COARSE GRAINED RECONFIGURABLE ARCHITECTURE FOR MULTICORE SYSTEM DESIGN

Prof. Radha K Chimurkar, Assistant Professor, ETC Department, Govindrao Wanjari College of Engineering & Technology, Nagpur, Maharashtra
Prof. Avishkar Wanjari, Assistant Professor, Electrical Engineering Department, Govindrao Wanjari College Of Engineering & Technology, Nagpur, Maharashtra

Abstract:
An increasingly significant area of study is reconfigurable computing. One way to significantly speed up an application is to move its computationally demanding parts onto reconfigurable hardware. As the ability to dynamically adapt a system to a range of diverse applications becomes the focus of system design, flexibility, scalability, and reconfigurability become increasingly important factors. Coarse-grained reconfigurable computing systems are much more performant than general-purpose systems, as indicated by numerous applications. In order to design the system with coarse-grained reconfigurable architecture, this study has been conducted.

Keywords: Coarse grain reconfigurable architecture, multi-core system, transistors, ASIC, Fine-grained reconfigurable architectures etc.

I. INTRODUCTION

In the past, clock rate increases and adding more transistors to chips were the only ways to improve processor performance. Nevertheless, there is a limit to this solution. Two major challenges for processor designs as transistor density rises are power dissipation and on-chip wire latency. Chip designers have to alter their processes because it results in excessive power consumption and high heat dissipation. Rather than using a single core, chips with multiple cores are the new trend. Better performance is provided by this new approach, which uses less power[1]. As single core processors quickly approach the physical boundaries of achievable complexity and speed, multi-core processing is an increasingly popular industry trend in the modern era. Power costs associated with improving single core performance could eventually outweigh the additional transistors used. Combining several cores on a die to increase processing capacity and throughput on a single chip is a logical use for the extra transistors. Increasing the number of processing units (cores) and, consequently, the possible computing capacity, is made possible by multi-core architectures [2][3].
In terms of computing technology, multicore processors are new; however, the ideas of parallel computing and multithreading are not. But these ideas have gained significance with the advent of multicore CPUs. Multi-core architectures are widely expected to surpass the conventional interpretation of Moore's Law, as more and more multi-core processor products from leading semiconductor companies are being released [4]. If two or more processors are added to a single integrated circuit to improve performance, lower power consumption, and more effectively process multiple tasks at once, the result is a multi-core system. Two or more independent cores in a single computing component constitute a multi-core system.
Multiple cores can be placed in a single die with their peripherals depending on the application thanks to the use of multi-core technology. The term "core" refers to a single processor. The integrated circuit dies consist of either a single die or multiple dies combined into a single chip package. A system with multiple cores can accomplish multiprocessing within a single physical package.
Only identical cores are present in homogeneous multi-core systems. Homogeneous multicore systems are those in which a single core design is deployed repeatedly. A homogeneous multi-core architecture has exactly the same number of processing cores: comparable frequencies, cache sizes, features, etc. The cores in heterogeneous multicore systems are not all the same. By this, it is meant

to imply that heterogeneous multi-core architectures employ a variety of cores. Performance is improved for domain-specific applications by heterogeneous multi-core architectures.
Three types of system architectures are most frequently utilized in data processing: Though versatile, general-purpose processors lack efficiency and provide inadequate performance for certain applications. Application-specific architectures are effective and perform well, but they are not adaptable. Reconfigurable systems have garnered more attention recently because of their efficiency and flexibility combined. Reconfigurable architectures are less flexible because they compromise between these two extremes. A hybrid strategy between application-specific designs and general-purpose processors is represented by reconfigurable systems.
Applications are executed quickly and with some flexibility thanks to reconfigurable architectures. The number of reconfigurable devices has increased dramatically over the past ten years to include a wide range of architectures and features; most recently, some of these devices have begun to incorporate reconfigurability into multicore designs [5]. According to their level of granularity, reconfigurable architectures are divided into two categories: fine-grained and coarse-grained.

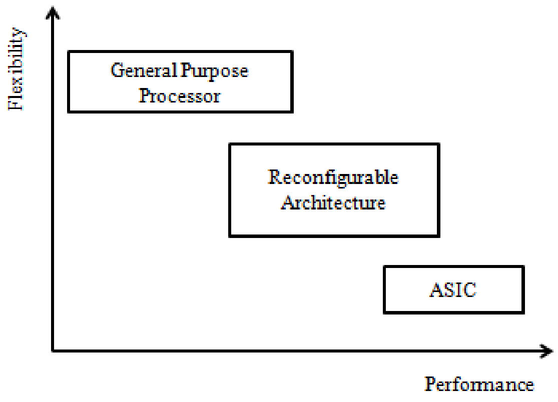


Figure 1: - Reconfigurable architecture vs General Purpose Processor or ASIC
Since they have been available for nearly thirty years, fine-grained reconfigurable architectures, or FGRAs, are an inherent component of digital system design and verification. Minimal-sized reconfigurable architectures that are fine-grained in nature comprise fundamental function units. Field-programmable gate arrays (FPGA), complex programmable logic devices (CPLD), and programmable array logic (PAL) are examples of the fine-grained devices. In addition to differing in architecture and configuration data storage methods, the devices are ranked in order of increasing complexity. While larger devices are volatile and require external programming, smaller devices use internal nonvolatile memory.
Nonetheless, FPGA architectures will be used as a model for fine-grained reconfigurable architectures here because they have the most adaptable architecture. Fine-grained basic logic blocks like transistors, NAND gates, multiplexer connections, and Look-Up Tables (LUTs) made up the majority of early FPGAs and Programmable Logic Devices (PLDs) [7]. Conventional FPGAs are field programmable gate arrays (FPGAs) with fine-grained logic that can be altered after product assembly. A range of configurable logic blocks (CLB) that mimic boolean logic and basic arithmetic using look-up tables (LUT) make up the FPGA architecture, which is depicted in Figure 2.

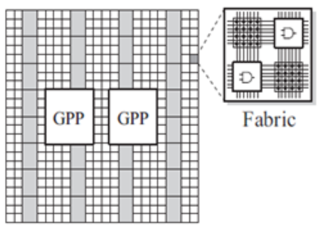


Figure 2: - An example of fine grained reconfigurable architecture, FPGA

II. LITERATURE REVIEW

The authors of the paper "Scalable Computing in the Multicore Era," Xian-He Sun, Yong Chen, and Surendra Byna, came to the conclusion that Amdahl's law does not apply to multi-core systems' scalability[1]. Three accelerated parallel processing models were examined in order to shed light on the largely unexplored topic of scalability of multi-core architecture.

1. Constant size
2. Set time
3. A boost in memory-bound speed.

When the number of cores n goes to infinity the speed up can grow linearly with n and thus multi-core architecture has scalability potential.

Yong Chen summarized that Hill and Marty presented a pessimistic view of multi-core scalability, citing Amdahl's law and the memory-wall problem in the paper "Reevaluating Amdahl’s law multi-core era" by Xian-He Sun [2]. High performance CPUs are increasingly following the trend of multi-core architecture. Although it is widely acknowledged that we have entered the multi-core era, there are questions about when or if we will actually reach the many core stage. Although the technology is there, big processor manufacturers are reluctant to produce processors with a lot of cores. This is an extremely fascinating phenomenon in which the 20-year-old parallel processing scalability debate appears to be repeating itself in history.

Several example architectures are provided in this paper by R. W. Hartenstein titled "A decade of reconfigurable computing: A visionary retrospective" to provide an overview of the advancements made in the field of coarse-grained reconfigurable computing [3]. The coarse-grained reconfigurable architectures are categorized as systems using mesh-based architectures, systems based on linear arrays, and systems using cross-bar switches based on the basic interconnect structure. The classification of coarse-grained reconfigurable architectures is based on the width of the data paths. The data path's width can vary from two to 32 bits. An architecture's data path width choice is a compromise between efficiency and flexibility.

A taxonomy of coarse-grained architectures from an architectural perspective and their related computation models are provided in the paper "Evolution in architectures and programming methodologies of coarse-grained reconfigurable computing" by Zain-ul-Abdin and Bertil Svensson. Emerging examples of these architectures are introduced in the 21st century [5]. The survey ends with a few predictions for the future in terms of programming models, architectural perspectives, and technological developments.

One of the characteristics of the hybrid architectures is that the coarse-grained reconfigurable array is tightly coupled to the host processor. The reconfigurable fabric has granularities ranging from fine to coarse, and its reconfiguration is managed by the host CPU, which is capable of carrying out calculations on its own. In terms of the interconnection network, the hybrid architectures combine global buses with nearest neighbor connectivity. Zippy and MorphoSys are two examples.

A configuration controller, which is incapable of performing any computations, is in charge of reconfiguring the array of functional units (FUs), which is made up of functional units with different levels of granularity. In contrast to hybrid architectures, arrays of functional units typically include dedicated configuration controllers as part of the reconfigurable array, and these controllers are limited to managing the sequencing of configurations; they cannot perform any computational tasks. The arrays of functional units also have a granularity ranging from fine grained to coarse grained. The functional unit arrays offer comparable interconnection options to those examined for hybrid architectures. Examples in this category include MATRIX.

III. METHODOLOGY

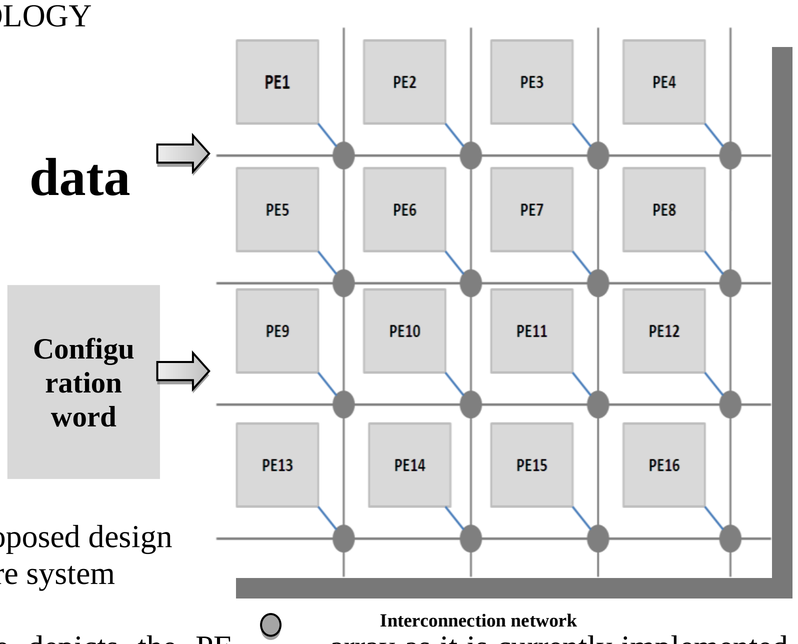


Figure 3: - Proposed design of multi-core architecture system

Figure 3 above depicts the PE array as it is currently implemented. It has a reconfigurable PE array that is more than enough to run most applications. Every PE is a reconfigurable, coarse-grained unit that can carry out different tasks. Via the interconnection network, it is connected to the closest neighboring PEs.
The following components make up the proposed architecture:
1) Processing Element; 2) Interconnection Network; and 3) Configuration Word
A vast array of Processing Elements (PEs) connected by a reconfigurable interconnect network and configuration memory make up the multi-core system architecture. The processing components are set up in a 4 x 4 matrix.

IV. DESIGN MODULES

4.1Processing Element

Three units make up the coarse-grained reconfigurable module that is the processing element that is designed. A logical unit can carry out logical operations like ANDing and ORing, an arithmetic unit can perform arithmetic operations like addition and multiplication, and a shifter block can carry out shifting operations.

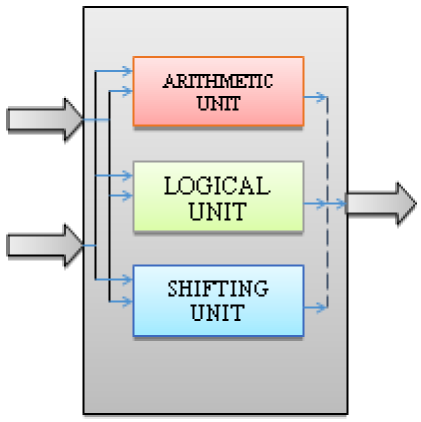


Figure 4: - Block diagram of Processing Element

4.2 Interconnection network

The communication between these functional blocks could be a significant bottleneck when there are more computational blocks available on a chip. Scalable on-chip communication mechanisms are therefore required. A programmable system for transferring data between the processing elements is called an interconnection network. Figure 5 illustrates the 4 to 1 multiplexer that serves as the interconnection network. It links the output of one processing element as an input to another processing element based on the input to select lines. The data from the left PE10, above PE7, diagonal upper left PE6, and one local input (PE11 will receive the data from these sources) are the inputs to the interconnection network. The output of the interconnection network is the input operand of the other processing element.

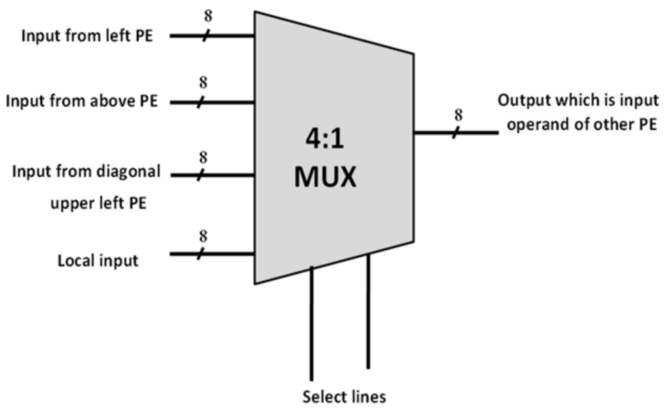

Input from left PE
8
Input from above PE
8
Input from diagonal upper left PE
8
Local input
8
4:1 MUX
8
Output which is input operand of other PE
Select lines

Figure 5: - Block diagram of Interconnection network

4.3 Configuration word:

The data needed to configure the interconnection network is stored in the configuration word. It is capable of reconfiguring itself by modifying the contents of the configuration word. It provides the data to the multiplexer's select lines so that it can operate properly.

V. DESIGNING OF VARIOUS MODULES

5.1 Creating the Processing Element Design

There are three units that make up the processing element. shifter block, logical unit, and arithmetic unit. The RTL schematic of all the blocks of processing element is shown in figure 6.

Three input ports (Operand1, Operand2, and Opcode) and one output port (Result) are present in each of these blocks. Eight bits make up the operand1, operand2, and opcode, and eight bits make up the output. These two operands carry out operations and provide the result based on the state of the opcode. The RTL schematic of processing element is shown in figure 7.

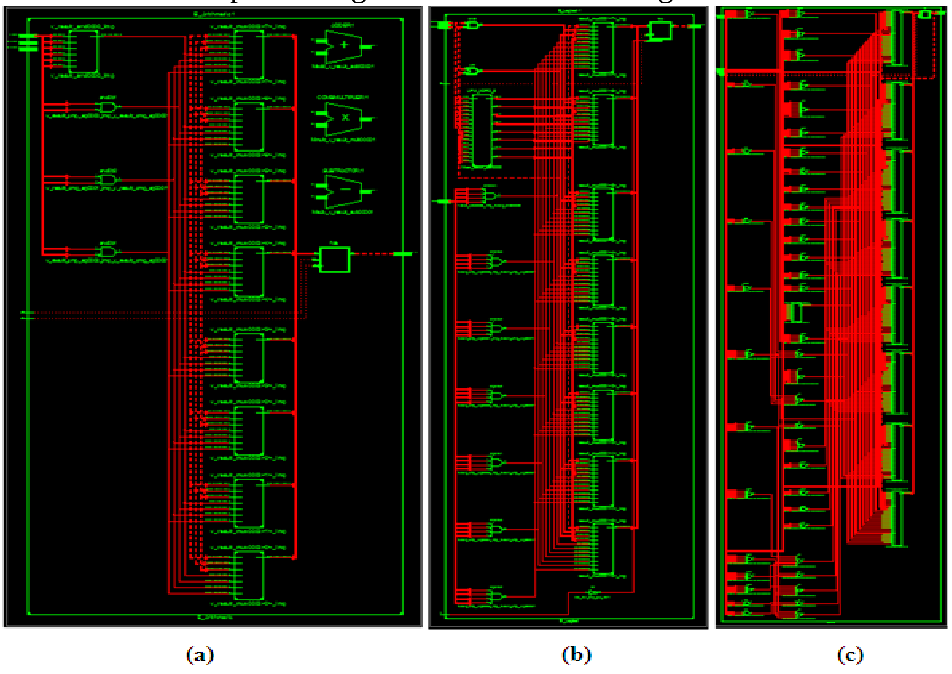


Figure 6: - RTL schematic of (a) Arithmetic unit (b) Logical unit (c) Shifter unit

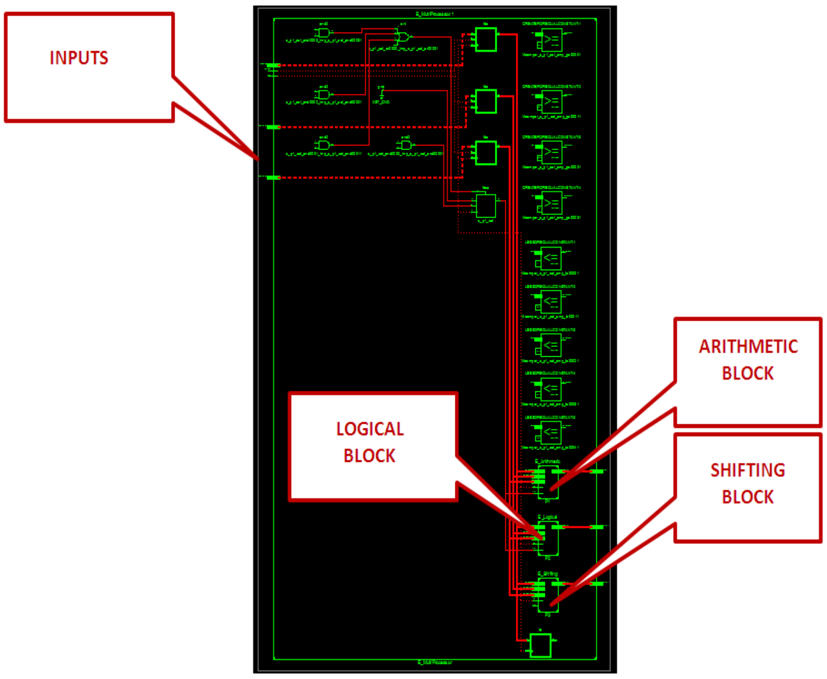


Figure 7: - RTL schematic of Processing element

The processing element has ports namely: data input opcode(7:0), operand1(7:0), operand2(7:0), clk, jmp_en and data output result(7:0), result1(7:0), result2(7:0)

data input operand1(7:0) and operand2(7:0) :- are the input 8 bit data on which the operation is to be performed.

data input opcode(7:0) :- decides which operation is to be performed on operand1 and operand2.

data input clk :- it is the clock signal.

data input jmp_en :- it is the enable signal.

data output result(7:0) :- it gives the output result of arithmetic block.

data output result1(7:0) :- it gives the output result of logical block.

data output result2(7:0) :- it gives the output result of shifting block.

5.2 Designing of Interconnection network

The interconnection network designed here is a 4 to 1 multiplexer. The are six ports namely: data input a(7:0), b(7:0), c(7:0), d(7:0) and data input sel(1:0), data output dout(7:0). The RTL schematic of interconnection network is shown below.

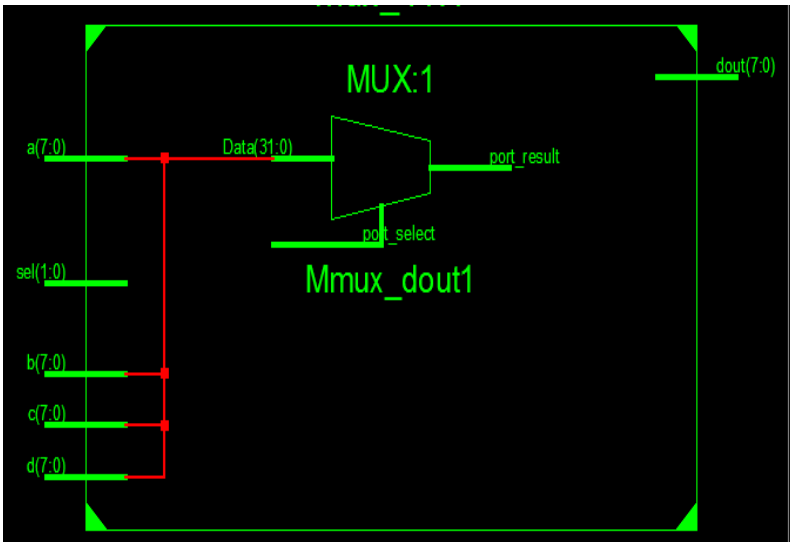


Figure 8: - RTL schematic of Interconnection network

data input a(7:0) :- input to the interconnection network from the left processing element.

data input b(7:0) :- input to the interconnection network from the left processing element.
data input c(7:0) :- input to the interconnection network from the left processing element.
data input d(7:0) :- input to the interconnection network of local operand.
data input sel(1:0) :- it is to select the input from the output of interconnection network and steer it to the output.
data output dout(7:0) :- output of the interconnection network which is the input operand of the other processing element.

5.3 Designing of Processing Element Array

The RTL schematic of processing element array which is a generic architecture is shown in figure 9 below.

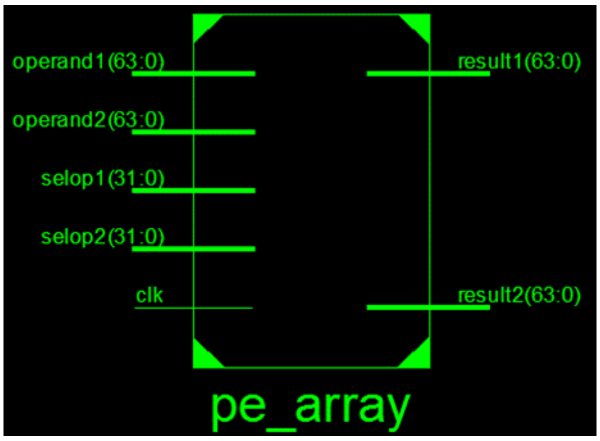


Figure 9: - Block schematic of Processing element Array

data input operand1(63:0) and operand2(63:0) :- are the inputs of 32 bit data on which the operation is to be performed.
data input selop1(31:0) and selop2(31:0) :- are the inputs given to select lines of interconnection network.
data input clk:- it is the clock signal.
data output result1(63:0) and result 2(63:0) :- are the output results

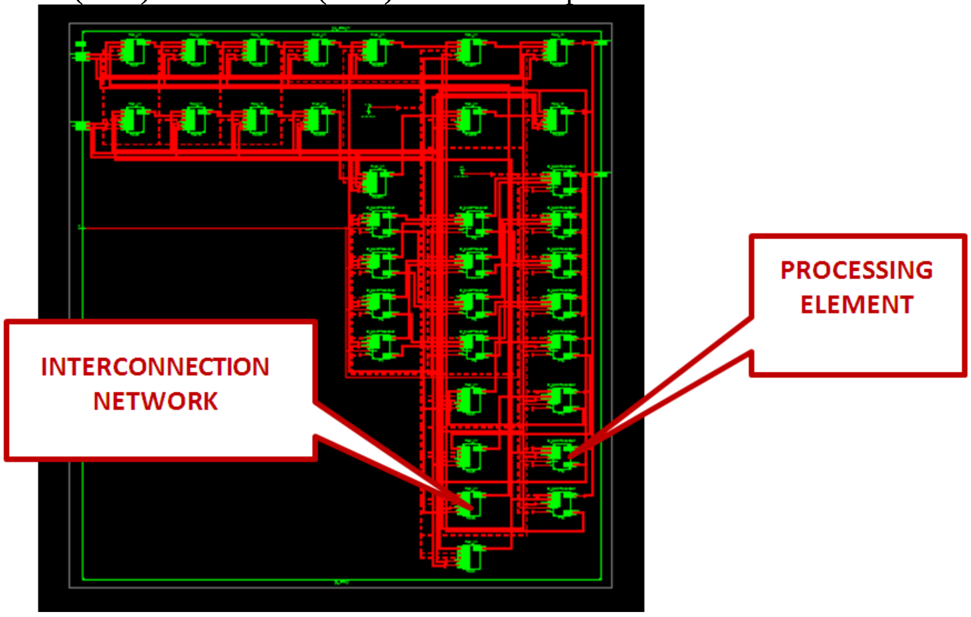


Figure 10: - RTL schematic of Processing element Array

5.4 Application design

Fast Fourier Transform (FFT), which is a quicker version of DFT, is a crucial method for analyzing the spectrum of digital signals. FFT is helpful in accelerating computation. It is known as a

decimation-in-time (DIT) FFT when used in the time domain. Decimation is the process of drastically lowering the quantity of computations done on time domain data. The butterfly structure is the fundamental component used in FFT calculations. The processing element's arithmetic block is the only one used for the butterfly operation. Using the following equations, the processing element determines the outcome of the radix 2 DIT butterfly structure depicted in figure 11.
Output 1 = input 1 + Input 2 x twiddle factor ……..(1)
Output 2 = input 1 - Input 2 x twiddle factor……..(2)
Where, twiddle factor is W_N^K

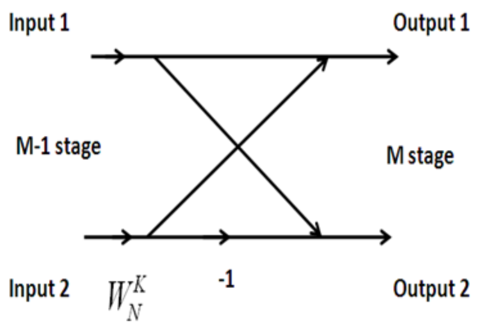
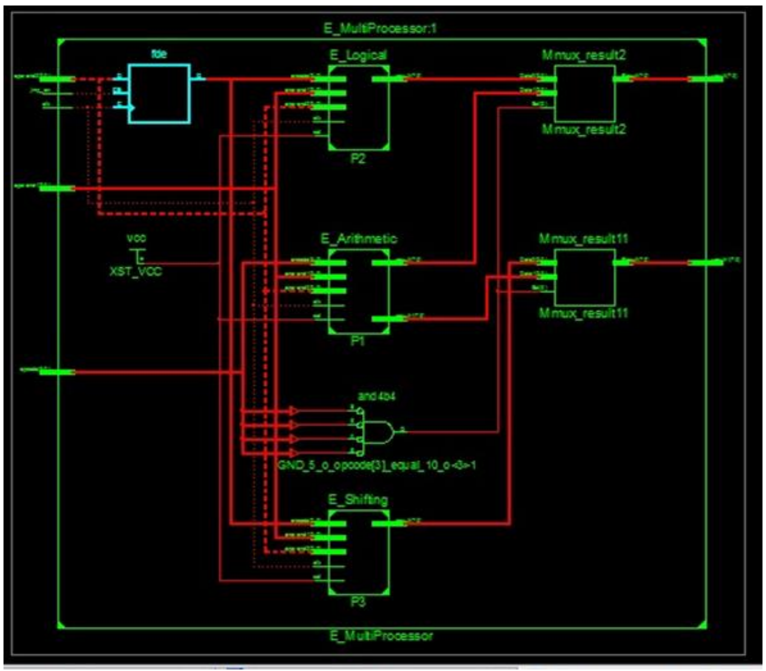


Figure 11: - Basic butterfly structure and its RTL schematic
It is possible to break down an N-point FFT into log2N stages. There are N/2 butterfly cells in each stage; each cell requires two additions and one multiplication. In order to apply the FFT effectively, the following observations are made: The quantity of butterflies in each stage is the same (number of butterflies = N/2, where N is the number of points). Each stage has the same number of DFT groups as (N/2stage). There is a 2stage-1 difference between the lower and upper legs. The group contains twice as many butterflies as stage-1.
Decimation in time FFT :
Number of stages = $\log_2 N$
Number of blocks/stage = N/2 ^{stage}
Number of butterflies/block = 2 $^{stage-1}$



Example : 8 point FFT

Number of stages :

- N $_{stages}$ = 3

Blocks/stage

- Stage 1: N $_{\text{blocks}}$ = 4
- Stage 2: N $_{\text{blocks}}$ = 2
- Stage 3: N $_{\text{blocks}}$ = 1

Butterflies/block

- Stage 1: N $_{\text{butterfly}}$ =1
- Stage 2: N $_{\text{butterfly}}$ = 2
- Stage 3: N $_{\text{butterfly}}$ = 4

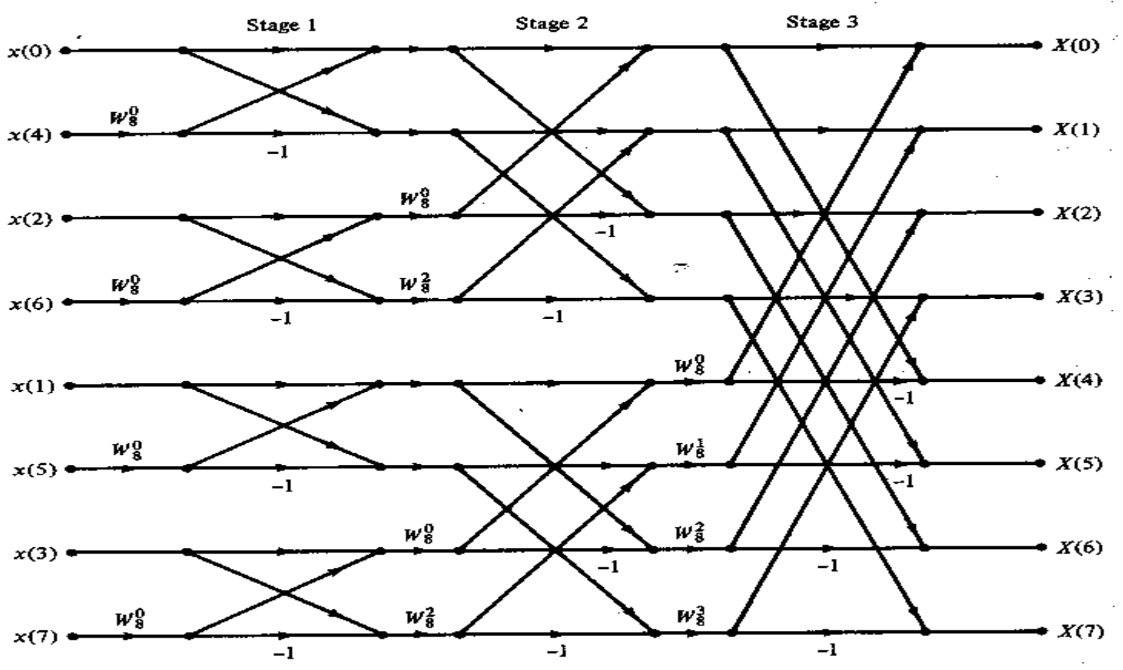
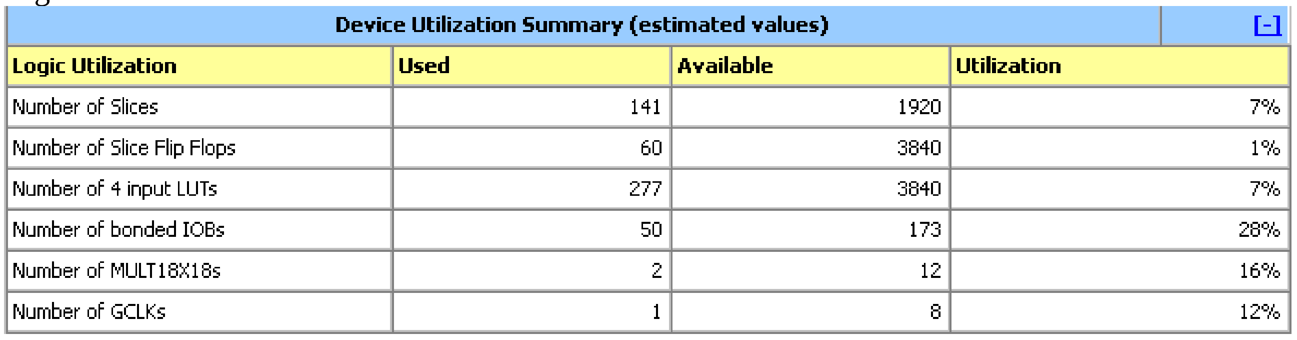


Figure 12: - 8 point decimation-in-time FF

VI. SIMULATION & RESULTS

The Xilinx ISE design suite 13.1 is used to synthesize and simulate each design module. Here, the target device is XC3S200-4FT256.



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	141	1920	7%
Number of Slice Flip Flops	60	3840	1%
Number of 4 input LUTs	277	3840	7%
Number of bonded IOBs	50	173	28%
Number of MULT18X18s	2	12	16%
Number of GCLKs	1	8	12%

Figure 13: - Synthesis report generated in Xilinx ISE 13.1 for Processing Element.

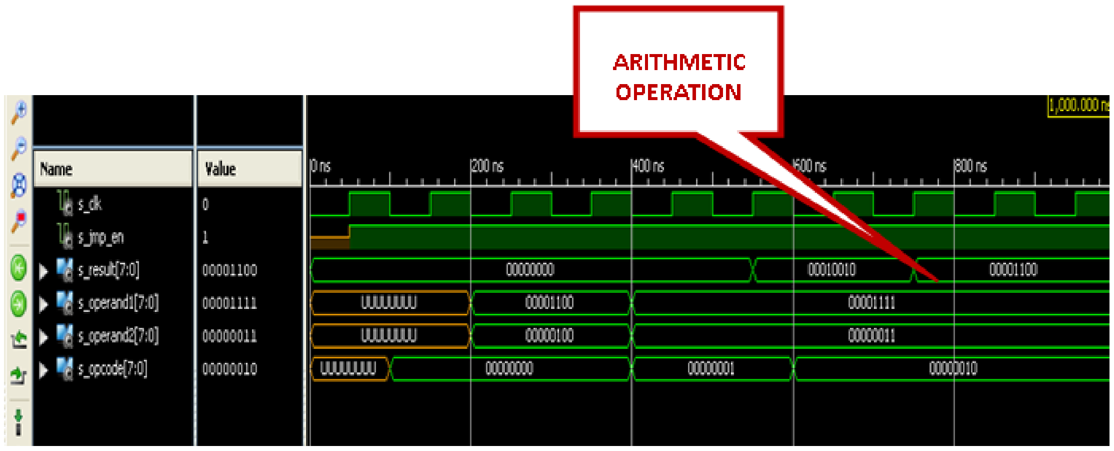
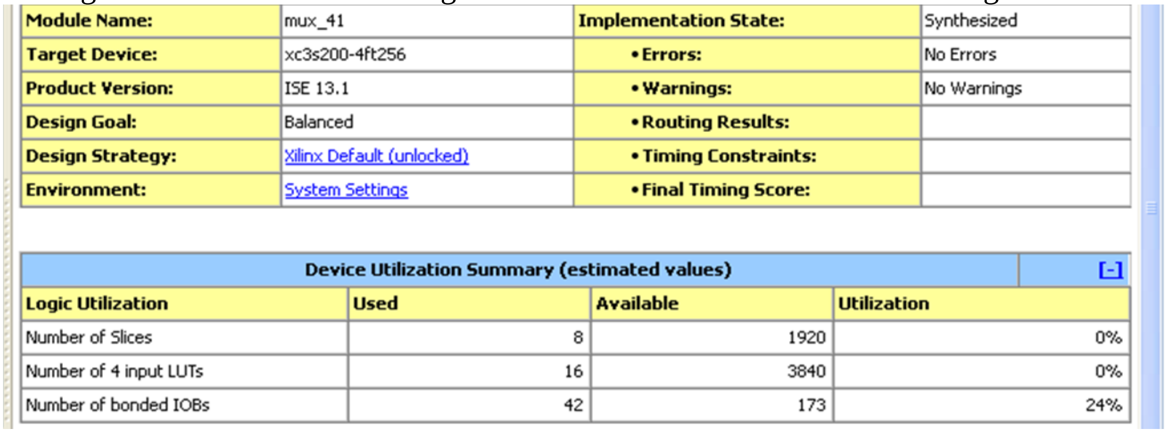


Figure 14: - Simulation result generated in Xilinx ISE 13.1 for Processing Element



Module Name:	mux_41	Implementation State:	Synthesized
Target Device:	xc3s200-4ft256	• Errors:	No Errors
Product Version:	ISE 13.1	• Warnings:	No Warnings
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	8	1920	0%
Number of 4 input LUTs	16	3840	0%
Number of bonded IOBs	42	173	24%

Figure 15: - Synthesis report generated in Xilinx ISE 13.1 for Interconnection network

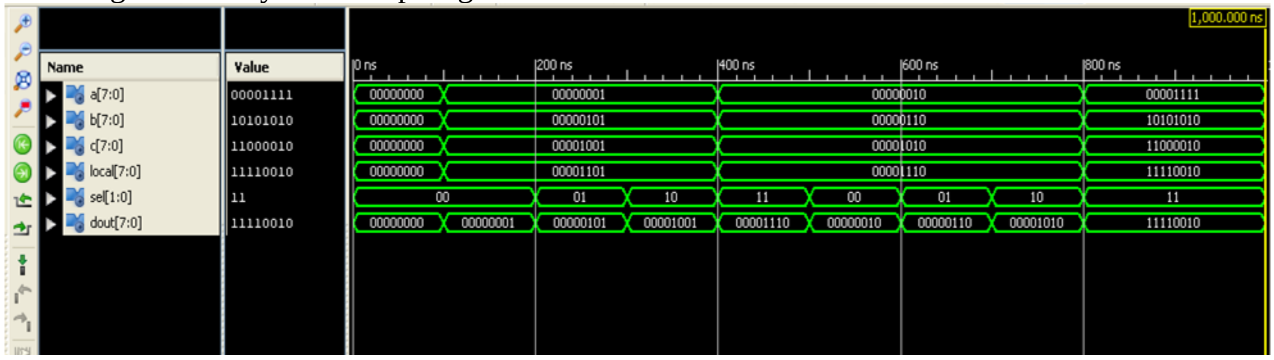
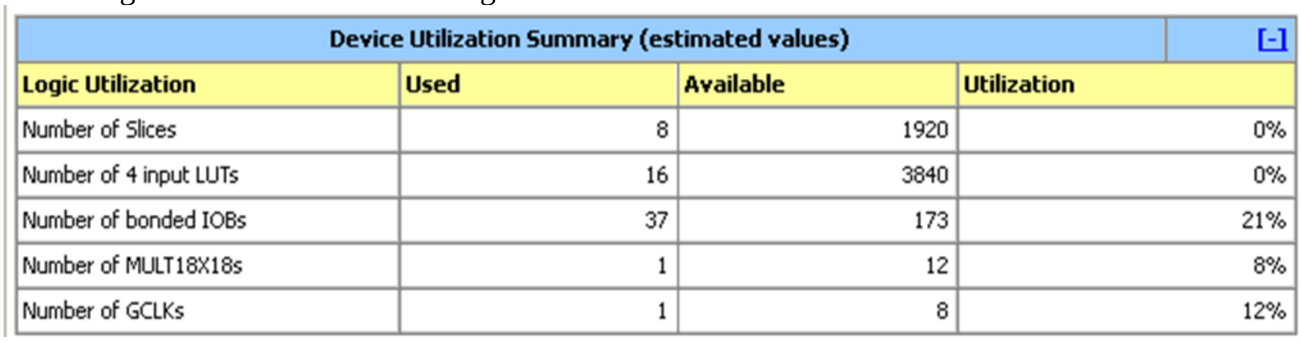


Figure 16: - Simulation result generated in Xilinx ISE 13.1 for Interconnection network



Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	8	1920	0%
Number of 4 input LUTs	16	3840	0%
Number of bonded IOBs	37	173	21%
Number of MULT18X18s	1	12	8%
Number of GCLKs	1	8	12%

Figure 17: - Synthesis report generated in Xilinx ISE 13.1 for PE array

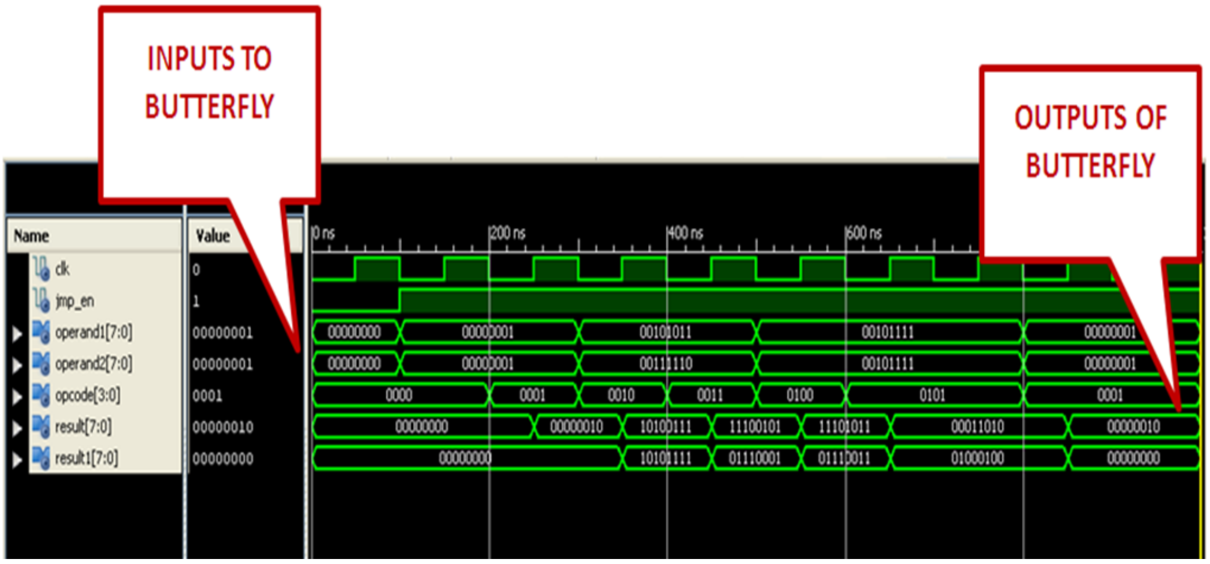


Figure 18: - Synthesis report generated in Xilinx ISE 13.1 for Butterfly Structure

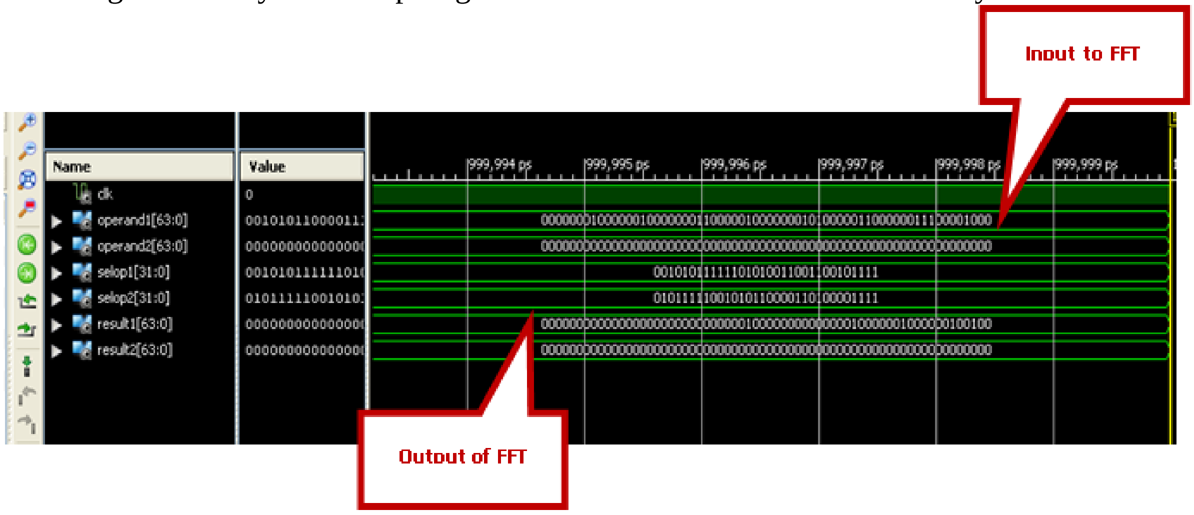
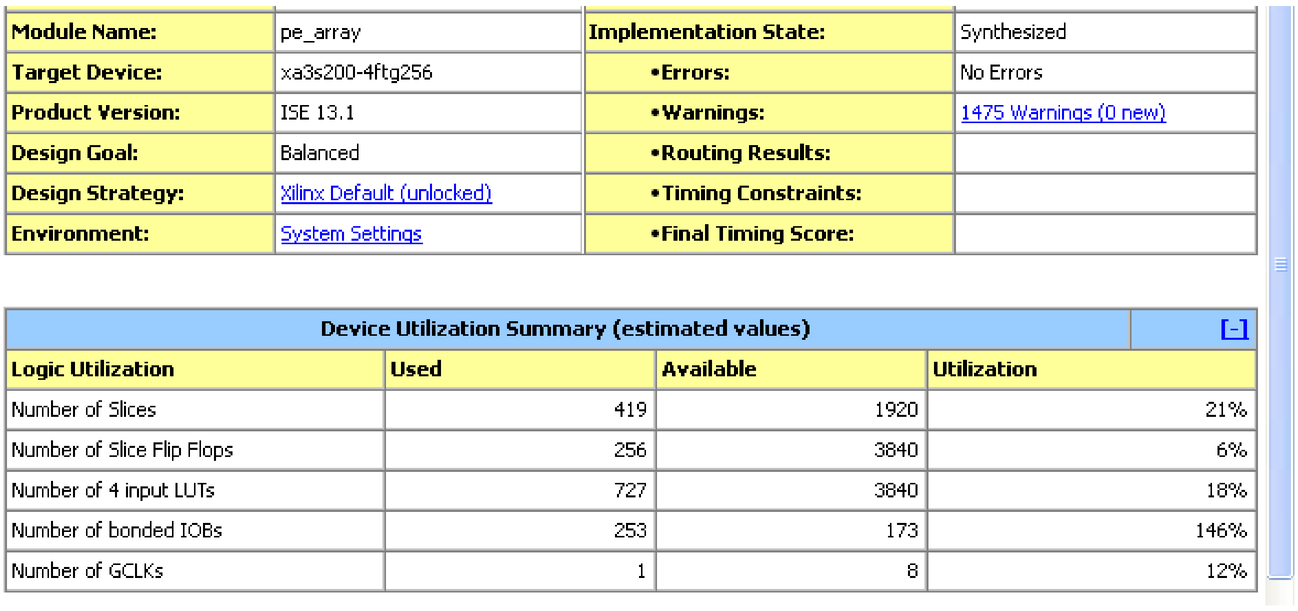


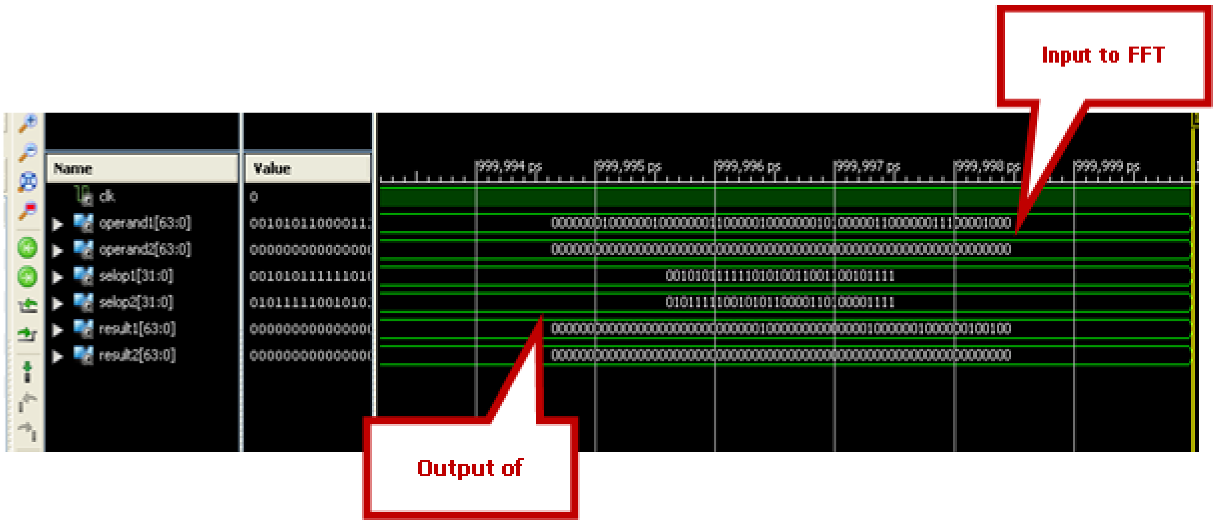
Figure 19:- Simulation result generated in Xilinx ISE 13.1 for Butterfly Structure



Module Name:	pe_array	**Implementation State:**	Synthesized
Target Device:	xa3s200-4ftg256	**•Errors:**	No Errors
Product Version:	ISE 13.1	**•Warnings:**	1475 Warnings (0 new)
Design Goal:	Balanced	**•Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	
Environment:	System Settings	**•Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	419	1920	21%
Number of Slice Flip Flops	256	3840	6%
Number of 4 input LUTs	727	3840	18%
Number of bonded IOBs	253	173	146%
Number of GCLKs	1	8	12%

Figure 20:- Synthesis report generated in Xilinx ISE 13.1 for PE array



Name	Value	
clk	0	
operand1[63:0]	001010110000011	0000000100000001000000011000000100000000101000000110000000111000001000
operand2[63:0]	000000000000000	0000000000000000000000000000000000000000000000000000000000000000
selop1[31:0]	001010111111010	00101011111101010011001100101111
selop2[31:0]	010111110010101	01011111001010101100001101000001111
result1[63:0]	000000000000000	0000000000000000000000000000000100000000000001000000010000000100100
result2[63:0]	000000000000000	0000000000000000000000000000000000000000000000000000000000000000

Figure 21:- Simulation result generated in Xilinx ISE 13.1 for Application design FFT
FFT Input = {8,7,6,5,4,3,2,1} and Assuming all Twiddle factor = 1
FFT Output = {36,16,8,0,4,0,0,0}

VII. CONCLUSION & FUTURE SCOPE

Xilinx ISE 13.1 is used to implement the suggested design modules. According to the synthesis report, on the target device xc3s200-4-ft256, a single processing element takes up 7% of the available space, whereas an array of 16 processing elements takes up 98% of the available space. The Fast Fourier Transform (FFT) is tested with various sample inputs and implemented using VHDL.

A heterogeneous multi-core system could replace the current multi-core homogeneous system, increasing functionality to a greater extent. (Different units are capable of carrying out various tasks.) Processing cores from could work closely together in such architectures while also carrying out specific tasks for which they were created.

The proposed multi-core architecture currently has a straightforward interconnection network and is challenging to scale to many-core architectures. Future research would employ more sophisticated interconnection methods, like network-on-chip, to enhance the architecture. User-defined applications can be developed using the proposed design to carry out specific tasks.

Wireless networking has many inherent functional and data parallelism applications, especially in the field of digital signal processing, which can be taken advantage of with a suggested multi-core architecture.

References

[1] "*Scalable Computing in the Multicore*" Era,Xian-He Sun, Yong Chen and Surendra Byna, in Proc. of International Symposium on Parallel Algorithms, Architectures and Programming (PAAP'08), 2008.
[2] "*Reevaluating Amdahl's law in the multi-core era*" Xian-He Sun_, Yong Chen, Computer Science Department, Illinois Institute of Technology, United States, J. Parallel Distrib. Comput. 70 (2010) 183_188
[3] R. W. Hartenstein, "*A decade of reconfigurable computing: A Visionary Retrospective*." in Proceedings of conference of Design, Automation and Test in Europe DATE, 2001, pp. 642–649.
[4] Hartenstein Reiner, "*Coarse grain reconfigurable architecture (embedded tutorial),*"6th Asia and South Pacific Design Automation Conference (ASP-DAC '01), 2001.