APPLICATION OF FIELD PROGRAMMABLE GATE ARRAY TO DIGITAL SIGNAL PROCESSING

O.A. Abisoye
Department of Computer Science, Federal University of Technology, Minna, Nigeria
E-mail: blesutunde@yahoo.com

Abstract
This work shows how one parallel technology Field Programmable Gate Array (FPGA) can be applied to digital signal processing problem to increase computational speed. The best algorithm for solving Digital Signal Processing Applications; Fast Fourier Transform (FFT) algorithm has shown significant speed improvement when implemented on a FPGA. The design methodology, the design tools for implementing DSP functions in FPGAs is discussed e.g. System Generator from Xilinx, Impulse C programming model etc. FPGA design in compares with other technolog) 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), of FPGA platform on FFT algorithm, 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.

Keywords: Platform Programmable Digital Signal Processors Digital Signal Processing (DSP), Field Programmable Gate Array (FPGA)

Introduction

Throughout the history of computing, digital signal processing applications have pushed the limits of computing power, especially in terms of real-time computation. While processed signals have broadly ranged from media driven speech, audio and video wave forms to specialized radar and sonar data, most calculations performed by signal processing systems have exhibited the same basic computational characteristics.

DSP algorithms have long been run on standard computers, on specialized processors called digital signal processors (DSPs) or on purpose built hardware such as Application Specific Integrated Circuits (ASICs). Recently, DSP has received increased attention due to rapid advancement in multimedia computing and high speed wired and wireless communication. Today, there are additional technologies used for digital signal processing including more powerful general purpose microprocessors, field programmable gate arrays(FPGAs), digital signal controllers(mostly for industrial applications such as motor control and stream processors).

The inherent data parallelism found in many Digital Signal Processing (DSP) functions has made DSP algorithms ideal candidates for hardware implementation, leveraging expanding Virtual Level Scale Integrated Circuit (VLSI) capabilities.

In Digital Signal Processing applications of FPGA includes: Digital image processing, Speech/Audio signal processing, Telecommunication, Biomedical, RADAR,SONAR, and Robotics

FPGAs are increasingly used in conventional High Performance computing applications where computational kernels such as FFT or convolution are performed on the FPGA instead of a microprocessor.

Objective

It is known that using Programmable Digital Signal Processors (PDSPs) and Application Specific Integrated Circuits (ASICs) more difficulties are still in existence to solve digital signal processing applications. To eradicate such difficulties, the possible solutions are been envisaged "Field *Programmable Logic Array.*

A radix 2 FFT algorithm was posed and then implements the algorithm on FPGA platform using Impulse Codeveloper from Xilinx Generator as simulator. The algorithm is translated into C++ program having a great deal of FPGA specific hardware knowledge.
The resulting optimized c code is compiled by the FPGA development tools (in particular the c-to hardware compiler) to create a parallel hardware/software implementation.

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.

Approaches to FPGA application development

Digital signal processing

Digital signal processing (DSP) is concerned with the representation of the signals digitally as sequences of numbers or symbols and the processing of these signals to extract information from the signals.

Digital signal processing key operations:
The basic DSP operations include convolution, correlation, filtering, transformations and modulation.

FPGA technology:

A **field-programmable gate array** 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.

DSP implementation

Digital signal processing is often implemented using specialized microprocessors such as the DSP56000, the TMS320, or the SHARC. These often process data using fixed-point arithmetic, although some versions are available which use floating point arithmetic and are more powerful. For faster applications with vast usage, ASICs might be designed specifically. For slow applications, a traditional slower processor such as a microcontroller may be adequate. For faster applications FPGAs might be used.

Technique for the implementation:

Fast Fourier Transform

Fast Fourier Transform (FFT) is a fast approach to compute Discrete Fourier Transform (DFT)**.** It is of O(nlog n) while DFT is of $O(n)^2$. The number of operations required is obviously of $O(n)^2$ order. But due to transform properties it is possible to reduce the number of operations to the order of $O(n\log_2 n)$. Historically, DFT is the origin discrete version of FFT from continuous version

For a continuous function of one variable f(x), the Fourier Transform F(k) will be defined as an integral of the form:

$$F(k) = \int f(x)\, e^{-\frac{i2\pi kx}{N}}\, dx \qquad \text{(i)}$$

Where F(k) is fourier transform of kth harmonic, x is consecutive voltage values $e^{-\frac{i2\pi}{N}}$ is twiddle factor and f(x)= x(nT), T is time series for n values

The transform operates in complex domain. Recall, that imaginary exponent could be written as:

$$e^{i\phi}=\cos(\phi)+i\sin(\phi) \qquad \text{(ii)}$$

For sampled function continuous transform (i) turns into discrete one:

$$F(k)=\sum_{n=0}^{N-1} f(n)\, e^{-\frac{i2\pi kn}{N}} \qquad \text{(iv)}$$

Expression (iv) is *discrete Fourier transform — DFT*. Here $\{f_0, f_1, \ldots, f_{N-1}\}$ is input discrete function and $\{F_0, F_1, \ldots, F_{N-1}\}$ is result of Fourier transform.

where $e^{-\frac{2\pi}{N}}$ is an Nth Primitive root of unity, Let us put N=8 and write down our DFT:

$$F_n = f_0 + f_1 e^{-i\frac{2\pi}{8}n} + f_2 e^{-i\frac{2\pi}{8}2n} + f_3 e^{-i\frac{2\pi}{8}3n} + f_4 e^{-i\frac{2\pi}{8}4n} + f_5 e^{-i\frac{2\pi}{8}5n} + f_6 e^{-i\frac{2\pi}{8}6n} + f_7 e^{-i\frac{2\pi}{8}7n} \quad \text{(viii)}$$

We can split the sum into two by separating odd and even terms and factoring out the latter sum:

$$F_n = \left[f_0 + f_2 e^{-i\frac{2\pi}{8}2n} - f_4 e^{-i\frac{2\pi}{8}4n} + f_6 e^{-i\frac{2\pi}{8}6n} \right] + e^{-i\frac{2\pi}{8}n} \left[f_1 + f_3 e^{-i\frac{2\pi}{8}2n} - f_5 e^{-i\frac{2\pi}{8}4n} - f_7 e^{-i\frac{2\pi}{8}6n} \right] \quad \text{(ix)}$$

Worked examples

1. With the assumption of this sequence {1,0,0,1} that has been processed. The data represents four consecutive voltages x(0) = 1, x(T) = 0, x(2T) = 0, x(3T) = 1, recorded vat time intervals T. The value X(k) is then calculated from N=4 : thus k= 0, k=1,k=2 and k=3(since N-1=3)

   X(k) =

   a. When k^{th} harmonic = 0 then we have

   =

   = x(0) + x(T) + x(2T) + x(3T)
   = 1 +0 + 0 +1 =2

Thus: When k^{th} harmonic = 0 then we have X(k) = 2 is entirely real of magnitude 2 and phase angle □(o)=0

   b. When k^{th} harmonic = 1 then we have

   X(k) =

= x(0) + x(T) + x(2T) + x(3T) = 1 + 0 +0 + 1( )

Since =Cos(□) +iSin(□) = 1+Cos(3 /2) –iSin(3 /2)

Then X(k) = 1 + Cos(□) **- i**Sin(□) = **1+i**

Thus X(1) = 1+i and is complex with magnitude √2 and phase angle □ = $\tan^{-1}1 = 45$

It has now been shown that the time series {1,0} has the Discrete Fourier Transform(DFT) by the complex sequence {2,1+i}

Impulse C

Impulse C is software to hardware compiler 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 is ideal for many DSP applications because 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.

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.

Discussion of results

Results show that the parallel implementation of FFT achieves linear speed-up and real-time performance for large matrix sizes. This was achieved by the use of FPGA technology that uses Impulse C tools as simulator.

Graphical tools (Fig 1,3) showing the source code(Fig 2,4) showing 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.

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

Pipeline stages

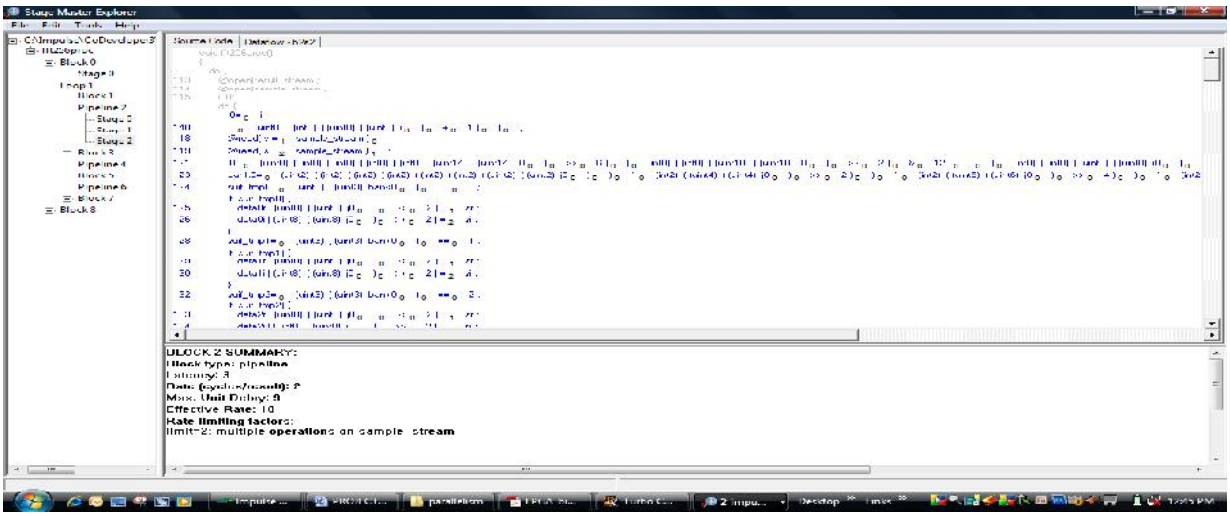


Fig 1. Showing the source code of pipeline2 generating latency of 3, effective rate 18, 2samples/cycle and Maximum Unit delay of 9

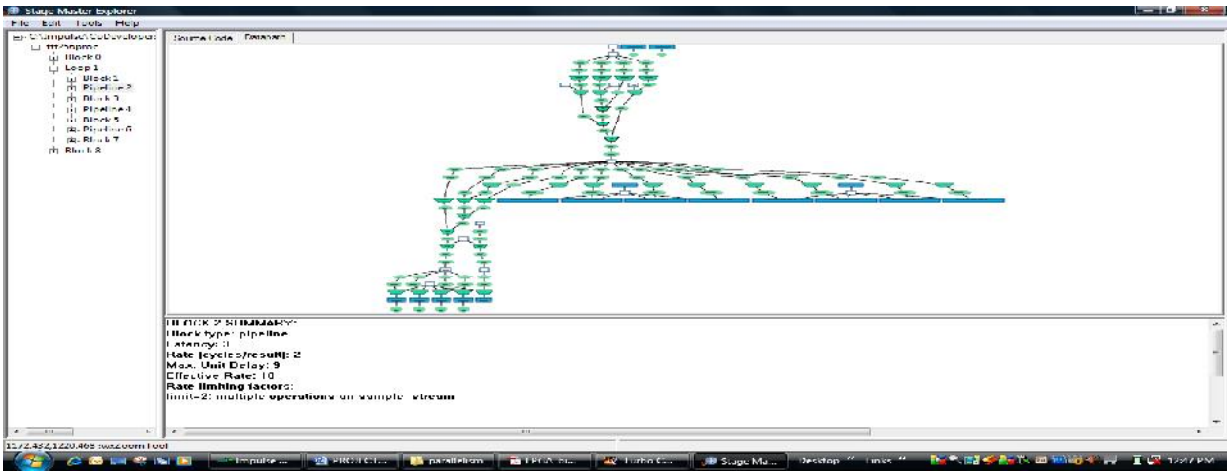


Fig 2. Showing the datapath of pipeline2 generating latency of 3, effective rate 18, 2samples/cycle and Maximum Unit delay of 9

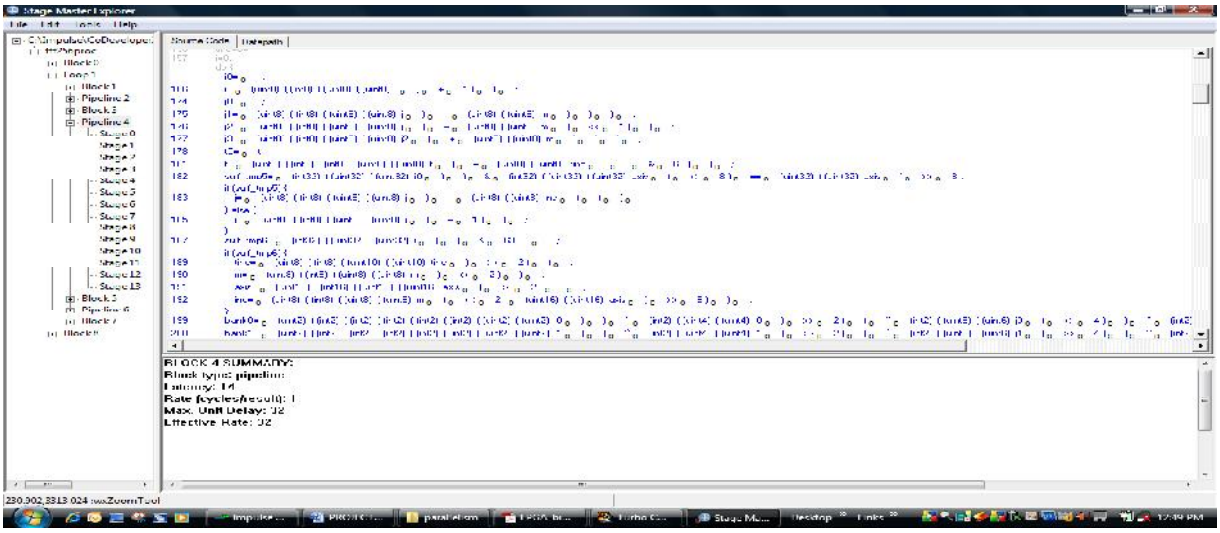


Fig 3. Showing the source code of pipeline4 generating latency of 14, effective rate 32, 1sample/clockcycle and Maximum Unit delay of 32

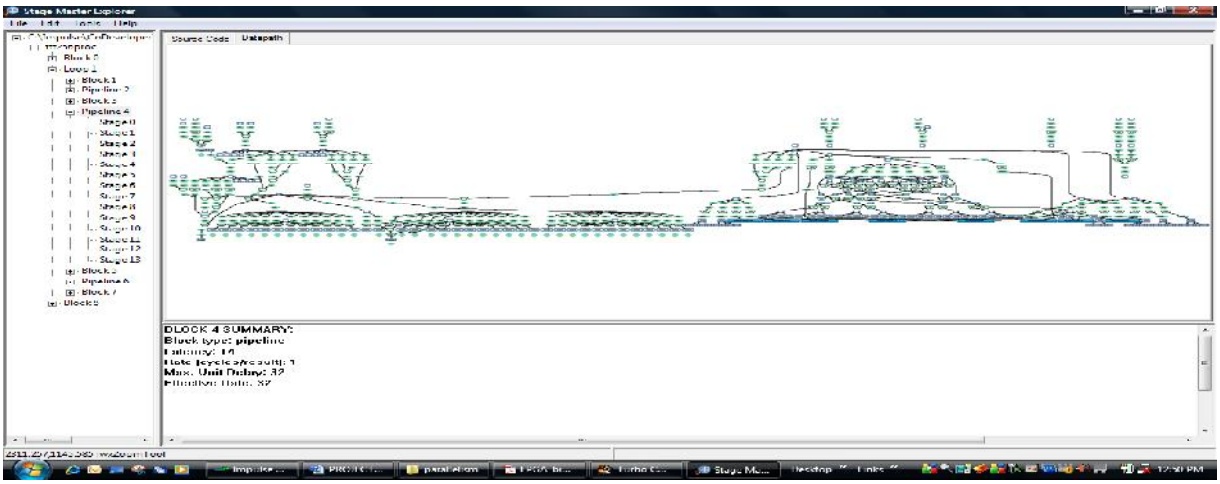
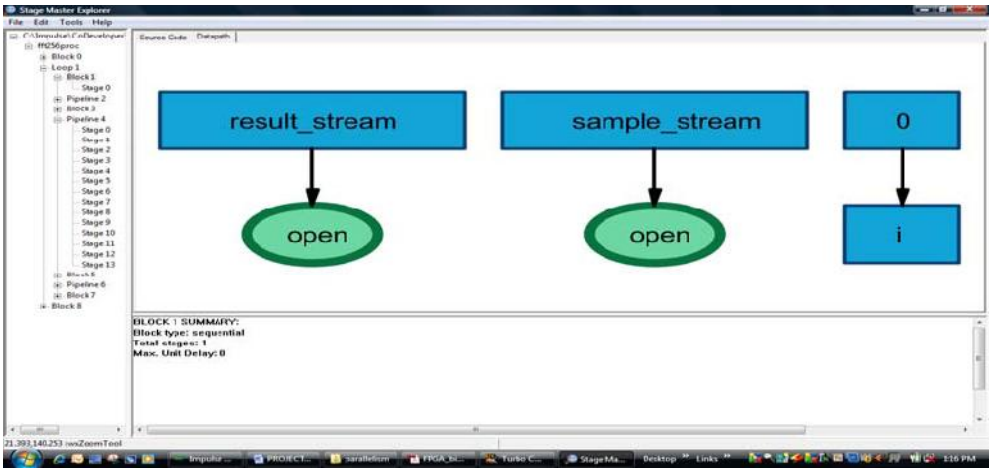


Fig 4. Showing the datapath of pipeline4 generating latency of 14, effective rate 32, 1sample/clockcycle and Maximum Unit delay of 32

With simulator (Block Stages) Fig 5. Max unit Delay of 0



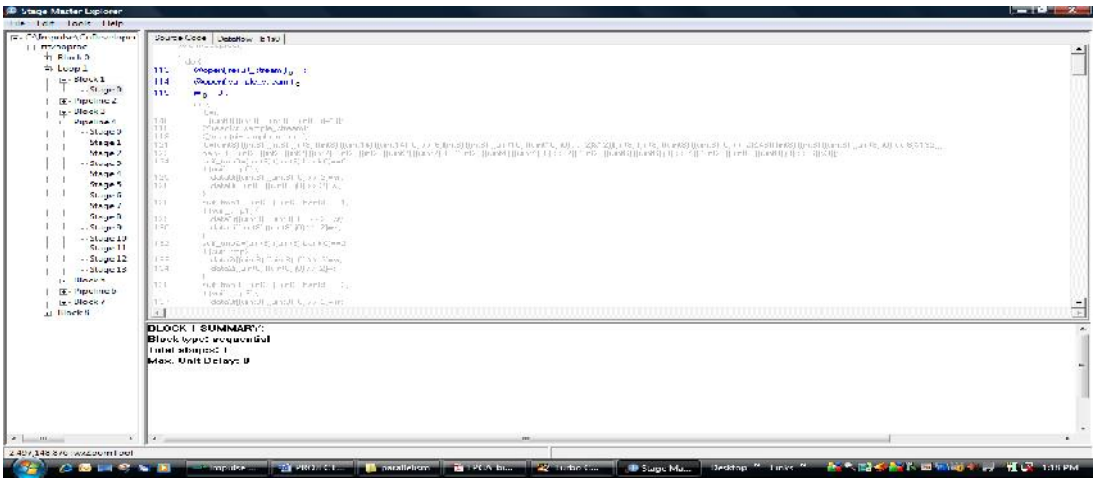


Fig 6. Showing the source code of the block stage

Without Simulator

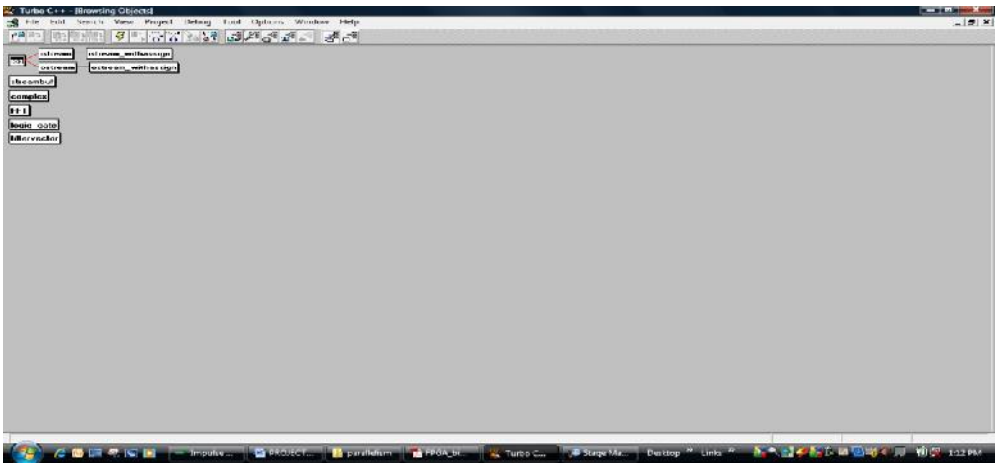


Fig 7. The modules or classes are shown while the pipeline rate, effective rate cannot be determined

8.0 Findings: Performance Evaluation/Comparative Analysis

	WITHOUT FPGA SIMULATOR	WITH FPGA SIMULATOR
1.	It makes use of Discrete Fourier Transform Algorithm Formula $X(k) = \sum_{n=0}^{N-1} X(n)\, e^{-\frac{i2\pi kn}{N}}$ to generate output filters.	It uses Impulse Co-Developer as simulator to generate output filter from input filters supplied.
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
	The number of samples generated per cycle can't be determined.	The number of samples generated per cycle determined.
4.	The code cannot be easily pipelined to speed up the	It uses Pragma CO-UNROLL & pragma CO-

	processing of filters.	PIPELINE to pipeline the code and processing of filters.
5.	It can easily be used for fixed format of filters.	It can be used for fixed & complex filters.
6.	No graphical representation of synthesis of code.	It generates graphical representation to show synthesis of code in blocks.
7.	The synthesis of code processing flow cannot be seen.	The synthesis of flow of blocks & statements can be shown
8.	Does not generate hardware description language.	It generates hardware description language
9.	It can easily be used to implement radix 2.	it can easily be used to implement radix 2, radix 4, algorithm.
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.

Conclusion

In conclusion, this project has described the benefits of using an FPGA as a DSP co-processor than conventional processors. We have shown that DSP algorithm can take advantage of FPGAs as a viable resource to improve highly computationally expensive digital signal processing by moving expensive computations from the CPU and into the specifically designed logic inside the FPGA and thus obtaining high performance at an economical price.

Algorithms such as Fast Fourier Transform have shown significant speed improvement when implemented on a FPGA. 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

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 digital signal processing applications rather than using conventional processors because it increases the computational speed of filters. FPGA technology is reliable and efficient compared to conventional processors.

References

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

Dag S. and William W. (2004). "*Digital Signal Processing and Applications"* (2nd ed. Elsevier.

Edition: 3, illustrated, revised Published by Springer, 2007, ISBN 3540726128,FPGAs accelerate time to market for industrial designs, EE Times 7/2/2004 http://www.us.design-reuse.com/articles/8190/fpgas-accelerate-time-to-market-for-industrial-designs.html

Gregory R.(1995): "A Guide to Using Field Programmable Gate Arrays (FPGAs) *for* Application-Specific Digital Signal Processing Performance"

Jason, C. and Kenneth, Y.(2000): *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

Maya G. and Paul S. 2006- "*Reconfigurable Computing- Digital Signal Processing Applications".*

Moreno, W.A.; Poladia, K.(1998): **"***Field programmable gate array design for an application specifics Signal Processing algorithms"* Devices, Circuits and Systems, 1998. 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.

Roger W. and John M. (2008). "*FPGA-based Implementation of Signal Processing Systems"*

Russel T. and Wayne, B. (1999), "Reconfigurable Computing For Digital Signal Processing: Survey. Department of Electrical and Computer Engineering, University of Massachusetts, Amherst. MA01003, USA.

Ryle,D. Popig, D. and Stahlberag, V. (2006). "Applying FPGA to Biological Problems"

Thompson, M.(2000): "*The Field-Programmable Gate Array "*(FPGA): Expanding Its Boundaries, *InStat Market* .

Uwe M. B. (2006): "*Digital Signal Processing With Field Programmable Logic Arrays*" 2006, 2nd Edition