# Field Programmable Gate Array (FPGA): A Tool For Improving Parallel Computations

By

Mrs. Abisoye Opeyemi A. (Msc) Computer Science; Department of Computer Science, Federal University of Technology, Minna, Niger State, Nigeria. E-mail: opeglo@yahoo.com.au, o.abisoye@futminna.edu.ng

E.O Justice Emuoyefarche (PhD) Computer Engineering Department; Ladoke Akintola University of Technology, Ogbomoso, Oyo State, Nigeria. E-mail: eojustice@gmail.com, eojustice@justice.com

**Abstract**

*Platform Multicore Processor,* Complex Programmable Logic Devices (CPLDs) *Application-Specific Integrated Circuit (ASIC), Graphic Processing Unit(GPU), Vector Processor, Massively parallel processor (MPP), and Symmetric Multiprocessor (SMP) have become key components for dealing with parallel applications. Within parallel computing, there is a specialized parallel device called Field Programmable Array (FPGA) remain niche areas of interest. While not domain-specific, it tends to be applicable to only a few classes of parallel problems. This research work explores the underlying parallel architecture of this Field Programmable Gate Array (FPGA). The design methodology, the design tools for implementing FPGAs is discussed e.g. System Generator from Xilinx, Impulse C programming model etc. FPGA design in compares with other technology) is envisaged. In this research work FPGA typically exploits parallelism because FPGA is a parallel device. With the use of simulation tool, Impulse Codeveloper (Impulse C), FPGA platform graphical tools that provide initial estimates of algorithm throughput such as loop latencies and pipeline effective rates are generated. Using such tools, you can interactively change optimization options or iteratively modify and recompile C code to obtain higher performance.*

**Index Terms**: *Application-specific integrated circuit (ASIC),* Complex Programmable Logic Devices (CPLDs, Field Programmable Gate Array (FPGA), *Graphic Processing Unit(GPU),* Impulse Codeveloper(Impulse C), *Massively parallel processor (MPP), Multicore Processor, Symmetric Multiprocessor (SMP), Vector Processor.*



## 1 INTRODUCTION

Parallel Computing allows for the dissection of large data sets into smaller sets, each to be handled by separate processors. This significantly increases the performance and handling of processing intensive application. With the emergence of multicore computers, we are facing the challenge of parallelizing performance-critical applications of all sorts. Compared to sequential applications, our repertoire of tools and methods for cost-effectively developing reliable, parallel applications is spotty. [5]

Within parallel computing, there is a specialized parallel device called Field Programmable Array (FPGA) that remain niche areas of interest. While not domain-specific, it tends to be applicable to only a few classes of parallel problems. The historical roots of FPGAs are in complex programmable logic devices (CPLDs) of the early to mid 1980s. A Xilinx cofounder, Ross Freeman, invented the field programmable gate array in 1985. CPLDs and FPGAs include a relatively large number of programmable logic elements. CPLD logic gate densities range from the equivalent of several thousand to tens of thousands of logic gates, while FPGAs typically range from tens of thousands to several million [11]

The primary differences between CPLDs and FPGAs are architectural. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers. The result of this is less flexibility, with the advantage of more predictable timing delays and a higher logic-to-interconnect ratio. The FPGA architectures, on the other hand, are dominated by interconnect. This makes them far more flexible (in terms of the range of designs that are practical for implementation within them) but also far more complex to design for. Another notable difference between CPLDs and FPGAs is the presence in most FPGAs of higher-level embedded functions (such as adders and multipliers) and embedded memories, as well as to have logic blocks implements decoders or mathematical functions. [15].

Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation and a growing range of other areas. FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs. As their size, capabilities, and speed increased, they began to take over larger and larger functions to the state where some are now marketed as full systems on chips (SOC).

FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture. One such area is code breaking, in particular brute-force attack, of cryptographic algorithms.

The inherent parallelism of the logic resources on the FPGA allows for considerable compute throughput even at a sub-500 MHz clock rate. For example, the current (2007) generation of FPGAs can implement around 100 single precision floating point units, all of which can compute a result every single clock cycle. The flexibility of the FPGA allows for even higher

performance by trading off precision and range in the number format for an increased number of parallel arithmetic units. This has driven a new type of processing called reconfigurable computing, where time intensive tasks are offloaded from software to FPGAs. [15]

## 2.0 AIMS AND OBJECTIVE

It is known that using *Multicore Processor, Application-specific integrated circuit (ASIC), Graphic Processing Unit(GPU), Vector Processor, Massively parallel processor (MPP), and Symmetric Multiprocessor (SMP)* more difficulties are still in existence to solve parallel applications. Such difficulties are low massive parallelism, inability to interactively change optimization options to obtain higher performance, inability to update the functionality after shipping. To eradicate such difficulties, the full architecture of "Field *Programmable Logic Array" is envisaged.*

Objectives

1. To explore challenges of parallel computations.
2. To describe the need of an effective parallel device to eradicate such challenges
3. To describe how typical parallel application is implemented on FPGA using Impulse Codeveloper from Xilinx Generator as simulator.
4. To write an algorithm for the application
5. To translate the algorithm into C++ program having a great deal of FPGA specific hardware knowledge.
6. To compile the resulting optimized c code by the FPGA development tools (in particular the c-to hardware compiler) to create a parallel hardware/software implementation.

## 3.0 RESEARCH METHOD

The execution of this research work is divided into phases and the goals are achieved through phases that include:

1. Description of the complete application in C++ language and use a standard C++ debugger to verify the algorithm.
2. Profiling the application to find the computational "hot spots".
3. Use of data streaming, message passing and/or shared memory to partition the algorithm into multiple communicating software and hardware processes.
4. Use of interactive optimization tools to analyze and improve the performance of hardware-accelerated functions.
5. Use of C++-to-hardware compiler to generate synthesizable hardware, in the form of hardware description language files.

## 4.0 CHALLENGES OF PARALLEL COMPUTATIONS.

1. Low massive parallelism
2. Inability to interactively change optimization options to obtain higher performance, inability to update the functionality after shipping.
3. Finding concurrency in a program - how to help programmers "think parallel"?
4. Scheduling tasks at the right granularity onto the processors of a parallel machine.
5. The data locality problem: associating data with tasks and doing it in a way that our target audience will be able to use correctly.
6. Scalability support in hardware: bandwidth and latencies to memory plus interconnects between processing elements.
7. Scalability support in software: libraries, scalable algorithms, and adaptive runtimes to map high level software onto platform details.

## 5.0 FPGA TECHNOLOGY:

A field-programmable gate array (FPGA) is a semiconductor device that can be configured by the customer or designer after manufacturing—hence the name "field-programmable". To program an FPGA you specify how you want the chip to work with a logic circuit diagram or a source code in a hardware description language (HDL). FPGAs can be used to implement any logical function that an application-specific integrated circuit (ASIC) could perform, but the ability to update the functionality after shipping offers advantages for many applications.

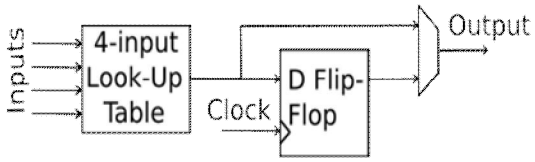
FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

At the highest level, FPGAs are reprogrammable silicon chips. Using prebuilt logic blocks and programmable routing resources, you can configure these chips to implement custom hardware functionality without ever having to pick up a breadboard or soldering iron.

### 5.1 FPGA Architecture:

The typical basic architecture consists of an array of configurable logic blocks (CLBs) and routing channels. Multiple I/O pads may fit into the height of one row or the width of one column in the array. Generally, all the routing channels have the same width (number of wires).An application circuit must be mapped into an FPGA with adequate resources.

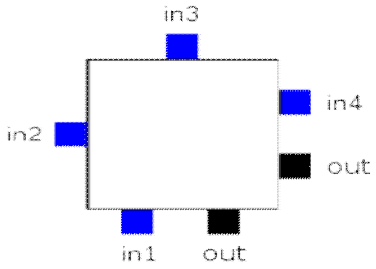
A classic FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop, as shown below. In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance.



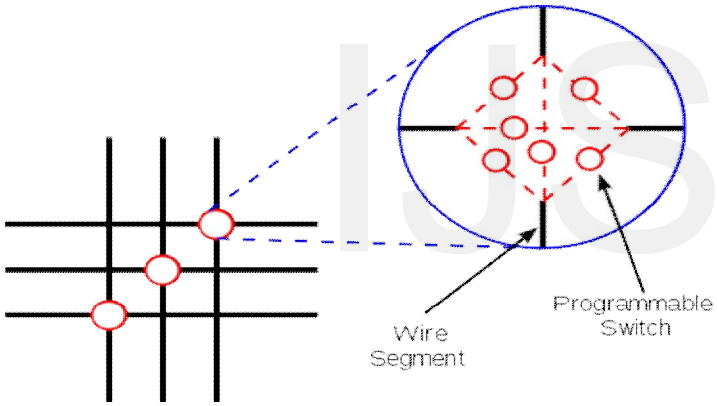

Typical logic block

There is only one output, which can be either the registered or the unregistered LUT output. The logic block has four inputs for the LUT and a clock input. Since clock signals (and often other high-fan-out signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed.

For this example architecture, the locations of the FPGA logic block pins are shown be-




low.




Logic Block Pin Locations Switch box topology

Each input is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block.Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it.

Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it.

Generally, the FPGA routing is unsegmented. That is, each wiring segment spans only one logic block before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths can be constructed. For higher speed interconnect, some FPGA architectures use longer routing lines that span multiple logic blocks.

Whenever a vertical and a horizontal channel intersect, there is a switch box. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. In this switch box topology, a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. The figure below illustrates the connections in a switch box.

### Switch Box Topology

Modern FPGA families expand upon the above capabilities to include higher level functionality fixed into the silicon. Having these common functions embedded into the silicon reduces the area required and gives those functions increased speed compared to building them from primitives. Examples of these include multipliers, generic DSP blocks, embedded processors, high speed IO logic and embedded memories.

FPGAs are also widely used for systems validation including pre-silicon validation, post-silicon validation, and firmware development. This allows chip companies to validate their design before the chip is produced in the factory, reducing the time to market

### 5.2 Benefits of FPGA Technology

1. Performance
2. Time to Market
3. Cost
4. Reliability
5. Long-Term Maintenance
6. Efficiency

1. Performance – Taking advantage of hardware parallelism, FPGAs breaks the paradigm of sequential execution and accomplishing more per clock cycle. BDTI, a noted analyst and benchmarking firm, released benchmarks showing how FPGAs can deliver many times the processing power per dollar of a DSP solution in some applications. Controlling inputs and outputs (I/O) at the hardware level provides faster response times and specialized functionality to closely match application requirements.

2. Time to market – FPGA technology offers flexibility and rapid prototyping capabilities in the face of increased time-to-market concerns. You can test an idea or concept and verify it in hardware without going through the long fabrication process of custom ASIC design. You can then implement incremental changes and iterate on an FPGA design within hours instead of weeks. Commercial off-the-shelf (COTS) hardware is also available with different types of I/O already connected to a user-programmable FPGA chip. The growing availability of high-level software tools decrease the learning curve with layers of abstraction and often include valuable IP cores (prebuilt functions) for advanced control and signal processing.

3. Cost – The nonrecurring engineering (NRE) expense of custom ASIC design far exceeds that of FPGA-based hardware solutions. The large initial investment in ASICs is easy to justify for OEMs shipping thousands of chips

per year, but many end users need custom hardware functionality for the tens to hundreds of systems in development. The very nature of programmable silicon means that there is no cost for fabrication or long lead times for assembly. As system requirements often change over time, the cost of making incremental changes to FPGA designs are quite negligible when compared to the large expense of repining an ASIC.

4. Reliability – While software tools provide the programming environment, FPGA circuitry is truly a "hard" implementation of program execution. Processor-based systems often involve several layers of abstraction to help schedule tasks and share resources among multiple processes. The driver layer controls hardware resources and the operating system manages memory and processor bandwidth. For any given processor core, only one instruction can execute at a time, and processor-based systems are continually at risk of time-critical tasks preempting one another. FPGAs, which do not use operating systems, minimize reliability concerns with true parallel execution and deterministic hardware dedicated to every task.

5. Long-term maintenance – As mentioned earlier, FPGA chips are field-upgradable and do not require the time and expense involved with ASIC redesign. Digital communication protocols, for example, have specifications that can change over time, and ASIC-based interfaces may cause maintenance and forward compatibility challenges. Being reconfigurable, FPGA chips are able to keep up with future modifications that might be necessary. As a product or system matures, you can make functional enhancements without spending time redesigning hardware or modifying the board layout.

6. Efficiency:-More computation with less power consumption. FPGA offers the best ratio of GFlops/Watt.

### 5.3 FPGA Design and Programming

FPGAs can be programmed with hardware description languages such as VHDL or Verilog. However, programming in these languages can be tedious. Several vendors have created C to HDL languages that attempt to emulate the syntax and/or semantics of the C programming language, with which most programmers are familiar. The best known C to HDL languages are Mitrion-C, Impulse C, DIME-C, and Handel-C.To simplify the design of complex systems in FPGAs, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called *IP cores*, and are available from FPGA vendors.

### 5.31 Impulse Co-Developer (Impulse C)

Impulse C is software to hardware compiler (simulator) that generates hardware to software interfaces. It is a set of library functions that support parallel programming for FPGAs using the C language. Impulse C optimizes C code for parallelism and generates HDL ready for FPGA synthesis. It moves compute –intensive functions to FPGA. The Impulse C approach focuses on the mapping of algorithms to mixed FPGA / processor systems. It creates highly parallel, dataflow-oriented application. Impulse C simplifies the creation of highly parallel algorithms, including mixed software / hardware algorithms, through the use of well defined data communication, message passing, and synchronization mechanisms.

## 6.0 EXPERIMENTATION WITH THE SIMULATOR

The source code is divided into various blocks and each block simulation in the hardware platform is shown. The pipeline stages, the latency, effective rate, and number of samples generated per each cycle for each block is shown in the source code. This will help in evaluating the acceleration and the performance of each algorithm.

Graphical Representation of Synthesis of Code using the Simulator (Impulse Codeveloper Pipeline stages

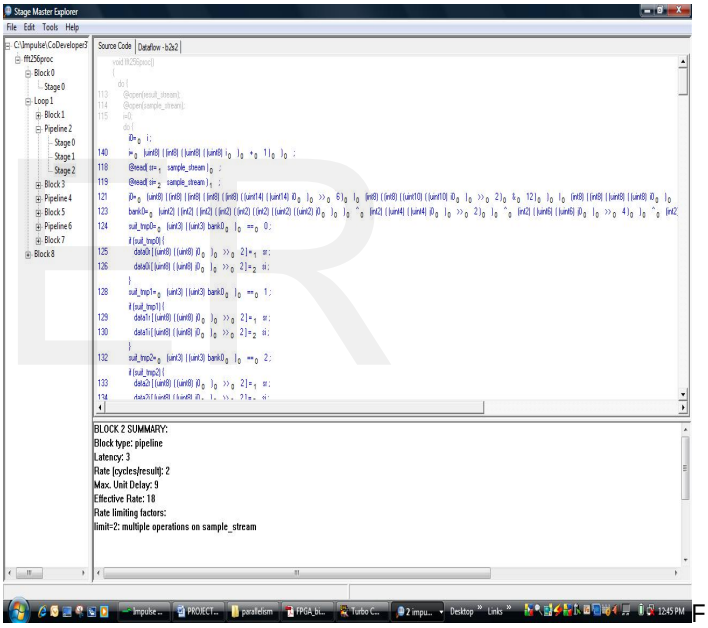



Fig 1a. Showing the source code of pipeline2, generating latency of 3, effective rate 18, 2samples/cycle and Maximum Unit delay of 9

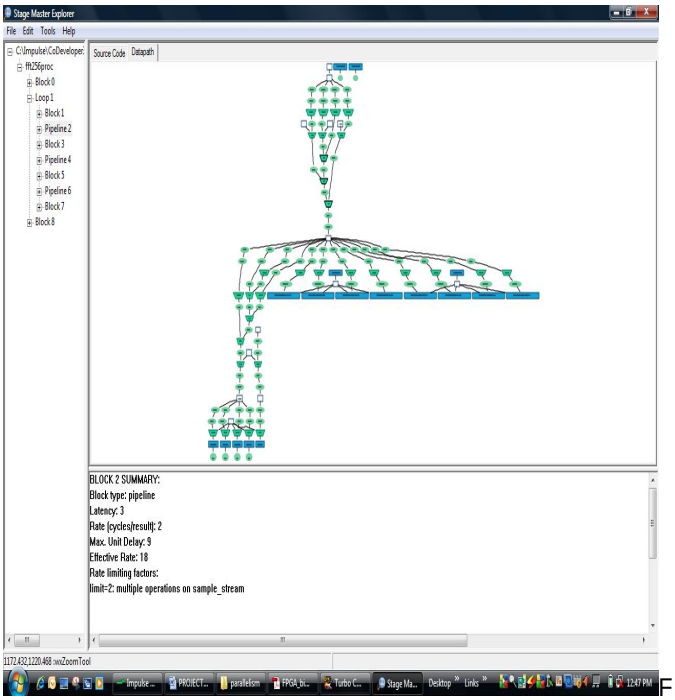



Fig 1b. Showing the datapath of pipeline2 generating latency of 3, effective rate 18, 2samples/cycle and Maximum Unit delay of 9

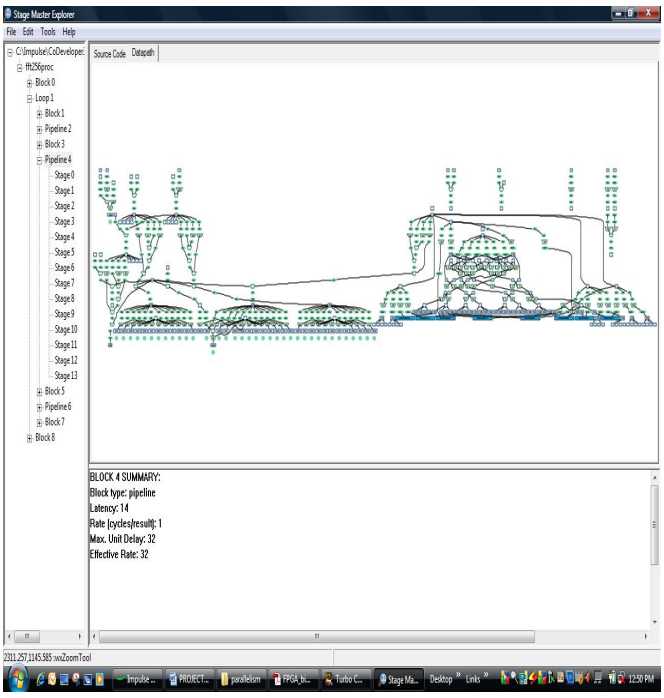



Fig 2b. Showing the datapath of pipeline4 generating latency of 14, effective rate 32, 1sample/clockcycle and Maximum Unit delay of 32

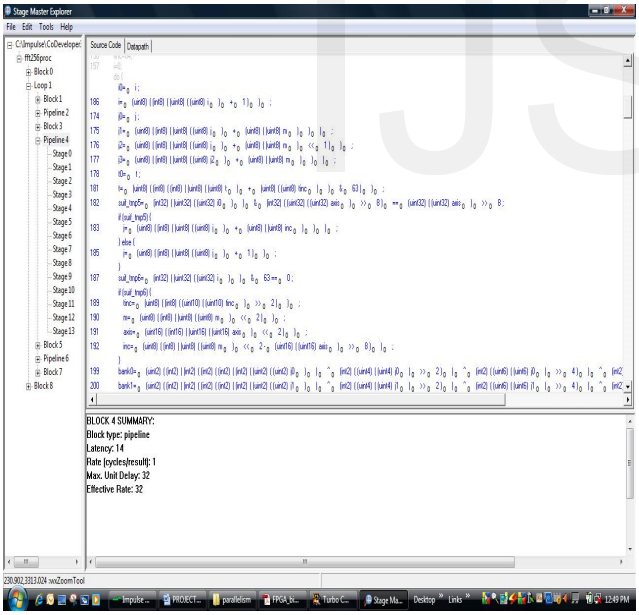



Fig 2a. Showing the source code of pipeline4 generating latency of 14, effective rate 32, 1sample/clockcycle and Maximum Unit delay of 32

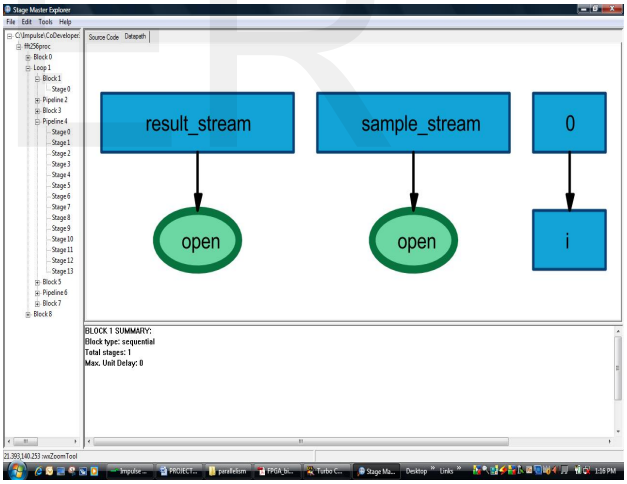



Fig 3. With simulator (Block Stages) , Max unit Delay of 0

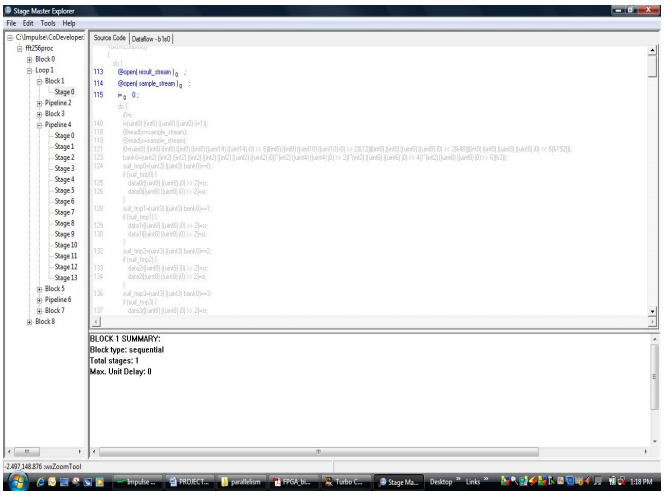


Fig 4 . Showing the source code of the block stage Without Simulator

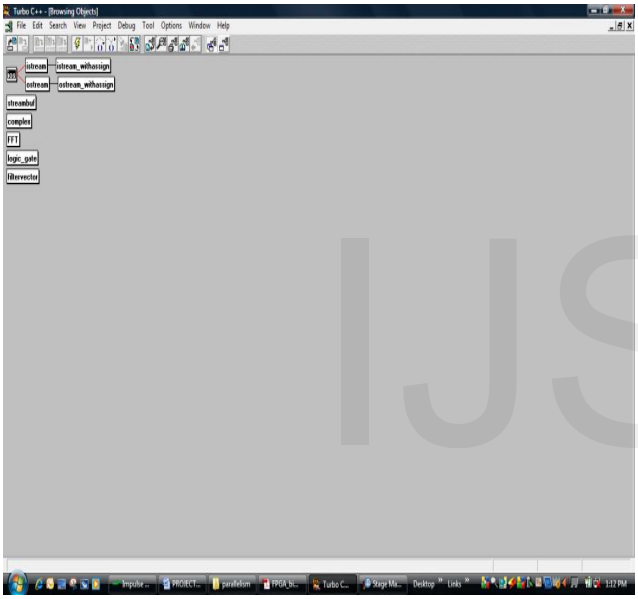


Fig 5 The modules or classes are shown while the pipeline rate, effective rate cannot be determined

## 7.0 DISCUSSION OF RESULTS

Results show the use of FPGA as a tool to improve parallel computations with the use of Impulse C tools as simulator. Graphical tools ( Fig 1a,2a) above show the source code wile (Fig 1b,2b) show the datapath and help to provide initial estimates of algorithm throughput such as loop latencies and pipeline effective rates. Using such tools, you can interactively change optimization options or iteratively modify and recompile C code to obtain higher performance. Such design iterations may take only a matter of minutes when using C, whereas the same iterations may require hours of even days when using VHDL or Verilog.

Moreover, Impulse C-tools uses optimization techniques to increase the performance of the code being used for an application without having a great deal of FPGA-specific hardware knowledge. We have also shown that pipelining introduces a potentially high degree of parallelism in the generated logic, allowing us to achieve the best possible throughput.

## 8.0 FINDINGS: PERFORMANCE EVALUATION / COMPARATIVE ANALYSIS

| | | |
|---|---|---|
| 1. | WITHOUT FPGA SIMULATOR | WITH FPGA SIMULATOR |
| 2. | It does not generate hardware program | It generates hardware program simultaneously |
| 3. | The number of adders, comparators cannot be calculated. | The number of adders, comparators can be calculated |
| 4. | The number of samples generated per cycle can't be determined. | The number of samples generated per cycle determined. |
| 5. | The code cannot be easily pipelined to speed up the processing of filters. | It uses pragma co-unroll & pragma co-pipeline to pipeline the code and processing of filters. |
| 6. | It can easily be used for fixed format of filters. | It can be used for fixed & complex filters. |
| 7. | No graphical representation of synthesis of code. | It generates graphical representation to show synthesis of code in blocks. |
| 8. | The synthesis of code processing flow cannot be seen. | The synthesis of flow of blocks & statements can be shown |
| 9. | Does not generate hardware description language. | It generates hardware description language |
| 10. | Presence of low level embedded functions | Presence of higher level embedded functions (such as adders & multipliers) and embedded memories as well as logic blocks to implement decoders or mathematical functions. |

## 9.0 CONCLUSION AND FURTHER RESEARCH

In conclusion, this research has described FPGA as a tool to improve to parallel applications by acting as a co-processor than conventional processors. We have shown that the underlying parallel architecture of FPGA helps in moving expensive computations from the CPU and into the specifically designed logic inside the FPGA and thus obtaining high performance at an economical price.

FPGAs are becoming easier to use as the development tools get better and as the prices on FPGAs falls smaller / denser chip manufacturing technology becomes available, thus making them affordable to use in more computing applications.

Therefore, trends of FPGAs have now proved a better alternative to traditional processors such as ASIC for a growing number of higher-volume applications. Further research can focus on , hardware or software interfacing and FPGA tool development.

## 10.0 RECOMMENDATION

Based on the results and findings above we now recommend the use of Field programmable gate array (FPGA) as the best technology to solve parallel applications rather than using conventional processors because it increases the computational speed . FPGA technology offers these advantages; reliability, low cost, high performance, long term maintenance, little time to market and efficient compared to conventional processors.

## REFERENCES

[1] Anthony S. & Lan P.(2006),- "The design of anew FPGA Architecture", Friday, Jan 20, BDTI Focus Report: FPGAs for DSP, Second Edition, BDTI Benchmarking.

[2] Acken, Kevin P.; Irwin, Mary Jane; Owens, Robert M. (1 January 1998). *The Journal of VLSI Signal Processing* 19 (2): 97–113. doi:10.1023/A:1008005616596

[3] D.Ryle, D.Popig, E.Stahlberag (2006). – "Applying FPGA to Biological Problems".

[4] D'Amour, Michael R., CEO DRC Computer Corporation. "Standard Reconfigurable Computing". Invited speaker at the University of Delaware, February 28, 2007.

[5] Gottlieb, Allan; Almasi, George S. (1989). *Highly parallel computing*. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1.

[6] Jason Cong & Kenneth Yan,(2000). Department of Computer Science, University of California, Los Angeles, *International Symposium on Field Programmable Gate Arrays Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays* Monterey, California, United States Pages: 75 - 82 Year of Publication: 2000 ISBN:1-58113-193-3

[7] Maslennikov, Oleg (2002). "Systematic Generation of Executing Programs for Processor Elements in Parallel ASIC or FPGA-Based Systems and Their Transformation into VHDL-Descriptions of Processor Element Control Units". *Lecture Notes in Computer Science*, 2328/2002: p. 272.

*[8]* M. Thompson,(2006). "*The Field-Programmable Gate Array* "(FPGA): Expanding Its Boundaries, InStat Market

[9] Moreno, W.A.; Poladia, K.,(1998). "*Field programmable gate array design for an application specifics Signal Processing algorithms"* Devices, Circuits and Systems. Proceedings of the 1998 Second IEEE International Caracas Conference on Digital Object Identifier 10.1109/ICCDCS.1998.705837 Volume 1 , Issue , 2-4 Mar 1998 Page(s):222 – 225 Research, April 2006.

[10] M. Thompson, (2004). *FPGAs accelerate time to market for industrial designs*, EE Times http://www.us.design-reuse.com/articles/8190/fpgas-accelerate-time-to-market-for-industrial-designs.html

[11] Peter Clarke(2001), "Xylinx, ASIC Vendors Talk Licensing", Retrieved February 10,2009.

[12] Patterson and Hennessy, (2007). *Computer organization and design : the hardware/software interface* (2. ed., 3rd print. ed.). San Francisco: Kaufmann. ISBN 1-55860-428-6. p. 537 & 714

[13] Kirkpatrick, Scott (2003). "Computer Science: Rough Times Ahead". *Science*, Vol. 299. No. 5607, pp. 668 - 669. DOI: 10.1126/science.1081623

[14] Shimokawa, Y.; Y. Fuwa and N. Aramaki, (1991). "A parallel ASIC VLSI neurocomputer for a large number of neurons and billion connections per second speed". *International Joint Conference on Neural Networks* 3: 2162–2167. doi:10.1109/IJCNN.1991.170708. ISBN 0-7803-0227-3.

[15] htttp://en.wikipedia.org/wiki/Field-Programmable-_gate_array