of Symmetry and Periodicity of [3]Twiddle Factors. The FFT Decimation approach is generally attributed to J.W Cooley and J.W. Tukey [3] although there are many variations, their work was first published in 1965 and it is thought that Gauss described the technique as long ago as 1805.

Traditionally DSP algorithms are implemented using general purpose DSP programmable chips. Alternatively for high performance applications, special purpose fixed function DSP chipsets or FPGAs are used. FPGAs are mainly reconfigurable silicon chips. In another way they are array of programmable logic cells interconnected by a matrix of programmable connections. Taking advantage of hardware parallelism, FPGA exceeds the computing power of digital signal processing by breaking the paradigm of sequential execution and accomplishing more per clock cycle. Thus using massive parallelism benefit from FPGA, we can implement high performance computing applications such as FFT. FPGA offers low power consumption, design flexibility and adaptability with optimal device utilization while conserving both board space and system power, which is often not the case with DSP chips, even with CPU and GPU [4]. When a design demands the use of a DSP, or time-to-market is critical, or design adaptability is crucial, then the FPGA may offer a better solution.

[1]If N = 1024 input samples, a DFT requires approximately 10^6 operations

[2]Whereas FFT requires approximately 500 * $\log_2^{1024}$ or 5000 operations which is a 200 fold reduction

[3]Discussed in section ш

This paper presents a method which realizes the FFT operation based on the FPGA [5]. The FFT design coded in Verilog HDL [6], synthesized using Xilinx [7] ISE 14.2 and verified on Modelsim_DE_10.0a.

II. FFT

The FFT arithmetic [8] is basically divided into two types, which is the decimation-in-time (DIT) and the decimation-in-frequency (DIF). This radix-2-DIT FFT is adopted in this paper. An 'N' point discrete Fourier transformation (DFT) of the input sequences x(n) is written as,

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

Where k, n = 0, 1, 2…N-1

$$W_N = e^{(-j\,2\pi/N)}$$

x(n) could be further divided into odd part and even part using radix-2 DIT in (1), taking advantage of periodicity and symmetry we can obtain the following equations

$$X(k) = \underbrace{\sum_{n=0}^{N/2-1}(x(2n)W_N^{2nk})}_{\text{Even terms}} + \underbrace{\sum_{n=0}^{N/2-1}(x(2n+1)W_N^{(2n+1)k})}_{\text{Odd terms}}\ \forall N \in \mathbb{Z} \quad \textit{... (2)}$$

Using $W^2 = (e^{-j2\pi/N})^2 = (e^{-j2\pi/(\frac{N}{2})})$ in (2) we get

X(k) = {(DFT of even indexed N/2 sequences) + W^k(DFT of odd indexed N/2 sequences)}

$$X(k) = E(k) + W^k F(k) \quad \textit{... (3)}$$

Now for the values of k ≥ N/2, the equation ***(2)*** for X(k) can be simplified by replacing k → (k + N/2) we get

Since, $W^{N/2} = (e^{-j2\pi/N})^{N/2} = -1$ we have

$$X(k + N/2) = E(k) - W^k F(k) \quad \textit{... (4)}$$

This is where *(3) & (4)* the reduction in complexity comes about: one large computation is reduced to several sequential smaller computations which lead to the radix-2 butterfly as shown

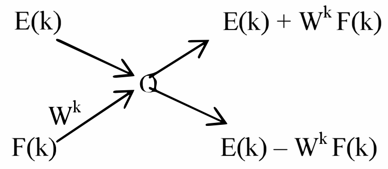

Fig.1 Basic Butterfly element

To summarize these steps for computing the DFT via decimation are

1. Shuffle input order in bit reversal form.
2. Compute N/2, two sample DFTs.
3. Compute N/4, four sample DFTs.
4. Continue until one, N-sample DFT is computed.

Fig.2 shows the 8 inputs decimation in time DFT operation diagram

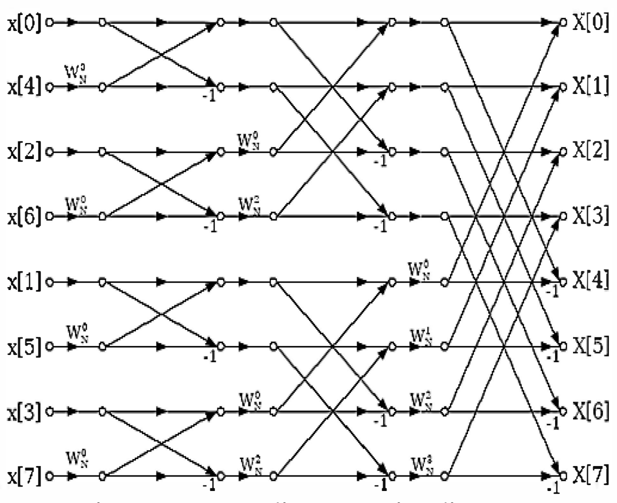

Fig.2 DIT-FFT radix-2 operation diagram [9]

W_N used in all the equations are also called as Twiddle Factors. Twiddle factor referred to the root of unity complex multiplicative constants in the butterfly operation. The realization of these twiddle factors can be done by using Coordinate Rotation for Digital Computers algorithm (CORDIC).

III. CORDIC

The CORDIC algorithm [10][11] is an ingenious method of computing various arithmetic functions using only straight forward binary operations with some pre computed values. The basis of this algorithm is the anticlockwise rotation of a point say 'p' through an angle θ as shown

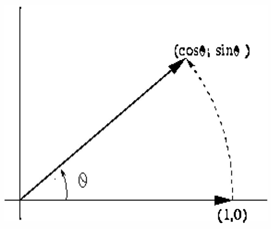

Fig.3 Rotation of a point with angle θ

Rotation matrix for computing the new point (x_{new} , y_{new}) from old point (x_{old} , y_{old}) is given by

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} \quad \text{... (5)}$$

The fundamental insight of the Cordic family of algorithms is that tan(∗) function can be expressed as 2 to the power –i i.e. 2^{-i} where $i \in \mathbb{Z}$.

Taking cosθ common from the above matrix *(5)* and rewriting in terms of tanθ i.e. 2^{-i}, we get

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = K_i \begin{bmatrix} 1 & -2^{-i} \\ 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{... (6)}$$

Where $K_i = \frac{1}{\sqrt{1+2^{-2i}}}$ or K = $\prod_{i=0}^{N-1} \cos\left(tan^{-1}(2^{-i})\right)$

N is the user dependent number of rotations

Or $x_{i+1} = K_i\,(x_i - y_i d_i 2^{-i})$ ... *(7)*

$y_{i+1} = K_i\,(y_i - x_i d_i 2^{-i})$ ... *(8)*

Here d_i is the sequence of directions defines the angle in a basis set defined by tan(2^{-i}). Alternatively d_i can be computed as rotations are performed using an additional accumulator.

$$Z_{i+1} = (Z_i - x_i d_i tan^{-1}(2^{-i})) \quad \text{... (9)}$$

d_i is +1 or -1 depending on the value of Z

If Z < 0 then d_i is +1 and if Z > 0 then d_i is -1

The CORDIC is mainly convergence algorithm depending on initial values here (1, 0), 'x' and 'y' values converges to the certain values for the given angle θ by converging phase difference to zero as shown in the Fig.4

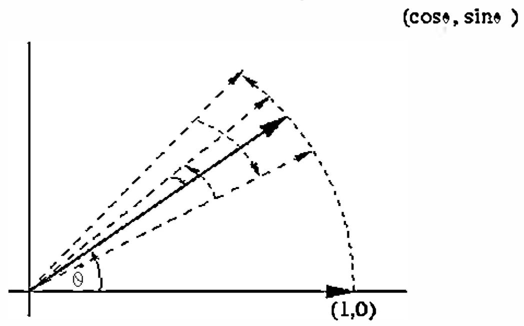

Fig.4 Convergence to a point

A small Look Up table (one entry per iteration) contains $tan^{-1}(2^{-i})$ values. The above equations and algorithm calculates sine and cosine values if the input angle present in first and fourth quadrants i.e. $0^{o} \leq \theta \leq 90^{o}$ and $0^{o} \leq \theta \leq -90^{o}$ and if the input angle is present in second and third quadrant i.e. $90^{o} < \theta < 270^{o}$, Coordinate rotation or pre-rotation should be done.

If $90^{o} < \theta < 180^{o}$ subtract pi from θ and update X to -Yin and Y to Xin. If $180^{o} < \theta < 270^{o}$ add pi to θ and update X to Yin and Y to -Xin to make the input angle in first or fourth quadrant and repeat the above algorithm of convergence.

The algorithm flow of our CORDIC design is as shown below

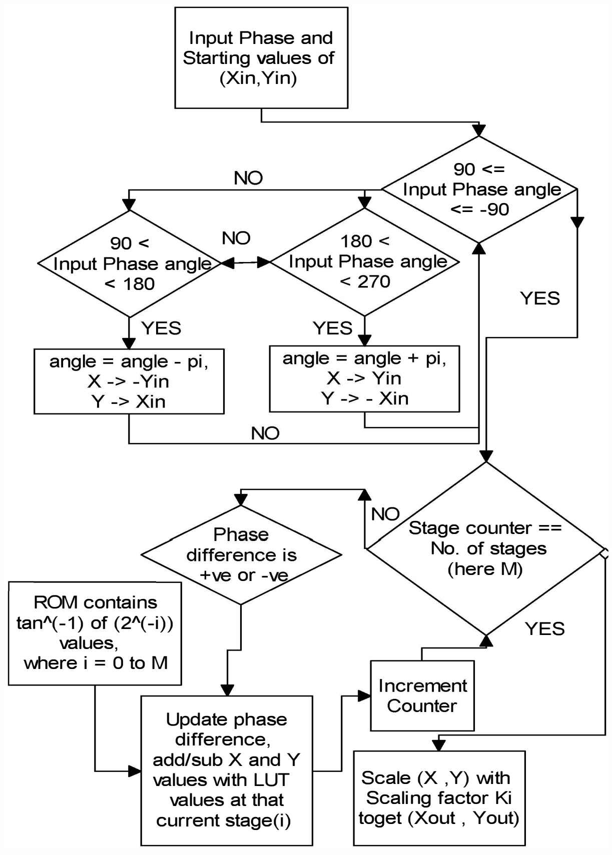
Input Phase and Starting values of (Xin,Yin)
90 <= Input Phase angle <= -90
NO
YES
90 < Input Phase angle < 180
NO
180 < Input Phase angle < 270
YES
YES
angle = angle - pi, X -> -Yin Y -> Xin
angle = angle + pi, X -> Yin Y -> - Xin
NO
Phase difference is +ve or -ve
NO
Stage counter == No. of stages (here M)
YES
ROM contains tan^(-1) of (2^(-i)) values, where i = 0 to M
Update phase difference, add/sub X and Y values with LUT values at that current stage(i)
Increment Counter
Scale (X ,Y) with Scaling factor Ki toget (Xout , Yout)

Fig.5 Flow chart of CORDIC algorithm

In our design we have taken M = 22 pipelined stages with 1 rotation in each stage. 'X', 'Y' values and phase difference are updated at each stage by adding or subtracting with the values present in the LUT. The final (X_{out}, Y_{out}) are scaled with K_i to get $(cos\theta, sin\theta)$ values.

CORDIC is not one algorithm but a family of algorithms for computing various trigonometrical and other functions.

The above CORDIC algorithm we implemented is to calculate cosine and sine values for the requirement of twiddle factors in FFT.

IV. Implementation of FFT

Efficient addressing is the key to the performance of any FFT architecture. In this DIT FFT architecture we efficiently did in place computation using dual port RAM's which best suits for radix-2 FFT. The proposed FFT system diagram is shown below

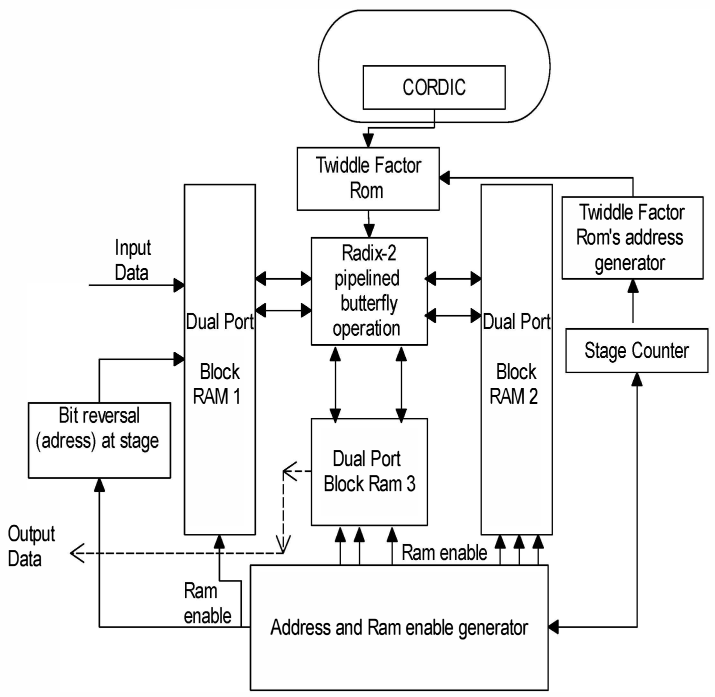

Fig.6 FFT system diagram

Fig.6 system diagram shows dual RAM ping pong architecture of an FFT where the data reading, processing and writing is done from one RAM to another at each stage and vice versa. We used an extra block RAM named 'Block RAM 3' as shown in Fig.6, so as to increase the throughput. Here user no need to wait until the first set data is read and he can read once the output is available while writing the second set of data.

We designed FFT architecture using with and without CORDIC block and verified both the results on FPGA. We observed that FFT [4]without CORDIC block for fixed N (here 1024) giving more precision in output compared to [5,6]with CORDIC. So, in our design we removed CORDIC block and we used pre computed twiddle factor values.

Radix-2 butterfly block as show in Fig.7 is 9 stages pipelined (6 clock cycles for multiplication, 2 clock cycles for addition and 1 clock cycle for signed shift operation) as it involves complex multiplications resulting bottle neck for maximum frequency of operation.

512 radix-2 butterflies operation takes 9 clock cycles for pipelining, 8 clock cycles for internal read and write delays and total of 10 such stages requires (5120+90+80) clock cycles,

[4]Twiddle factor values are pre computed using MATLAB
[5]Percentage error with Cordic is 3.21743
[6]CORDIC best suits when 'N' is variable

1024 clock cycles to load the input data which is nearly 6320 clock cycles for one complete FFT operation.

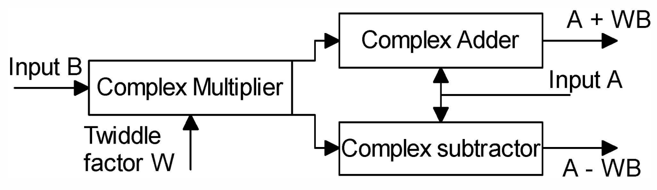

Fig.7 Radix-2 butterfly operation

As told in DFT summarizing steps, input should be stored in the reversal address of a one bit incremental counter.

Bit reversal block takes care of reversing input addresses.

index	address
000	000
100	001
010	010
110	011
001	100
101	101
011	110
111	111

Table.1 Three bits example of bit reversal operation

1. Address generation for Twiddle factor ROM

As mentioned, efficient addressing is important for high throughput and performance of this design. In our design we used only simple shift operations to generate addresses to both Twiddle factor ROM and in place computation Block RAM's.

The logic to generate twiddle factors addresses to ROM (addr_twid as shown in Fig.8) is as follows:

For [7]n = 2 the address at each clock are (0 at first stage), (0, 1 at second stage).

For n = 3 address are (1st stage: 0),(2nd stage: 0,2),(3rd stage: 0,1,2,3).

For n = 4 address are (1st stage:0),(2nd stage: 0,4),(3rd stage:0,2,4,6),(4th stage: 0,1,2,3,4,5,6,7)

And so on…

As shown in Fig.8 initial shifter value is loaded as N/2 (Number of twiddle factors for given 'N') with left shifted by one.

The one bit counter 'K' is designed such that the logic is as follows:

For n = 2 the counter values at each clock are (K = 0, 0, 0.. at first stage), (K= 0, 1, 0, 1, at second stage).

[7]If $N = 2^n$ then 'n' represents number of stages for given N

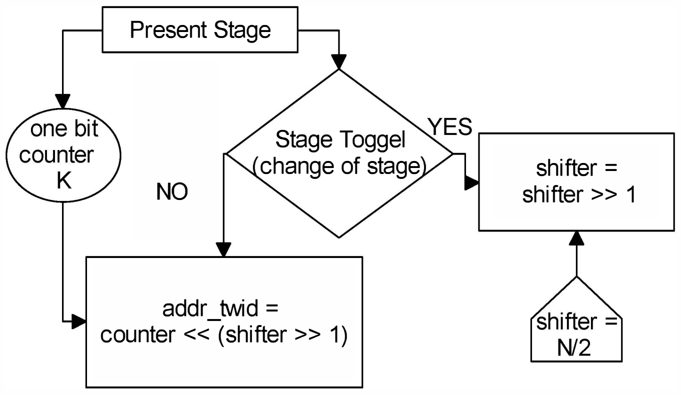

Fig.8 Logic flow for address generation to the twiddle factor ROM.

For n = 3 address are (1st stage: K= 0, 0, 0…), (2nd stage: K= 0, 1, 0, 1…), (3rd stage: K= 0, 1, 2, 3, 0, 1, 2, 3…).

For n = 4 address are (1st stage: K= 0, 0, 0…),(2nd stage: K= 0, 1, 0, 1…),(3rd stage: K= 0, 1, 2, 3, 0, 1, 2, 3…), (4th stage: K= 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7).

And so on...

At each stage toggle (i.e. is change of stage) shifter shifts its value left by 1(or divided by 2). The shifter values for N = 16 are as follows:

At stage 1: shifter value = 8.
At stage 2: shifter value = 4.
At stage 3: shifter value = 2.
At stage 4: shifter value = 1.

2. Address generation for Block RAM's

In radix-2 FFT, if the first input address (for the value of input A as in Fig.7) to the Block RAM 1 is 'M' (Address A)then the second input address (for the value of input B as in Fig.7) can be calculated as '$M + 2^n/2$' (Address B) where 'n' here is present stage.

At stage n = 1: if M = (0, 2, 4, 6…) then $(M + 2^n/2)$ = (1, 3, 5, 7…)
At stage n = 2: if M = (0, 1, 4, 5…) then $(M + 2^n/2)$ = (2, 3, 6, 7…)
At stage n = 3: if M = (0, 1, 2, 3…) then $(M + 2^n/2)$ = (4, 5, 6, 7…)
And so on…

The logic flow to generate value of 'M' is shown in Fig.9, 'tc' in Fig.9 is the terminal counter which resets to '1' if its value equals to the value of offset.

It adds offset address value to the present address and resumes the normal counter operation for address and terminal counter.

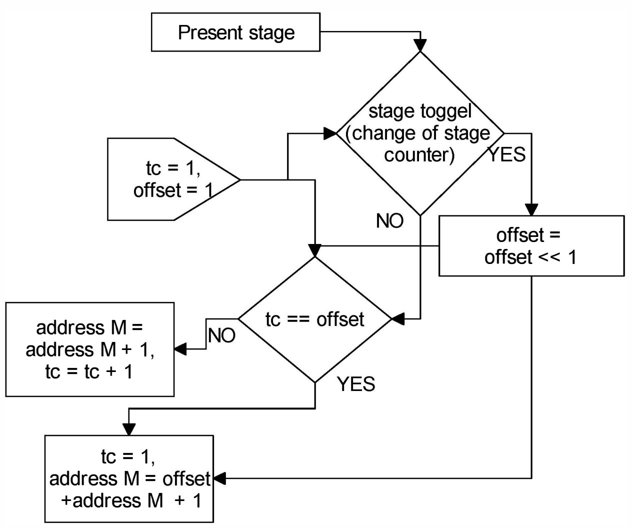
Present stage
stage toggel (change of stage counter)
YES
NO
tc = 1, offset = 1
offset = offset << 1
tc == offset
NO
YES
address M = address M + 1, tc = tc + 1
tc = 1, address M = offset +address M + 1

Fig.9 Logic flow for address generation to the Block RAM's.

3. *Fixed Point Arithmetic*

The name itself suggests that decimal point is fixed. To represent a real number we can use Fixed Point or Floating Point arithmetic in scientific computations.

In the current design we used Fixed Point arithmetic in which the number of binary bits before and after a decimal point is fixed.

We tested by converting input real values into fixed point values in Q.Q format (16.16) i.e. for 32 bit data, first bit represents sign, followed by next 15 bits represents Integer part and next 16 bits represents fractional part.

E.g. if Q = 4 i.e. a real number 2.5 and -2.5 can be represented in 4.4 format is given by 0010.1000 and 1101.1000 (2's complement form).

V. Simulation and Synthesis results

1. *Simulation results:*

The number of clock cycles (clock latency) to complete one set of 1024 point FFT (output is available to read at the output RAM or block RAM 3 in Fig.6) is about 6320.

The time taken by the FFT processor to complete one set of 1024 point FFT is 16.376 micro seconds as shown in Fig.11.

2. *Synthesis results:*

We implemented this FFT architecture on Pico M503 board with Xilinx xc6vlx550t-2ff1759 virtex-6 FPGA. We designed Pico interface for both single core and multicore ([8]3 cores) FFT working at [9]360 MHz and [9]340 MHz respectively and the results are verified with MATLAB. The values of both the output for the given input within boundary values, the [10]percentage error is minimized and in the order of 0.314%.

3. *Device Utilization Summary:*

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	2,633	687,360	1%
Number of Slice LUT'S	1,883	343,680	1%
Logic distribution			
Number of occupied Slices	722	85920	1%
Number of RAMB36E1	8	632	1%
Number of DSP48E1's	17	864	1%

Table.2 Device Utilization Summary

VI. FFT I/O port description

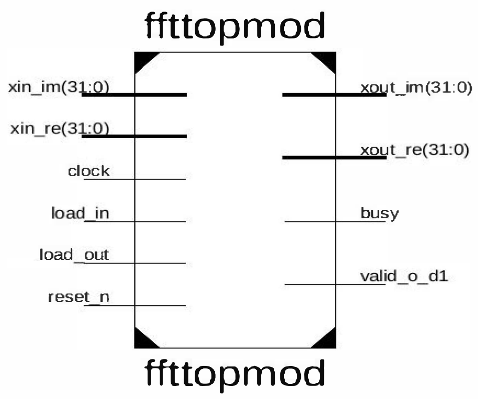

Fig.10 FFT Black Box

- xin_im and xin_re are the imaginary and real part of the input data with bit width of 32 each.
- clock is input clocking signal.
- load_in is single bit input signal, upon active high represents input data is valid.
- load_out is single bit input signal, upon active high represents user requesting the output data from the FFT.

[8]Three 1024 point FFT Cores are sufficient for continuous inflow of input data as the first core is free while writing into the third core

[9]Frequency of operation is with interface

[10]Percentage error calculated using formula $\sum_{i=1}^{1024}\left(\frac{A(i)-B(i)}{B(i)}\right)$

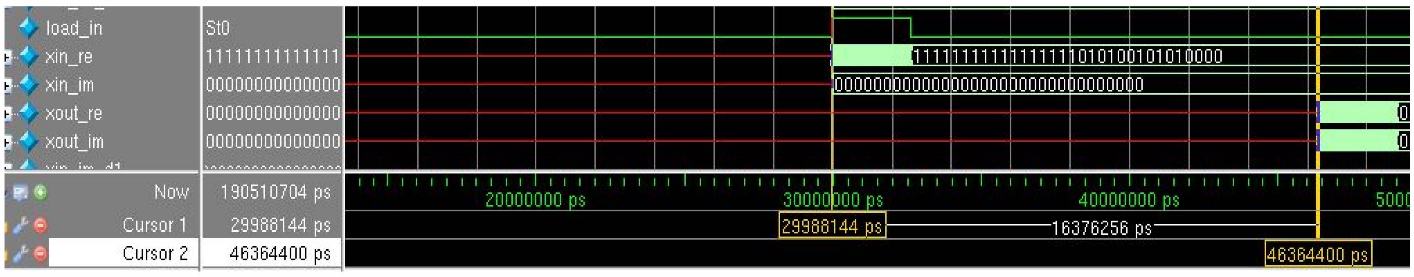

Fig.11 Simulation results showing input and output with time taken to calculate one set of 1024 point FFT.

- reset_n is single bit active low input signal which resets all internal operations.
- xout_im and xout_re are the imaginary and real part of the output data with bit width of 32 each.
- busy is single bit output signal, upon high represents that internal FFT calculation is going on.
- valid_o_d1 is single bit output signal, upon high represents that output data is valid.

VII. Comparison with previous works

	Xilinx[7]	[11]**This work**
Algorithm	Radix-2	Radix-2
Chip Type	Virtex-6 xc6vlx75t	Virtex-6 xc6vlx550t
Number of Points	1024	1024
Operating frequency	395 MHz	385 MHz
Execution Time	23.87µs	**16.376µs**
Number of Slices	2028	2633

Table.3 Comparison with previous Works

VIII. Conclusion

In this paper we presented 1024 point radix-2 DIT FFT processor with efficient addressing logic using well known Ping-Pong Burst I/O architecture which supports 32 bit real and 32 bit imaginary values. We Implemented FFT with and without CORDIC algorithm in which we found higher precision by using pre computed twiddle factor values. By efficient addressing using only simple shift operations, we reduced number of clocks for reading and processing of address logic, achieved area and frequency optimized FFT core with minimum number of clock latency and maximum clock frequency (385 MHz). Results of the FFT are compared with MATLAB FFT operator and calculated percentage error.

[11]Although there are previous works using Ping-Pong radix-2 FFT, but implemented on different virtex architectures.

Acknowledgment

The research work presented here was supported by Centre for Development of Advanced Computing (CDAC), Bengaluru. The authors would like to thank Dr. N. Sarat Chandra Babu for giving the continuous support for the research work.

References

[1] A.V Oppenheim, R.W. Schaefer, and J.R. Buck, “Discrete-time Signal Processing”. Prentice-Hall, Englewood Cliffs, NJ, 1998.

[2] Anders E. Zonst, “Understanding the FFT, A tutorial on the algorithm and software for laymen,” Citrus press.

[3] J. Cooley and J. Tukey, “An algorithm for the machine calculation of complex fourier series,” Mathematics of Computation, vol. 19, no. 90, pp. 297–301, 1965.

[4] Srinidhi Kestury, John D. Davisz, and Oliver Williamsz “BLAS Comparison on FPGA, CPU and GPU”, Proceedings 2010 IEEE Annual Symposium on VLSI, pp288-293.

[5] Uwe Meyer-Baese, “Digital Signal Processing with Field Programmable Gate Arrays,” Springer International Publication.

[6] Weng Fook Lee, “Verilog Coding for Logic Synthesis,” A John Wiley & Sons, Inc., Publications.

[7] Xilinx: Fast Fourier Transform logic IP core v7.1 Product specifications DS260 (1st March 2011).

[8] Charles Van Loan, “Computational Frame works for the Fast Fourier Transform,” SIAM series on Frontiers in Applied Mathematics.

[9] Zahra Haddad Derafshi, Javad Frounchi, Hamed Taghipour “FPGA Implementation of a 1024-Point Complex FFT Processor” Proceeding 2010 IEEE Second International Conference on Computer and Network Technology pp312-315.

[10] J.E.Volder, “The CORDIC trigonometric comp. technique” IRE Trans. Elec. Comp. (1959), vol. EC-8-3, pp. 330-334.

[11] F.Angarita, A Perez-Pascual, T.Sansaloni, J.Valls, “Efficient Fpga Implementation of Cordic Algorithm for Circular and Linear Coordinates”, Proceedings 2005 International Conference on Field Programmable Logic and Applications, pp535-538.