International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-3 Number-4 Issue-13 December-2013

Low Power Implementation of Fast Fourier Transform Processor on FPGA

Shashank Gupta

Abstract

DFT(Discrete Fourier Transform) is a fundamental principle of DSP whose applications vary from Spectral analysis, Data compression, solving Partial Differential Equations, convolution and multiplication of large numbers. Despite its enormous potential in theoretically solving many DSP problems, it is of very little use in practical because of its extremely expensive hardware implementation. It is due to its complexity $O(N^2)$, N being number of data points. To address this problem Fast Fourier Transform (FFT) was introduced. This algorithm uses the symmetry and periodicity properties of Twiddle Factor involved with DFT to reduce the number of calculations drastically. For N=1024, FFT is more than 200 times faster than DFT. In this paper we focus on implementing FFT for a processor, by applying Cooley-Tukey Algorithm to improve the speed of computation at expense of minimum power. This paper discusses in detail about the core FFT block and auxiliary blocks of Testbench like Buffer Ram, Complex Multiplier and Bit Shifter. The simulation has been done in Xilinx ISE with verification on two different FPGA platforms. The correctness of our algorithm is demonstrated via output waveforms

Keywords

Fast Fourier Transform, Butterfly Element, Complex Multiplier, Radix-2 Algorithm.

1. Introduction

Discrete Fourier Transform is extremely important in the area of frequency (spectrum) analysis because it transforms a discrete signal in time domain to its discrete frequency domain representation. It decomposes a sampled signal in terms of sinusoidal (complex exponential) components. This discrete-time to discrete-frequency transformation is essential or we wouldn't be able to compute Fourier transform with a microprocessor or DSP based system.

Shashank Gupta, Department of Electronics Engineering, Indian Institute of Technology, Banaras Hindu University, Varanasi, India.

The properties - symmetry and periodicity of the DFT are exploited to significantly reduce its computational requirements. The resulting algorithm is named as Fast Fourier Transforms (FFTs). It is the speed and discrete nature of the FFT that allows us to analyze a signal's spectrum with Matlab or in real-time on the SR770 spectrum analyzer. [1]

The 'Radix 2' algorithms is particularly useful if N is a power of 2 ($N=2^p$). If we assume that algorithmic complexity provides a direct measure of execution time and that the relevant logarithm base is 2, then, ratio of execution times for the DFT (complexity $O(N^2)$) vs. Radix 2 FFT (O(N log N))increases tremendously with increase in N. FFT relies on the recursive decomposition of an N point transform into 2 (N/2) point transforms. The above process of decomposition can be applied to any composite (non prime) N. The method is particularly simple if N is divisible by 2 and if N is a regular power of 2, the decomposition can be applied repeatedly until the trivial '1 point' transform is reached. A sequence x(n) of 256 complex-valued numbers will be computed by 256-point DFT and gives another seq. of data X(k) of length 256 by following rule:

$$X(k) = \Sigma_{0 \le n \le 255}\ x(n)\, e^{-j2\pi nk/256} \ ; k = 0 \text{ to } 255 \quad (1)[3]$$

To simplify the notation, the complex-valued phase factor $e^{-j2\pi nk/256}$ is usually defined as $W_{256}^{\ n}$, where

$$W_{256} = \cos(2\pi/256) - j \sin(2\pi/256) \quad (2)[3]$$

We see FFT algorithms take advantage of the symmetry and periodicity properties of $W_{256}^{\ n}$ to greatly reduce the number of calculations that the DFT requires. A FFT implementation has the real and imaginary components of $W_N^{\ n}$ which are called twiddle factors. In the processor for FFT256 a radix-16 FFT algorithm is used. It divides DFT into two smaller DFTs of the length 16, as it is shown in the formula:

$$X(k) = X(16r+s) = \Sigma_{0 \le m \le 15}\, W_{16}^{\ mr}\, W_{256}^{\ ms}\, \Sigma_{0 \le m \le 15}\, x(16l+m)\, W_{16}^{\ sl},$$
$$r = 0 \text{ to } 15,\ s = 0 \text{ to } 15 \quad (3)[4]$$

which shows that 256-point DFT is divided into two smaller 16-point DFTs. This algorithm is illustrated

by the graph which is shown in the Fig.1. The input complex data x(n) are represented by the 2-D array of data x(16l+m). The columns of this array are computed by 16-point DFTs. The results of them are multiplied by the twiddle factors W_{256}^{ms} and the resulting array of data X(16r+s) is derived by 16-point DFTs of rows of the intermediate result array.The 16-point DFT, named as the base FFT operation, is implemented by the Winograd small point FFT algorithm, which provides the minimum additions and multiplications (only 10 complex multiplications to the factor W_{16}^{sl}). As a result, the radix-16 FFT algorithm needs only 256 complex multiplications to the twiddle factors W_{256}^{ms} and a set of multiplications to the twiddle factors W_{16}^{sl} instead of 65536 complex multiplications in the original DFT. So 896 complex multiplications will be required for the well known radix-2 256-point FFT algorithm. [4]

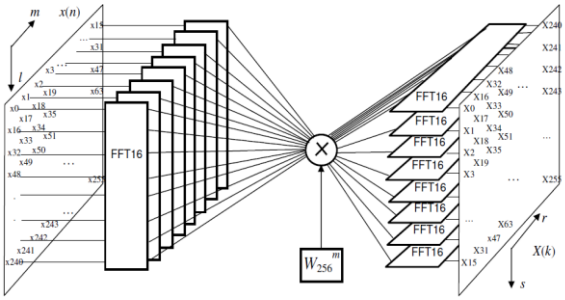

Figure 1: Graph of the FFT 256 algorithm

2. Salient Features of Processor

· 256 -point radix-8 forward and Inverse FFT.
· Pipelined mode operation, each result is outputted in one clock cycle with the latent delay from input to output being equal to 580 clock cycles where simultaneous loading/ downloading are supported.
· Input data, output data, and coefficient widths are parameterisable in range 8 to 16 and more.
· 2 and 3 data buffers have been used.
· FFT for 10 bit data and coefficient width is calculated on Xilinx XC4SX25-12 FPGA at 250MHz clock cycle, and on Xilinx XC5SX25-12 FPGA at 300 MHz clock cycle, respectively. [5]
· FFT unit for 10 bit data and coefficients, and 2 data buffers occupies 1652 CLB slices, 4DSP48 blocks, and 2.5 kb of RAM in Xilinx XC4SX25 FPGA, and 670 CLB slices 4 DSP48E blocks, and 2.5 kb of RAM in Xilinx XC5SX25 FPGA, data buffers are implemented on the distributed RAM.

Excessive pipelined calculations: A datapath called FFT16 computes every base FFT operation. This datapath will calculate the 16-point DFT in a pipelined manner. Therefore in each clock cycle one complex number is read from the input data buffer RAM and the complex result is written in the output buffer RAM. The 16-point DFT algorithm is divided into several stages which are implemented in the stages of the FFT16 pipeline. An increased clock frequency up to 200 MHz and higher would be supported. The latent delay of the FFT16 unit from input of the first data to output of the first result is equal to 30 clock cycles.[7]

Computations of high precision: The result truncation after multiplication to the factors W_{256}^{ms} serves as the main error source in computation. Since most of the base FFT calculations are additions, they can be calculated without errors. The FFT results have the data bit width which is higher in 4 digits than the input data bit width. This provides the high data range of results when the input data is the sinusoidal signal. The maximum result error is less than the 1 least significant bit of the input data. To provide the proper bandwidth of the resulting data, the normalizing shifters have been attached to the outputs of FFT16 pipelines. The overflow detector outputs provide the opportunity to assign the proper shift left bit number for these shifters.[6]

Low hardware volume: The FFT256 processor has the minimum multiplier number equal to 4. This fact makes this core attractive to implement in ASIC. When configuring in Xilinx FPGA, these multipliers are implemented in 4 DSP48 units respectively. We can select the input data, output data, and coefficient widths which provide application dynamic range needs. This can minimize both logic hardware and memory volume.[7]

3. Radix–2 Algorithm (Cooley-Tukey Algorithm) & The Butterfly element

The radix-2 decimation-in-frequency (DIF) FFT is an important algorithm obtained by the divide & conquer approach. Fig.2 shows the first stage of the 8-point DIF algorithm. This whole process involves $\log_2 N$ stages of decimation and each stage involves N/2 butterflies of the type shown in Fig. 3. Consequently, $(N/2)\log_2 N$ complex multiplications shall be required for the N-point DFT compute via this algorithm. We observe that the output sequence occurs in bit-

reversed order with respect to the input. Furthermore, if we are not rigid in the requirement of having computations occurrence in place, it is also possible to have both the input and output in normal order.[3]

Finally the algorithm is modified for co-efficient reordering, depending on the relative Hamming distances between them. This reordering minimizes switching activity and hence lowers the power consumption overall.[2]

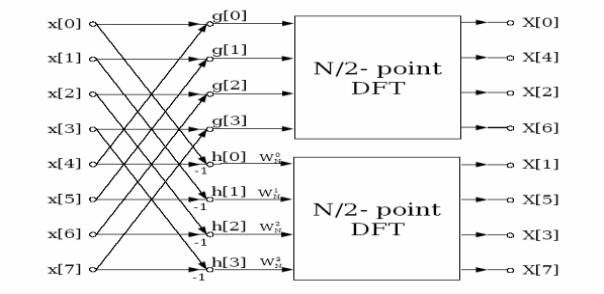

Figure 2: I[st] stage of the 8-point DIF algorithm

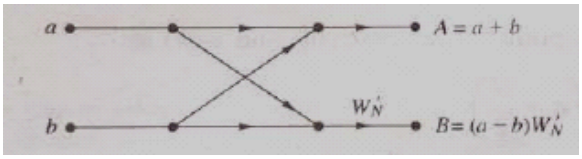

Figure 3: Butterfly element of Processor

4. Processor Architecture

There are 2 ways to implement the stages say for 8-point FFT. Two extreme cases are presented:

A) REUSE SINGLE Way – Involves only one stage iteratively, once for every decimation. This circuit is hardware efficient as there is need of only one set of 12-bit adders and subtractors. The first stage requires only 2 CORDICs(COordinate Rotation DIgital Computer)[8]. Each CORDIC computation takes 8 clock pulses. The second and third stages do not require any CORDIC, although in this structure using CORDIC they will need to rotate data by 0^0 or -90^0, which will take 16 (8 for the second and 8 for third stage) clock pulses. The entire process of rotation by 0^o or -90^o can rather be easily achieved by 2's complement and BUS exchange which would require much less hardware. Besides, while one set of data is being computed, we are left with no option but to wait for it to get completely processed for 36 clock cycles before inputting the next dataset. So, the total computation time spent will be= 24 clock cycles with sixteen 12 bit adders and subtractors needed.

B) FULLY SPREAD Way – Involves three separate stages, one each for every decimation. This other extreme needs 3 sets of sixteen, 12-bit adders. The complexity of implementation would definitely be reduced and delay would drastically cut down as each stage would be separated from the other by a bank of registers. This will be further aided by one data set serially streamed into the input registers after 8 clock pulses of the previous set. The net effect is that at a time we can have 3 stages working simultaneously.

However, this architecture is ***not*** taken into consideration as a valid option simply because of the immense hardware required. Besides, it would give improvement of merely 1 clock cycle over the architecture discussed below which we have used in terms of the total time taken. Thus, Time Taken for computation = 8 clock cycles No. of 12 bit adders and subtractions = 40 A Comparison has been made in Fig. 4.

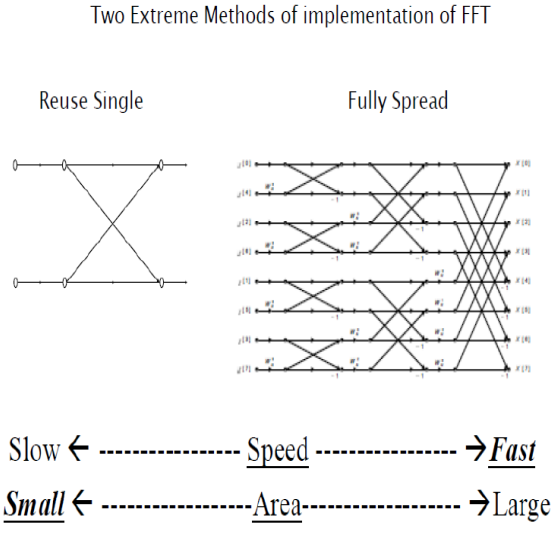

Figure 4: Analysis of 2 extreme implementation methods

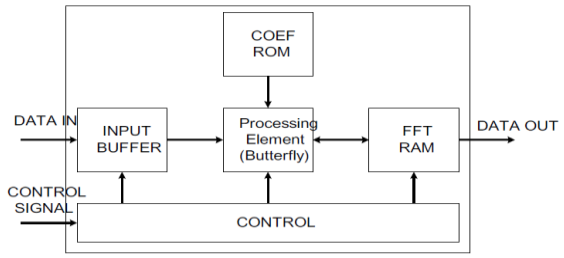

Figure 5: Block Diagram for the Processor

5. FFT Code and implementation

The datapath FFT16 implements the 16-point FFT algorithm in the pipelined mode. For 46 clock cycles, 16 input complex data are calculated but each new 16 complex results are outputted every16 clock cycles. Calculations by our Algorithm are: [4]

$t1:=x(0) + x(8);$ $m4:=x(0) – x(8);$
$t2:=x(4) + x(12);$ $m12:= – j*(x(4)–x(12));$
$t3:=x(2) + x(10);$ $t4:=x(2) – x(10);$
$t5:=x(6) + x(14);$ $t6:=x(6) – x(14);$
$t7:=x(1) + x(9);$ $t8:=x(1) – x(9);$
$t9:=x(3) + x(11);$ $t10:=x(3) – x(11);$
$t11:=x(5) + x(13);$ $t12:=x(5) – x(13);$
$t13:=x(7) + x(15);$ $t14:=x(7) – x(15);$
$t15:=t1 + t2;$ $m3:= t1 – t2;$
$t16:=t3 + t5;$ $m11:= –j*(t3 – t5);$
$t17:=t15 + t16;$ $m2:= t15 – t16;$
$t18:=t7 + t11;$ $t19:= t7 – t11;$
$t20:=t9 + t13;$ $t21:= t9 – t13;$
$t22:=t18 + t20;$ $m10:= – j*(t18 – t20);$
$t23:=t8 + t14;$ $t24:= t8 – t14;$
$t25:=t12 + t10;$ $t26:= t12 – t10;$
$m0:=t17 + t22;$ $m1:=t17 – t22;$
$m13:= –j*$ sin(p/4)*($t19 + t21$); $m5:=$ cos(p/4)*($t19 – t21$);
$m6:=$ cos(p/4)*($t4 – t6$); $m14:=–j*$ sin(p/4)*($t4 + t6$);
$m7:=$ cos(3p/8)*($m24+m26$); $m15:= –j*$ sin(3p/8)*($t23 + t25$);
$m8:=$ (cos(p/8) + cos(3p/8))*$t24$; $m16:= –j*$ (sin(p/8) – sin(3p/8))*$t23$;
$m9:=–$ (cos(p/8) - cos(3p/8))*$t26$; $m17:= –j*$(sin(p/8) + sin(3p/8))*$t25$;
$s7:= m8 – m7;$ $s15:= m15 – m16;$
$s8:= m9 – m7;$ $s16:= m15 – m17;$
$s1:=m3 + m5;$ $s2:=m3 – m5;$
$s3:=m13 + m11;$ $s4:=m13 – m11;$
$s5:=m4 + m6;$ $s6:=m4 – m6;$
$s9:=s5 + s7;$ $s10:=s5 – s7;$
$s11:=s6 + s8;$ $s12:=s6 – s8;$
$s13:=m12 + m14;$ $s14:=m12 – m14;$
$s17:=s13 + s15;$ $s18:=s13 – s15;$
$s19:=s14 + s16;$ $s20:=s14 – s16;$
$y(0):=m0;$ $y(8):=m1;$
$y(1):=s9 + s17;$ $y(15):=s9 – s17;$
$y(2):=s1 + s3;$ $y(14):=s1 – s3;$
$y(3):=s12 – s20;$ $y(13):=s12 + s20;$
$y(4):=m2 + m10;$ $y(12):=m2 – m10;$
$y(5):=s11 + s19;$ $y(11):=s11 – s19;$
$y(6):=s2 + s4;$ $y(10):=s2 – s4;$
$y(7):=s10 – s18;$ $y(9):=s10 + s18;$

Where x and y are input and output arrays of the complex data, t1,…,t26, m1,…, m17;s1,…,s20 are the intermediate complex results. We see the algorithm contains only 20 real multiplications to the untrivial coefficients:
$\sin(\pi/4) = 0.7071;$ $\sin(3\pi/8) = 0.9239;$
$\cos(3\pi/8) = 0.3827;$ (cos(p/8) + cos(3p/8)) =1.3066;
(sin(p/8) – sin(3p/8)) = 0.5412;
and 156 real additions and subtractions.

Working clock cycles from 0 to 15 are counted bya counter named “ct”. So a single inferred adder adds x(0) + x(8) in one cycle, x(1) + x(9) in the next cycle, D(1) + D(5) in another cycle and so on, and x(7) + x(15) in the final cycle of the sequence of cycles deriving the results t1,t7,t9,…,t13 respectively. Four constant multipliers are used to derive the multiplication to 5 different coefficients. Implementation of the multiplication to the coefficient 0.7071is done in the pipelined manner. The multipliers use the adder tree, which adds the multiplicand shifted to different bit numbers. For example, for short input bit width the coefficient 0.7071 is approximated as 0.101101012, for long input bit width it is approximated as 0.101101010000001012. The long coefficient bit width is set by parameter FT256bitwidth_coef_high. The first kind of the constant multiplier occupies 3 adders, and the second one occupies 4 adders. The importance of the long coefficient selection is seen from the following fact. When the input bit width is 16 and higher, the selection of the long coefficient bit width decreases the FFT256 result error in two times. The FFT16 unit implements both FFT and inverse FFT depending on the parameter FFT256paramifft. Practically the inverse FFT is implemented on the base of the direct FFT by the inversion of operations in the final stage of computations for all the results except $y(0)$, $y(8)$. For example, $y(1):=s9 + s17;$ is substituted to $y(1):=s9 – s17;$

The FFT16 unit starts its operation by the START impulse. The first result is preceded by the RDY impulse which is delayed from the START impulse to 30 clock impulses. The output results have the bit width which is in 4 higher than the input data bit width. That means that all the calculations except multiplication by coefficients like 0.7071 are implemented without truncations, so the FFT256 results have the minimized errors comparing to other FFT processors.

6. Symbols and Signals Description

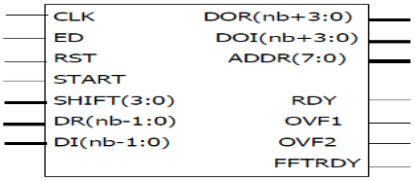

Figure 6: FFT256 core Symbol

Table 1: FFT256 core signal description

SIGNAL	TYPE	DESCRIPTION
CLK	input	Global clock
RST	input	Global reset
READY	input	FFT Start
ED	input	Input data and operation enable strobe
DR [nb-1:0]	input	Input data real sample
DI [nb-1:0]	input	Input data imaginary sample
SHIFT	input	Shift left code
RDY	output	Result ready strobe
WERES	output	Result write enable strobe
FFTRDY	output	Input data accepting ready strobe
ADDR [7:0]	output	Result number or address
DOR [nb+3:0]	output	Output data real sample
DOI [nb+3:0]	output	Output data imaginary sample
OVF1	output	Overflow flag
OVF2	output	Overflow flag

nb and *nb*+4 bit twos complement complex integers respectively denote Input and output data The twiddle coefficients are numbers of *nw* – bit width. Where $nb \leq 16$, $nw \leq 16$.

The nature of application where it is used drives the core interconnection. The calculation of the unlimited data stream which are inputted in each clock cycle is considered. In Fig.7 this interconnection is shown.
In this case data source is DATA_SRC and the analog-to-digital converter, FFT256 is the core, which is customized as one with 3 inner data buffers.

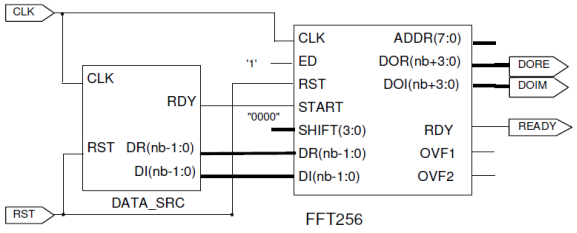

Figure 7: Simple Core interconnection

The impulse START initiates the FFT algorithm. After the READY impulse the corresponding results are generated which is followed by the address code ADDR. For the global synchronization, START is essential and could be generated once before the system operation. The input data have the natural order, and can be numbered from 0 to 63. When 3 inner data buffers are configured then the output data have the natural order. When 2 inner data buffers are configured then the output data have the 8-th inverse order, i.e. here the order will be 0,8,16,...56,1,9,17,...

7. Waveforms – Input & Output of FFT256 core

After the falling edge of the START signal the input data array starts to be taken as input. The data samples are latched by the each rising edge of the clock signal CLK when the enable signal ED is active high. When all the 256 data are inputted and the START signal is low then the data samples of the next input array start to be inputted.

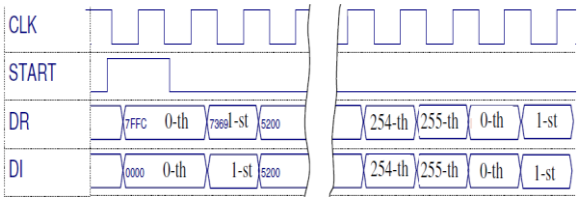

Figure 8: Input Waveform

The throughput of the processor is slowed down when ED signal is controlled. The input waveforms are shown in fig.9 when the ED signal is the alternating signal with the 2 times less than one of clock signal frequency. The sampling of input data is done when ED is high and the clock signal is rising.

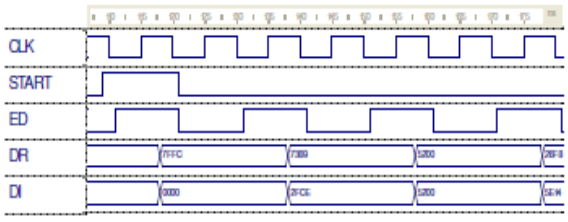

Figure 9: Input Waveform when the throughput is slowed down in 2 times

Also, after the RDY signal the result samples start to be generated. They are followed by the result number

which is given by the signal ADDR (Fig.10). When the START signal is not active for the long time period then just after output of the 255-th couple of results the 0-th couple of results for the next data array is outputted

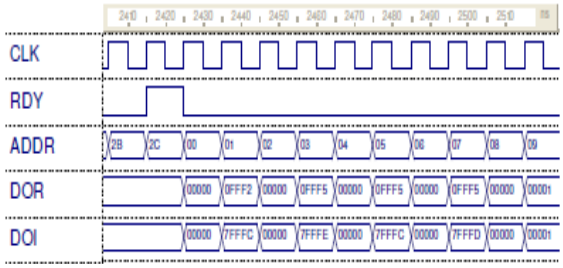

Figure 10: Output Waveform

The latent delay of the FFT256 processor core is estimated by ED = 1 as the delay between impulses START and RDY, and it is equal to 839 clock cycles when 3 buffer units are used and to 580 clock cycles when 2 buffer units are instantiated. When the throughput is slowed down by the signal ED controlling then the result output is slowed down respectively, as it is shown in Fig.11.

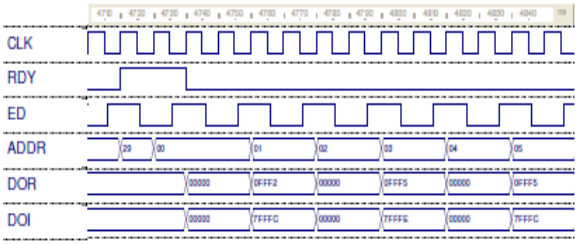

Figure 11: Output Waveform when the throughput is slowed down in 2 times

8. Other components of Testbench - Data Buffer, Bit Shifter & Complex Multiplier

BUFRAM256: BUFRAM256 is the data buffer, which consists of the two port synchronous RAM of the volume 512 complex data, and the write-read address counter. The real and imaginary parts of the data are stored in the natural ascending order as in Fig. 12 By the START impulse the address counter is reset and then starts to count (signal addrw). The input data DR and DI are stored to the respective address place by the rising edge of the clock signal.

After writing 256 data beginning at the START signal, the unit outputs the ready signal RDY and starts to write the next 256 data to the second half of the memory. At this time point, it outputs the data stored in the memory's first half. The reading of the next array starts as soon as this data reading is finished. This process continues till the next RST or START signal are entered. The reading address sequence is 16-th inverse order, i.e. the order is 0,16,32,...240,1,17,33,... . Really the reading address is derived from the writing address by swapping 4 LSB and 4 MSB address bits. The reading waveforms are illustrated by the Fig.13. Implementation of BUFRAM256 unit is possible in 2 ways. The first way consists in use of the usual one-port synchronous RAMs. Then BUFRAM256 consists of 2 parts, firstly one data array is stored into one part of the buffer, and another data array is read from the 2nd part of the buffer, then these parts are substituted by each other. The second way consists in use of the usual 2-port synchronous RAM with a single clock input. Such a RAM is usually instantiated as the Block RAM or the dual ported Distributed RAM in the Xilinx FPGAs.

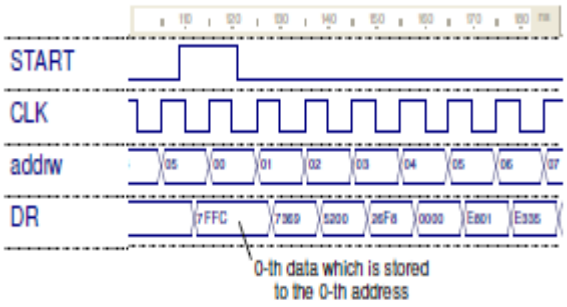

Figure 12: Waveforms of data writing to BUFRAM256

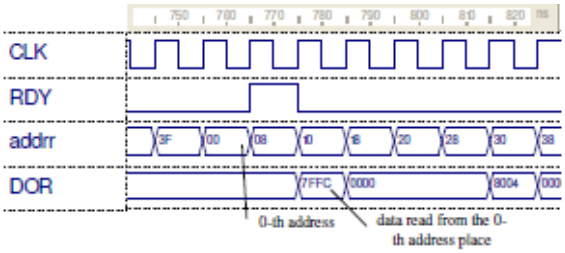

Figure 13: Waveforms of data reading to BUFRAM256

Cnorm – shifter to 0,1,2,3 bit left shift: During computations in FFT16 the data magnitude increases up to 16 times, and the FFT256 result can increase up

to 256 times depending on the spectrum properties of the input signal. Therefore, to prevent the signal dynamic bandwidth loose, the output signal bit width must be at least in 8 bits higher than the input signal bit width. To prevent this bit width increase, to provide the proper signal dynamic bandwidth, and to ease the next computation of the derived spectrum, the CNORM units are attached to the outputs of the FFT16 units. CNORM unit provides the data shift left to 0,1,2, and 3 bits depending on the code SHIFT. The input data width is nb+3 and the output data width is nb+2, where nb is the given processor input bit width. The SHIFT inputs of two CNORM stages are concatenated to the 4-bit input SHIFT of the FFT256 core, 2 LSB bits control the first stage, and 2 MSB bits do the second stage.

Rotator256: The unit ROTATOR implements the complex vector rotating to the angles W_{256}^{ms}. The complex twiddle factors are stored in unitWROM256 Here the row and column indexes are *m* and *s* respectively. These coefficients are read in the natural order addressing by the 8-bit counter addrw. The complex vector rotating is implemented by the usual schema of the complex number multiplier which contains 4 multiply units and 2 adders.

9. TestBench Simulation Observations

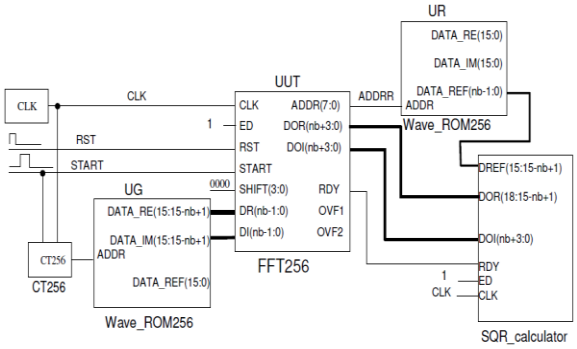

Figure 14: TestBench

The units UG and UR are implemented as ROMs which contain the generating waveforms (UG) and the reference waveform (UR). They are instantiated as a component Wave_ROM256. The tables of sums of up to 4 sine and cosine waves and respective frequency bins are generated in Perl. The table length is set as $n = 256. The samples of these tables are outputted to the outputs DATA_IM, DATA_RE, and DARA_REF of the component Wave_ROM256 respectively. The counter process CT256 generates the address sequence to the UG unit starting after the START impulse. The UG unit outputs the testing complex signal to the UUT unit (FFT256) with the period of 256 clock cycles.

When the FFT result is ready then UUT generates the RDY signal after that it generates the address sequence ADDR of the results. This sequence is the input data for the UR unit which outputs the correct real samples (bins) of the spectrum. We note that because the input data is the complex sine wave sum then the imaginary part of the spectrum must be a sequence of zeros. The sum of square differences of spectrum results and the reference samples is calculated by a process named SQR_calculator. This is commenced after the impulse RDY and continues till 256 clock cycles.

Then the result is divided to 128 and outputted in the message in the console of the simulator. For eg. the message: "rms error is 1 lsb" means that the square of the residue mean square error is equal to 1 LSB of the spectrum result which is acceptable.

10. Conclusion

When the model FFT256 is correct and its bit widths are selected correctly then the rms error is not succeeded by1 or 2. When this model is not correct then the message will be a huge positive or negative integer, or 'X'. The model correctness can be proven or investigated by looking at the input and output waveforms. Fig.15 illustrates the waveforms of the input signals, and Fig.16 of the output waveforms.

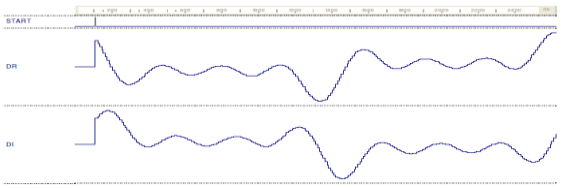

Figure 15: Input Waveform for Testbench

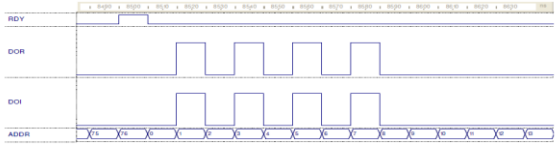

Figure 16: Output Waveform for Testbench

Performance of Implementation Data on Xilinx :
Table 2 illustrates the performance of the FFT256 core with two data buffers based on Block RAMs in Xilinx Virtex device when implementing 256-point FFT for 10 and 16-bit data and coefficients. Four DSP48 units have been used in whole project. The results are derived using the Xilinx ISE 9.1 tool.[5]

Table 2: Performance of Implementation Data

Target device and its data bit width	**XC 4VSX25-12(10 Bit)**	**XC 4VSX25-12(15 Bit)**	**XC 5VLX30-3 (10 Bit)**	**XC 5VLX30-3(15 Bit)**
Area, Slices	4791 (46%)	6945 (67%)	1739 (36%)	2434 (50%)
RAMBs	3	4	3	3
Max system clock	206 MHz	194 MHz	244 MHz	234 MHz

References

[1] Bracewell, R. N.: 'The Fourier Transform and its applications', The McGraw-Hill Companies, Inc, 2000, ISBN: 0-07-303938-1.
[2] Ma, Y., Wanhammar, L.: 'A hardware Efficient Control of Memory Addressing for High-Performance FFT Processors', IEEE Transaction on Signal Processing, Vol. 48, No. 3, March 2000.
[3] Proakis, J. G.: 'Digital Signal Processing, Principles, algorithms and applications', Prentice Hall, Inc., 1996, ISBN: 0-13-394289-9.
[4] Nussbaumer H. J.: 'Fast Fourier Transform and Convolution Algorithms' (Springer-Verlag: Berlin, 1990).
[5] Xilinx Inc.: 'Xilinx Virtex-E Databook', http://www.xilinx.com, 2000-2001, (Acc 2001-02-05).
[6] Eric Ericson Fabris; Gustavo André Hoffmann; Altamiro Susin; Luigi Carro : 'A bit-serial FFT processor' UFRGS Federal Univ. – Microelectronics Group, 2003.
[7] M. Hasan and T. Arslan, Department of Electronics and Electrical Engineering, The University of Edinburgh:'A Coefficient Memory Addressing Scheme for VLSI Implementation Of FFT Processors', The King's Buildings, Mayfield Road, Edinburgh EH9 3JL, Scotland, UK, 0-7803-7448-7/02 *2002* IEEE.
[8] Ray Andraka, "A survey of CORDIC Algorithms for FPGA based computers". Proceedings of the 1998 ACM/SIGDA sixth International Symposium on Field Programmable Gate Array.

Shashank Gupta completed his Bachelors in Electronics Engineering from Indian Institute of Technology, Banaras Hindu University, Varanasi, India. His major fields of interests are Communication and Embedded Systems. He did his summer internships at National University of Singapore and IRISA Labs, France. His paper titled : Optimization of Resonant frequency in co-axial probe feed Microstrip Patch antenna using Differential Evolution (DE) algorithm was accepted in 2nd International Conference on advances in Computer, Electronics and Electrical Engineering – CEEE, Mumbai, which was later published in International journal of Advancements in Electronics and Electrical Engineering Vol2, Issue 2.