FPGA Implementation of 1024-point Radix-4 FFT core using Xilinx VHDL

Ram B. Alapure, Kushal M. Ghadge

***Abstract*— Design and optimized implementation of a 16-bit 1024-point FFT processor is presented in this paper. The system architecture of a FFT core is based on a radix-4 algorithm. The target FPGA is an XC3S500E Spartan-3E from Xilinx working at clock frequency of 50MHz and fully tested by method of co-simulation using Xilinx ISE 14.2 and tested on real hardware using Spartan-3E starter kit.**

***Index Terms*—DFT, DIF, Fast Fourier Transform (FFT), Field Programmable Gate Array (FPGA), VHDL.**

I. INTRODUCTION

The Fast Fourier Transform (FFT) is a conventional method for an accelerated computation of the Discrete Fourier Transform (DFT) [1], which has been used in many applications such as spectrum estimation, fast convolution and correlation, signal modulation, etc. Even though FFT algorithmically improves computational efficiency, additional hardware-based accelerators are used to further accelerate the processing through parallel processing techniques [2].

A large number of applications in signal processing and modern communication system uses the Fast Fourier transform (FFT) as one of its central blocks. Since the appearance of the Cooley-Tukey fast algorithm[1], which dramatically reduced the number of operations from N^2 to N $Nlog_2 N$, several works have tried to accelerate the computation speed in both hardware and software versions. Hardware implementations have demonstrated to be very efficient and faster than their software counterparts when running at a given clock speed. This because having dedicated hardware to perform the required operations allows for completing calculation using a lower number of clock cycles. Field programmable gate arrays (FPGA) have facilitated the hardware implementation of FFTs allowing for scalable and non-scalable designs. As FPGA technology evolves increasing density and speed, it is expected to enable larger and faster designs [3].

1024-point FFT computation is considered the main basic algorithm for several DSP applications. Different FFT algorithms have been proposed to exploit certain signal properties to improve the trade-off between computation time and hardware requirements. Radix-4 based algorithms improve computation time in a factor of two, compared with radix-2 based algorithms, increasing hardware requirements by the same factor.

Considering that low-cost, high-density reconfigurable devices are already available, an optimized price/performance HDL core development of the 1024-point radix-4 FFT is feasible [4, 5].

Manuscript received February 15, 2015.

Ram B. Alapure, Electronics & Telecommunication Dept., Government College of Engineering, Aurangabad, India.

Kushal M. Ghadge, Electronics & Telecommunication Dept., Government College of Engineering, Aurangabad, India.

II. RADIX-4 FFT

The N-point discrete Fourier transform (DFT) is defined by equation (1),

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

Where

$$W_N^{kn} = e^{-2\pi\left(\frac{kn}{N}\right)} = \cos\left(2\pi\frac{kn}{N}\right) - j\,sin\left(2\pi\frac{kn}{N}\right) \qquad (2)$$

The DFT calculation demands a complex implementation (requires N^2 complex multiplications and N(N-1) complex additions), so we have to find a more efficient way to perform this calculation. The FFT was proven to be a faster and more efficient algorithm to compute Fourier transform. We use the decimation in Frequency (DIF) radix-4, which is the most used to calculate the FFT because of it's reduce computational complexity [4, 5].

The radix-4 DIF FFT divides an N-point discrete Fourier transform (DFT) into four N/4-point DFTs, then into 16 N/16-point DFTs, and so on. In the radix-2 DIF FFT, the DFT equation is expressed as the sum of two calculations. One calculation sum for the first half and one calculation sum for the second half of the input sequence [6]. Similarly, the radix-4 DIF FFT expresses the DFT equation as four summations, and then divides it into four equations, each of which computes every fourth output sample.

X(4k), X(4k+1), X(4k+2), X(4k+3) are N/4-point DFTs. Each of their N/4 points is a sum of four input samples x(n), $x\left(n + {}^{N}/_{4}\right), x\left(n + {}^{2N}/_{4}\right), x\left(n + {}^{3N}/_{4}\right)$, each multiplied by either +1,-1,j or –j. The sum is multiplied by a twiddle factor ($W_N^0, W_N^n, W_N^{2n}, W_N^{3n}$). Figure 1.

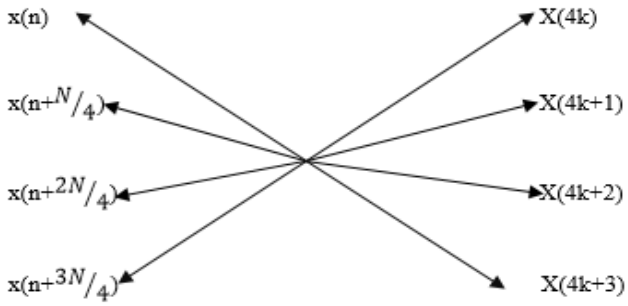

Figure 1. Radix-4 DIF FFT Dragonfly.

The four N/4-point DFTs together make up an N-point DFT. Each of these N/4-point DFTs is divided into four N/16-point DFTs. Each N/16 DFT is further divided into four

N/64-point DFTs, and so on, until the final decimation produces four-point DFTs. The four-point DFT equation makes up the dragonfly calculation of the radix-4 FFT.

When using Radix-4 decomposition, the N-point FFT consists of $log_4 N$ stages, with each stage containing N/4 Radix-4 dragonflies. From the formulas we calculate the 1024-point FFT. That will be 5 stages where dragonflies will run for the 1024-points.

Compared with the Radix-2 algorithm, we will get a more complex algorithm but with less computational cost. For 1024-point sequence, Radix-2 would require 40960 additions and 20480 multiplications whereas Radix-4 requires 30720 aditions and 5120 multiplications [5].

III. Architecture

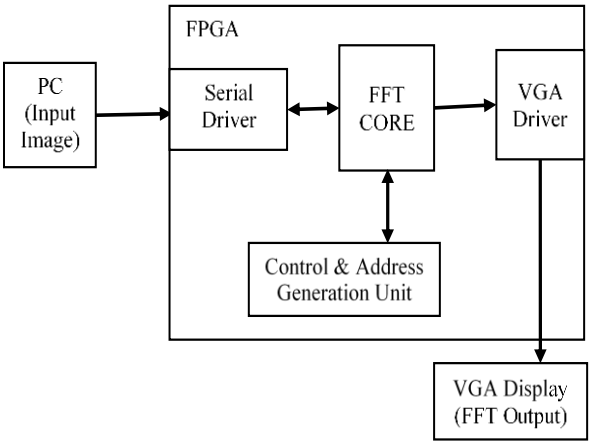

Figure 2. Block Diagram of System

The 1024-point FFT processes 1024 complex samples, with 16-bit length. Those samples are store in the memory RAM1, each one with a direction. The dragonfly takes 4 samples and operates, this process is repeated 256 times and it's stored in the memory RAM2. Then it replaces the RAM1 with the RAM2 information and process the dragonfly again. This process is done 5 times in order to finish the calculations of the Radix-4 FFT. We can see the flowchart in the Figure 3.

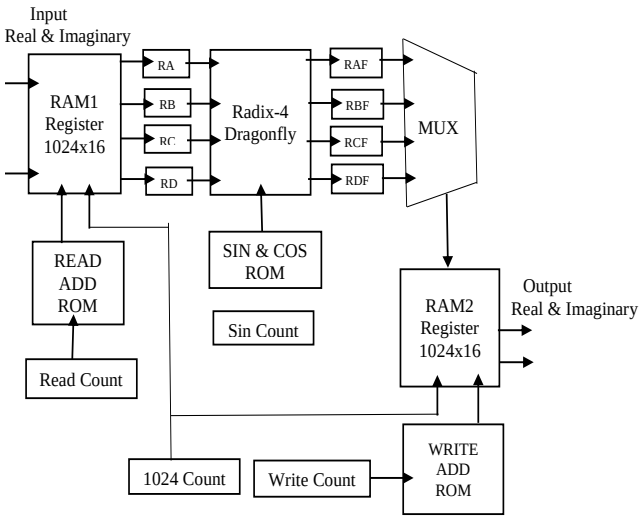

Figure 3. Radix-4 DIF FFT flow chart

IV. FFT CORE

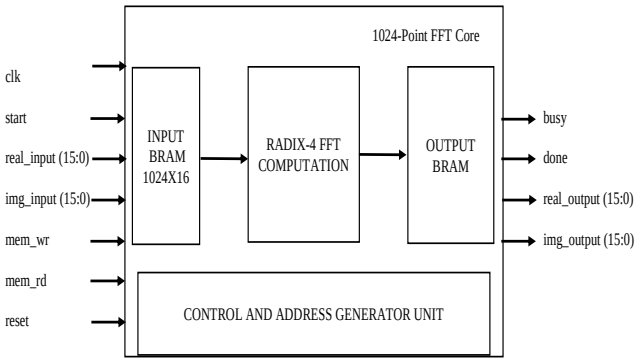

Figure 4. 1024-Point FFT core

The FFT core receives the input data in natural order, and it generates the output in bitreversed order. Input data arrives at clock rate. All the data needed to compute each FFT arrives as a block. In order to ease the integration of the FFT a validation signal, DATA IN VALID, will be set high during the arrival of valid data at the input of the FFT core. Similarly, when valid output data are ready at the output of the FFT core, a validation signal DATA OUT VALID is set high.

The FFT processor employs fixed-point arithmetic. Input and output data are represented using dbw bits. The twiddle factors have been quantized with tbw bits. The core scales data appropriately during internal operations to avoid overflow.

The FFT Core computes a 1024-point complex forward FFT. The input data is a vector of 1024 complex values represented as 16-bit 2's complement numbers – 16-bits each, for the real and imaginary components of a data sample. The 1024 element complex output vector is also represented using 16 bits for each of the real and imaginary components of an output sample. The FFT core is partitioned into three major modules namely Memory section, Multiply and Accumulate, and Control and Address Generator.

A. Memry section

The core has three memory areas – two 1024 x16 BRAMs for storing the input samples, two 1024 x 16 BRAMs for storing the output results and two 768 x 16 ROMs for storing the Twiddle Factors.

B. Multiply and Accumulate Section

This is the heart of the FFT core as FFT basically involves multiplication and adding. It also includes the pipelines needed for the data to be properly present at the Multiplier and adder inputs.

V. FPGA IMPLEMENTATION

In the first implementation of the FFT processor core in VHDL some problems occurred due to the differences between MATLAB and VHDL. One difference is caused by the concurrency of VHDL simulations, which caused a mismatch between data and control signals. There were three possible solutions: insert delay elements in the data path, change the control unit, or to insert asynchronous interfaces in the problem areas. The first option is the easiest to implement, but adds unnecessary hardware. The second is possible, but

could be hard to get correct. The last option is the most efficient from design time point of view [8].

Xilinx hardware and software tools were used during the design process. Xilinx ISE served as the VHDL modeling environment and a Xilinx Spartan 3E starter kit used as the target hardware. This specific board inlcudes an XC3S500E-4FG320 FPGA.

All phase factors are stored in a ROM memory, whose contents is initialized at the moment of downloading the configuration file into the FPGA. The ROM, as well as the RAM data memories (Mem1 and Mem2), were implemented using pre-designed blocks available in the Xilinx Core Generator tool. Nonetheless, this design is not limited to the memory resources inside the FPGA. External memory can be used for cases when small FPGAs with limited memory were chosen, or when the core were part of a complete system and chip resources were shared with other components.

The elemental fixed point operators in the design are also Xilinx units. These were used as building blocks for the complex operators needed to implement the radix-4 butterfly. For the case of the complex multiplier, four real multipliers and two adder/subtracter units were used to compute the result. This block has the highest cost in terms of resource consumption. In addition, the implementation needed rounding to cast the output of each multiplier back to the input precision. Truncation, although feasible, would cause a greater error during the calculation. Rounding to the nearest integer was used instead.

Using fixed point full-scale arithmetic implies that the word length will be different at the input and output sides of the core. The output width is dependent on the number of stages (transform size) and can be calculated as:

output width = input width + num stages+ 1

The output width specifies the minimum requirement for the data word width that can be used in the data memories [3].

VI. VALIDATION, TESTING AND RESULTS

A. Validation and Testing

The design validation was performed at two levels. The first was focused on simulation and the second on implementation. For both levels, input vectors were read from a coe file and results written to another file which was compared with the expected results. All input vectors and expected results were generated with MATLAB. Results were obtained with the FFT MATLAB command. For simulations, a testbench read the input into the design and wrote the FFT result to an output file to be later compared with the expected results.

B. Results

The time-computation performance of the FFT is estimated by the Xilinx ISE 14.2 for the FPGA Spartan-3E and gives the result of clock period 6.3ns (frequency 158.73MHz).

The implementation reaches a very near result in compare with the Xilinx FFT Core [9] and better result than other FFT

TABLE I. Used Resource Report

Utilization	Xilinx FFT Core	FFT Core[4]	Case study
Slice Registers	47%	93%	41%
Slice LUTs	43%	84%	62%
LUT-FF Pairs	9%	43%	9%
RAM/FIFO	16%	66%	66%
Average	38%	72%	45%

VII. CONCLUSION

The developed FFT core has a higher time-performance than similar DSP and PC systems, and although it has lower time-performance compared with the commercially available core from Xilinx®, it has the advantage of using lower resources making it feasible to be implemented in lower-cost FPGA families like the Spartan-3E.

REFERENCES

[1] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” Math. Comput., vol. 19, pp. 297-301, 1965.

[2] A. Suleiman; H. Saleh; A. Hussein; D. Akopian, “A family of scalable FFT architectures and an implementation of 1024-point radix-2 FFT for realtime communications”, In Proc. of IEEE Int. Conf. on Computer Design, ICCD-2008. pp.321-327, Oct. 2008

[3] Agenor Polo, Manuel Jimenez, David Marquez and Domingo Rodriguez, “An Address Generator Approach to the Hardware Implementation of a Scalable Pease FFT Core”, Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on DOI: 10.1109/MWSCAS.2012.6292149 Publication Year: 2012, Page(s): 832- 835.

[4] J. A. Vite-Frias, R. J. Romero Troncoso and A. Ordaz Moreno, "VHDL Core for 1024-Point Radix-4 FFT Computation", IEEE International Conference on Reconfigurable Computing and FPGAs.

[5] Adriana Bonilla R., Roberto J. Vega L., Karlo G. Lenzi and Luís G. P. Meloni, “Design and Implementation of Fast Fourier Transform Algorithm in FPGA”, XXX SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES – SBrT’12, 13-16 DE SETEMBRO DE 2012, BRASÍLIA, DF

[6] A. V. Oppenheim and R. W. Shafer, Discrete-Time Signal Processing, 2nd Upper Saddle River, NJ: Prentice Hall, 1998.

[7] Vıctor Montano and Manuel Jimenez, “Design and Implementation of a Scalable Floating-point FFT IP Core for Xilinx FPGAs”, In Proc. of the 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS-2010), pp.533-536, Aug. 2010.

[8] Weidong Li, Jonas Carlsson, Jonas Claeson, and Lars Wanhammar, “ A GALS Based 16-Point Pipeline FFT Core” Proceeding in IEEE NorChip Conf.

[9] Xilinx® LogiCORE IP, Fast Fourier Transform V7.1, Xilinx® 2011.

Ram B. Alapure (M.E. final year student), Electronics & Telecommunication Dept.,Government College of Engineering, Aurangabad, India,

Kushal M. Ghadge (M.E. final year student), Electronics & Telecommunication Dept., Government College of Engineering, Aurangabad, India,