Nahrain University, College of Engineering Journal (NUCEJ) Vol.13 No.2, 2010 pp.150-157

Run-Time Reconfigurable FFT Engine

Ahmad F. Al-Allaf
Lecturer
Technical college-MOSUL
Dept of Computer eng.
CTE
Ahmadalallaf@yahoo.com

Shefa A. Dawwd
Lecturer
College of Engineering
Dept of Computer Eng.
MOSUL University
shefadawwd@yahoo.com

Abstract

This paper develops a system level architecture for implementing a cost-efficient, FPGA-based real-time FFT engine. This approach considers both the hardware cost (in terms of FPGA resource requirements), and performance (in terms of throughput). These two dimensions are optimized based on using run time reconfiguration, double buffering technique and the "hardware virtualization" to reuse the available processing components. The system employs sixteen reconfigurable parallel FFT cores. Each core represents a 16 complex point parallel FFT processor, running in continuous real-time FFT engine. The architecture support transform length of 256 complex points, as a demonstrator to the idea design, using fixed-point arithmetic and has been developed using radix-4 architecture.

The parallel Booth technique for realizing the complex multiplier (required in the basic butterfly operation) is chosen. That
is to save a lot of hardware compared to other techniques. The simulation results that have been performed using VHDL
modeling language and ModelSim software
shows that the full design can be implemented using single FPGA platform requiring about 50,000 Slices.

Keywords*:* Fast Fourier Transform, Radix-4, FPGA, Run time Reconfiguration

1.0 Introduction

The fast Fourier transform [1] is one of the fundamental algorithms in Digital Signal Processing (DSP) including acoustics, optics, telecommunications, speech, signal and image processing. Using this transform, signals can be
moved from time domain to the frequency domain where many digital signal processing techniques such as filtering and correlation can be performed with fewer operations.

Many different FFT implementations have been developed for the digital signal processors and dedicated FFT processor ICs [2,3,4,5,6,7,8,9,10,11]. However, decreasing the cost with the growing in the capacity, FPGAs have become a good candidate for implementing FFT engines. Despite of that, High performance, large-scale DSP applications still cannot fit in a single FPGA and require carful design considerations. In this context, the hardware implementation of the FFT algorithm can be done in either fixed or floating point. Floating-point arithmetic requires much more area per operation (adders and multipliers). Furthermore, it has much higher demands on memory capacity and bandwidth.

To overcome this challenge, fitting the whole Fourier transform processor on a single FPGA chip is approached along two paths: First, the algorithm itself is examined and optimized specifically to minimize the hardware resources. Second, applying different techniques on the architecture, to reduce the required hardware. With respect to second path; one of the viable solutions is to use the Reconfigurable Computing (RC) in the field of Field Programmable Gate Arrays (FPGAs) which potentially can offers high performance at lower chip area and power consumption.

This paper presents a high performance FFT architecture focusing on finding the most suitable structure for implementing efficient and cost effective FFT engine using RC technique. The system uses radix-4 architecture with fixed point arithmetic which is sufficient in many domains.

The rest of this paper is organized as follows. A background and an overview of the related works are described in Section 2. Review of different hardware implementations of the FFT algorithm and the proposed reconfigurable implementation of the 256 complex points 1D-FFT algorithm are given in section 3. The architecture of the proposed FFT system is presented in section 4. Section 5 focuses on the architecture of the FFT core including the implementation of the butterfly element and the

hardware multiplier. Finally in Section 6 and 7, performance analysis and some conclusions are offered.

2.0 Background And Related Work

The Fast Fourier Transform (FFT) algorithm used for implementation in the proposed system is based on the radix-4 butterfly, which represent the heart of the FFT algorithm. It takes four complex input data words, computes the FFT, and produces four complex output data words. Figure (1), illustrates the single flow graph of the radix-4 butterfly unit.

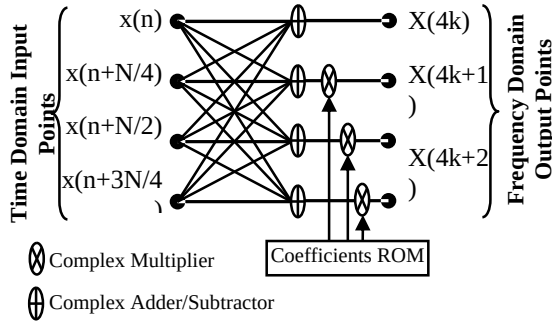

Fig.1 Single flow graph of radix-4 butterfly unit.

Each butterfly requires four complex adder/subtractors and three complex multipliers. The mathematical model for each radix-4 butterfly is:

$$X(4k) = \sum_{n=0}^{N/4-1}\left[x(n) + x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) + x\left(n+\frac{3N}{4}\right)\right] W_N^0 W_{N/4}^{kn} \quad (1)$$

$$X(4k+1) = \sum_{n=0}^{N/4-1}\left[x(n) - jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) + jx\left(n+\frac{3N}{4}\right)\right] W_N^n W_{N/4}^{kn} \quad (2)$$

$$X(4k+2) = \sum_{n=0}^{N/4-1}\left[x(n) - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) - x\left(n+\frac{3N}{4}\right)\right] W_N^{2n} W_{N/4}^{kn} \quad (3)$$

$$X(4k+3) = \sum_{n=0}^{N/4-1}\left[x(n) + jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) - jx\left(n+\frac{3N}{4}\right)\right] W_N^{3n} W_{N/4}^{kn} \quad (4)$$

Many different researches for the implementation of FFT algorithms on FPGA have been proposed since the introduction of this technology. For example, The design of a parametrisable architecture on an FPGA, using Handel-C language, was presented in [12] for implementation of different types of FFT algorithms. Kee et al[13] uses approach involves two orthogonal methods - FFT inner loop unrolling and outer loop unrolling - to achieve cost-optimized FFT implementations on FPGA. In outer loop unrolling of the targeted FFT, he realizes parallelism by instantiating multiple processing cores (dedicated hardware subsystems) across FFT butterfly stages. While in unrolling of the FFT inner loop, he allocates multiple cores within each stage.

Andraka et al[14] describes a technique, for implementing FFT algorithm on FPGA, that is a hybrid of fixed point and floating point operations designed to significantly reduce the overhead for floating point. In Ma's scheme[15], an FFT core that involves a single butterfly unit was developed. He uses an efficient method for in-place memory management. But the overall approach is limited in terms of throughput improvement.

To achieve an effective balance between hardware costs and performance features, Nordin et al [16] presented a parameterized soft core generator for the FFT based on the Peace FFT algorithm by running multiple butterflies simultaneously with a scalable stride permutation.

In XU et al. [17], an FPGA-based reconfigurable, hierarchical-SIMD (H-SIMD) machine with its codesign of the Pyramidal Instruction Set Architecture (PISA) was proposed. He assumes a multiple FPGA board where each FPGA is configured as a separated SIMD machine to implement 2D FFT. While Jackson et al. [18] proposed a systolic structure for high throughput FFT implementation. Finally, Kamalizad et al[19], mapped the FFT to the MorphoSys reconfigurable computing platform to achieve high performance FFT architecture.

3.0 Hardware Implementations Of The Fft Algorithm

In general, there are four different ways for hardware implementations of the FFT algorithm:

Serial FFTs: The computations are implemented in a number of iterations using only single butterfly unit and single memory unit.

Pipeline FFTs: They utilize concurrent processing of different stages to achieve high throughput.

Parallel FFTs: Each stage in the FFT is computed with a set of processing elements and the result is fed back to the same processing elements for the computation of the next stage.

Fully parallel-pipeline FFTs: The operations in the signal flow graph are mapped completely to a hardware structure.

The serial implementations sufure from low throughput while the pipeline FFT architectures are

suitable for continuous I/O with high throughput but yield larger latency. The parallel FFT architectures increase the parallelism within a stage but require buffer for the continuous throughput. Fully parallel-pipeline FFT architectures are hardware intensive and not suitable for implementation on FPGA especially for large transform length.

The design of the FFT engine, proposed in this paper, is implemented using reconfigurable parallel architecture. The architecture considered focuses on minimizing and optimizing the hardware resources without large scarifying in performance. The signal flow graph for implementing 256 complex point decimation in frequency (DIF) readix-4 FFT algorithm in the proposed system is shown in figure 2.

The input data are grouped in 16 blocks; each block consists of 16 complex points which then distributed to 16 FFT processors (FFT cores) for execution. The computation of 256 point radix-4 FFT algorithm requires four stages implementing using only 16 FFT cores. The result of each stage is stored and then reused by the same hardware to execute the next stage. The time required to compute an entire stage is the same for all stages. Data exchange is required between each group of four FFT cores after executing stage-1 and stage-2 of the algorithm as shown in figure 2.

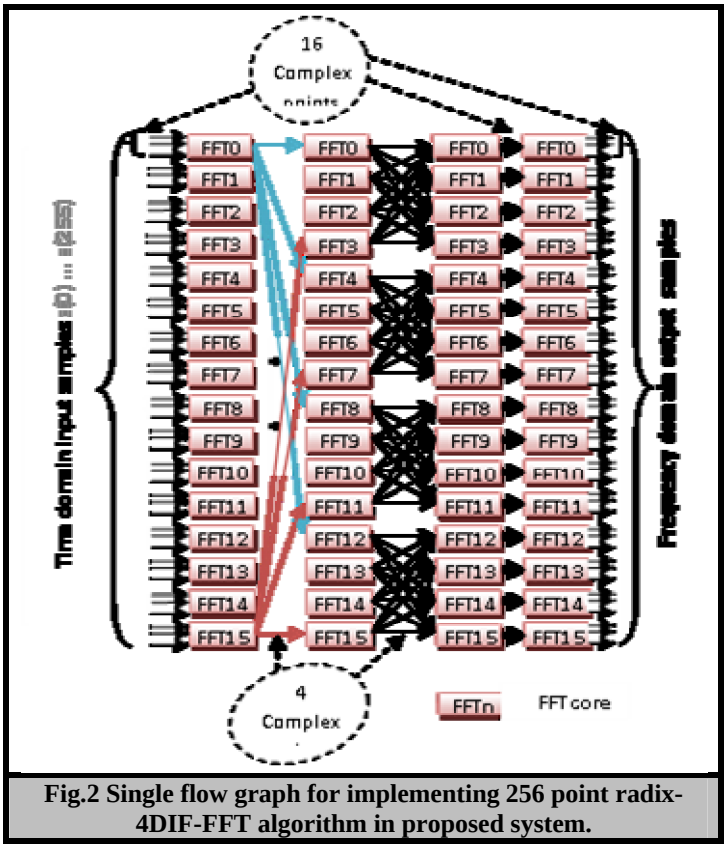

Fig.2 Single flow graph for implementing 256 point radix-4DIF-FFT algorithm in proposed system.

4.0 Proposed Fft Architecture

Figure 3 shows the architecture of the proposed FFT machine which can be divided into four main building blocks:

The data processing part and consists of 16 FFT cores, each core represents 16 complex point parallel FFT processor. The storage system which represented by the FIFO buffers. The routing structure (buses and input/output system) to routes data between the FFT cores and between the FFT engine and the external memories. Finally, Global and Control Configuration Unit (GCCU) to produce the global clock and control signals and to manages the configuration process.

Each FFT core consists of four redix-4 butterflies, and every core has a pair of FIFO input buffers each of size 16 complex points (16FIFO). The FPGA receives input samples from external memory and distributed it to one of the two sets of 16FIFO input buffers. By utilizing this double buffering structure, the two sets of 16FIFO input buffers are used to feed the FFT cores alternatively, thus solving the problem of data latency in data distributing process. Moreover, double buffering architecture allows subsequent input blocks to be processed in a

continuous, so that all of the butterflies in all FFT cores can be engaged all the time.

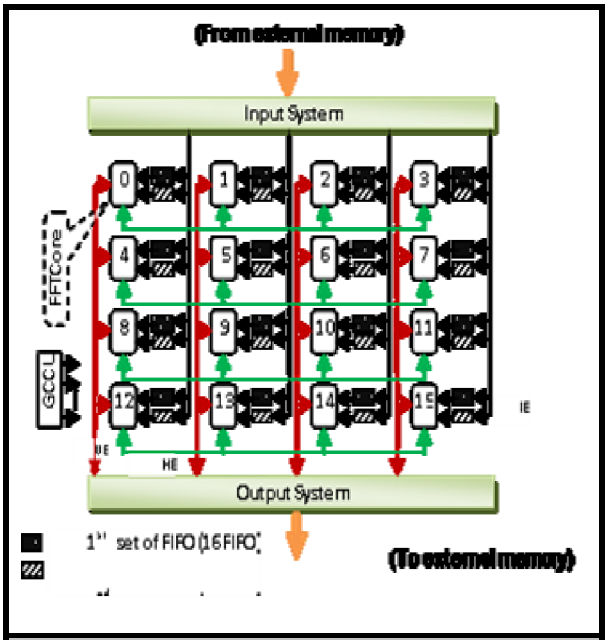

Fig.3 Architecture of the FFT Engine.

The execution of the algorithm passes in three phases:

- In data load phase, the input data is loaded (by the input system) from external memory and grouped in digit reverse order to a 16 block. These blocks are stored in the first set of 16FIFO input buffers.
- After full frame has been loaded, the FFT is computed on the stored data. When the FFT computation is complete, the result is sent back to the same set of 16FIFO input buffers. This represents the computation phase.
- In the last phase - result store phase - the result is read out from the 16FIFO. input buffers (by the output system) and then sent to the external memories.

Using the decimation in frequency (DIF) FFT algorithm, a digit reversal is required to reorder the input data which is done (under the control of GCCU) during data load phase, by the input system. Therefore no extra hardware or additional memory resource or time overhead is required to reorder the input samples.

After the data load phase, and while the first frame of 256 points, stored in the 1st set of 16FIFO input buffers, take part in calculations the next frame of 256 points are loaded to the 2nd set of 16FIFO input buffers. When the calculations of the first frame are finished, the FFT engine copies the result to the 1st 16FIFO input buffers. This result is then transferred, in next time, to the external memories and the FFT cores starts directly to compute the FFT algorithm on the next frame stored in the 2nd set of 16FIFO input buffers. Thus, data input, computation and data output operations are overlapped, so that the FFT processor is never left in an idle state waiting for an I/O operation. This provides high throughput rates for real-time applications, in which the input data is a sequential stream.

Furthermore, to speed up the operation of data transfer between the FFT cores and the I/O system and improving the throughput, each four FFT cores share a common bus for data input (Input Buses, IB). Considering that there are four separate external memory models to feed the FFT engine with the I/O samples in parallel. A torus network is chosen to connecting the FFT cores to facilities data exchange between the stages and for the data output. As shown in figure 3, four Horizontal Buses (HB) and four Vertical Buses (VB) connect each group of four FFT cores horizontally and vertically. This allows the sharing of data among neighboring cores which reduce the communication overhead.

Since the target output is the frequency components of the input signal. Therefore, the amplitude of the input signal is effectiveless and can be normalized. Based on this fact, we assumed that the input signal is in the range ±1, and using fixed point implementations, each of the real and imaginary parts of the I/O data are represented in 18-bit format with one bit for sign, one bit for integer and 16-bit for fraction. Therefore, all data pathways (buses) are also in 18-bit two's complement signed format. During the FFT computation results at a particular stage are scaled and truncated and then stored in FIFO buffers. In the following stage these results are read from these buffers. New stage computations are performed and new results are scaled and truncated again and moved back to the buffers.

The vertical buses are also used as data output buses to upload the result of computation to the external memories through the output system. The input and output systems are a set of input/output circuitry within the FPGA. These systems are used to upload and download, in parallel, the input/output data to and from four external memories using the separate input/output buses.

The Global Control and Configuration Unit (GCCU) is a state machine provides a number of control signals to coordinate and to synchronize the activity of different units in the FFT engine and to initiate processing and monitor its completion. The

GCCU provides different clock signals to keep track of data input/output in FIFO buffers and switching between the 16FIFO input buffers during data load/store phase and FFT computation phase. It implies that a particular stage of the FFT computation is done, either the input or output process is done, and the FFT computation process is accomplished. It is also responsible of unscrambling (digit-reversal) of input data at the beginning of each FFT execution. Finally, the GCCU controls and configures the input/output system to route the input and output data between FFT cores and external world.

5.0 Fft Cores

The FFT cores are responsible for performing the butterfly computations needed for the FFT algorithm. The FFT core consists of four DIF-FFT radix-4 butterflies (which are referred as the basic radix-4 butterfly (BR4B) processing element) in a full parallel configuration, local FIFO buffers of size 16 complex points (Local 16FIFO) to store the intermediate results of the computation, and the Local Control and Configuration Unit (LCCU). The local data pathways within the core are also in the form of 18- bit two's complement signed numbers. The block diagram representation of the FFT core is depicted in figure 4.

In the beginning of processing of a new frame, the input data are loaded from one of the two 16FIFO input buffers and distributed to the operand registers within each BR4B. During the computation, the intermediate results of the BR4B units are stored in local 16FIFO buffer. Then routed through the VB or HB to other cores (data exchange), depending on the stage of computations. After the end of computation of the current frame, the result is return back to the same 16FIFO input buffer, (replacing the input frame) in which is sent through the VB to the external memories in the next time.

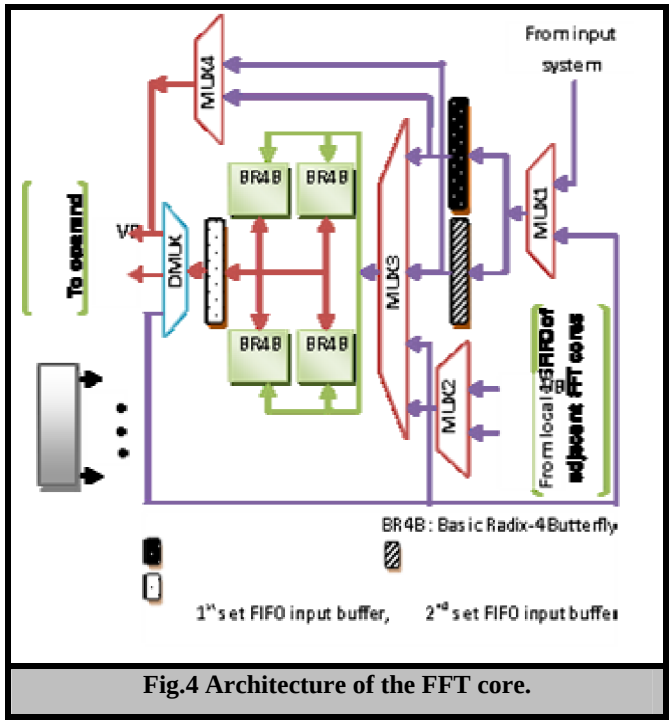

Fig.4 Architecture of the FFT core.

As mentioned above, the result of the BR4B is stored in local 16FIFO buffer and then sent to one of the storage sources depending on which stage is being executed as shown figure 2 and figure 4.

The LCCU is mainly responsible for sequencing the execution of local hardware on the FFT core during different execution phases. It sends number of control signals to set the MUXs and DMUX within the core to appropriate configuration to rout the data between components of the core. Moreover, it sends enable signals to the FIFO buffers of the core and to the operand registers of the BR4B units to perform the data exchange between the FFT cores. Finally, is also responsible of producing counting signals to address the coefficient ROM within every butterfly to provide the multipliers with the correct twiddle factors.

5.1 Basic Radix-4 Butterfly (BR4B) elements

The radix-4 "butterfly" operations represent one of the most efficient methods of performing the FFT calculation. The main advantages in utilizing a radix-4 butterfly operation is that it has better speed performance, in spite of its major complexity, and require less hardware compared to Radix-2. It requires 3 complex multiplies and 4 complex additions. Therefore, the total cost in complex multipliers is 75% of radix-2 FFT, although it uses the same number of complex additions [20]. Also,

Radix-4 algorithm has better signal to noise ratio than that of radix-2 algorithm [21].

Figure 5 shows the main components of the basic radix-4 butterfly (BR4B) elements. Each BR4B consists of three 18*8bit booth complex multipliers, and 18-bit adders/subtractors. Also each BR4B has eight 18-bit operand registers that accept (under the control of LCCU) the real and imaginary parts of four complex input points. Coefficients (or twiddle factors) are pre-calculated and stored in local coefficient ROM as 8-bit two's complement signed fixed-point words in each butterfly to achieve parallel access to twiddle factors for all butterflies. The adder/subtractors perform the butterfly operations on the data stored in the operand registers. The result is then send to the complex multiplier to multiply the it by the twiddle factors.

The 18 bit results of the butterfly operation are scaled by ¼ by applying right shift. The 26 bit result, growth of the fractional bits created from the multiplication, are truncated to return the data word sizes back to 18-bit, which is sufficient in most practical cases. The result are then stored in local 16FIFO buffers for further processing in next stages.

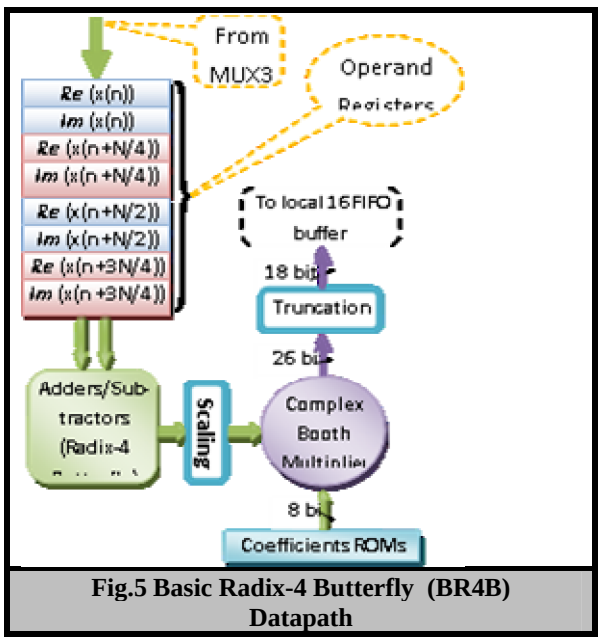

Fig.5 Basic Radix-4 Butterfly (BR4B) Datapath

The scaling of the intermediate results in FFT computation is necessary in order to prevent overflows which can be handled in three ways[22]:

- Performing the calculations with no scaling and carrying all significant integer bits to the end of the computation
- Scaling at each stage using a fixed-scaling schedule
- Scaling automatically using block floating point

The second option is chosen in this work. Therefore the scaling factor of ¼ is required after each butterfly operation to avoid overflow with fixed-point arithmetic. The design of the butterfly unit is simulated using the gate level simulator ModelSim. The functional simulation is performed to confirm the correct operation of the design.

5.2 Complex multiplier

The complex multiplier is the key component in the data processing. The direct implementation of complex multiplier requires 4 real multipliers, one adder and one subtractor. Furthermore, the multiplications are the most power dissipating arithmetic operations. Today's FPFA contain a number of speed optimized signal processing building blocks, such as multipliers, RAM blocks or I/O structures with propagation delays in the range of a few nanoseconds [23]. However, in this system we intended to design the FFT engine to be as a part of larger system. Therefore we do not use any of embedded hardware multipliers which are left to be used by other parts of the system and we designed our custom hardware multiplier. Based on that, and in order to minimize the area cost and the total power consumption and also to simplify the implementations, the parallel booth multiplier technique is used.

The simple serial by parallel booth multiplier is particularly well suited for FPGA implementation without carry chains because all of its routing is to nearest neighbors with the exception of the input. The number of real multipliers can be reduced to 3 with a simple transformation at the cost of extra additions. Thus, the complex multiplier used in this work requires only three real multipliers and four adder/subtractors. Several tests were performed in order to verify the complex multiplier functionality, besides the time performance analysis. The total area of the implemented multiplier highlights the advantages over traditionally implementation of the complex multipliers specially with the increasing of multiplier operand sizes. A comparison in area cost with respect to different operand size between two types of complex multipliers (Booth Complex Multiplier-BCM, and Ripple Complex Multiplier-RCM) is shown in figure 6.

6.0 Simulation Results

The simulation is based on using VHDL and ModelSim softwares. The inputs to the FFT are 18 bits wide, sixteen bits of fraction and one bit for integer and one bit for sign (assuming that the input signal in the range between (+1 and -1). Basic radix-4 butterfly element has been implemented on Spartan-3E(XC3S500E) evaluation board of 4656

slices requiring 657 slices. Based on this implementation, it can be estimated that one FFT core approximately requires (657*4) slices, which consumes 56 % of the total number of slices. Therefore, the entire proposed system (with 16 FFT cores) can be implemented using a single FPGA platform of more than 50,000 slices. As shown in figure 6, the component utilization, using the parallel Booth technique for realizing the complex multiplier save a lot of hardware compared to other techniques such as RCM.

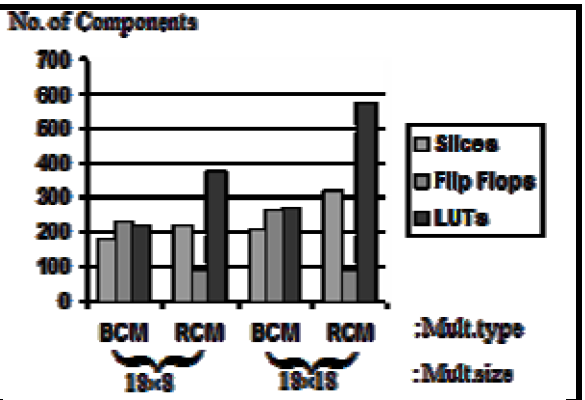

Fig.6 Component utilization for Spartan-3E (XC3S500E) for two types of multipliers with respect to the operand size.

7.0 Conclisions

This paper shows proposed architecture for the development of a 256-point radix-4 FFT engine for applications in hardware digital signal processing, targeting low-cost FPGA technologies. The architecture supports scaled fixed point arithmetic methods. The approach proposed in this paper use reconfigurable computing to carefully integrate two orthogonal methods for trading-off hardware cost and performance. This type of implementation leads to decrease in a silicon area at the cost of increasing in processing time. However, different methods are used to increase the performance such as using double buffering technique and parallel butterflies' execution.

The double buffering technique, by using two sets of FIFO buffers, overcomes the data I/O latency problem. The switching between those pairs of data FIFO buffers overlaps data communications with computations. Thus, hiding the communication overheads leading to improve the performance. The global controller (GCCU) lies in the FPGA and controls all the transactions between the FPGA and the external world. Data coming from external memory is distributed into the 1st set of high- speed FIFO buffers on the FPGA. When the FFT cores finish their current FFT implementations, they will switch to 2nd set of FIFO buffers to begin the FFT computation on a new frame of 256 complex points. Thus real-time processing is achieved.

In the proposed implementation, the communications among the butterflies are based on a nearest neighbor's grid interconnection. Data needed by every butterfly can be routed from its neighbor by using a set of operand registers and FIFO buffers.

The proposed system can offer acceptable throughput rates in relation to the other conventional FFT implementations such as DSP processors or ASIC FFT systems.

References

[1] J.W. Cooley and J. W. Tukey, "An Algorithm for the Machine Computation of the Complex Fourier Series," Math.of Computation, Vol. 19, pp. 297-301, April 1965.

[2] Datasheet, "Analog Devices DSP Selection Guide 2002 Edition", Analog Devices, 2002.

[3] Datasheet, "TI C62x and C67x DSP Benchmarks", Texas Instruments, 2002.

[4] Datasheet, "Motorola DSP 56600 16-bit DSP Family Datasheet", Motorola Ltd.,2002.

[5] M. Wosnitza: "High Precision 1024-point FFT Processor for 2-D Object Detection", Ph.D. SBN 3-89649-443-0, Swiss Federal Institute of Technology (ETH) 1999.

[6] Baas, B. M., "A low-power, High-Performance 1024-point FFT Processor", IEEE Journal of Solid State Circuits. pp. 380-387, 1999.

[7] Lecce, V. D. and D. E. Sciascio, "A VLSI Implementation of a Novel Bit-Serial Butterfly Processor for FFT", Proceedings of the 5th Euro computer conference, Comp Euro'91, Advanced Computer Technology, Reliable Systems and Applications, No. 1991, pp. 875-879, 1991.

[8] Chen, T. and L. Zho, "An Expandable Column FFT Architecture Using Circuit Switching Networks", Journal of VLSI Signal Processing, Vol. 6, No. 3, pp. 243-257, Dec. 1993.

[9] Szwarc, V. , L. Desormeaux, W. Wong, S. P. C. Yeung, H. C. Chan and A. T. Kwasnievski, " A Chipset for Pipeline and Parallel Pipeline FFT Architectures", Journal of VLSI Signal Processing, Vol. 6, No. 3, pp.253-265, Dec. 1994.

[10] Storn, R. "Radix-2 FFT- Pipeline Architecture with Reduced Noise to Signal Ratio", IEE Proceedings - Vision, Image and Signal Processing , Vol. 141, No. 2, pp.81-88, April, 1994.

[11] Melander, J. , T. Widhe, P. Sanbarg, K. Palmkvist, M.Vesterbacka and L. Wanhammar, "Implementation of a Bit- Serial FFT Processor With a Hierarchical Control Structure", Proceedings - EECCTD' 95 European Conference on Circuit

Theory and Design, I. T. U. , pp. 423-426, Sept. 1995.
[12] I. S. Uzun , A. Amira and A. Bouridane, "FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing", VISP(152), No. 3, pp. 283-296, June, 2005,.
[13] Hojin Kee, Newton Petersen, Jacob Kornerup, and Shuvra S. Bhattacharyya, "Systematic Generation of FPGA-based FFT Implementations," In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, Las Vegas, Nevada, March,2008.
[14] Raymond J. Andraka "Hybrid Floating Point Technique Yields 1.2 Gigasample Per Second 32 to 2048 point Floating Point FFT in a single FPGA", DSP magazine, pp. 42-44. April, 2007
[15] Y. Ma, "An Effective Memory Addressing Scheme for FFT Processors," IEEE Transactions on Signal Processing, Vol. 47, Issue 3, pp. 907-911, March, 1999.
[16] G. Nordin, P. A. Milder, J. C. Hoe, M. Puschel, "Automatic Generation of Customized Discrete Fourier Transform IPs", Design Automation Conference, pp. 471- 474, 2005.
[17] Xizhen XU*y and* Sotirios G. ZIAVRAS, "A Coarse-Grain Hierarchical Technique for 2-Dimensional FFT on Configurable Parallel Computers.," IEICE Transc. Inf. & Syst., Vol. E89D, No. 2 Feb. 2006.
[18] P. A. Jackson, C. P. Chan, J. E. Scalera, C. M. Rader, and M. M. Vai, "A Systolic FFT Architecture for Real Time FPGA Systems", High Performance Embedded Computing Workshop, 2004.
[19] A. H. Kamalizad, C. Pan, and N. Bagherzadeh " Fast parallel FFT on a reconfigurable computation platform" In Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-*PAD'03)*, pages 254–259, Sao Paulo, SP - Brazil, November, 2003.
[20] http://cnx.rice.edu/content/m12027/
[21] Weidong Li and Lars Wanhammar, "Efficient Radix-4 and Radix-8 Butterfly Elements", available at : http://www .es.isy.liu.se/publications/papers_and_reports/1999/weidongl_NorChip99.pdf
[22] Ishaan L. Dalal and Fred L. Fontaine "A Reconfigurable FPGA-based 16-Channel Front-End for MRI,", at the 40th Asilomar Conference on Signals, Systems and Computers in Pacific Grove, CA, on October 30, 2007.
[23] Virtex-II ProO Platform FPGA Handbook, Xilinx, Inc., San Jose, CA, 2002.
[24] S. Sukhsawas and K. Benkrid , "A High-level Implementation of a High Performance Pipeline FFT on Virtex-E FPGAs" In Proceedings of the IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design (ISVLSI 04), pp. 229–232, Lafayette, LA, February, 2004.
[25] Ediz Cetin, Richard C. S. Morling and Izzet Kale, "An Integrated 256-point Complex FFT Processor for Real-time Spectrum Analysis and Measurement," IEEE Proceedings of Instrumentation and Measurement Technology Conference, Vol. 1, pp. 96-101,Ottawa, Canada, May 19-21, 1997.
[26] L. Wanhammar, "*DSP Integrated Circuits*," Academic Press, 1999.

ماكنة FFT قابلة لإعادة التشكيل في زمن التنفيذ

احمد فالح محمود العلاف
مدرس/ الكلية التقنية- الموصل
قسم هندسة الحاسبات

شفاء عبد الرحمن داوؤد
مدرس/ جامعة الموصل- كلية الهندسة
قسم هندسة الحاسبات

الخلاصة:

في هذا البحث تم تطوير معمارية جديدة بمستوى النظام لماكنة FFT قليلة الكلفة لمعالجات الزمن الحقيقي باستخدام دوائر ال . FPGA التصميم المقترح يأخذ بنظر الاعتبار تحقيق اقل الكلفة (بدلالة متطلبات موارد ال (FPGA واعلى انجاز (بدلالة ال throughput) وذلك من خلال استخدام خاصية اعادة التشكيل في زمن التنفيذ ، خاصية الخزن المزدوج ، وخاصية اعادة استخدام المكونات المادية لدوائر ال . FPGA المنظومة المقترحة تستخدم ١٦ وحدة FFT متوازية قابلة لاعادة التشكيل . كل من هذه الوحدات هي عبارة عن معالج FFT متوازي بطول ١٦ نقطة مركبة . لذا فأن المعمارية المقترحة تستطيع تنفيذ خوارزمية FFT بطول ٢٥٦ نقطة مركبة والذي استخدم كمثال لتوضيح فكرة التصميم. ان خوارزمية FFT التي تم اعتمادها في هذا البحث تستخدم معمارية فراشة الاساس-٤ وحسابات النقطة الثابتة.

في هذا التصميم تم اختيار اسلوب بوث (Booth) المتوازية في بناء دوائر الضرب المركبة المطلوبة لانجاز عمليات الفراشة (وحدة البناء الاساسية لخوارزمية ال (FFT والتي تختصر المكونات المادية المطلوبة لتصميم دوائر الضرب قياسا للاساليب الاخرى. لقد اشارت نتائج المحاكات التي تمت باستخدام لغة VHDL و برنامج Model Sim الى ان التصميم الكامل للمنظومة يتطلب بحدود ٥٠٠٠٠ وحدة (Slices) ويمكن تنفيذها في FPGA واحدة.