FPGA implementation of a frame delay

Kwasi Gyening Afrifa
University of Windsor

FPGA IMPLEMENTATION OF A FRAME DELAY

By

Kwasi Gyening Afrifa

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

September, 2010

Your file Votre référence
ISBN: 978-0-494-70581-0
Our file Notre référence
ISBN: 978-0-494-70581-0

DECLARATION OF ORIGINALITY

ABSTRACT

The objective of this thesis is to investigate the applicability of Field Programmable Gate Arrays (FPGAs) for frame delay implementation. FPGAs are programmable devices that can be directly configured by the end user without the use of an integrated circuit fabrication facility. They offer the designer the benefits of custom hardware, eliminating high development costs and manufacturing time. Frame delays are easier to realize using R/W memory where data is written into the memory and read out for each frame. FPGAs are used in a Quartus II environment as it is easy to perform frame delay implementation using schematic entry procedure. Since FPGAs use look-up tables as configurable logic blocks, they are considered as an appropriate choice for frame delay based designs.

DEDICATION

To our Lord Jesus Christ and to my mother Mrs Joyce Nortey Johnson.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to both of my co-supervisors Dr. Maher Sid-Ahmed and Dr. Rashid Rashidzadeh for giving me the opportunity to work with them. I am also grateful to them for their encouragement, support and guidance during my research work presented in this thesis. I would also like to thank the thesis committee members Dr. Gaspar and Dr. Tepe for their comments and advice.

I am also thankful to my parents Mr. Edward G. Afrifa and Mrs. Joyce Johnson, for their prayers and moral support. A lot of thanks to my siblings: Ama Asantewaa and Afua for their support and affection. I would also like to express my appreciation to Mr. and Mrs. Masse for their kindness. My thanks also go to Mr. and Mrs. Balga for the accommodation afforded me during my stay in Canada. I also extend my gratitude to Andria Ballo who helped ensure I completed my studies. I also thank my friends and colleagues who helped make my stay in this university a memorable one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
CAD	Computer-Aided Design
CLB	Configurable Logic Block
clk	clock
CPLD	Complex Programmable Logic Device
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
EPROM	Erasable Programmable Read-Only Memory
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDL	Hardware Description Language
IIR	Infinite Impulse Response
I/O	Input/Output
IOE	Input/Output Element
LAB	Logic Array Block
LE	Logic Element
LED	Light Emitting Diode
LIFO	Last In First Out

LUT	Look-up Table
PAL	Programmable Array Logic
PLA	Programmable Logic Array
PLL	Phase-Locked Loop
PLD	Programmable Logic Device
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
R/W	Read/Write
sclr	Synchronous clear
SDRAM	Synchronous Dynamic Random Access Memory
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
SSI	Small Scale Integration
VHDL	(Very-high speed integrated circuit) hardware description language
VLSI	Very Large Scale Integration
wren	write enable

CHAPTER 1

INTRODUCTION

1.1 Motivation

In this thesis, a new application for FPGAs is explored. FPGAs are considered for frame delay implementation. This thesis is to help realize a cost effective and easy to use frame delay which can be used to implement or realize 3-D filters, improve the quality of real time video streams and help reduce noise. Three-dimensional (3-D) digital filters are of increasing interest because of their potential applications, the space-time domain of the 3-D signals corresponds to the two spatial dimensions and the temporal dimension. Example of this is a time varying image, such as a video sequence. A video sequence is a three-dimensional signal, with the three dimensions being horizontal orientation, vertical orientation and time, and is said to be in spatio-temporal domain.

In digital video processing, 3-D IIR filters are useful for applications involving the selective enhancement of objects having various passband linear velocities in the presence of other objects having stopband velocities and noise [7, 8, 9, 10]. In [5] the first-order 3-D IIR frequency-planar filter is an important building block for many high-order 3-D IIR digital filters. In digital plane-wave filtering, highly-selective 3-D IIR beam and cone filters are useful for applications involving the selective enhancement of broadband spatio-temporal plane-waves based on their directions of arrivals in the presence of plane-waves, interference and additive white Gaussian noise.

In [19] systolic hardware realization structure permits the 2-D input data to be scanned row-wise and broadcasted on value at a time to various processing elements likewise 3-D input data is sent frame by frame to the memory and received after delay in the time domain.

Sid-Ahmed in [20] presented the realization of 3-D filters along with the hardware implementation of frame delays using read/write memory arrays. Three-dimensional filters were devised to operate on a sequence of images. These could provide, in some respect, an added advantage over 2-D filters when dealing with a sequence of images. An application that has been studied in the literature [6], [11] is the removal of moving objects along a given path and at a given speed from a sequence of images. Another application is the increase of frame rate in television images.

Three-dimensional filter realization enables frame delays to be obtained. Frame delay is achieved by sending sequence of frames to be written into memory and later read out. The time lapse between the frame being written and the read out one is basically the frame delay. The parallelism feature invites systolic architecture approach for VLSI realization of frame delay operations.

There are numerous attractive features in FPGAs, which encouraged the frame delay implementation. The arrangement of arrays of logic blocks in FPGAs appears systolic in structure, where logic blocks can be treated as Processing Elements. Additionally some FPGAs have ample flip-flops which make them suitable for pipelined systolic architectures. Altera FPGAs use logic elements as configurable logic blocks, and have abundant flip-flops to support pipelining. These favourable features make Altera FPGAs an ideal choice for implementing frame delays.

It is inefficient to directly retarget a full or semi-custom implementation of frame delay design to FPGA technology. In a full-custom implementation, all parts of the circuit, the logic, the routing circuitry are carefully tailored to meet a set of specific requirements. An FPGA is a prefabricated chip with programmable logic blocks and routing connections. There are density and performance penalties associated with user-programmable routing in FPGAs. For good FPGA implementation, the perspective is to efficiently utilize the available configurable logic blocks and the programmable routing connections between them. Minimizing the number of configurable logic blocks needed to realize a given function, is a very area-efficient solution for FPGAs. Reducing the number of CLBs is also effective in terms of reducing the routing performance penalty, as the connections between the CLBs are reduced if the number of CLBs is reduced.

Frame delay is realized using R/W memory. The sequence of frames are sent to a RAM one frame at a time and stored before they are called out as output. There is a counter which help to create locations for data to be sent to the memory and a clock which helps data to be written into the memory and then to read the data after another clock signal. This frame delay would be implemented in the Quartus II environment for synthesis in a reconfigurable hardware.

1.2. Thesis Organization

This thesis consists of 5 chapters. In the second chapter a background of frame delay and material to understand FPGA technology are presented. It covers programmable devices, explanation on FPGA architectures and how to implement the circuit.

In Chapter 3, the implementation of frame delay in Altera EP2C35 series FPGAs is dealt with. It shows the method of creating a frame delay and how to use the Quartus II schematic diagram to implement the proposed frame delay method.

Chapter 4 includes simulation results of the implemented frame delay. This chapter shows how FPGAs implement frame delays efficiently with its accompanying results.

The final chapter presents the conclusion and shows some directions that can be taken for future research work.

CHAPTER 2

BACKGROUND

2.1 Introduction to FPGA Technology

This chapter provides a background of frame delay and materials to understand the FPGA technology. The evolution of programmable devices is reviewed and then goes on with three-dimensional filters so as to show frame delays. This frame delay would later be implemented with the aid of an FPGA. The FPGA industry sprouted from programmable read-only memory (PROM) and programmable logic devices (PLDs). A PROM is a one-time programmable device that consists of a two dimensional array of memory cells and a decoder. Based on the address line inputs, the decoder selects and outputs one row of the memory on the data lines. To implement logic functions, address lines are used as the logic circuit inputs and each data line can implement a separate logic function. PROMs are most suited for implementing memory in applications such as microcontroller based systems.

The next generation of programmable devices are known as Programmable Logic Devices (PLDs). The basic PLDs are Programmable Logic Arrays (PLAs) and Programmable Array Logic (PALs) devices. PLAs have an AND-plane which can give any product term of the inputs as an output and an OR-plane which can generate any sum term of the product terms. Thus a PLA is optimally suitable for implementation of sum-of-products forms of Boolean expressions. The two programmable planes in the PLA are costly to build and they introduce considerable propagation delays for signals. In PALs, only the AND-plane is programmable and the OR-plane is fixed. So PALs are inexpensive to manufacture and they provide very high speed-performance of implemented circuits. PALs with registered outputs are used in developing many finite-state machines. PALs and PLAs are collectively referred to as Simple Programmable Logic Devices (SPLDs) in the literature. They are best used for implementing control circuitry.

Due to the limited logic capacity of SPLDs, denser Complex Programmable Logic Devices (CPLDs) were introduced by the microelectronics community. CPLDs have a hierarchical arrangement of multiple SPLDs on a single chip, and the logic capacity of CPLDs can advance to 5000 logic gates. CPLDs can also sustain system clock rates above 100 MHz. In order to use the merits, and remove the demerits of SPLDs and FPGAs, one of the semiconductor companies, Xilinx, introduced a new invention of programmable devices called Field Programmable Gate Arrays (FPGAs). Presently FPGAs can support logic capacities up to millions of gates. FPGA implementation of circuits has high operating speeds.

2.2 FPGA

FPGAs are one of the fastest growing segments of the semiconductor industry. They are programmable devices that can be directly configured by the end user without the application of an integrated circuit fabrication facility. They offer the designer the benefits of custom hardware, eliminating high development costs and turnaround time and thereby easily evolving as a lower cost alternative for VLSI implementation of circuits. They were first introduced in 1985 by Xilinx, and since then have quickly gained widespread acceptance as an excellent technology for implementing moderately large digital circuits in low production volumes. Since then, many different FPGAs have been developed by number of companies such as Actel [1], Altera [2], Motorola, QuickLogic and Crosspoint Solutions. Fig. 2.1 shows a conceptual diagram [4] of a typical FPGA.

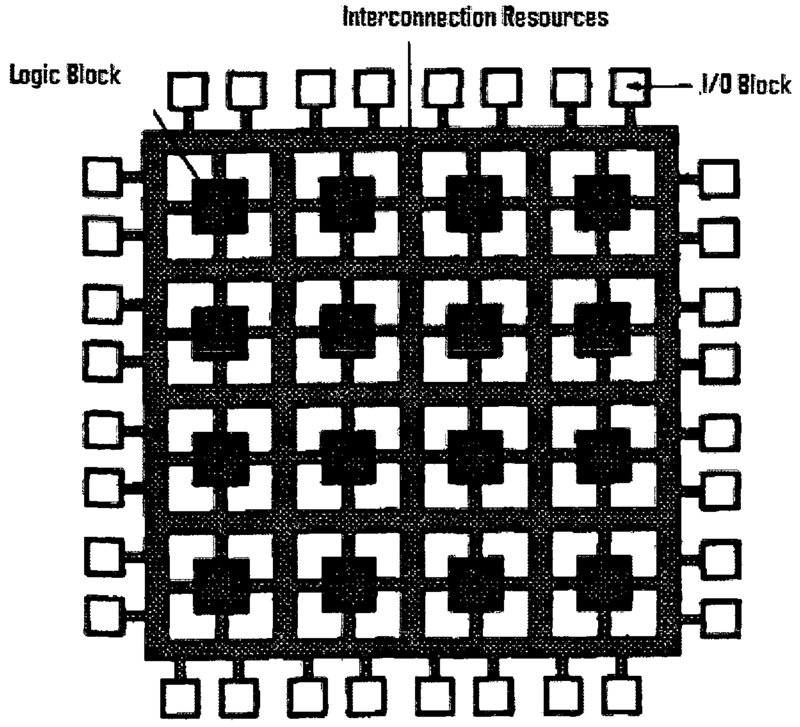


Fig.2. 1 Conceptual diagram of a typical FPGA architecture

An FPGA generally consists of a regular array of logic blocks called Configurable Logic Block (CLB) that can be programmed to implement combinational and sequential logic functions, I/O block and user-programmable routing channels [3, 12, 13, 14] that provides connections between the logic blocks. The interconnect resources comprises segments of wire, where the segments may be of different lengths. The interconnect resources include programmable switches that serve to connect the logic blocks to one another or one wire segment to another. Logic circuits are implemented in the FPGA by partitioning the logic into individual blocks and then interconnecting the blocks as required via the switches.

The structure and content of the interconnect resources in an FPGA is called its routing architecture. The routing architecture consists of wire segments and programmable switches. There exists many diverse ways to design the structure of a routing architecture, some FPGAs offer simple connection between blocks, and others provide less, but more complex routes. There are many advantages in using FPGA technology and they can be utilized in almost all the applications that currently use PLDs and Small Scale Integration (SSI) logic chips. Some of the applications are listed as follows.

Application Specific Integrated Circuits: An FPGA can be thought of as a general medium for implementation of ASICs. Some examples that have been reported are: a 1 megabit FIFO controller, a DRAM controller and a graphics engine.

Random logic implementation: Random logic circuitry is habitually implemented using PALs. If the speed of the circuit is not of immense concern then such circuits can be implemented advantageously with FPGAs. A single FPGA can implement a circuit that might require between ten to twenty PALs.

Prototyping: FPGAs are very appropriate for prototyping logic designs. The low outlay of implementation and short instance needed to physically realize a given design, provide them massive advantages over traditional approaches for building prototype hardware.

On-site Reconfiguration of Hardware: Some FPGAs can be reprogrammed unlimited number of times. Reprogrammability is a very striking feature where hardware has to be changed dynamically, or where hardware has to be modified to different user applications.

There are disadvantages for using FPGAs. The two main disadvantages of FPGAs are their relatively low speed of operation, and relatively low logic density. The propagation delays in FPGAs are adversely affected by the inclusion of programmable switches, which have considerable resistance and capacitance, in the connections between logic blocks. Logic density is reduced because the programmable switches and coupled programming circuitry require a great deal of chip area compared to the metal connections.

2.3 Programming Technologies

It is constructive to gain a better understanding of how FPGAs are made field-programmable. The term programmable switch actually refers to the programmable elements of the FPGA chip and a typical FPGA may contain 100000 of them. Programming elements are implemented using different technologies [4], such as static RAM cells and EPROM transistors. Regardless of the technology of implementation the programming elements are all configurable in one of two states: ON or OFF. The desirable properties of the programming elements are that they should consume as little chip area as possible, they should have a low ON resistance and a very high OFF resistance and it should be possible to reliably fabricate a large number of them on a single chip.

2.3.1 Static RAM Programming Technology

This technology is used in FPGAs produced by numerous companies: Concurrent Logic, and Xilinx. In these FPGAs, programmable connections are made using pass-transistors, transmission gates or multiplexers that are all controlled by SRAM cells. The RAM cell controls the pass-gates to be turned on or off. When off, the pass-gate presents a very high resistance between the two wires to which it is attached and the wires are hence disconnected. When the pass gate is turned on, it forms a relatively low resistance connection between the two wires. The chip area required by the static RAM approach is relatively large as at least five transistors are needed for each RAM cell, along with additional transistors for the pass-gates or multiplexers. The foremost advantage of this technology is that it provides an FPGA that can be reprogrammed very quickly.

2.3.2 EPROM Programming Technology

EPROM programming technology is used in FPGAs manufactured by Altera Corporation [2] and Plus Logic. The technology is the same as that used in EPROM memories. Unlike a simple MOS transistor, an EPROM transistor comprises two gates, a floating gate and a select gate. The floating gate is not electrically connected to any circuitry. In its unprogrammed state, no charge exists on the floating gate and the transistor can be turned ON in the normal fashion using the select gate. However, when the transistor is programmed by causing a large current to flow between the source and drain, a charge is trapped under the floating gate. The charge has the effect of permanently turning the transistor OFF. In this way, the EPROM transistor can function as a programmable element. An EPROM transistor can be re-programmed by first removing the trapped charge from the floating gate. EPROM transistors in addition to serving as programmable element can be used as "pull down" devices for logic block inputs. As long as the transistor is not programmed into the OFF state, the word line can cause the bit line, which is connected to a logic block input, to be pulled to logic zero.

2.4 Look-up table (LUT) based FPGA Architectures

This section gives a description of LUT-based architectures using the Xilinx FPGA family as an example. The general architecture of a Xilinx FPGA [4] is shown in Fig. 2.2

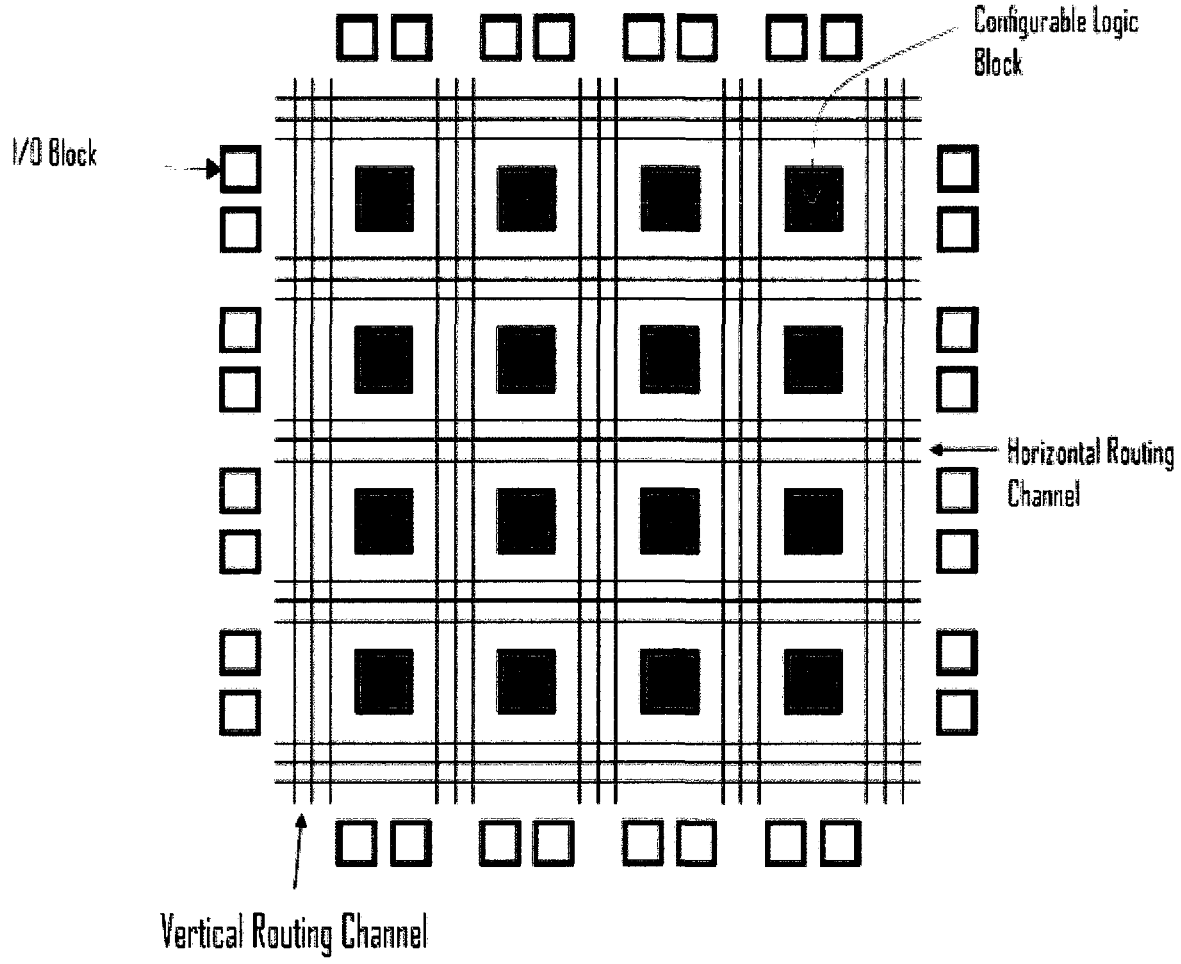


Fig.2. 2 Architecture of a Xilinx FPGA

It comprises of a two-dimensional array of programmable blocks, called Configurable Logic Blocks (CLBs), with horizontal routing channels between rows of blocks and vertical channels between columns. Programmable resources are controlled by static RAM cells. The gate count measure is given in terms of "equivalent to a mask-programmable gate array of the same size".

A CLB in the XC2000 family [15] consists of a 4-input look-up table with two outputs, and a D flip-flop. The look-up table can produce any function up to four variables or any two functions of three variables. Either the CLB ouputs can be combinational, or one output can be registered. The XC2000 routing architecture employs three types of routing resources: direct interconnect, general purpose interconnect and long lines. At every intersection of four CLBs, switch matrices are present; they hold a number of routing switches that can interconnect the wiring segments on its four sides. Longer wires are formed by connecting general purpose wiring segments through switch matrices. Connections that are required to reach several CLBs with low skew can use the long lines, which navigate at most one routing switch to span the entire length or width of the FPGA. The XC3000 [15] is an enhanced version of the XC2000, featuring a more complex CLB and more routing resources. XC4000 CLBs include on-chip static memory resources. An optional mode for each CLB makes the memory look-up tables in the function generators usable as either a 16 x 2 or 32 x 1 bit array of Read/Write memory cells. The inputs to the function generators act as address lines, selecting a particular memory cell in each look-up table. On-chip RAM is very useful for DMA counters, LIFO stacks and FIFO buffers.

There are three types of routing resources in the XC4000 [16], they are single length lines, double length lines and long lines. The single length lines are a framework of horizontal and vertical lines that intersect at a switch matrix between each block. The double length lines consist of a framework of metal segments twice as long as the single length lines; i.e., a double length lines runs past two CLBs before entering a switch matrix. Long lines forms a grid of metal interconnect segments that run the entire length or width of the array. Longlines can be driven by global buffers, designed to distribute clocks and other high fanout control signals throughout the array with negligible skew. Communication between longlines and single length lines is controlled by programmed interconnect points at the line intersections. Double length lines do not connect to other lines.

2.5 Cyclone II Architecture

This section gives a description of architectures using the Altera Cyclone II FPGA family as an example. The general architecture of an Altera FPGA is shown in the block diagram in Fig.2.3

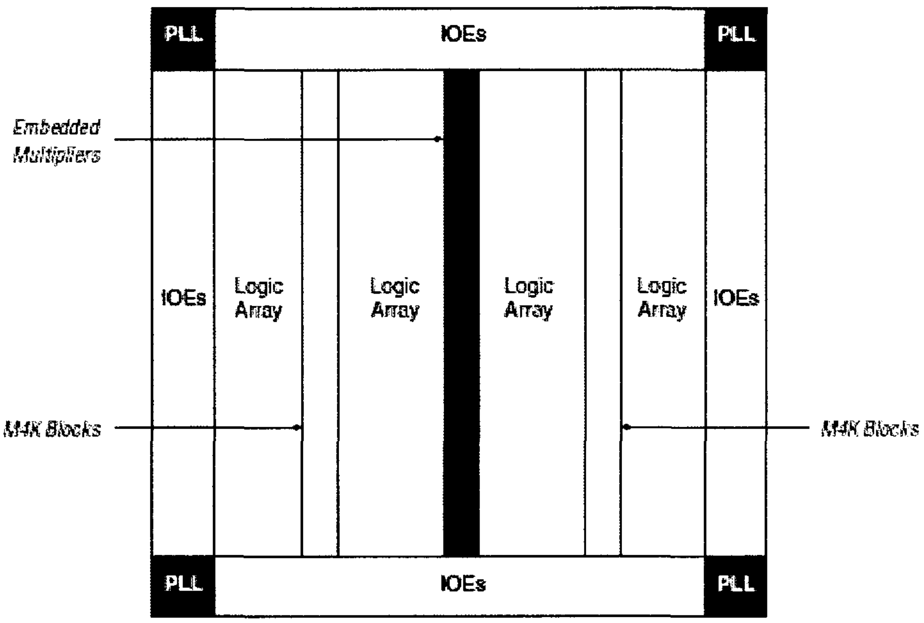


Fig.2. 3 Architecture of an Altera FPGA

Cyclone II devices contain a two-dimensional row and column-based architecture to implement custom logic. Column and row interconnects of varying speeds provide signal interconnects between logic array blocks (LABs), embedded memory blocks and embedded multipliers. The logic array consists of LABs, with 16 logic elements (LEs) in each LAB. An LE is a small unit of logic providing efficient implementation of user logic functions. LEs operate in either normal mode or arithmetic mode. LABs are grouped into rows and columns across the device.

The devices provide a global clock network and up to four phase-locked loops (PLLs). The global clock network consists of up to 16 global clock lines that drive throughout the entire device. The global clock network can provide clocks for all resources within the device, such as input/output elements (IOEs), LEs, embedded multipliers, and embedded memory blocks. The global clock lines can also be used for other high fan-out signals.

Cyclone II PLLs provide general-purpose clocking with clock synthesis and phase shifting as well as external outputs for high-speed differential I/O support.

M4K memory blocks provide dedicated true dual-port, simple dual-port, or single-port memory up to 36-bits wide. The M4K memory blocks include registers that synchronize writes and output registers to pipeline designs and improve system performance. The output registers can be bypassed, but input registers cannot. Each embedded multiplier block can implement up to either two 9x9-bit multipliers or one 18x18-bit multiplier. Embedded multipliers are arranged in columns across the device. Each device I/O pin is fed by an IOE located at the ends of LAB rows and columns around the periphery of the device. Each IOE contains a bidirectional I/O buffer and three registers for registering input, output, and output-enable signals.

The FPGA configuration is normally specified using a hardware description language (HDL), comparable to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could execute. FPGAs contain programmable logic components called logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together.

Most recent FPGAs have the ability to be reprogrammed at “run time” and this has lead to the notion of reconfigurable computing or reconfigurable systems- CPUs that reconfigure themselves to suit the task at hand. Historically, FPGAs have been slower, less energy efficient and generally achieved less functionality than their fixed ASIC counterparts. A combination of volume, fabrication improvements, research and development, and the I/O capabilities of new supercomputers have largely closed the performance gap between ASICs and FPGAs.

Xilinx claims that several market and technology dynamics are changing the ASIC/FPGA paradigm:

- Integrated circuit costs are rising aggressively
- ASIC complexity has bolstered development time and costs
- Financial constraints in a poor economy are driving low-cost technologies
- Research and Development resources and head count are decreasing

Some FPGAs have the capability of partial re-configuration that lets one portion of the device be re-programmed while other portions continue running. The inherent parallelism of the logic resources on an FPGA allows for substantial computational throughput even at low MHz clock rates. 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.

Implementation of large digital circuits such as full systems-on-chips in modern high-density FPGAs is enabled by sophisticated CAD tools. The process of mapping a circuit on an FPGA is divided into a series of sequential subproblems which make the procedure tractable. In the first phase, a designer describes a circuit in a hardware description language (HDL) such as Verilog or VHDL.

The second phase is synthesis stage which contains several steps. In the first step HDL is converted into a netlist of basic gates which is subsequently minimized. Then, the basic netlist is mapped onto a netlist of FPGA logic cells. The third step is packing, in which the logic cells are packed into the logic clusters. In the third phase, for each logic cluster from the netlist it is decided where it is going to be placed on an FPGA device. After locations of logic blocks are determined, a router finds a path between each connected logic block in the netlist and determines how the routing channels should be configured to implement the connection [21]. Routing algorithms are usually timing-driven as most of the delay in FPGAs stems from the programmable routing. Once it is ascertained that the synthesized circuit meets all requirements of the specifications the chip programming file is generated.

2.6 Frame Delay

A frame delay is obtained during the realization of a three dimensional filter. Assuming images are formed from the x and y axes then as these image frames move along the temporal axis there are some delays between these frames which become the frame delays. As frames move along the time axis there is a lapse of time between frames sent and received hence providing the frame delay. A 3-D digital filter is a system that, when given a sequence of 3-D input numbers, produces a sequence of 3-D output numbers subject to a specified set of rules providing some expected changes to the characteristics of the 3-D input signal. Three-dimensional digital filters has applications of removal of moving objects along a given path at a given speed from a sequence of images [17] as well as increasing frame-rate in television images. Filters of any dimension are traditionally divided into two categories: non-recursive filters and recursive filters. Non-recursive filters, also known as Finite Impulse Response (FIR) filters, produce an output which is weighted average of present and previous values. Recursive filters also known as Infinite Impulse Response (IIR) filters, produce an output that is a weight average of present and past values as well as past output values [18].

Certain types of three-dimensional filters are defined by linear time-invariant function in z_1, z_2 and z_3. In this case z_1 is considered as a pixel delay, z_2 as a line delay, and z_3 as a frame delay [18]. Fig. 2.4 shows a sequence of images represented by f(x,y,t). A three-dimensional z-transform of f(x,y,t) would yield F (z_1,z_2,z_3) as depicted in Fig.2.4

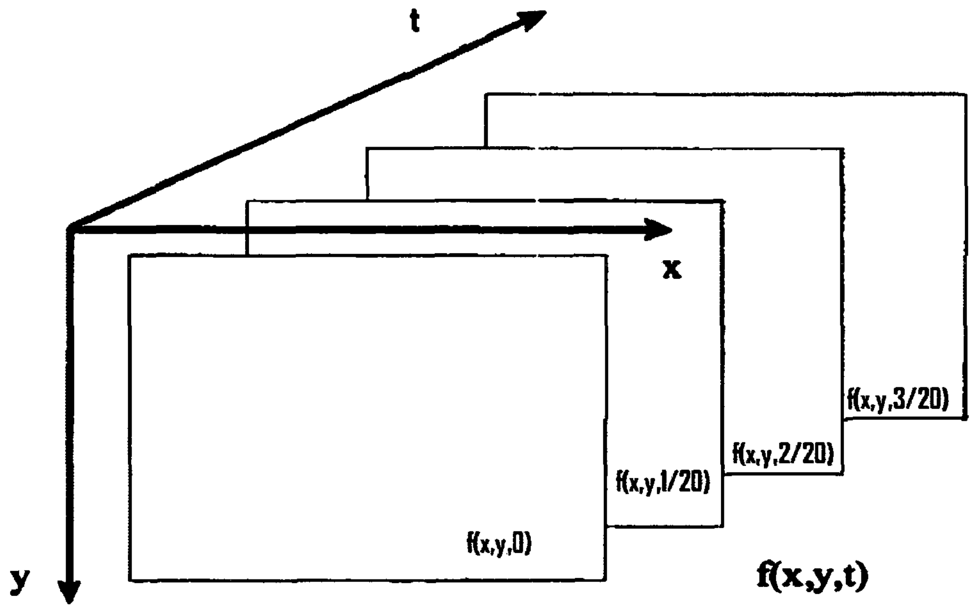


Fig.2. 4 Representation of moving images

Digital video is an example of a moving digital image sequence, with each frame of video representing a separate two-dimensional digital image. These images change as a function of time, and it is this temporal variation which represents the third dimension in digital video.

For a 1x1x1 FIR filter

$$H(z_1,z_2,z_3) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1} h(i,j,k)z_1^{-i}z_2^{-j}z_3^{-k} \tag{2.1}$$

$$= \sum_{i=0}^{1}\sum_{j=0}^{1}[h(i,j,0)\, z_1^{-i}z_2^{-j} + h(i,j,1)z_1^{-i}z_2^{-j}z_3^{-1}]$$

$$= \sum_{i=0}^{1}\sum_{j=0}^{1} h(i,j,0)z_1^{-i}z_2^{-j} + \sum_{i=0}^{1}\sum_{j=o}^{1} h(i,j,1)z_1^{-i}z_2^{-j}z_3^{-1}$$

$$H(z_1,z_2,z_3) = H_1(z_1,z_2) + H_2(z_1,z_2)z_3^{-1} \tag{2.2}$$

$$where\ H_1(z_1,z_2) = \sum_{i=0}^{1}\sum_{j=0}^{1} h(i,j,0)z_1^{-i}z_2^{-j}$$

$$and\ H_2(z_1,z_2) = \textstyle\sum_{i=0}^{1}\sum_{j=0}^{1} h(i,j,1)z_1^{-i}z_2^{-j}$$

$$for\ Y(z_1,z_2,z_3) = H(z_1,z_2,z_3)X(z_1,z_2,z_3)$$

$$= H_1(z_1,z_2)X(z_1,z_2,z_3) + H_2(z_1,z_2)X(z_1,z_2,z_3)z_3^{-1} \tag{2.3}$$

Given that $X(z_1,z_2,z_3)$ is the input and $Y(z_1,z_2,z_3)$ is the output of the 3-D filter, equation (2.3) can be realized as shown in Fig. 2.5

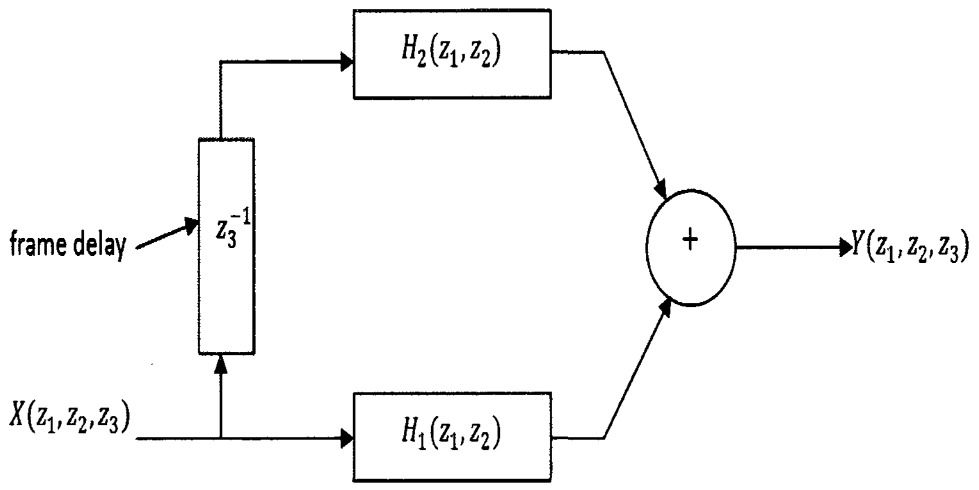


Fig.2. 5 Realization of a 3-D FIR filter

CHAPTER 3

DESIGN AND METHODOLOGY

3.1. Platform and Methodology

The main motivation of this work is to implement frame delay in a realistic hardware environment. The platform and methodology was selected to implement frame delay and run it on an FPGA device. Altera Cyclone II FPGA device is the chosen basis platform. This high-density device is intended for a full system-on-a chip implementation as it contains a balance of memory and logic resources. Table 3.1 contains the characteristics of the FPGA platform selected. Also the Altera tools that are used by the platform are listed in Table 3.2.

Table 3. 1 FPGA device and FPGA board characteristics

FPGA device	Altera Cyclone II EP2C35F672C6
M4K RAM blocks (4Kbits plus 512 parity bits)	105
Logic Elements (LEs)	33216
Total RAM bits	483840
Embedded Multipliers	35
PLLs	4
Maximum user I/O pins	475

Table 3. 2 Design Tools

Synthesis CAD tool	Altera Quartus II 9.1
System-on-a chip design tool	Altera SOPC Builder
Embedded processor	Altera Nios II
Software compilation tool	Altera Nios II Embedded Design Suite

3.2. FPGA Implementation Procedure

An FPGA design code can be implemented with the aid of a text editor or a schematic editor or a look-up table using simulation software such as Active-HDL, Quartus II or ModelSim. After the verification of the design, compilation including synthesis and floor planning are performed. Thus when developing a DSP system, two roles are needed to fulfill the FPGA development. One is the DSP engineer who is assigned to design and simulate the system before it is implemented. Another FPGA designer is assigned to design the structure from VHDL code to compilation and programming. Since both the coding work of development and system design are tedious, the process is divided between two different engineers, one focuses on the system and other focuses on the FPGA logic.

Altera has developed different software to improve this process and save time. Altera makes use of Simulink of Mathwork's MATLAB GUI interface, and it is also developed a block set called DSP builder. The schematic components are combined to generate this complex block set similar to other Simulink block set. Then in this GUI interface, the design generates a model file. The model file is translated to an HDL file, and the fitter can perform the compilation. During this process, the tedious HDL coding process becomes a process of pulling, plugging and debugging. This new procedure hides the HDL coding process so that the designer does not need to write HDL codes. This lessens the development and makes the process easier so that one engineer can do the job. In this interface, the synthesis process can be done by calling synthesis software such as Quartus II from Altera.

In Altera's GUI environment, DSP Builder links Mathworks MATLAB and Simulink software with the Altera Quartus II software. DSP system design in Altera FPGA requires both high-level algorithm and hardware description language (HDL) development tools. The Altera DSP Builder incorporates these tools by combining the algorithm development, simulation and verification abilities of the Mathworks MATLAB and Simulink system-level design tools with VHDL and Verilog HDL design flows, including the Altera Quartus II software.

The DSP Builder begins the Quartus II compilation automatically. It provides arithmetic and logical operators for use with the Simulink software. The DSP Builder Signal Compiler block reads Simulink model files that are built using DSP Builder, and generates VHDL and Verilog HDL files and Tcl scripts for the synthesis, hardware implementation and simulation. It also generates VHDL or Verilog HDL testbench or Quartus II vector file from MATLAB and Simulink test vectors.

Also the DSP Builder contains bit-cycle-accurate Simulink blocks and takes advantage of key device feature such as embedded memory ,built-in PLLs, or. DSP blocks. It provides faster performance and richer instrumentation of hardware co-simulation by implementing parts of the design in an FPGA using Hardware In the Loop (HIL) feature. The HIL block enables FPGA hardware accelerated co-simulation with Simulink.

The DSP Builder supports the SignalTap II logic analyzer an embedded signal analyzer that provides signals from the Altera device on the development board, and imports the data into the MATLAB workspace to facilitate visual analysis.

3.3. Procedure For Creating Frame Delay

A frame delay can be realized using R/W memory as shown in fig. 3.1 for a frame size of 512x512 pixels. The setup basically works as a FIFO shift register and operates as follows:

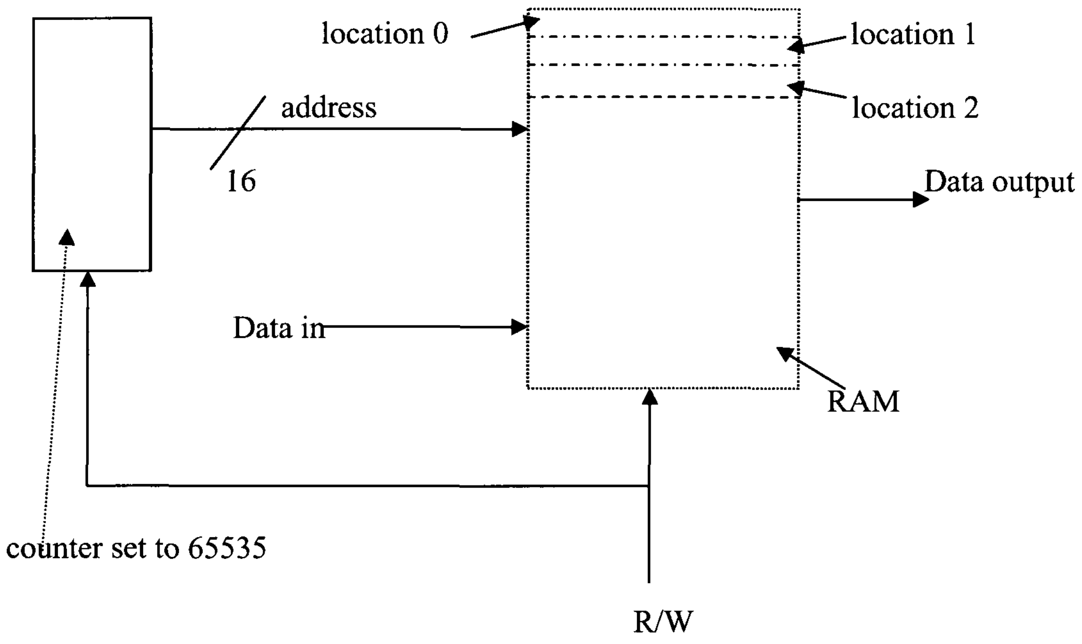


Fig.3. 1FIFO shift register realized using R/W memory

The clock signal, driving the R/W memory and counter, sets the R/W memory to the read mode during the period when the clock signal is high, and to write mode during the period when the clock signal is low. Before the circuit is switched on, the 16-bit counter is set to 2^{16}-1= 65535. At the first clock cycle and during the read portion, the output read is that of location zero of the R/W memory, which should contain a zero value at the start.

During the write portion of the cycle, data is written at that same location. In the second clock cycle, the counter is incremented to 1, reading of the second location first takes place in memory during the period when the clock signal is high, which is again zero, and writing is carried out during the second half of the cycle, at the same location of the R/W memory. This continues until the counter is at 65535. The following cycle resets the counter to zero, the value that was written at location zero in the first cycle is read out, and a new value is written in, and so on.

3.4. Altera FPGA Board for Frame Delay Implementation

The Altera FPGA board for the implementation of frame delay is shown in Fig.3.2. The board provides an ideal environment for hardware implementations of digital logic circuits and systems. It depicts the layout of the board and indicates the location of the connectors and key components. This board has many features that allow the user to implement a wide range of designed circuits, from simple circuits to various multimedia projects.

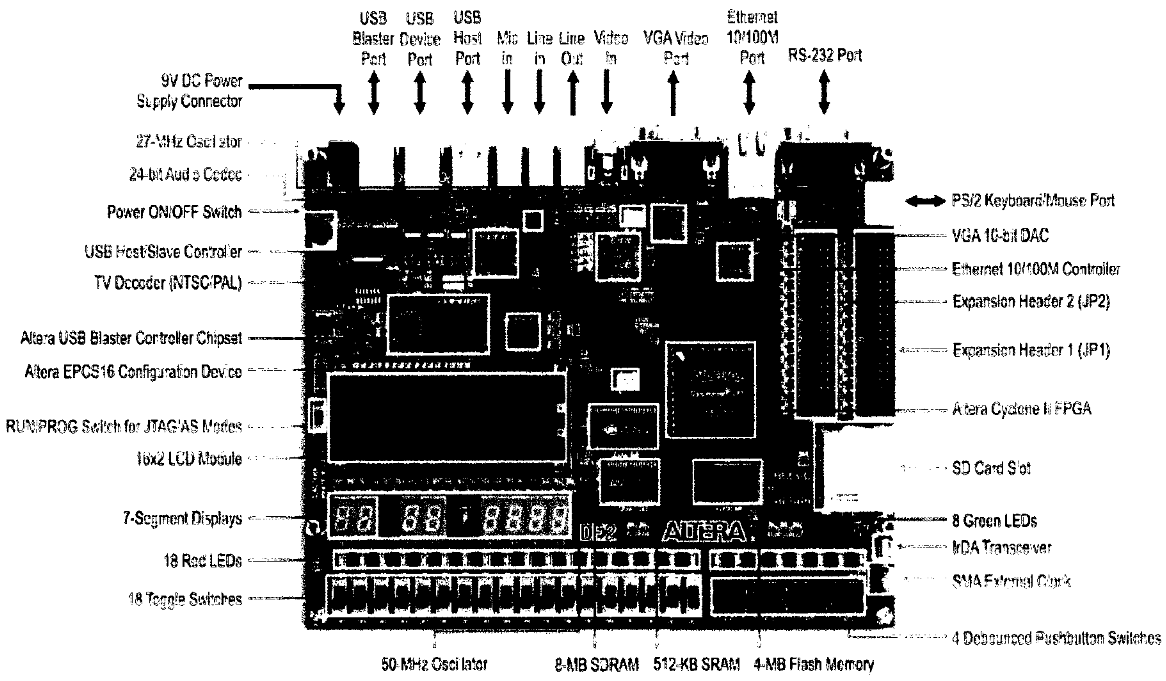


Fig.3. 2 Altera FPGA Development Board

The board is provided with some hardware such as the following:

- Altera Cyclone II 2C35 FPGA device
- Altera Serial Configuration device - EPCS16
- USB Blaster (on board) for programming and user API control; both JTAG and Active Serial (AS) programming modes are supported
- 512-Kbyte SRAM
- 8-Mbyte SDRAM
- 4-Mbyte Flash memory (1 Mbyte on some boards)
- 4 pushbutton switches
- 18 toggle switches
- 18 red user LEDs and 9 green user LEDs
- 50-MHz oscillator and 27-MHz oscillator for clock sources

In support to these hardware features, the board has software support for standard I/O interfaces and a control panel facility for accessing various components. In order to provide maximum flexibility for the user, all connections are made through the Cyclone II FPGA device. Thus, the user can configure the FPGA to implement any system design. The Quartus II synthesis environment is used to implement the frame delay in a reconfigurable hardware. The environment is shown in Fig.3.3 which is a schematic entry implementation of the frame delay.

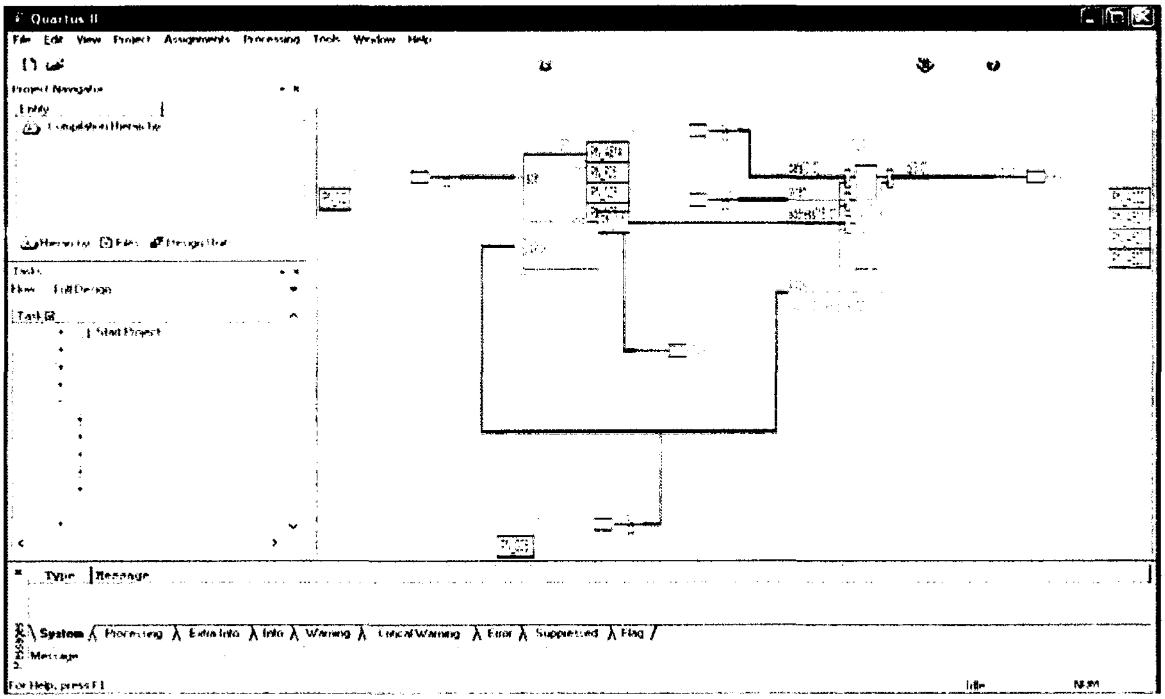

Quartus II
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Hierarchy Files Design Units
Tasks
Flow Full Design
Start Project
Type Message
System Processing Extra Info Info Warning Critical Warning Error Suppressed Flag
Message
For Help, press F1
Idle
NUM

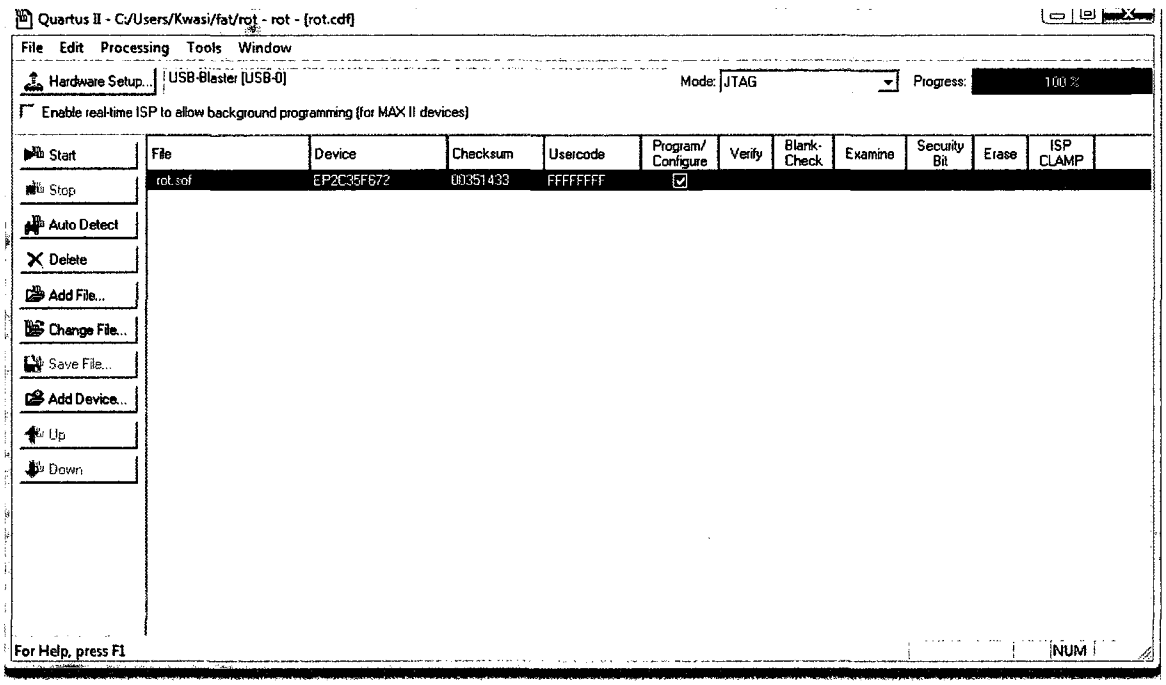
Fig.3. 3 Quartus II Environment

After a project is opened in the Quartus environment, the block diagram as shown in Fig.3.3 is designed by using MegaWizard Plug-in Manager to create the 16-bit counter, the clock and RAM. The counter has a synchronous clear (sclr) which is used to clear the counter before it provides the address locations for the RAM. The sclr is then assigned as

PIN_V2 which is toggle switch 17 on the FPGA board. The RAM also has the write-enable (wren) which is on when set to 1 and off when set to 0 at which the already written data would be read. It was assigned as PIN_V1 which is toggle switch 16. Also the toggle switches 0-3 represent the input data which is in binary form. The clock (clk) is assigned PIN_G26 which is KEY0 one of the debounced pushbutton switches. The output is read using the fifth segment of the Seven Segment Display. The first four segments of the Seven Segment Display provide the address locations for storing the data in the memory. The Seven Segment Display is in the hexadecimal form.

The block diagram named rot.bdf is processed by going to Processing and then compiled. Also the compilation can be done by using the VHDL code which is shown in thesis as Appendix A.

After the compilation is done, the function Tools is selected and then Programmer is clicked on. The FPGA board is connected to the computer in the Quartus II environment, the Hardware Setup is set to USB-Blaster as seen in Fig.3.4. The function rot.sof is selected after which Add File is activated. The function is run by clicking Start and the board is ready to be used when Progress reads 100%.



Quartus II - C:/Users/Kwasi/fat/rot - rot - [rot.cdf]

USB-Blaster [USB-0]

Mode: JTAG

Progress: 100 %

Enable real-time ISP to allow background programming (for MAX II devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
rot.sof	EP2C35F672	00351433	FFFFFFFF	☑						

Fig.3. 4 Quartus Programmer Interface

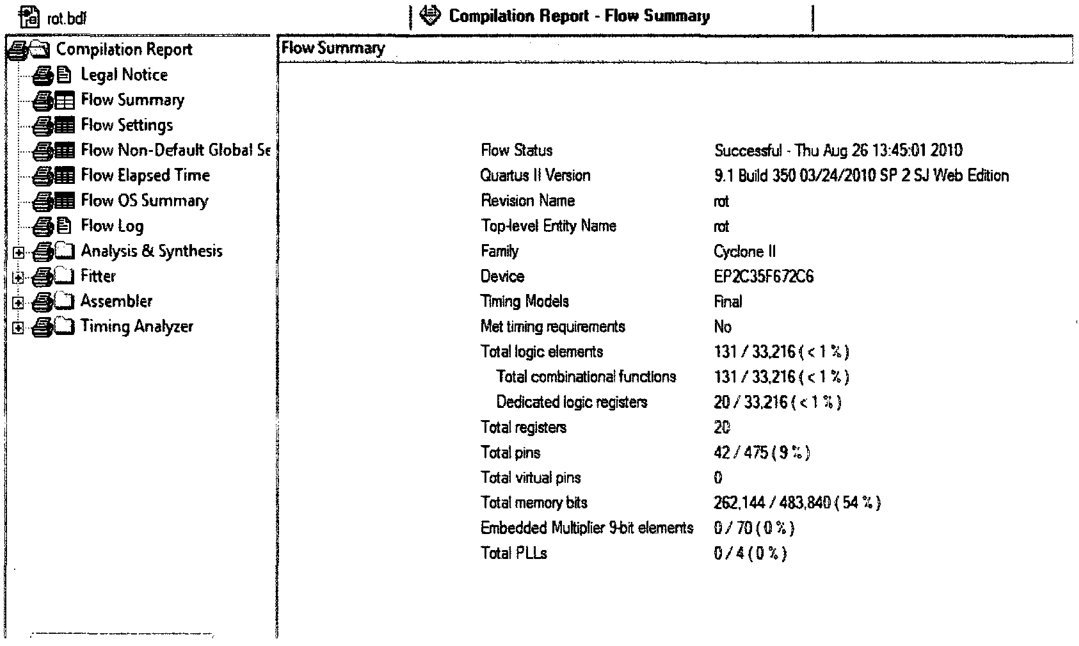
The FPGA board is now turned on data written into the various address locations and later read out.

CHAPTER 4

SIMULATIONS AND RESULTS

4.1. Compilation Results

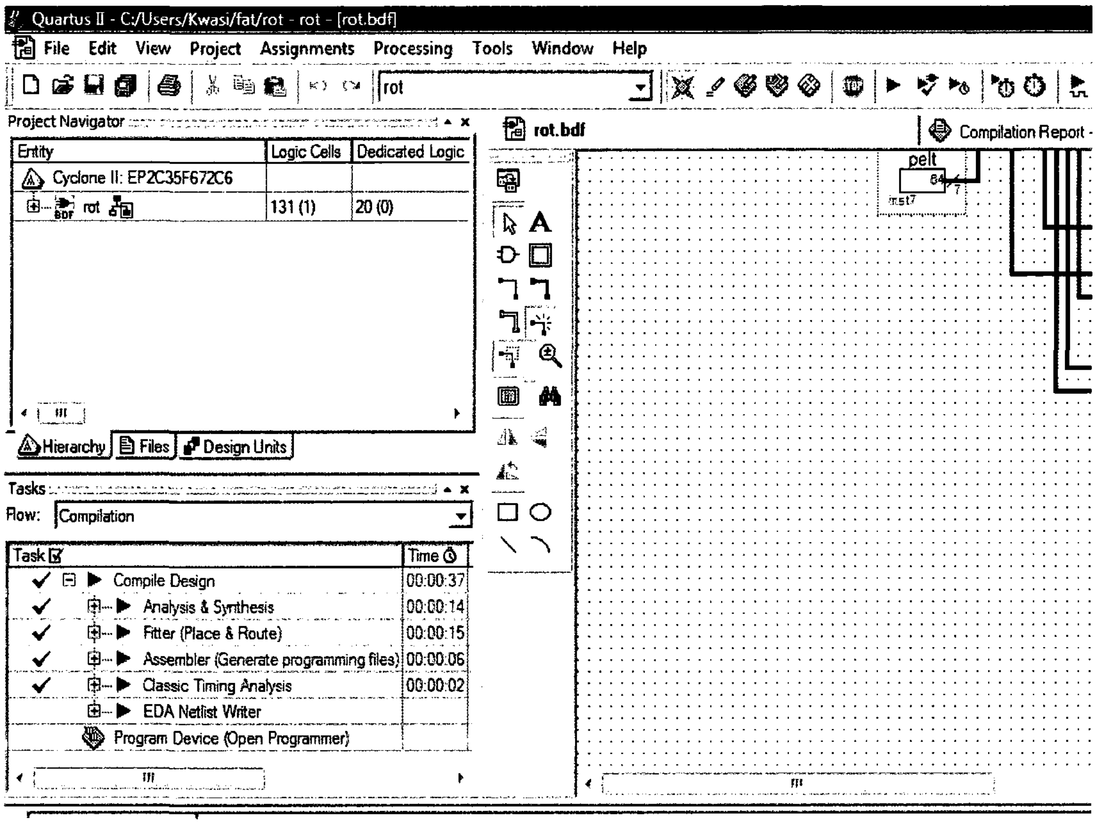
The compilation produces a compilation report for the FPGA implementation block diagram which is named rot.bdf.



Flow Status	Successful - Thu Aug 26 13:45:01 2010
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	rot
Top-level Entity Name	rot
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	No
Total logic elements	131 / 33,216 (< 1 %)
Total combinational functions	131 / 33,216 (< 1 %)
Dedicated logic registers	20 / 33,216 (< 1 %)
Total registers	20
Total pins	42 / 475 (9 %)
Total virtual pins	0
Total memory bits	262,144 / 483,840 (54 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Fig.4. 1 Compilation Report

Fig.4.1. shows the type of device used, the total number of logic elements used, total registers, pins and total memory bits. It is seen from Fig.4.1. that 131 logic elements are used in the implementation which is less than 1% of the total available. Also the total number of registers is 20 with the memory used being 262144 bits which is 54% of total memory bit space available. The number of LUTs used for the implementation is 111 covering about 1% of the LUTs available. Fig.4.2 shows the time for the compilation to be completed that is it takes about 37 seconds to complete the Analysis & Synthesis, Fitter, Assembler and the Classic Timing Analysis.


Quartus II - C:/Users/Kwasi/fat/rot - rot - [rot.bdf]
File Edit View Project Assignments Processing Tools Window Help
rot
Project Navigator
rot.bdf
Compilation Report -
pelt
inst7
Hierarchy
Files
Design Units
Tasks
Flow: Compilation

Entity	Logic Cells	Dedicated Logic
Cyclone II: EP2C35F672C6		
rot	131 (1)	20 (0)

Task	Time
✓ Compile Design	00:00:37
✓ Analysis & Synthesis	00:00:14
✓ Fitter (Place & Route)	00:00:15
✓ Assembler (Generate programming files)	00:00:06
✓ Classic Timing Analysis	00:00:02
EDA Netlist Writer	
Program Device (Open Programmer)	

Fig.4. 2 Time for Complete Compilation

4.2. Implementation Results

After the compilation and the Quartus Programmer interface is accessed and turned on together with the Altera board then 4-bit data is written into the memory from location 0 to location 65535. Location 0 is represented as 0000 in hexadecimal form whilst Location 65535 is represented as FFFF in hexadecimal form. The toggle switches 0 to 3 which are the inputs are turned on or off to represent binary digits 1 and 0 respectively.

Table 4. 1 Data input and location

Data Input(Binary)	Location(Decimal)	Location(Hexadecimal)
1001	0	0000
0001	1	0001
0100	2	0002
1000	3	0003
1111	4	0004
0000	5	0005
0010	6	0006
1101	7	0007
●	●	●
0110	6899	1AF3
●	●	●
0111	65451	FFAB
●	●	●
0101	65535	FFFF

After the sclr is turned on together with the clock, the wren is turned on again but in this instance becomes the read enable and the following is seen on the FPGA board.

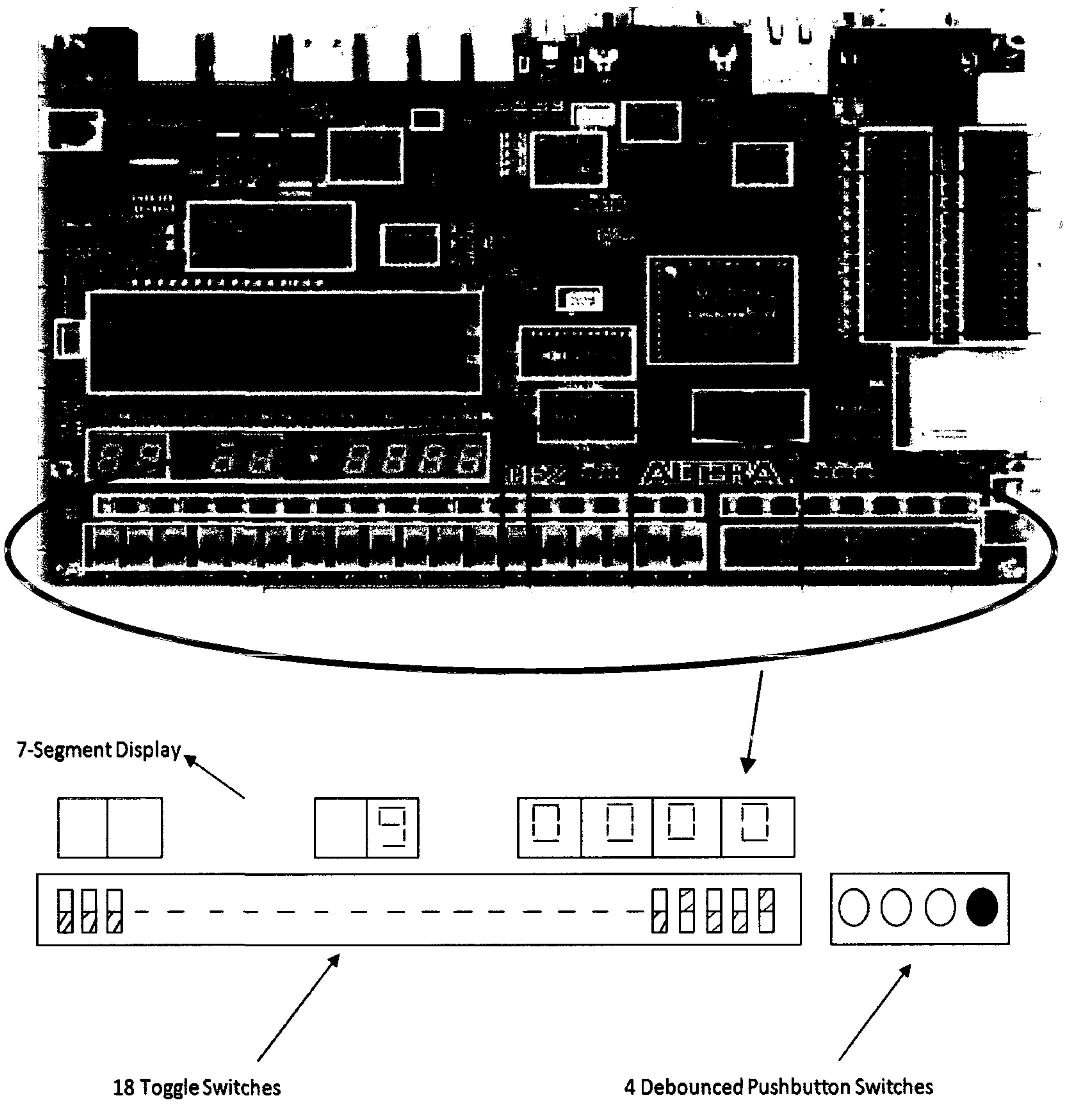


Fig.4. 3 Location 0 with data input 9

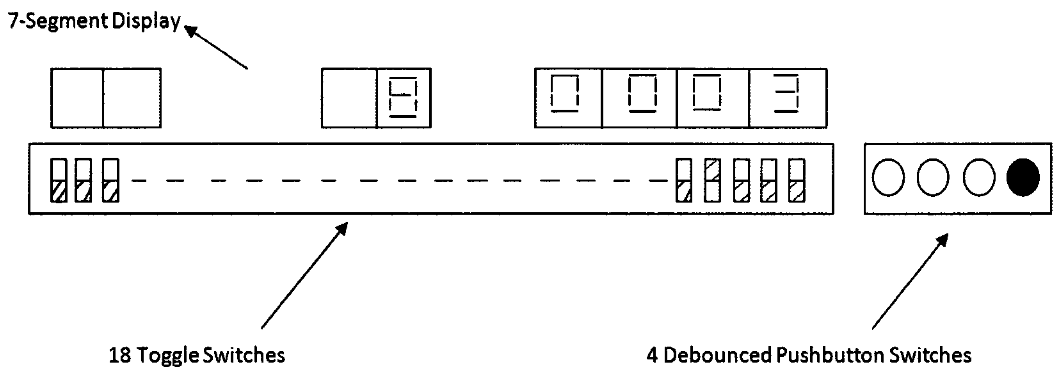


Fig.4. 4 Location 3 with data input 8

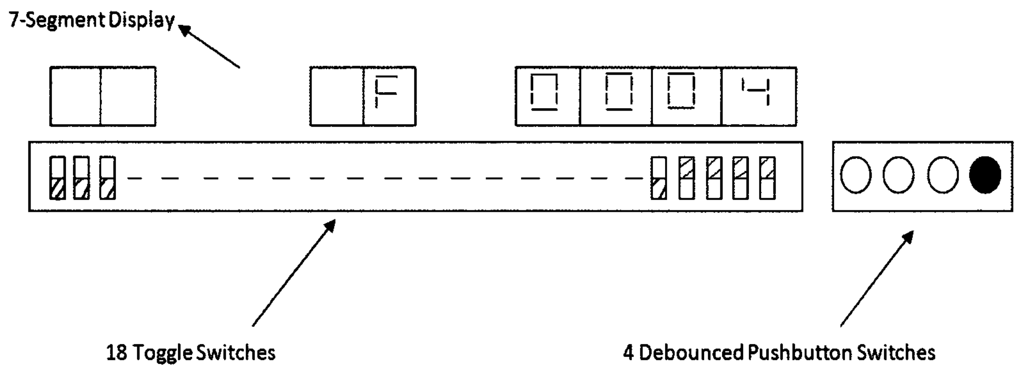


Fig.4. 5 Location 4 with data input 15

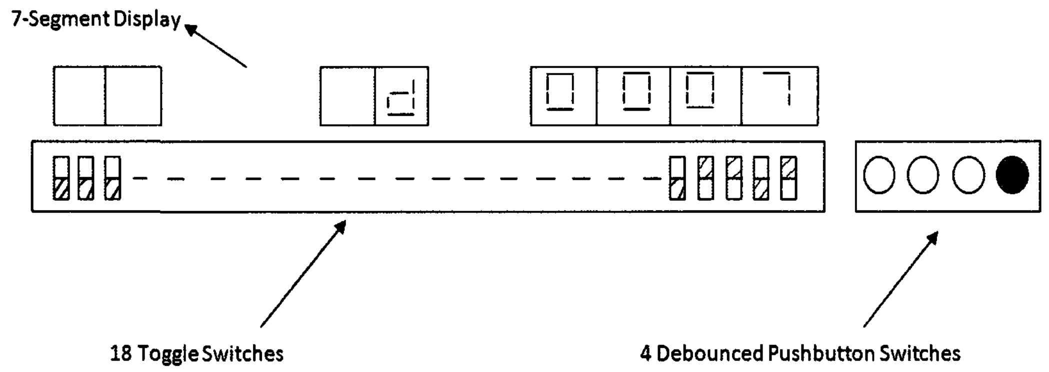


Fig.4. 6 Location 7 with data input 13

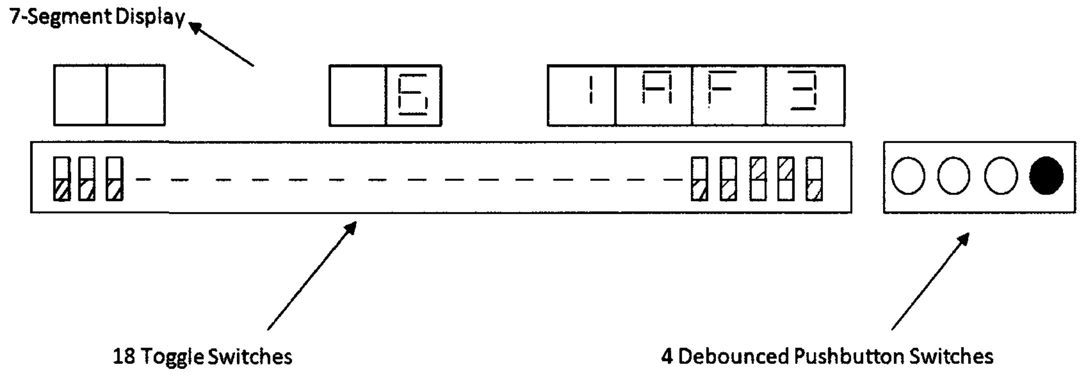


Fig.4. 7 Location 6899 with data input 6

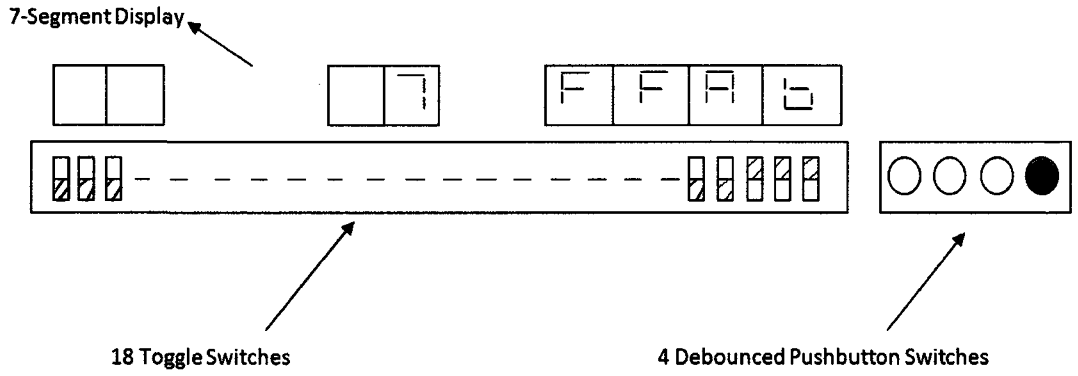


Fig.4. 8 Location 65451 with data input 7

The Classic Timing Analyzer is selected from Processing and produces a report. The report shows a Clock period of 5ns and indicates the maximum frequency of 200 MHz. The Total Thermal Power Dissipation obtained from the Power Analyzer was 114.64mW for the implementation of frame delay.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

This chapter presents the summary of the main conclusions of this thesis and also provides recommendations for future work directions.

5.1. Conclusions

In this thesis, the design and implementation of the frame delay using a Field Programmable Gate Array (FPGA) was performed. The implementation of frame delay for realistic performance experiments and architecture exploration is designed by the use of Altera FPGA board. During the same period, high-performance in a complexity-efficient manner is achieved, that is, consume minimal chip resources while achieving maximum operating frequency of about 200 MHz.

Altera Cyclone II FPGAs was chosen as the target hardware technology. The advantages of the FPGAs such as cost and reconfigurability allows for the easy implementation of frame delay. The FPGA platform is used as the target deployment platform which makes it easier for the counter, clock and memory to be employed for the implementation. This approach validates the possibility of an efficient hardware implementation of a frame delay.

The implemented frame delay can be used to realize a real-time three-dimensional filter. The procedure used is computationally simple and easily implemented mainly by the use of the schematic process in Quartus II.

The total power dissipation was 114.64mW which shows that the frame delay implementation consumes very little power for the entire process. The implementation consumes less than 1% of logic elements. It also uses 54% of the total memory bits and about 9% of the total pins.

5.2. Future Recommendations

There are some suggestions for future work. It would be of immense importance to investigate the implementation of frame delay in other hardware environment. Also frame delay should be used to implement video streams with high quality. This frame delay should be applied in the realization of cone and beam three-dimensional filters. Frame delay should be performed on coloured 3-D TV on different FPGA platforms. During the implementation of frame delay it is found out that there is a period of delay before the data is read out and it would make sense for data to be manipulated in this instance before it is called out.

APPENDIX A

VHDL CODE FOR IMPLEMENTATION

```
-- Copyright (C) 1991-2010 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.


-- PROGRAM          "Quartus II"
-- VERSION          "Version 9.1 Build 350 03/24/2010 Service Pack 2 SJ Web
Edition"
-- CREATED          "Tue Aug 31 14:08:47 2010"


LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
LIBRARY work;

ENTITY rot IS

        PORT

        (

                wren :  IN  STD_LOGIC;

                sclr :  IN  STD_LOGIC;

                clk :  IN  STD_LOGIC;

                Din :  IN  STD_LOGIC_VECTOR(3 DOWNTO 0);

                M :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);

                N :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);

                Q :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);

                R :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0);

                T :  OUT  STD_LOGIC_VECTOR(6 DOWNTO 0)

        );

END rot;

ARCHITECTURE bdf_type OF rot IS

COMPONENT ox

        PORT(sclr : IN STD_LOGIC;

                 clock : IN STD_LOGIC;

                 q : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)

        );

END COMPONENT;

COMPONENT cow
```

```
        PORT(wren : IN STD_LOGIC;
                clock : IN STD_LOGIC;
                address : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
                data : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt2
        PORT(           result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt3
        PORT(           result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt4
        PORT(           result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT wet
        PORT(data0x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
                data10x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
                data11x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
```

```
			data12x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data13x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data14x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data15x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data1x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data2x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data3x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data4x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data5x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data6x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data7x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data8x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			data9x : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
			sel : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
			result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
	);
END COMPONENT;
COMPONENT pelt5
	PORT(		result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
	);
END COMPONENT;
COMPONENT pelt6
	PORT(		result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
          );
END COMPONENT;
COMPONENT pelt7
          PORT(       result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
          );
END COMPONENT;
COMPONENT pelt8
          PORT(       result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
          );
END COMPONENT;
COMPONENT pelt9
          PORT(       result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
          );
END COMPONENT;
COMPONENT pelt10
          PORT(       result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
          );
END COMPONENT;
COMPONENT pelt11
          PORT(       result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
          );
END COMPONENT;
COMPONENT pelt13
```

```
        PORT(        result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt14
        PORT(        result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt15
        PORT(        result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt12
        PORT(        result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt
        PORT(        result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
COMPONENT pelt1
        PORT(        result : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END COMPONENT;
```

```
SIGNAL      F :  STD_LOGIC_VECTOR(15 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_81 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_82 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_83 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_84 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_85 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_86 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_87 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_88 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_89 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_90 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_91 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_92 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_93 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_94 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_95 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_96 :  STD_LOGIC_VECTOR(6 DOWNTO 0);

SIGNAL      SYNTHESIZED_WIRE_80 :  STD_LOGIC_VECTOR(3 DOWNTO 0);


BEGIN

b2v_inst : ox

PORT MAP(sclr => sclr,

                  clock => clk,
                q => F);

b2v_inst1 : cow

PORT MAP(wren => wren,

                clock => clk,

                address => F,

                data => Din,

                q => SYNTHESIZED_WIRE_80);

b2v_inst13 : pelt2

PORT MAP(           result => SYNTHESIZED_WIRE_89);

b2v_inst22 : pelt3

PORT MAP(           result => SYNTHESIZED_WIRE_90);

b2v_inst23 : pelt4

PORT MAP(           result => SYNTHESIZED_WIRE_91);

b2v_inst24 : wet

PORT MAP(data0x => SYNTHESIZED_WIRE_81,

                data10x => SYNTHESIZED_WIRE_82,

                data11x => SYNTHESIZED_WIRE_83,

                data12x => SYNTHESIZED_WIRE_84,

                data13x => SYNTHESIZED_WIRE_85,

                data14x => SYNTHESIZED_WIRE_86,

                data15x => SYNTHESIZED_WIRE_87,

                data1x => SYNTHESIZED_WIRE_88,

                data2x => SYNTHESIZED_WIRE_89,
                data3x => SYNTHESIZED_WIRE_90,
                data4x => SYNTHESIZED_WIRE_91,
                data5x => SYNTHESIZED_WIRE_92,
                data6x => SYNTHESIZED_WIRE_93,
                data7x => SYNTHESIZED_WIRE_94,
                data8x => SYNTHESIZED_WIRE_95,
                data9x => SYNTHESIZED_WIRE_96,
                sel => F(3 DOWNTO 0),
                result => M);
b2v_inst25 : wet
PORT MAP(data0x => SYNTHESIZED_WIRE_81,
                data10x => SYNTHESIZED_WIRE_82,
                data11x => SYNTHESIZED_WIRE_83,
                data12x => SYNTHESIZED_WIRE_84,
                data13x => SYNTHESIZED_WIRE_85,
                data14x => SYNTHESIZED_WIRE_86,
                data15x => SYNTHESIZED_WIRE_87,
                data1x => SYNTHESIZED_WIRE_88,
                data2x => SYNTHESIZED_WIRE_89,
                data3x => SYNTHESIZED_WIRE_90,
                data4x => SYNTHESIZED_WIRE_91,
                data5x => SYNTHESIZED_WIRE_92,
                data6x => SYNTHESIZED_WIRE_93,
                    data7x => SYNTHESIZED_WIRE_94,
                    data8x => SYNTHESIZED_WIRE_95,
                    data9x => SYNTHESIZED_WIRE_96,
                    sel => F(7 DOWNTO 4),
                    result => N);
b2v_inst26 : wet
PORT MAP(data0x => SYNTHESIZED_WIRE_81,
                    data10x => SYNTHESIZED_WIRE_82,
                    data11x => SYNTHESIZED_WIRE_83,
                    data12x => SYNTHESIZED_WIRE_84,
                    data13x => SYNTHESIZED_WIRE_85,
                    data14x => SYNTHESIZED_WIRE_86,
                    data15x => SYNTHESIZED_WIRE_87,
                    data1x => SYNTHESIZED_WIRE_88,
                    data2x => SYNTHESIZED_WIRE_89,
                    data3x => SYNTHESIZED_WIRE_90,
                    data4x => SYNTHESIZED_WIRE_91,
                    data5x => SYNTHESIZED_WIRE_92,
                    data6x => SYNTHESIZED_WIRE_93,
                    data7x => SYNTHESIZED_WIRE_94,
                    data8x => SYNTHESIZED_WIRE_95,
                    data9x => SYNTHESIZED_WIRE_96,
                    sel => F(11 DOWNTO 8),
            result => R);
b2v_inst27 : wet
PORT MAP(data0x => SYNTHESIZED_WIRE_81,
            data10x => SYNTHESIZED_WIRE_82,
            data11x => SYNTHESIZED_WIRE_83,
            data12x => SYNTHESIZED_WIRE_84,
            data13x => SYNTHESIZED_WIRE_85,
            data14x => SYNTHESIZED_WIRE_86,
            data15x => SYNTHESIZED_WIRE_87,
            data1x => SYNTHESIZED_WIRE_88,
            data2x => SYNTHESIZED_WIRE_89,
            data3x => SYNTHESIZED_WIRE_90,
            data4x => SYNTHESIZED_WIRE_91,
            data5x => SYNTHESIZED_WIRE_92,
            data6x => SYNTHESIZED_WIRE_93,
            data7x => SYNTHESIZED_WIRE_94,
            data8x => SYNTHESIZED_WIRE_95,
            data9x => SYNTHESIZED_WIRE_96,
            sel => F(15 DOWNTO 12),
            result => T);
b2v_inst36 : pelt5
PORT MAP(         result => SYNTHESIZED_WIRE_92);
b2v_inst37 : pelt6
```

```
PORT MAP(            result => SYNTHESIZED_WIRE_93);

b2v_inst38 : pelt7

PORT MAP(            result => SYNTHESIZED_WIRE_94);

b2v_inst39 : pelt8

PORT MAP(            result => SYNTHESIZED_WIRE_95);

b2v_inst40 : pelt9

PORT MAP(            result => SYNTHESIZED_WIRE_96);

b2v_inst41 : pelt10

PORT MAP(            result => SYNTHESIZED_WIRE_82);

b2v_inst42 : pelt11

PORT MAP(            result => SYNTHESIZED_WIRE_83);

b2v_inst44 : pelt13

PORT MAP(            result => SYNTHESIZED_WIRE_85);

b2v_inst45 : pelt14

PORT MAP(            result => SYNTHESIZED_WIRE_86);

b2v_inst46 : pelt15

PORT MAP(            result => SYNTHESIZED_WIRE_87);

b2v_inst47 : pelt12

PORT MAP(            result => SYNTHESIZED_WIRE_84);

b2v_inst6 : wet

PORT MAP(data0x => SYNTHESIZED_WIRE_81,

                data10x => SYNTHESIZED_WIRE_82,

                data11x => SYNTHESIZED_WIRE_83,
                    data12x => SYNTHESIZED_WIRE_84,
                    data13x => SYNTHESIZED_WIRE_85,
                    data14x => SYNTHESIZED_WIRE_86,
                    data15x => SYNTHESIZED_WIRE_87,
                    data1x => SYNTHESIZED_WIRE_88,
                    data2x => SYNTHESIZED_WIRE_89,
                    data3x => SYNTHESIZED_WIRE_90,
                    data4x => SYNTHESIZED_WIRE_91,
                    data5x => SYNTHESIZED_WIRE_92,
                    data6x => SYNTHESIZED_WIRE_93,
                    data7x => SYNTHESIZED_WIRE_94,
                    data8x => SYNTHESIZED_WIRE_95,
                    data9x => SYNTHESIZED_WIRE_96,
                    sel => SYNTHESIZED_WIRE_80,
                    result => Q);
b2v_inst7 : pelt
PORT MAP(           result => SYNTHESIZED_WIRE_81);
b2v_inst8 : pelt1
PORT MAP(           result => SYNTHESIZED_WIRE_88);
END bdf_type;
```

REFERENCES

[1] Actel, "FPGA Data Book and Design Guide," 1994

[2] Altera Data Book, 1993

[3] Betz, V., Rose, J. And Marquardt, A, "Architecture and CAD for Deep-Submicron FPGA" Kluwer Academic Publishers, Norwell, MA, U.S.A., 1999

[4] Brown, D.S., Francis, R.J., Rose, J and Vranesic, Z.G, "Field Programmable Gate Arrays", 1992

[5] Bruton, L.T. and Bartley, N.R., "Three-Dimensional Image Processing Using the Concept of Network Resonance" IEEE Transactions on Circuits and Systems, Vol. 32, Issue 7, pp. 664-672, July 1985

[6] Bruton, L. and Bartley, N., "The design of highly selective adaptive three-dimensional recursive cone filters" IEEE Transactions on Circuits and Systems, Vol. 34, Issue 7, pp. 775-781, July 1987

[7] Bruton, L. T. and Kuenzle, B., "3-D IIR filtering using decimated DFT-polyphase filter bank structures" IEEE Transactions on Circuit and Systems I: Regular Papers, Vol. 53, Issue 2, pp. 394-408, 2006

[8] Bruton, L.T. and Kuenzle, B., "A novel low-complexity spatio-temporal ultra wide-angle polyphase cone filter bank applied to subpixel motion discrimination" IEEE International Symposium on Circuits and Systems, ISCAS 2005, Vol.3, pp. 2397-2400, Kobe, Japan, 2005

[9] Bruton, L.T. and Zhang, Y., "Applications of 3-D LCR networks in the design of 3-D recursive filters for processing image sequences" IEEE Transactions on Circuits and Systems for Video Technology, Vol.4, Issue 4, pp. 369-382, 1994

[10] Bruton, L.T. and Fowlow, T.J, "The design and application of a high quality three dimensional linear trajectory filter" IEEE International Symposium on Circuits and Systems, Vol.2, pp. 1033-1036, 1998

[11] Bruton, L.T. and Knudsen, K.S., "Mixed Multidimensional Filters" Proceedings of the 33rd Midwest Symposium on Circuits and Systems, Vol.1, pp. 80-83, 1990

[12] Kuon, I. and Rose, J., " Measuring the Gap between FPGAs and ASICs", Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field programmable gate arrays, pp. 21-30, ACM Press, New York, NY, U.S.A.,2006

[13] Lin, M., "The amorphous FPGA architecture", Proceedings of the 16th International ACM/SIGDA Symposium on Field programmable gate arrays, pp.191-200, ACM Press, New York, NY, U.S.A., 2008

[14] Rose, J., El Gamal, A. and Sangiovanni-Vincentelli, A., "Architecture of Field Programmable Gate Arrays" Proceedings of the IEEE, Vol.81, Issue 7, pp.1013-1029, July 1993

[15] Xilinx, "The Programmable Logic Data Book", 1993

[16] Xilinx, "The XC4000 Data Book", 1992

[17] Sid-Ahmed, M.A., "Image Processing-Theory, Algorithms & Architectures" 1994

[18] El-Feghi, I. S.,"Design of Three-Dimensional Digital Filters," MASc Thesis, University of Windsor, Windsor, 1999

[19] Sid-Ahmed, M.A., "A systolic realization for 2-D digital filters" IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.37, Issue 4, pp.560-565, April 1989

[20] Sid-Ahmed, M.A., "Systolic and semi-systolic realizations of three dimensional filters" IEEE Transactions on Consumer Electronics, Vol.40, Issue 2, pp.107-113, 1994

[21] Capalija, Davor, "Microarchitecture and FPGA implementation of the multi-level computing architecture" MASc. Thesis, University of Toronto, Toronto, 2008

VITA AUCTORIS

Kwasi Gyening Afrifa was born in April, 1982 in Accra, Ghana. He attended Prempeh College for his high school education. He graduated from the Kwame Nkrumah University of Science and Technology in 2005 where he obtained his BSc. in Electrical and Electronics Engineering. He is currently a candidate for the Master's degree in Electrical and Computer Engineering department at the University of Windsor.