An Experimental Study on Field Programmable Gate Arrays Routing Resources

by
Serafeim Radis

Department of Electronic Engineering
Technical University of Crete
Chania, Greece

July 2005

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) are integrated circuits which can be programmed to implement virtually any digital circuit. This programmability provides a low-risk, low-turnaround time option for implementing digital circuits. This programmability comes at a cost, however. Typically, circuits implemented on FPGAs are three times as slow and have only one tenth the density of circuits implemented using more conventional techniques. Much of this area and speed penalty is due to the programmable routing structures and the quantity of routing resources contained in the FPGA.

In this study, we focus on commercially available FPGAs examining their capability to handle designs, as far as speed is concerned, of various sizes and bus widths. For this purpose, a large number of experiments have been conducted. Great attention has been given to the varying factors of logic utilization and bus width and their relation with the results.

Many graphs developed from the results give an overall view of this delicate subject. Comments on and analysis of the results was also conducted and led to interesting conclusions. Some of these conclusions confirmed our expectations and beliefs on this subject. But some others posed questions worthy to draw our attention. Generally speaking, we believe we shed some light concerning this subject. More study however should be done.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks and appreciation to my academic supervisor Professor Dionysis Pneymatikatos, who has provided a continual source of guidance, advice, encouragement, and friendship throughout the period of this study. It has been my privilege to work with him.

I would also like to thank John Sourdis for providing me some designs and giving helpful tips.

Very special, special, special thanks (with sugar on top) to each of my friends personally for supplying me with poetry, laughter, strength and support through all of these unforgettable years.

My father and mother have always supported my studies and deserve much credit for enabling me to reach this milestone.

TABLE OF CONTENTS

1 Introduction

1.1 Introduction to Field-Programmable Gate Arrays

Since their inception in 1985, Field Programmable Gate Arrays (FPGAs) have emerged as a leading choice of technology for the implementation of many digital circuits and systems. New commercial architectures offer variety of features such as densities of up to 10 million system gates, on-chip single or dual port memories, digital phase lock loops for clock management, and system performance up to 311 MHz [1-3], making FPGAs ideal not only for glue logic but also for the implementation of entire systems.

Field Programmable Gate Arrays (FPGAs), Complex Programmable Logic Devices (CPLDs), and Masked Programmable Gate Arrays (MPGAs) are part of a programmable logic family which provides an alternative way of designing and implementing digital circuits. The traditional approach consists of designing a chip to meet a set of specific requirements which cannot be changed once the design is complete. In contrast, a programmable logic device can be programmed “in the field”, leading to lower time to market, and lower non-recurring engineering (NRE) costs. In addition many programmable devices can be re-programmed many times, leading to a faster recovery from design errors, and the ability to change the design as requirements change, either late in the design cycle, or in subsequent generations of a product.

Of all programmable devices, one of the most common is the Field Programmable Gate Array (FPGA). Compared to other programmable devices, an FPGA offers the highest logic density, a good speed-area trade-off, and a very general architecture suitable for a wide range of applications. FPGAs are being used in prototyping of designs,

communication encoding and filtering, random logic, video communication systems, real time image processing, device controllers, computer peripherals, pattern matching, high speed graphics, digital signal processing and the list goes on. PLD shipments from various vendors are expected to exceed $3 billion for 1999 [4]. The advantages of FPGAs come at a price, however. The nature of their architecture, and the abundance of user-programmable switches makes them relatively slow compared to some other devices in the programmable logic family. Regardless, FPGAs have a significant impact on the way a digital circuit design is done today.

Logic is implemented in FPGAs using many Basic Logic Elements (BLEs), known also as *Logic Cells*. Each BLE can implement a small amount of logic and optionally a flip-flop. These BLEs are grouped into *Configurable Logic Blocks(CLBs)*. The exact number of BLEs in each CLB varies from vendor to vendor (4 to 32 BLEs per logic block is common).

Within a logic block, the BLEs are connected using a programmable routing structure called an *Interconnect Matrix*, as shown in Figure 1. The Interconnect Matrix is a switch which connects:

• BLE outputs to BLE inputs within the same logic block, and

• Logic Block inputs to BLE inputs.

There are two types of Interconnect Matrices: *fully connected*, in which every connection between BLE output and BLE input and between logic block input and BLE input is possible, and *partially depopulated*, in which only certain connections between BLEs and the logic block input pins are possible.

Connections between the Logic Blocks are made using fixed wires/tracks. These tracks are connected to each other, and to the logic block inputs and outputs using programmable switches. These programmable switches are usually implemented using a pass transistor controlled by the output of a static RAM cell. By turning on the appropriate programmable switches, appropriate connections can be made between the logic block inputs and outputs.

Figure 1 shows an Island-Style FPGA architecture. In an island-style architecture the Logic Blocks are organized in an array separated by horizontal and vertical programmable routing channel. A *Switch Block* is used to make connections between the tracks in adjacent channels. Figure 1 shows only one switch block and four logic blocks, but real FPGAs consist of many switch blocks and logic blocks.

The Interconnect Matrix within a logic block and the fixed tracks and programmable switches outside of the logic blocks are collectively known as FPGA routing structures (often referred to as FPGA “routing architecture”). In most commercially available FPGAs, these routing structures consume the majority of the chip area, and are responsible for most of the circuit delay. As FPGAs are migrated to even more advanced technologies, the routing fabric becomes even more important [5]. Thus the optimization of these structures is very important. An FPGA with poor routing structure may suffer either in terms of routability, speed, or density. The more flexible the routing architecture, the more routable the device will be (i.e the more likely it is that a given connection between BLEs can be implemented). On the other hand, a more flexible architecture generally requires more transistors, consuming more chip area than is

needed, and adding extra parasitic capacitance to the wires connecting BLEs, slowing down the circuit more than is necessary.

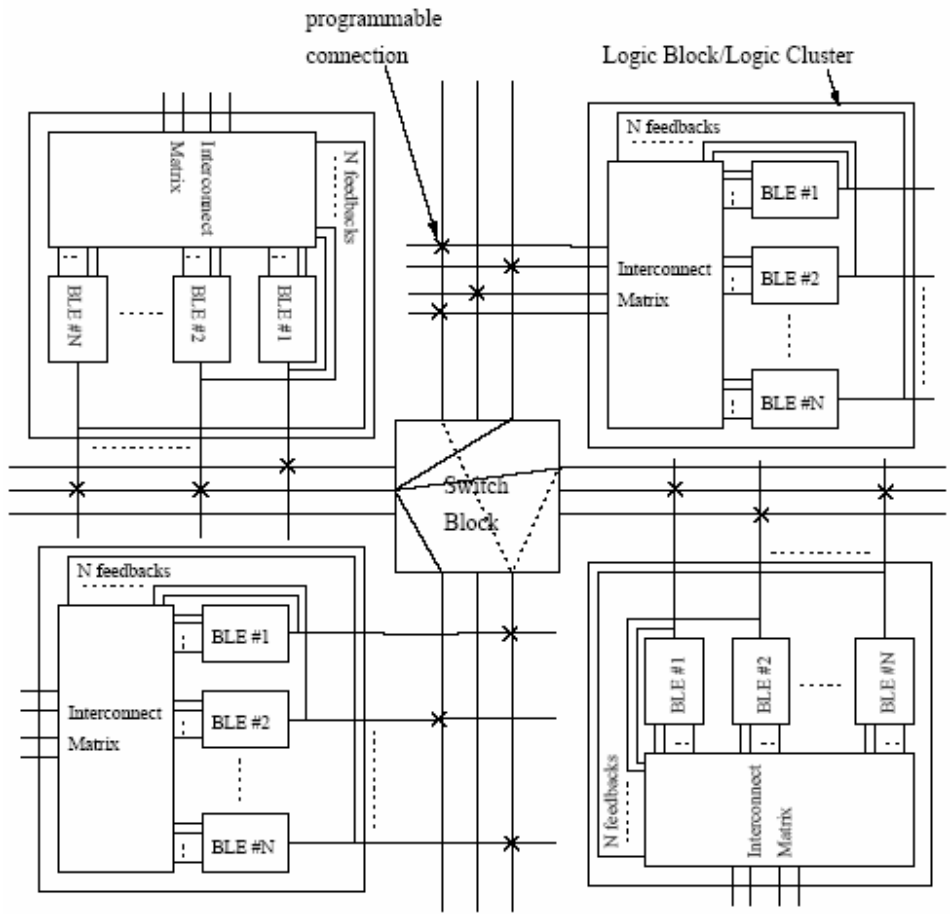


Figure 1 – **FPGA Architecture**

1.2. Organization of this study

This study is organized as follows: Chapter 2 provides background information, including FPGA architectures, general approaches to routing problems, routing algorithms and the definitions of basic terminology. It also describes representative examples of commercially available FPGAs, including a brief description of the routing architecture contained in each chip.

Chapter 3 gives a detailed overview of the experimental procedure and the experimental results and states comments and remarks over the results.

Chapter 4 presents conclusions and future work, and precisely states the contributions of this work.

Finally, Appendix presents the entirety of the graphs that were produced during this study.

2 Background Information

Introduction

This chapter provides some necessary background information that is assumed in various discussions. Section 2.2 provides an FPGA architecture overview. Section 2.3 introduces the two main fields of research, FPGA routing algorithms and FPGA routing architecture. Section 2.4 describes several commercially available FPGA devices to provide a point of reference for the FPGA model that is used throughout this work.

2.1 FPGA Architecture Overview

There are many different FPGA architectures available from various vendors including Xilinx [1], Altera [2], Actel [3], Lucent [6], QuickLogic [7], and Cypress [8]. Although the exact structure of these FPGAs varies from vendor to vendor, all FPGAs consist of three fundamental components: Logic Blocks, I/O blocks, and the Programmable Routing. What comprises of a logic block, and how the programmable routing is organized defines a particular architecture. A logic block is used to implement a small portion of the circuit being implemented using an FPGA. The programmable routing is used to make all the required connections among various logic block and the required connections to the I/O (input/output) blocks. Many commercially available FPGAs use an Island-style architecture in which logic blocks are organized in an array separated by horizontal and vertical programmable routing channel, as shown in Figure 2.

2.1.1 Logic Resources

The typical FPGA has a Basic Logic Element(BLE) with one or more 4-input LUT(s), optional D flip-flops (DFF), and some form of fast carry logic (Figure 3). These BLEs are grouped in Configurable Logic Blocks (CLBs or just Logic Block). The LUTs

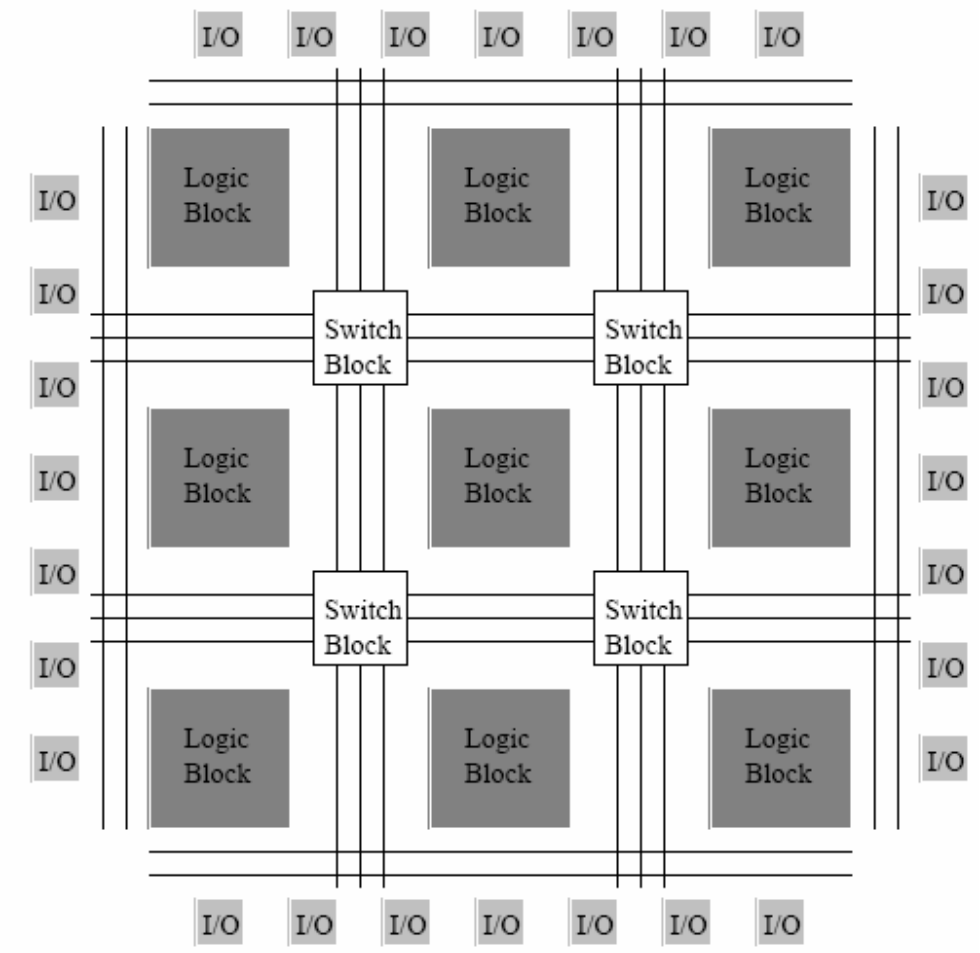


Figure 2 – **Island Style FPGA Architecture**

allow any function to be implemented, providing generic logic. The flip-flop can be used for pipelining, registers, stateholding functions for finite state machines, or any other situation where clocking is required. Note that the flip-flops will typically include programmable set/reset lines and clock signals, which may come from global signals routed on special resources, or could be routed via the standard interconnect structures from some other input or logic block. The fast carry logic is a special resource provided in the cell to speed up carry-based computations, such as addition, parity, wide AND operations, and other functions. These resources will bypass the general routing structure, connecting instead directly between neighbors in the same column. Since there are very

few routing choices in the carry chain, and thus less delay on the computation, the inclusion of these resources can significantly speed up carrybased computations.

Each Logic Block also contains an Interconnect Matrix. The Interconnect Matrix is a switch which connects BLE outputs to BLE inputs and Logic Block inputs to BLE inputs within each Logic Block, as shown in Figure 4.

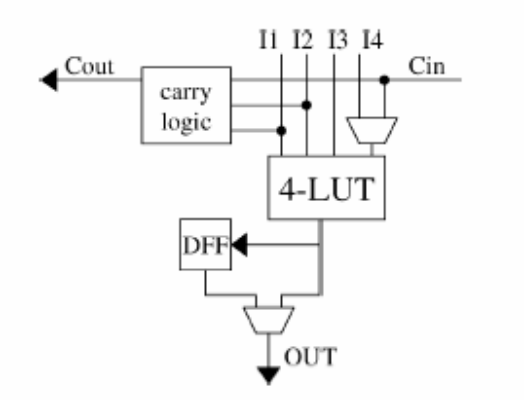


Figure 3 – **Basic Logic Element**

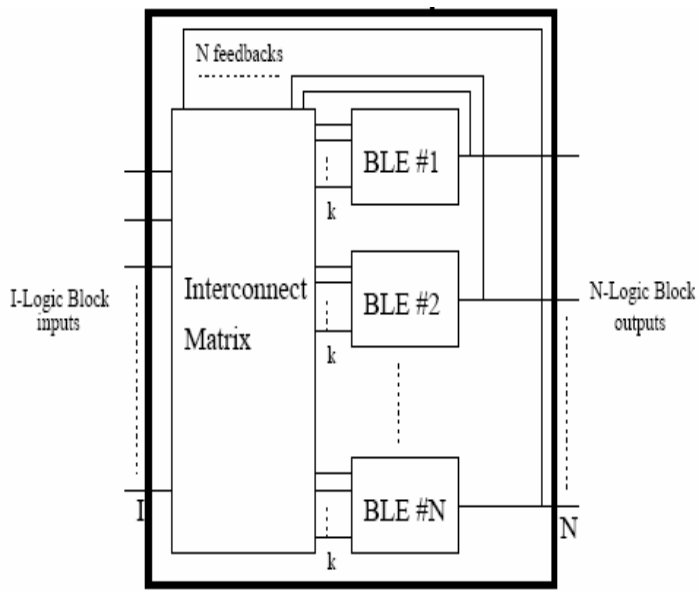


Figure 4 – **Logic Block**

2.1.2 Routing Resources

The programmable routing in an FPGA consists of two categories: (1) routing within each Configurable Logic Block, and (2) routing between the Configurable Logic Blocks. Figure 5 shows a detailed view of the routing for a single tile. Normally, an FPGA is created by replication of such a tile (a tile consists of one Logic Block and it's associated routing).

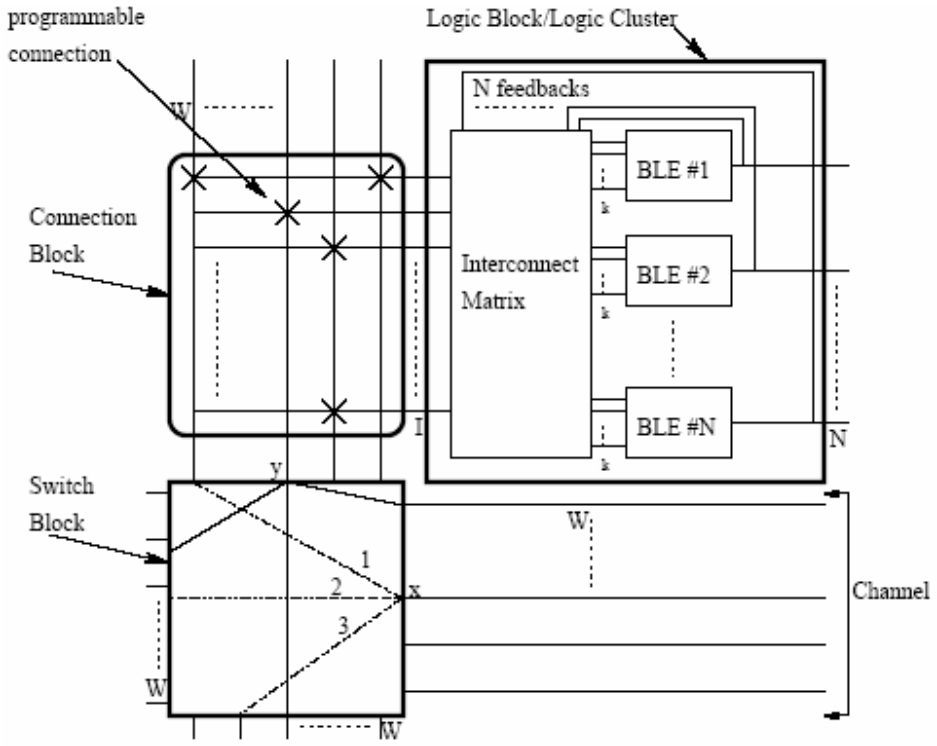


Figure 5 – **Detailed Routing Architecture**

The programmable routing within each Logic Block consists of the Interconnect Matrix. The programmable routing between the Logic Blocks consists of fixed metal tracks, Switch Blocks, Connection Blocks, and the programmable switches. The fixed metal tracks run horizontally and vertically, and are organized in channels; each channel contains the same number of tracks for the architecture that we investigated. A Switch

Block occurs at each intersection between horizontal and vertical routing channels, and defines all possible connections between these channels. The Connection Block (shown in Figure 5) defines all the possible connections from a horizontal or vertical channel to a neighboring logic block. The connections in the switch blocks and connection blocks are made by programmable switches. Part of the programmable routing also lies within each logic block, determining how different components are connected within the logic block. This Island-Style architecture is a very general version of most commercial architectures.

A programmable switch (Figure 6) consists of a pass transistor controlled by a static random access memory cell (in which case, the device is called a SRAM-based FPGA), or an anti-fuse (such devices are referred to as anti-fuse FPGAs), or a non-volatile memory cell (such devices are referred to as floating gate devices). Since SRAM-based FPGAs employ static random access memory (SRAM) cells to control the programmable switches, they can be reprogrammed by the end user as many times as required and are volatile. Of the three categories, SRAM-based FPGAs are most widely used and hence we will limit our discussion and investigations to SRAM-based devices.

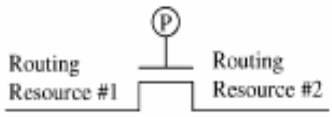


Figure 6 – **Programmable Routing Connection**

2.2 FPGA CAD Flow

In this section we present an overview of the entire CAD process that is necessary to implement a circuit in an FPGA. A typical CAD system for FPGAs consists of several

interconnected programs as illustrated in Figure 7. The input to the CAD system is a functional description of a network, usually expressed in a standard format such as boolean equations. The equations are read by a logic optimization [9] [10] tool, which performs manipulations of the equations so as to optimize area, delay, or a combination of area and delay. This step usually performs the equivalent of an algebraic minimization of the boolean equations and is appropriate when implementing a circuit in any medium, not just FPGAs. To transform the boolean equations into a circuit of FPGA logic cells, the optimized network is fed to a technology mapping program [11] [12] [13]. This step maps the equations into logic cells, which also presents opportunity to optimize, either to minimize the total number of logic cells required (area optimization) or the number of logic cells in time-critical paths (delay optimization). The circuit of logic cells is then passed to a placement program [14] [15] [16], which selects a specific location in the FPGA for each logic cell. Typical placement algorithms usually attempt to minimize the total length of interconnect required for the resulting placement.

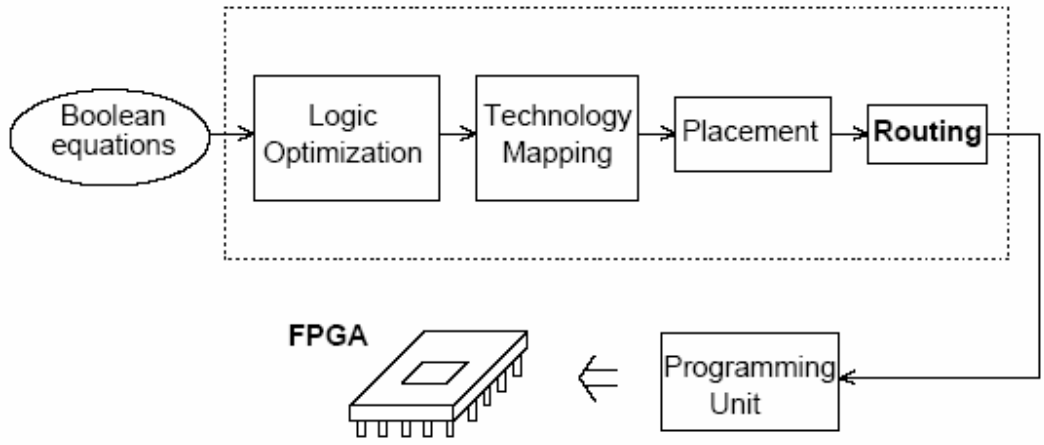


Figure 7 **– A typical FPGA CAD system**

The final step in the CAD system is performed by the routing software, which allocates the FPGA's routing resources to interconnect the placed logic cells. The routing

tools must ensure that 100 percent of the required connections are formed, and may be required to maximize the speed performance of time-critical connections. Finally, the CAD system's output is fed to a programming unit that is used to configure the FPGA.

2.3 Routing Algorithms

In current FPGAs, the most of the tile area is devoted to routing resources and most of the delay of circuits implemented on FPGAs is due to routing. Software that performs automatic routing has existed for many years, with the first algorithms designed to route printed circuit boards. Over the years there have been many publications concerning routing algorithms, so that the problem is well defined and understood.

2.3.1 Routing Terminology

The following list gives common routing terms, as they are defined for FPGA routing in this study:

- *Pin* - a logic cell input or output.
- *Connection* - a pair of logic cell pins that are to be electrically connected.
- *Net* - a set of logic cell pins that are to be electrically connected. A net can be divided into one or more connections.
- *Wiring segment* - a straight section of wire that is used to form part of a connection.
- *Routing switch* - a device that is used to electrically connect two wiring segments.
- *Track* - a straight section of wire that spans the entire width or length of a routing

channel. A track can be composed of a number of wiring segments of various lengths.

- *Routing channel* - the rectangular area that lies between two rows or two columns of logic cells. A routing channel contains a number of tracks.

2.3.2 General Approach to Routing - PATHFINDER Algorithm

Routing is one of the most challenging problems of FPGAs. This problem of routing FPGAs can be stated simply as that of assigning signals to routing resources in order to successfully route all signals while achieving a given overall performance. The first goal, complete routing of all signals, is difficult to achieve in FPGAs because of the lack of routing resources. The usual approach to achieving this goal is to minimize the use of routing resources by constructing minimum routing trees for each signal. Although this reduces the demand for routing resources, signals will still compete for the same resources and the challenge is to find a way to allocate resources so that all signals can be routed. The second goal, minimizing delay, requires the use of minimum delay routes for signals, which in general are much more expensive in terms of routing resources than minimum routing trees. Thus the solution to the entire routing problem requires the simultaneous solution to two interacting and competing subproblems.

Various algorithms, concerning the FPGA routing problem, have been presented so far. Most of them [17], [18], [19] targeted in a standard rip-up and retry approach by ordering

the nets to be routed such that critical nets are routed most directly. The most efficient algorithm up to date is that presented by McMurchie and Ebeling [20], the Pathfinder algorithm.

PathFinder is an iterative algorithm that balances the competing goals of eliminating congestion and minimizing delay of critical paths in an iterative framework. In this framework, signals are allowed to share routing resources initially, but subsequently must negotiate with other signals to determine which signal needs the shared resource most. A timing analysis is performed every iteration to apply pressure continuously to routes that can potentially become critical if left unchecked. PathFinder is derived from an iterative scheme for the global routing of custom IC's developed by Nair [21]. This scheme differs in several aspects from most forms of rip-up and retry.

PathFinder is composed of two parts: a signal router, which routes one signal at a time using a shortest-path algorithm, and a global router, which calls the signal router to route all signals, adjusting the resource costs in order to achieve a complete routing. The signal router uses a breadth-first search to find the shortest path given a congestion cost and delay for each routing resource. The global router dynamically adjusts the congestion penalty of each routing resource based on the demands signals place on that resource. During the first iteration of the global router there is no cost for sharing routing resources and individual routing resources may be used by more than one signal. However, during subsequent iterations the penalty is gradually increased so that signals in effect negotiate for resources. Signals may use shared resources that are in high demand if all alternative routes utilize resources in even higher demand; other signals will tend to spread out and use resources in lower demand. The global router reroutes signals using the signal router until no more resources are shared. The use of a cost function that *gradually* increases the penalty for sharing is a significant departure from Nair's algorithm, which assigns a cost of infinity to resources whose capacity is exceeded. In addition to

minimizing congestion, the signal router ensures that the delays of all signal paths stay within the critical path delay. For multiple sinks, low congestion cost can be achieved by a minimum Steiner tree, but this can result in long delays. Low delay can be achieved by a minimum-delay tree, but this may mean competition by many signals for the same routing resources. To achieve a balance, the signal router uses the relative contribution of each connection in the circuit (i.e. source-sink pair) to the overall delay of the circuit to determine how to trade off congestion and delay. Thus, every connection on the longest path has a slack ratio of 1, while connections on the least critical paths have slack ratios close to 0. The inverse of the slack ratio gives the factor by which the delay of a path can be expanded before the circuit is slowed down. The key idea behind the signal router is that connections with a slack ratio close to 1 will be assigned greater weight in negotiating for resources and consequently will be routed directly (i.e. using a minimum-delay route) from source to sink. Connections with a small slack ratio will have less weight and pay more attention to congestion-avoidance during routing.

Since Pathfinder was presented, there have been many router implementations based on it. The VPR (versatile place and route) router [22], which is such an implementation of the PathFinder algorithm, is known to be the best routing tool for FPGAs to date.

2.4 Commercially Available FPGAs

This section provides a detailed description of two commercially available FPGA families, including those from Xilinx Co and Altera. These particular FPGAs have been chosen because they are representative examples of state-of-the-art devices and they are

in widespread use. Each device is described in terms of its general architecture, its choice of programmable cell, its routing architecture, and its CAD routing tools. Enough details are given, and in some cases specific comments are made, to show how the routing resources and architecture of each device relates to the research contained in this study.

2.4.1 XILINX FPGAs

The general architecture of Xilinx FPGAs is shown in Figure 8. It consists of a two-dimensional array of programmable cells, called Configurable Logic Blocks (CLBs), with horizontal routing channels between rows of cells and vertical routing channels between columns. Programmable resources are configured by Static RAM cells, and each routing switch is implemented as a specially designed transistor controlled by an SRAM bit. There are two families of Xilinx FPGAs described here, called the VIRTEX II and SPARTAN 3. Table 1 gives an indication of the logic capacities of each family by showing the number of CLBs and an equivalent gate count. The design details of the Xilinx CLB and routing architecture for each family will each be described in turn.

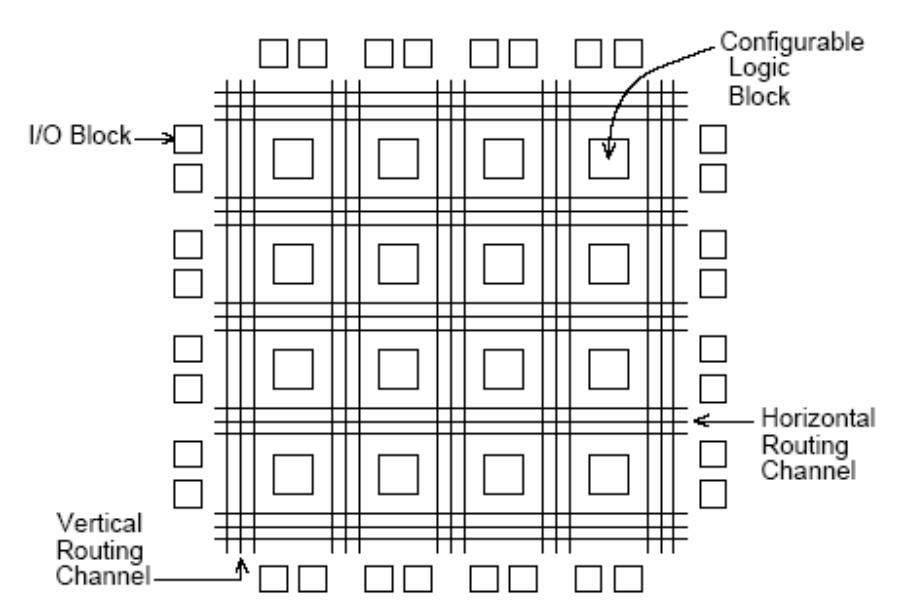


Figure 8 - **General Architecture of Xilinx FPGAs**

Family	Number of CLBs	Number of Gates
VIRTEX II	64 – 11,648	40K – 8M
SPARTAN3	192 – 8,320	50K – 5M

Table 1 - **Xilinx FPGA Logic Capacities**

2.4.1.1 XILINX VIRTEX II

The Virtex-II family is a platform FPGA developed for high performance from low-density to high-density designs that are based on IP cores and customized modules. The family delivers complete solutions for telecommunication, wireless, networking, video, and DSP applications, including PCI, LVDS, and DDR interfaces.

Virtex-II devices are user-programmable gate arrays with various configurable elements. The Virtex-II architecture is optimized for high-density and high-performance logic designs. As shown in Figure 9, the programmable device is comprised of input/output blocks (IOBs) and internal configurable logic.

The internal configurable logic includes four major elements organized in a regular array.

• Configurable Logic Blocks (CLBs) provide functional elements for combinatorial and synchronous logic, including basic storage elements. BUFTs (3-state buffers) associated with each CLB element drive dedicated segmentable horizontal routing resources.

• Block SelectRAM memory modules provide large 18 Kbit storage elements of dual-port RAM.

• Multiplier blocks are 18-bit x 18-bit dedicated multipliers.

• DCM (Digital Clock Manager) blocks provide self-calibrating, fully digital solutions for

clock distribution delay compensation, clock multiplication and division, coarse- and fine-grained clock phase shifting.

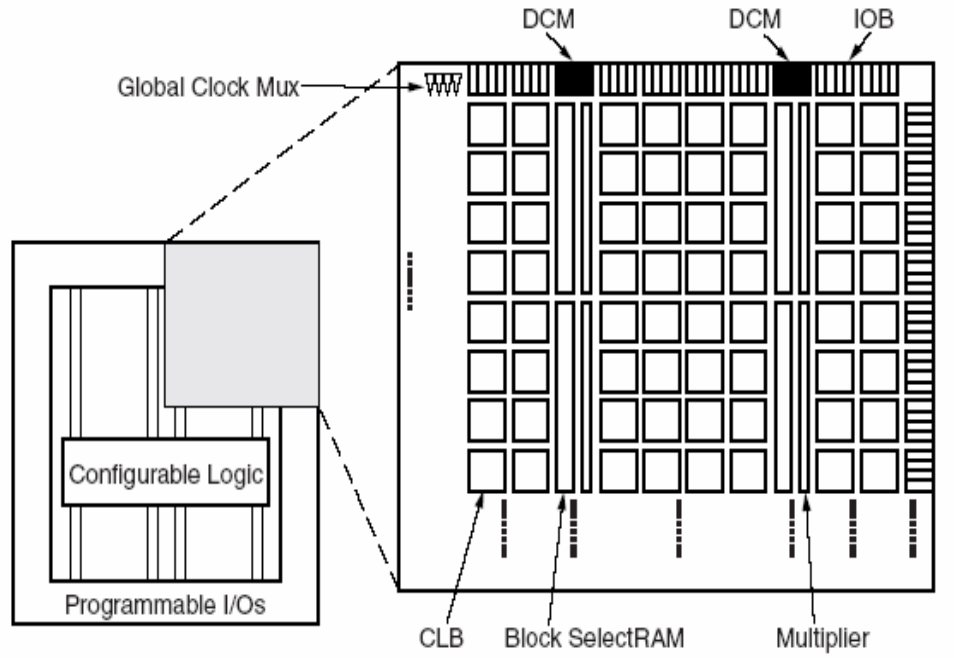


Figure 9 - **Virtex-II Architecture Overview**

The Virtex-II configurable logic blocks (CLB) are organized in an array and are used to build combinatorial and synchronous logic designs. Each CLB element is tied to a switch matrix to access the general routing matrix, as shown in Figure 10. A CLB element comprises 4 similar slices, with fast local feedback within the CLB, and two 3-state buffers. The four slices are split in two columns of two slices with two independent carry logic chains and one common shift chain.

Each slice includes two 4-input function generators, carry logic, arithmetic logic gates, wide function multiplexers and two storage elements. As shown in Figure 11, each 4-input function generator is programmable as a 4-input LUT, 16 bits of distributed SelectRAM memory, or a 16-bit variable-tap shift register element. In addition, the two storage elements are either edge-triggered D-type flip-flops or level-sensitive latches.

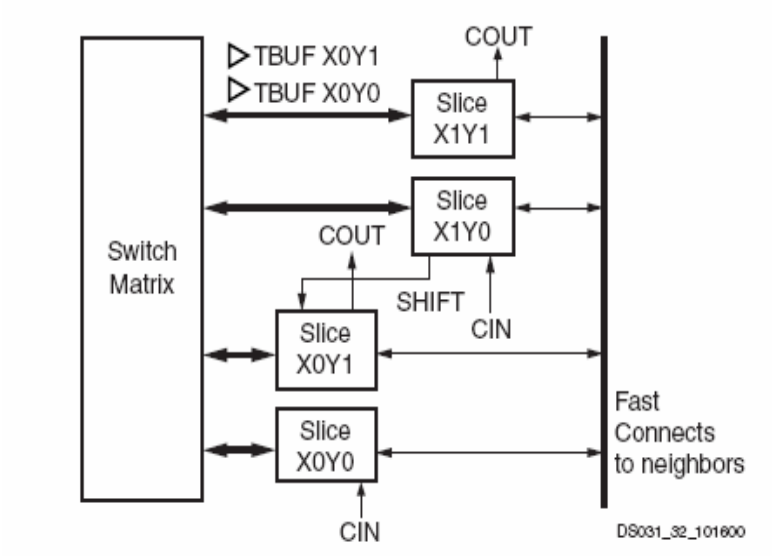


Figure 10 - **Virtex-II CLB Element**

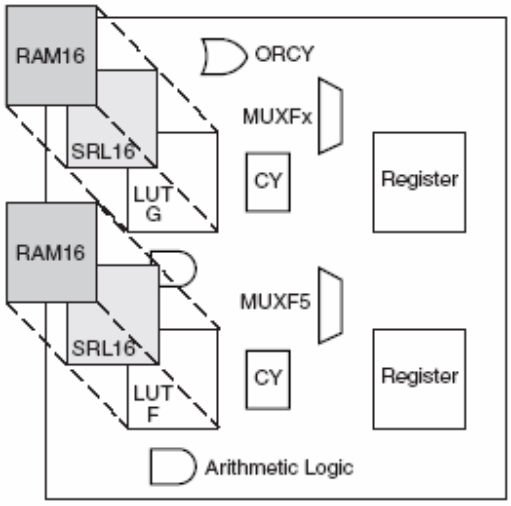


Figure 11 - **Virtex-II Slice Configuration**

Interconnection Scheme – Routing Resources

Local and global Virtex-II routing resources are optimized for speed and timing predictability, as well as to facilitate IP cores implementation. Virtex-II buffered

interconnects are relatively unaffected by net fanout and the interconnect layout is designed to minimize crosstalk. Virtex-II Active Interconnect Technology is a fully buffered programmable routing matrix. All routing resources are segmented to offer the advantages of a hierarchical solution. Virtex-II logic features like CLBs, IOBs, block RAM, multipliers, and DCMs are all connected to an identical switch matrix for access to global routing resources, as shown in Figure 12.

Most Virtex-II signals are routed using the global routing resources, which are located in horizontal and vertical routing channels between each switch matrix. As shown in Figure 13, Virtex-II has fully buffered programmable interconnections, with a number of resources counted between any two adjacent switch matrix rows or columns. Fanout has minimal impact on the performance of each net.

• The long lines are bidirectional wires that distribute signals across the device. Vertical and horizontal long lines span the full height and width of the device.

• The hex lines route signals to every third or sixth block away in all four directions. Organized in a staggered pattern, hex lines can only be driven from one end. Hex-line signals can be accessed either at the endpoints or at the midpoint (three blocks from the source).

• The double lines route signals to every first or second block away in all four directions. Organized in a staggered pattern, double lines can be driven only at their endpoints. Double-line signals can be accessed either at the endpoints or at the midpoint (one block from the source).

• The direct connect lines route signals to neighboring blocks: vertically, horizontally, and diagonally.

• The fast connect lines are the internal CLB local interconnections from LUT outputs to LUT inputs.

In addition to the global and local routing resources, dedicated signals are available.

• There are eight global clock nets per quadrant

• Horizontal routing resources are provided for on-chip 3-state busses. Four partitionable bus lines are provided per CLB row, permitting multiple busses within a row.

• Two dedicated carry-chain resources per slice column (two per CLB column) propagate carry-chain MUXCY output signals vertically to the adjacent slice.

• One dedicated SOP chain per slice row (two per CLB row) propagate ORCY output logic signals horizontally to the adjacent slice.

• One dedicated shift-chain per CLB connects the output of LUTs in shift-register mode to the input of the next LUT in shift-register mode (vertically) inside the CLB.

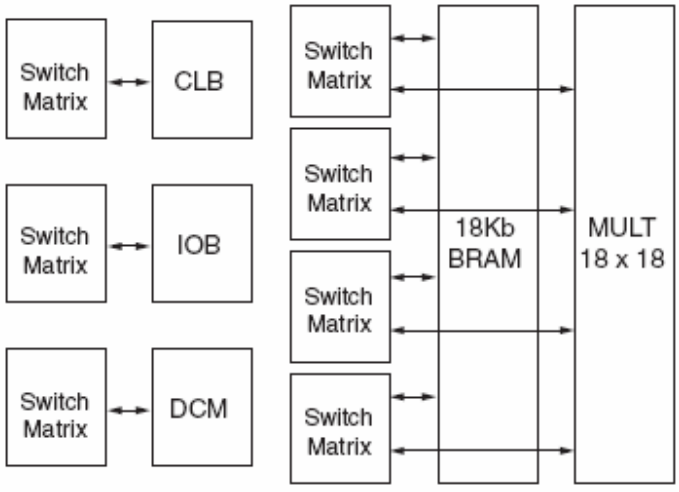


Figure 12 - **Active Interconnect Technology**

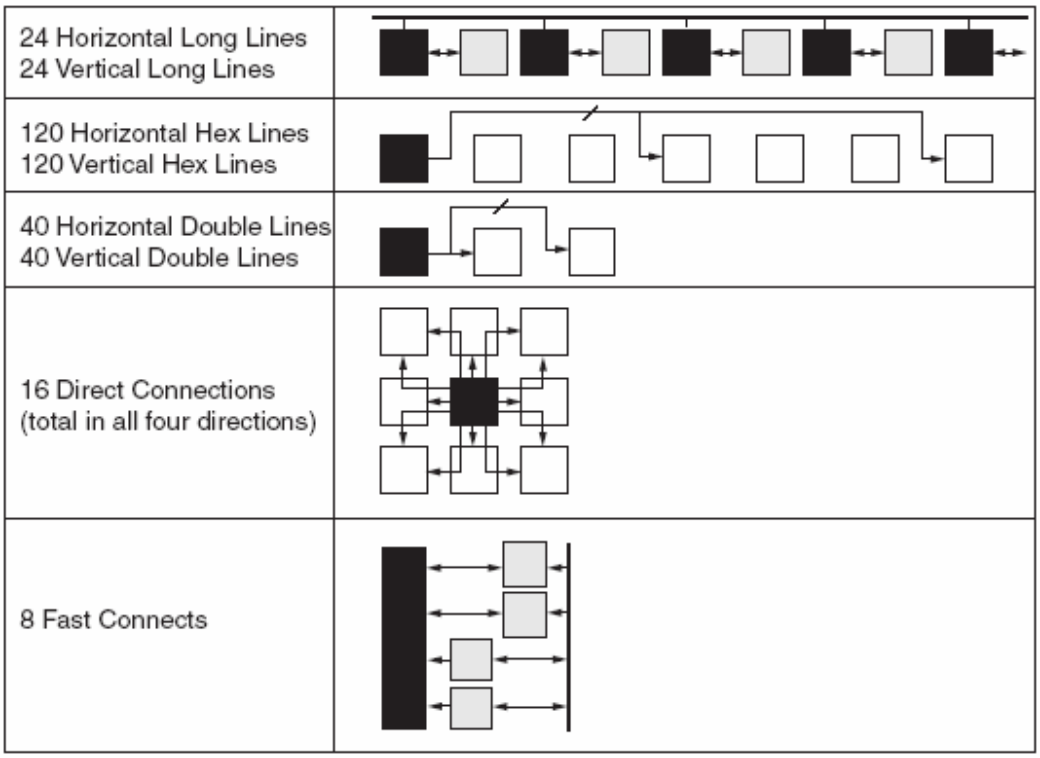


Figure 13 - **Hierarchical Routing Resources**

2.4.1.2 XILINX SPARTAN 3

The Spartan™-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates, as shown in Table 1. The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment.

The Spartan-3 family architecture consists of five fundamental programmable

functional elements:

• Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.

• Input/Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-four different signal standards, including seven high-performance differential standards, are available. Double Data-Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.

• Block RAM provides data storage in the form of 18-Kbit dual-port blocks.

• Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.

• Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

These elements are organized as shown in Figure 14. A ring of IOBs surrounds a regular array of CLBs. The columns of block RAM range from one to four, for the members of the family. Each column is made up of several 18K-bit RAM blocks; each block is associated with a dedicated multiplier. The DCMs are positioned at the ends of the outer block RAM columns.

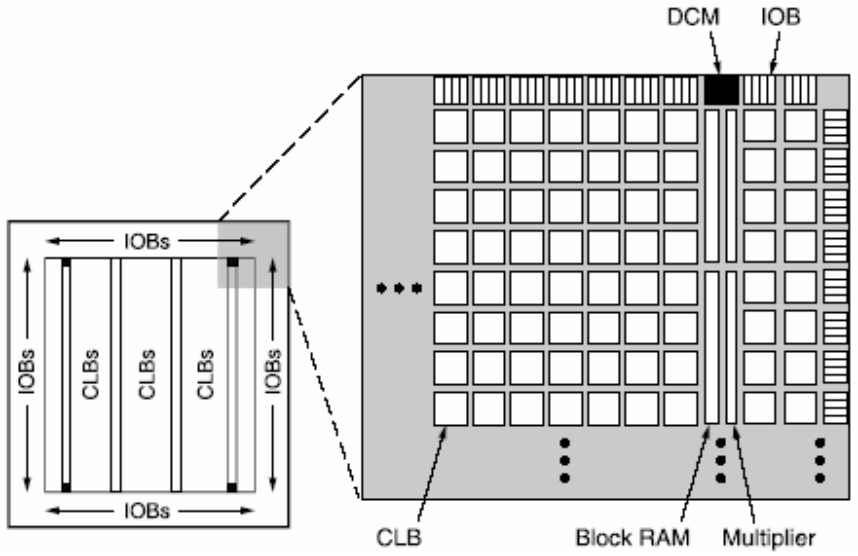


Figure 14 - **Spartan-3 Family Architecture**

The Configurable Logic Blocks (CLBs) constitute the main logic resource for implementing synchronous as well as combinatorial circuits. Each CLB comprises four interconnected slices, as shown in Figure 15. These slices are grouped in pairs. Each pair is organized as a column with an independent carry chain.

All four slices have the following elements in common: two logic function generators, two storage elements, wide-function multiplexers, carry logic, and arithmetic gates, as shown in Figure 16. Both the left-hand and right-hand slice pairs use these elements to provide logic, arithmetic, and ROM functions. Besides these, the left-hand pair supports two additional functions: storing data using Distributed RAM and shifting data with 16-bit registers.

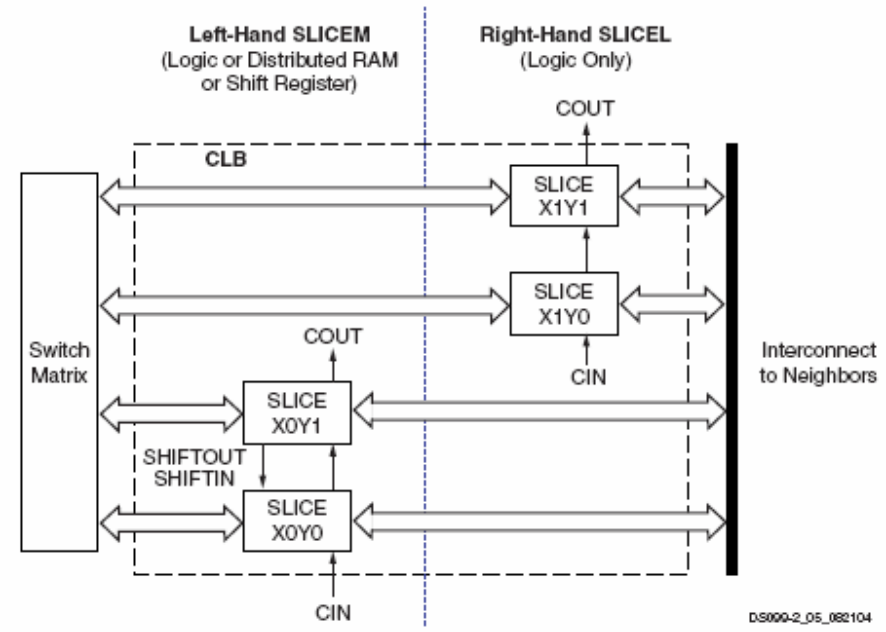


Figure 15 - **Arrangement of slices within the CLB**

Interconnection Scheme – Routing Resources

The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

Interconnect (or routing) passes signals among the various functional elements of Spartan-3 devices. There are four kinds of interconnect: Long lines, Hex lines, Double lines, and Direct lines.

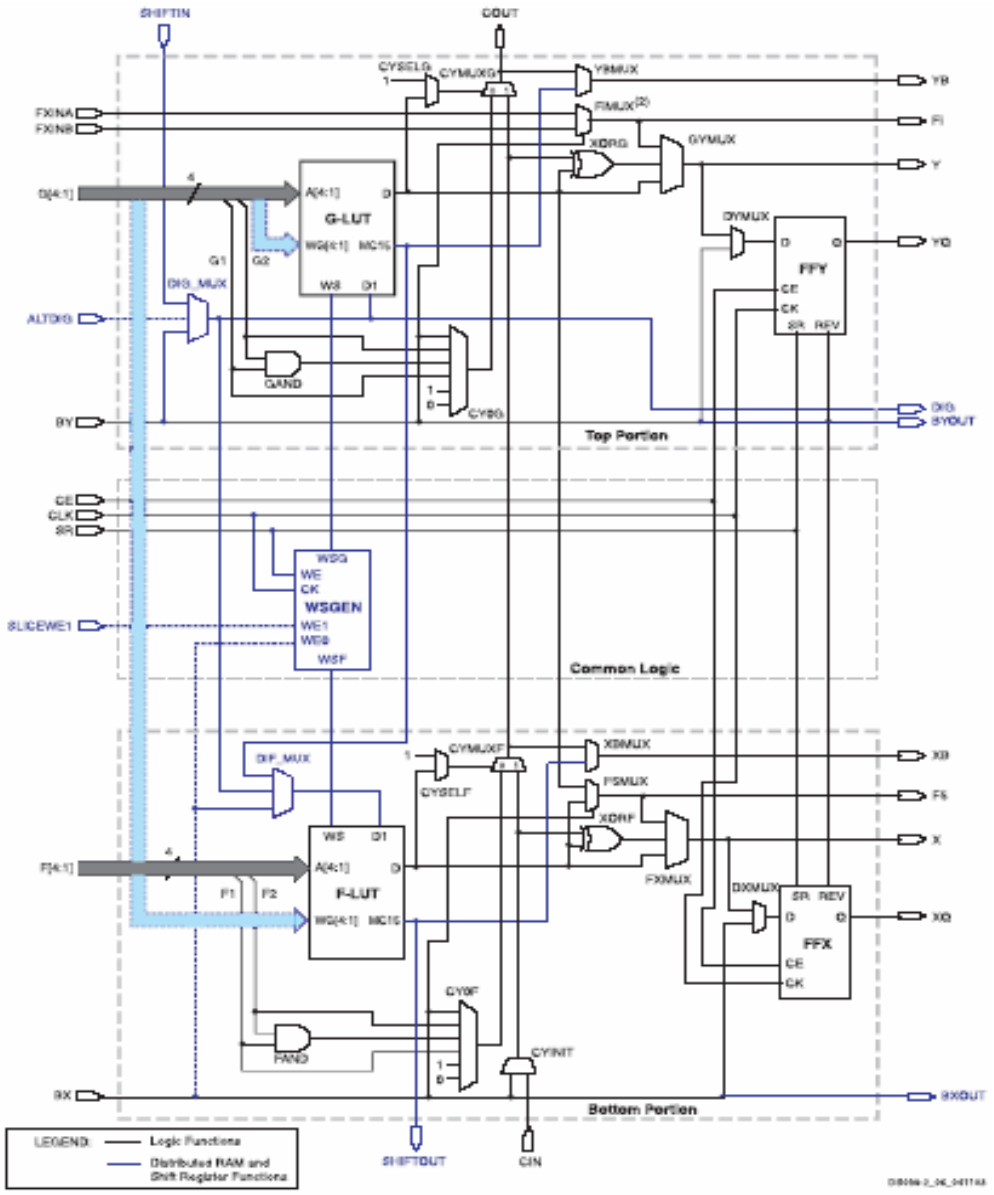


Figure 16 - **Simplified diagram of a Slice**

Long lines connect to one out of every six CLBs (see Figure 17a). Because of their low capacitance, these lines are well-suited for carrying high-frequency signals with minimal loading effects (e.g. skew). If all eight Global Clock Inputs are already committed and there remain additional clock signals to be assigned, Long lines serve as a good alternative.

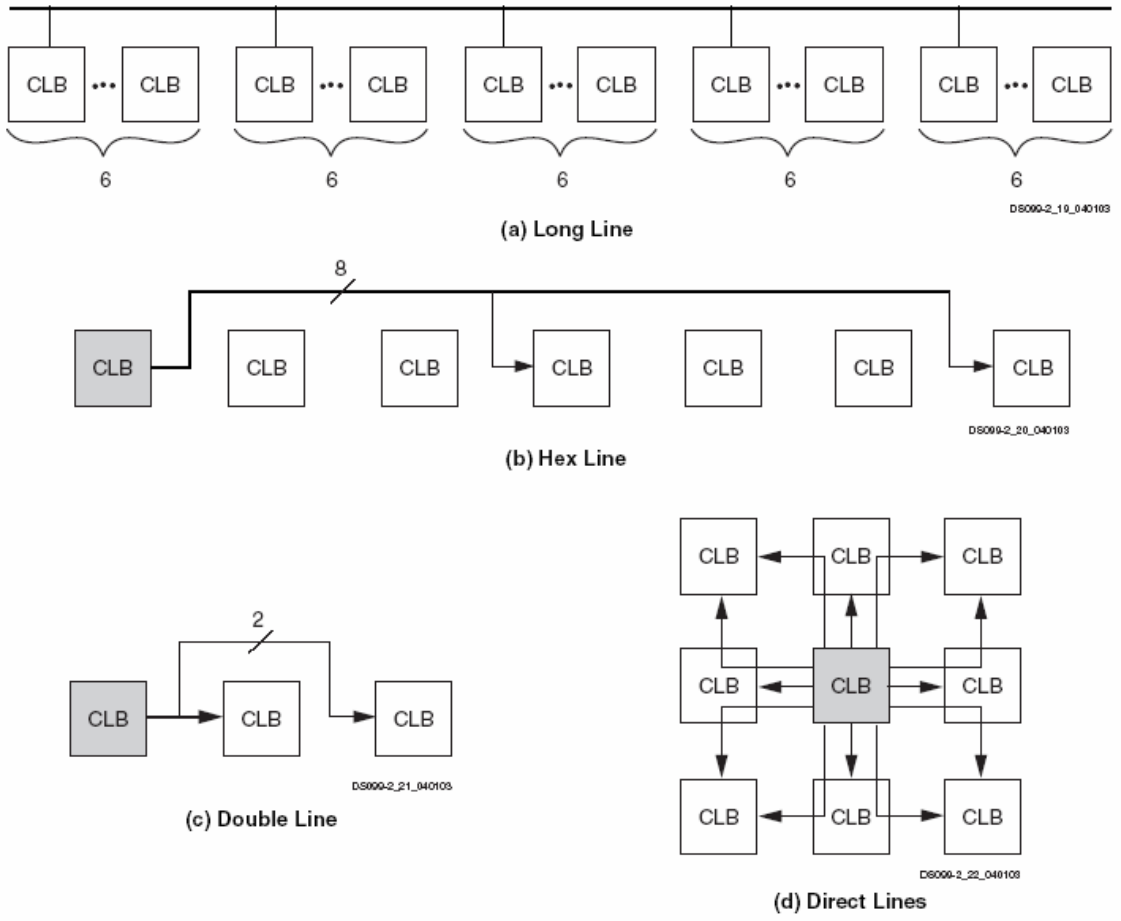


Figure 17 - **Types of Interconnect**

Hex lines connect one out of every three CLBs (see Figure 17b). These lines fall between Long lines and Double lines in terms of capability: Hex lines approach the high-frequency characteristics of Long lines at the same time, offering greater connectivity.

Double lines connect to every other CLB (see Figure 17c). Compared to the types of lines already discussed, Double lines provide a higher degree of flexibility when making connections.

Direct lines afford any CLB direct access to neighboring CLBs (see Figure 17d). These lines are most often used to conduct a signal from a "source" CLB to a Double, Hex, or Long line and then from the longer interconnect back to a Direct line accessing a "destination" CLB.

2.4.3 ALTERA FPGAs

Altera FPGAs [2] are considerably different from the others discussed above because they resemble large Programmable Logic Devices. Nonetheless, they are functionally equivalent to FPGAs because they employ a two-dimensional array of programmable cells and a programmable routing structure, they can implement multi-level logic, and they are user-programmable. Altera's general architecture, which is based on an EPROM programming technology, is illustrated in Figure 18. It consists of an array of programmable cells, called Logic Array Blocks (LABs), interconnected by a routing resource called the Programmable Interconnect Array (PIA).

The Altera LAB is by far the most complex logic cell of any of the FPGA families describe. A LAB can be thought of as an efficient PLD, as will be explained in the following paragraphs. Each LAB, as seen in Figure 19, consists of two major blocks, called the Macrocell Array and the Expander Product Terms.

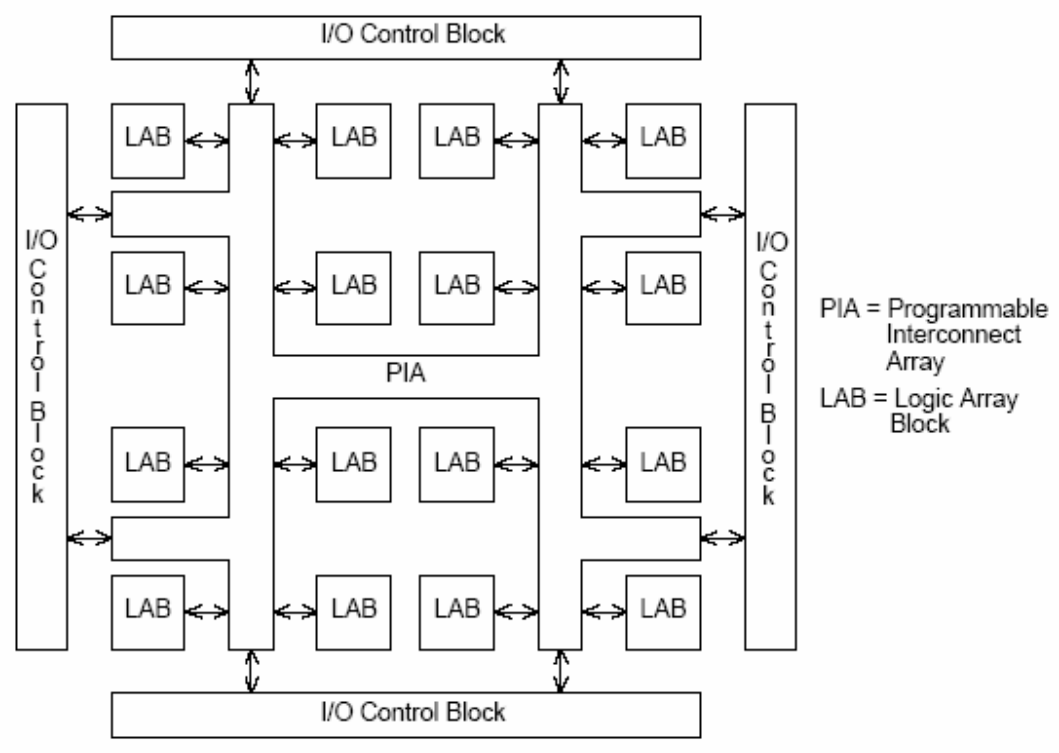


Figure 18 - **General Architecture of Altera FPGAs**

The Macrocell Array is a one-dimensional array of elements called Macrocells, where the number of elements in the array varies with each Altera device. As illustrated in Figure 20, each Macrocell comprises three wide AND gates that feed an OR gate which connects to an XOR gate, and a flip-flop. The XOR gate generates the Macrocell output and can optionally be registered. In Figure 20, the inputs to the Macrocell are shown as single-input AND gates because each is generated as a wired-AND (called a pterm) of the signals drawn on the left-hand side of the figure. A p-term can include any signal in the PIA, any of the LAB Expander Product Terms (described below), or the output of any other Macrocell. With this arrangement the Macrocell Array functions much like a PLD, but with fewer product terms per register (there are usually at least eight product terms per register in a PLD). Altera claims [2] that this makes the LAB

more efficient because most logic functions do not require the large number of p-terms found in PLDs and the LAB supports wide functions by way of the Expander Product Terms.

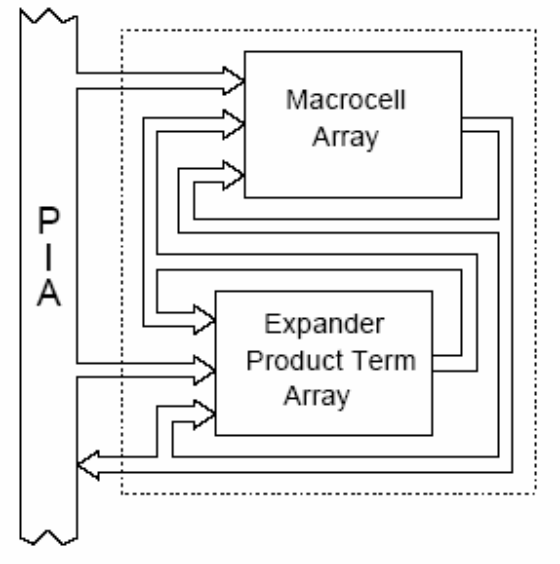


Figure 19 - **Altera LAB**

As illustrated in Figure 20, each Expander Product Terms block consists of a number of p-terms (the number shown in the figure is only suggestive) that are inverted and fed back to the Macrocell Array, and to itself. This arrangement permits the implementation of very wide logic functions because any Macrocell has access to these extra p-terms.

The Altera routing structure, the PIA, consists of a number of long wiring segments that pass adjacent to every LAB. The PIA provides complete connectivity because each LAB input can be programmably connected to the output of any LAB, without constraints. With this arrangement, routing an Altera FPGA is trivial, since there are no routing constraints. However, as mentioned previously for Actel FPGAs, this level of connectivity is excessive and could probably be reduced, given an appropriate routing

algorithm.

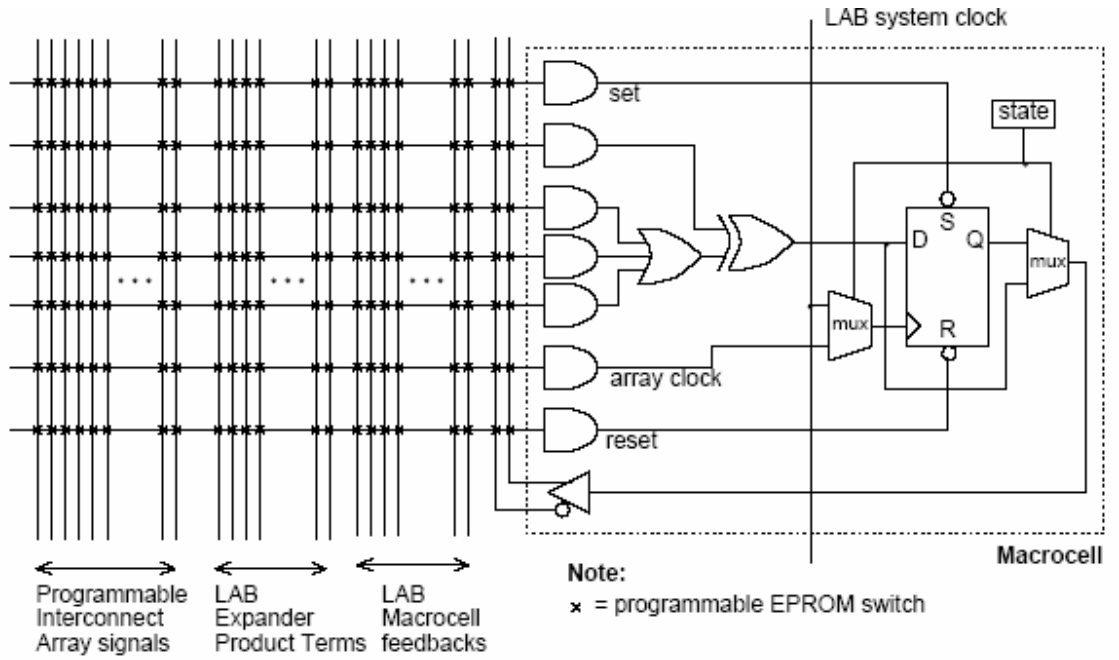


Figure 20 - **Altera Macrocell**

3 Experimenting on Field-Programmable Gate Array Routing

In Chapter 2, we presented background information concerning FPGAs in general, and specially routing and commercially available FPGAs. In this chapter we give the objects of this study and present a detailed overview of the experimental procedure followed and of the experimental results acquired.

3.1 General Approach and Problem Definition

The problem of routing efficiently designs in FPGAs is very challenging, as mentioned in the previous chapter. There are various factors that affect this problem, like routing resources, routing algorithms and routing architectures. Our study concentrated most on the first factor. The object was to make a detailed examination of FPGA routing resources, based on many experimental results. The experiments focus on the relationship between the length of a connection and its delay. This kind of examination could lead to useful conclusions and could arise some important questions.

In order to complete such an experimental procedure, CAD system tools were necessary. We decided to use Xilinx CAD system tools and especially *Xilinx ISE 6.1i* version. This version provides all the necessary tools, a very good environment and efficient documentation, so it fitted perfectly to our study.

For the purpose of this study three commercially available FPGAs were chosen: xc2v1500-6bg575, xc2v3000-6bf957 and xc3s1500-4fg676. In figure 21, we give an ordering example of a VirtexII. VirtexII and Spartan3 are two well known FPGA device families used for a wide variety of designs. So these three devices suited exactly our needs.

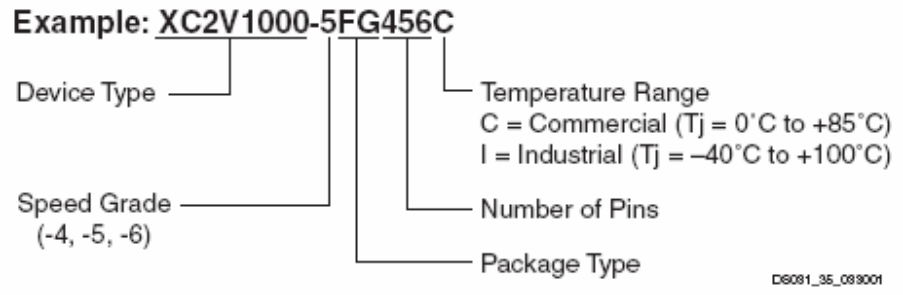


Figure 21 – **Virtex II Ordering Example**

The experimental procedure consisted of two basic phases, in order to achieve better results:

1. In the first phase, we experimented in almost empty FPGAs using a very simple design we created for our study. So, logic utilization was very close to 0%.
2. In the second phase, similar experiments were made. But this time we used a second larger design[23] in addition. So changing this design gave us the opportunity to have various values of utilization for each device.

The experimental procedure of each phase is presented in detail in the following section.

3.2 Experimental Procedure – Experimental Results

Description of phase 1

The first step was to create a very simple design which would be used for the experiments. The purpose of this simplicity was our will to be able to manually place and route the logic elements of the design. The design is actually two inverters in a row, which means that the output is same as the input. We also have two registers, one right after the input and one before the output. And, of course, DFFs. We used three bus widths for the input and output: 1, 8 and 32 bits. For the creation of the design we used *Project Navigator* which is the Xilinx CAD tool offered for this purpose. Using the Project

Navigator we also synthesized the design and completed it's logic optimization. The exact design in VHDL is presented here.

Our Design in VHDL

```
entity our_design is
    Port ( A : in std_logic;
           Reset : in std_logic;
           Clk : in std_logic;
           B : out std_logic );
end our_design;

architecture Structural of our_design is

component reg_bit
       Port ( X : in std_logic;
          Reset : in std_logic;
          Clk : in std_logic;
          Y : out std_logic);
end component ;

component invert
       Port ( X : in std_logic;
          Reset : in std_logic ;
          Clk : in std_logic ;
          Y : out std_logic);
end component ;

signal temp_a, temp_b, temp_c : std_logic ;

begin

       rg_a : reg_bit port map (A, Reset, Clk, temp_a) ;
       inv_a: invert port map ( temp_a, Reset, Clk, temp_b) ;
       inv_b: invert port map ( temp_b, Reset, Clk, temp_c) ;
       rg_b : reg_bit port map    (temp_c, Reset, Clk, B) ;

end Structural;
```

After the logic optimization, the next step was to place and route the design. The Xilinx tool offered for this purpose is the *FPGA Editor*, a very versatile tool which gives you the choice either to auto-place and route or manually place and route a design. In figure 22, the basic window of FPGA Editor is presented, showing an already placed design.

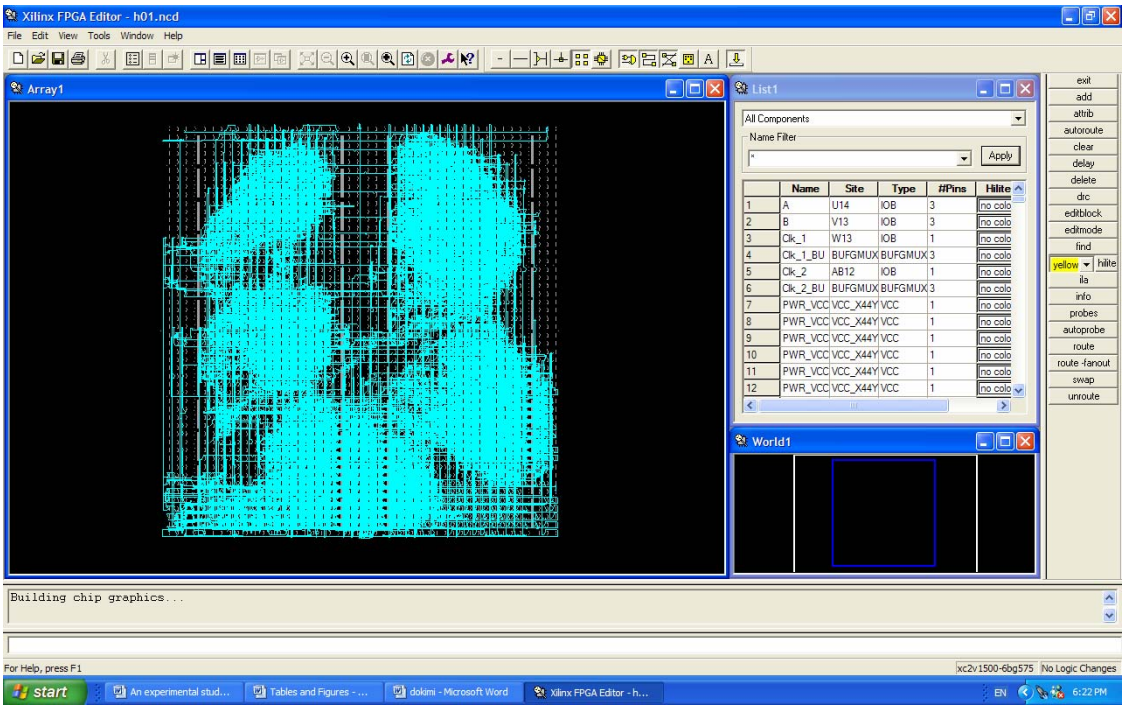


Figure 22 – **FPGA Editor (main window)**

Our design needs only three logic cells in order to be fully placed after the logic optimization has taken place. The idea was to manually place and route the design many times, producing in this way many maps, which will differ in certain characteristics. The characteristic that we changed was the length between two of the logic cells. Because the logic cells are connected (form a connection), we will refer to this characteristic as Connection Length.

In the beginning, we placed the logic to nearby logic cells in one side of the FPGA. Then, keeping one slice in its place, we changed the placement (and so the routing) of the second increasing each time the connection length (in the same direction).The increment value was one CLB length. This process continued until we reached the other side of the FPGA. Figure 23 depicts this process, in order to be more

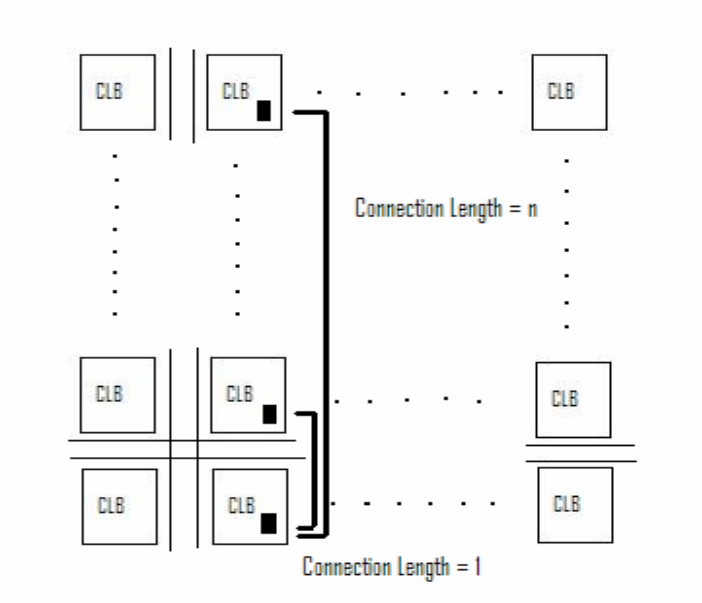


Figure 23 – **Experimental Process**

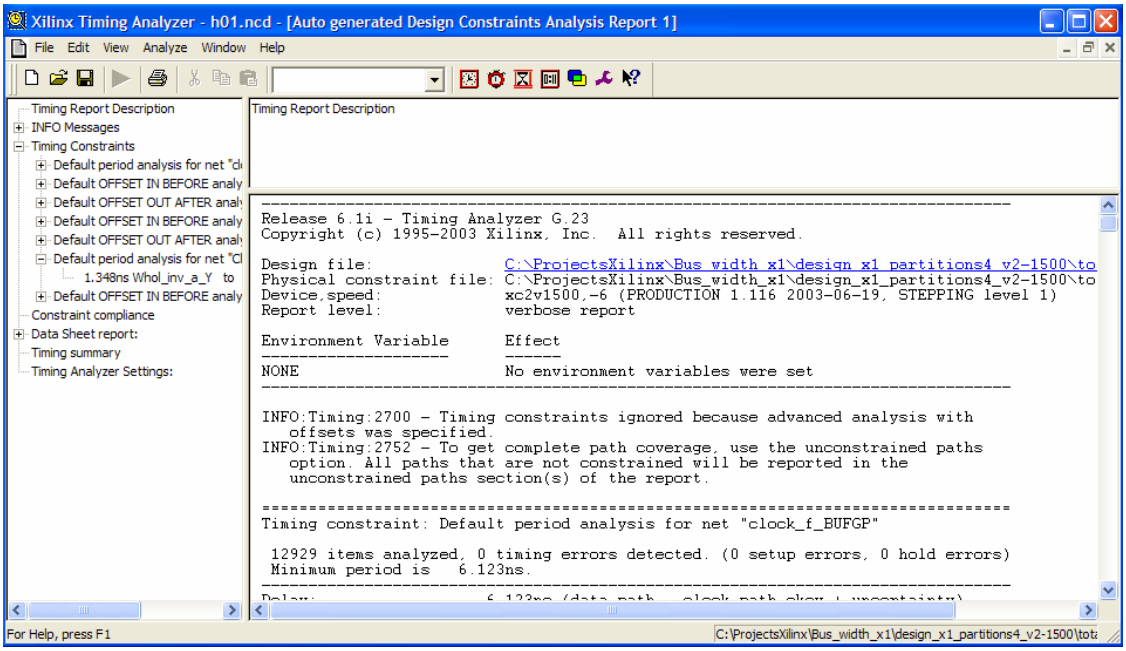


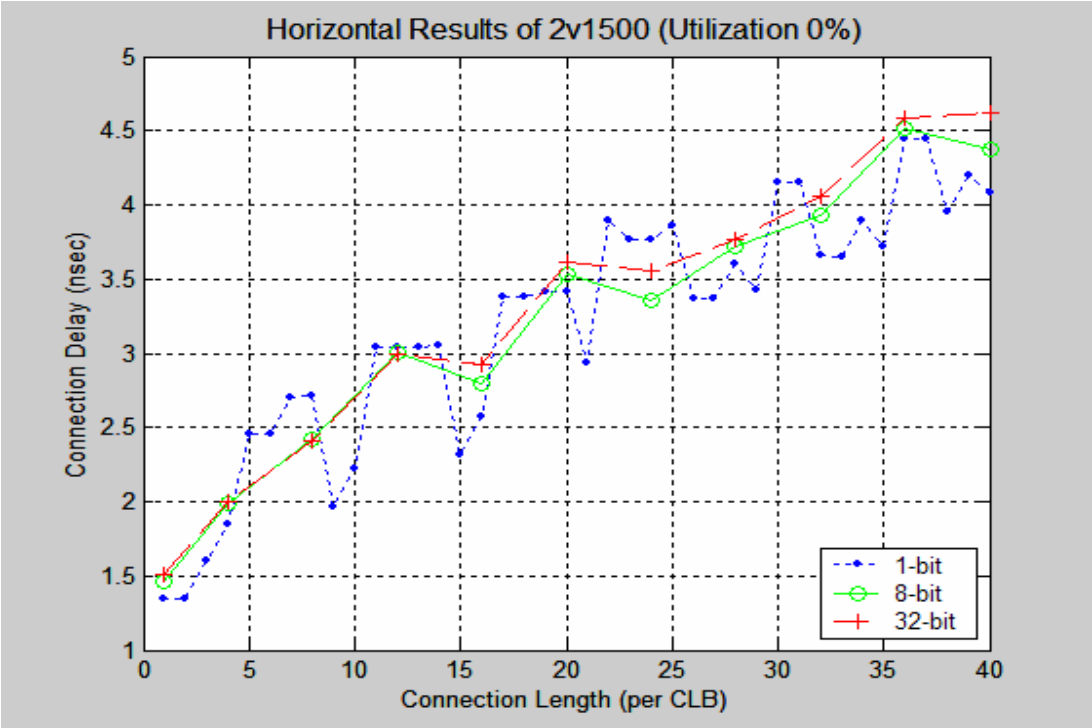
Figure 24 – **Timing Analyzer (main window)**

understandable. This procedure was followed for both the vertical and horizontal dimension of the FPGA. We describe the procedure for bus width 1-bit. The same exact procedure was followed for bus width 8 and 32 bits. The difference is that we have 8 and 32 pairs of logic cells for bus width 8 and 32 bit, respectively.

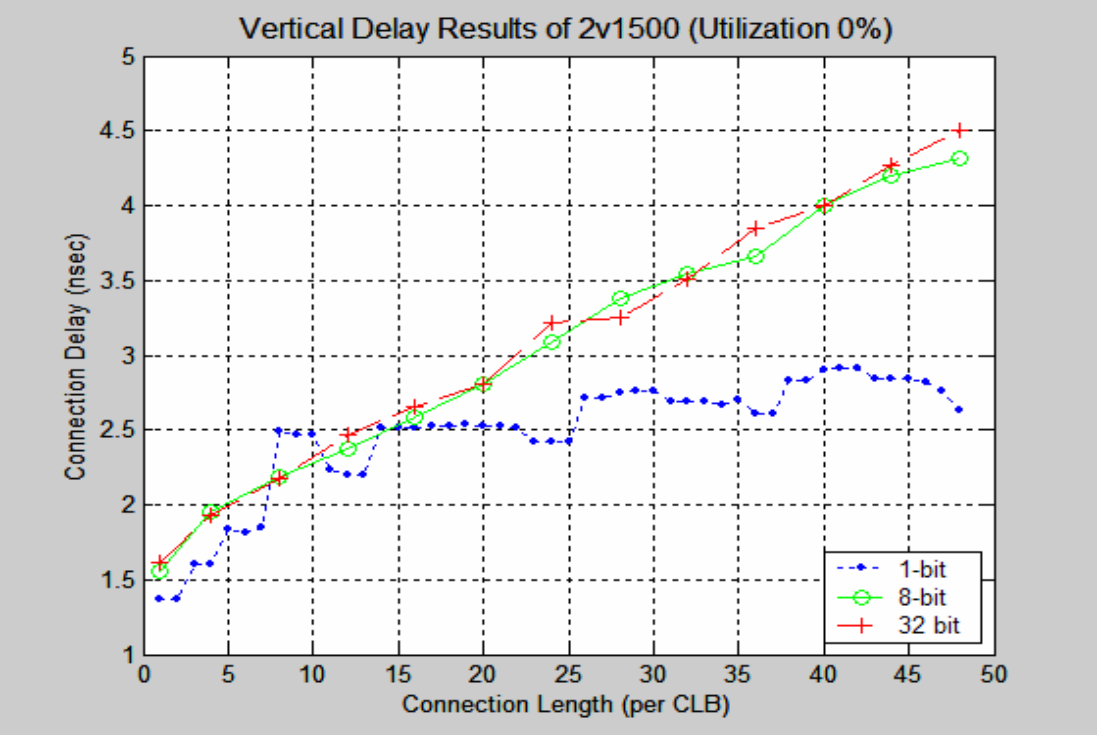
So for each of the three devices, we produced many maps following the idea mentioned above. In the next step, timing analysis of each map had to be performed in order to get the delay results of the connection we wanted. This step was accomplished using *Timing Analyzer* of Xilinx CAD tools. Figure 24 shows the main window of Timing Analyzer having performed a timing analysis.

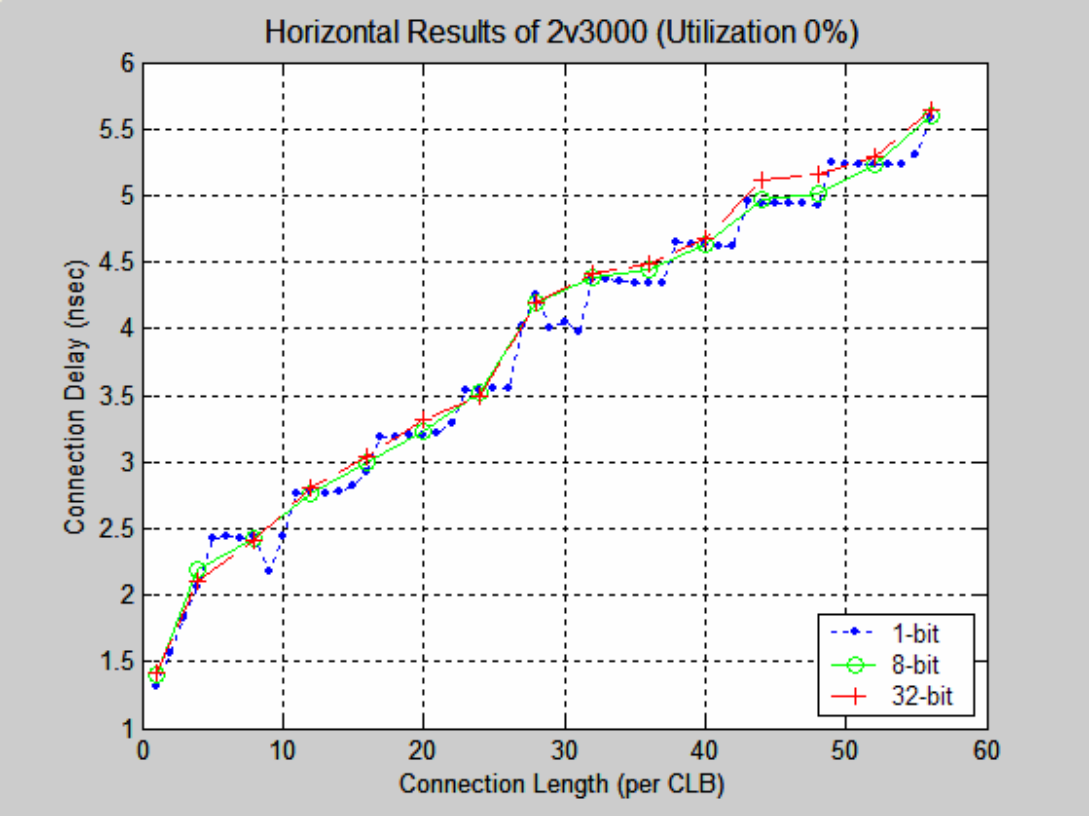
Lastly, after performing the analysis we gathered the experimental results. These results are presented in Figures 25 - 30 below. There are two figures for each of the devices, one for each dimension. The results analysis and the charts are made in *Matlab R12*.

In this point, we are obligated to examine as an example a chart in detail, in order to become more understandable for anyone. The heading of each chart gives three data. The exact device which corresponds to each chart. The logic utilization value of the device for this chart. And the dimension of the device (horizontal or vertical) we refer. As far as axes are concerned: X-axis shows the connection length per Configurable Logic Block (as explained above) and Y-axis shows the equivalent connection delay in nanoseconds. In the bottom right corner the legend is presented. The legends gives a label for each single graph of the chart concerning the bus width (e.g. 1-bit) and shows the kind of line and spots for each graph, respectively. Finally the graphs appear. Each spot represent a result and the spots are connected with a line.

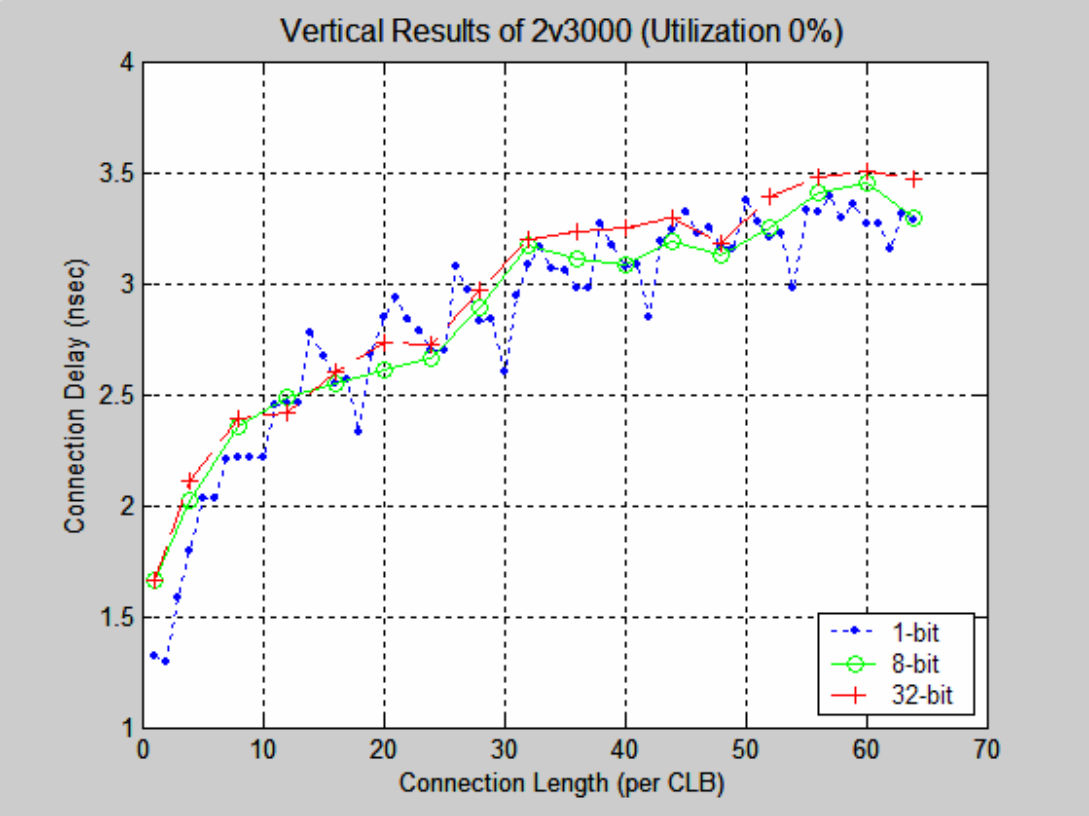


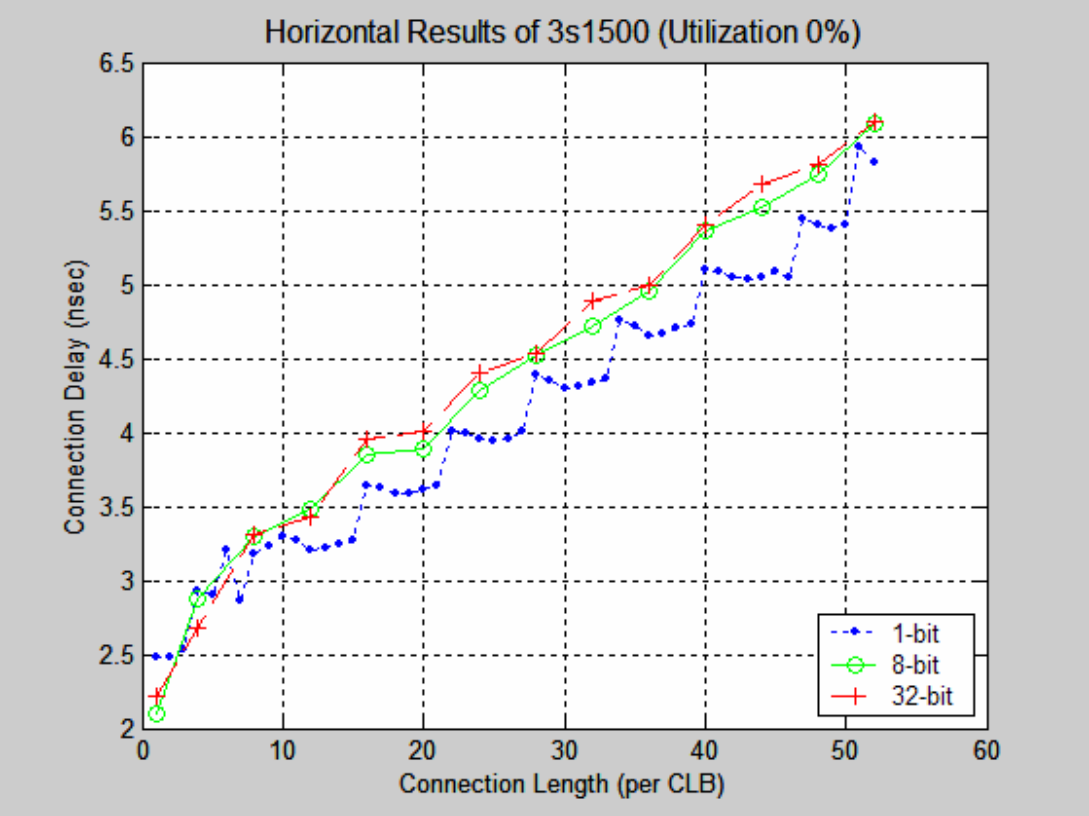
Figures 25 – 26



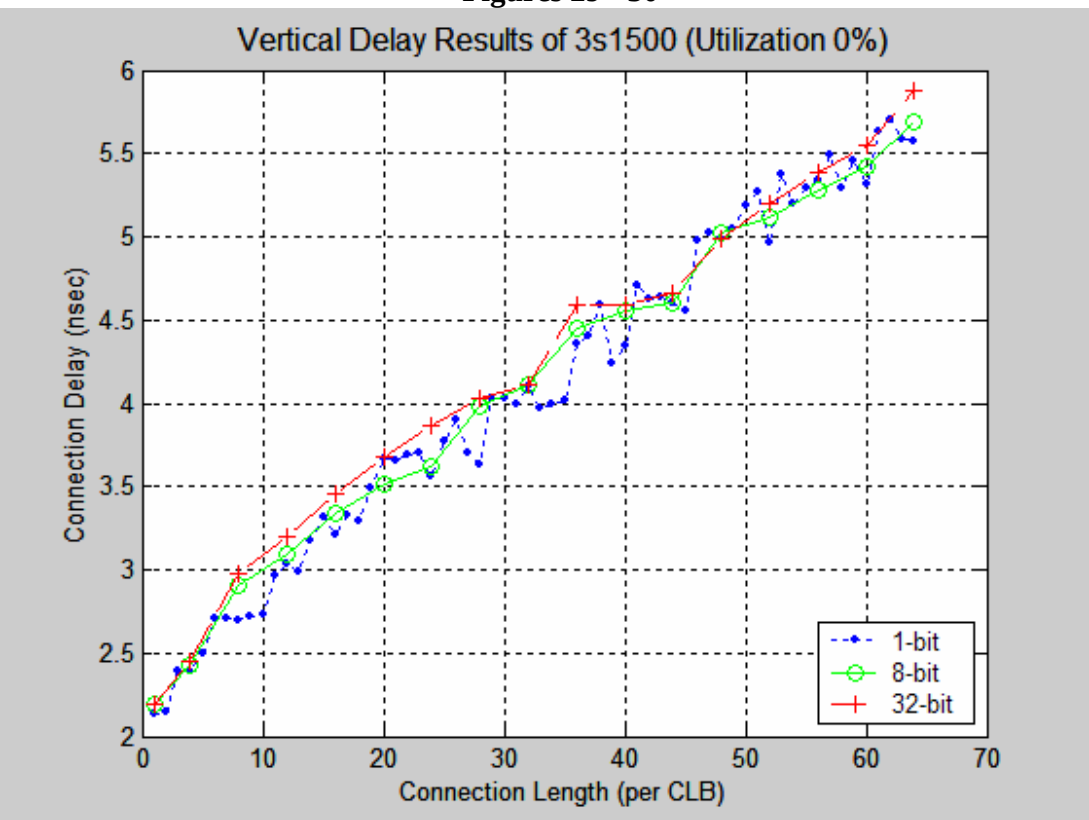


Figures 27 – 28





Figures 29 - 30



We described above in detail the whole experimental procedure of the first phase of our study which led to the results presented in the figures above. Examining closely these figures, we came up with some interesting comments and remarks concerning our field of research that should be stated.

1. Generally, the connection delay increases as the connection length increases. Specifically, we observe that there is an increment of about 200 to 300 % of the connection delay value between the lowest and the highest connection length value. This applies for all the devices, bus widths and dimensions with only one exception. In the horizontal results of device Virtex2-3000 (Figure 27) the increment approaches 400 %.

2. Comparing the graphs of same device and dimension, but with the bus width varying, we observe that they are generally similar. 32-bit and 8-bit designs have slightly larger values, as expected, from the 1-bit design. There is an exception in Figure 26 (vertical results of device Virtex2-1500) where we see that as the connection length increases the delay difference between 1-bit design and the others, is also increasing (reaching 80% approximately).

3. Something interesting comes up if we examine the 1-bit design graphs. We notice that for 4 or 5 consecutive connection lengths they have the same delay and after there is sharp increment. This is well understood in Figures 27 and 29.

Description of phase 2

The object of the second phase was to widen the experimental field in FPGAs that have a portion of their resources, both logic and routing, already utilized. In this way, experimenting on various values of utilization, we acquire an overall point of view. For this purpose we had to use a large design which could be easily modified. A design with versatile code. The design we chose is a pattern matching [23] one, that fulfilled the characteristics mentioned above. The design was modified many times according to our needs. Each time we synthesized it and made the logic optimization using the Project Navigator. These modifications gave utilization values for each device from 35% to 80% more or less. The exact values are presented in Table 2.

Similar to the first phase, placement and routing had to be done. Using the FPGA Editor we, firstly, auto-placed and routed the pattern matching design. Then, we manually placed and routed our own simple design. The experiments were taken place following the exact same procedure described in phase 1 for each value of utilization.

Device Type	Device Utilization Values
xc2v1500-6bg575	41, 53, 62, 75, 85 (%)
xc2v3000-6bf957	33, 40, 45, 51, 59, 66, 71 (%)
xc3s1500-4fg676	36, 43, 49, 55, 64, 71, 77 (%)

Table 2- **Utilization Values for each Device**

We did not interfere with the placement and routing of the large design at all during the experiments. The fact that a considerable part of the devices was utilized limited the number of maps that could be acquired. Not all connection lengths could produce a map because simply, we could not place the logic element of our design

there. This becomes even more obvious with the placement and routing of our 8-bit and 32-bit design where we need to find more than one empty logic cells.

After having completed the placement and route step we had enough maps to be analyzed based on time. As in the previous phase Timing Analyzer was used to produce the connection delay results. We have to mention here that for the 8 and 32 bit design Timing Analyzer produces 8 and 32 values respectively as results. In these cases we made further analysis and acquired the median value as an overall result of the connection delay for each connection length, and this last result is the one depicted in the charts. Using MatlabR12 we completed the last step which was the creation of the charts.

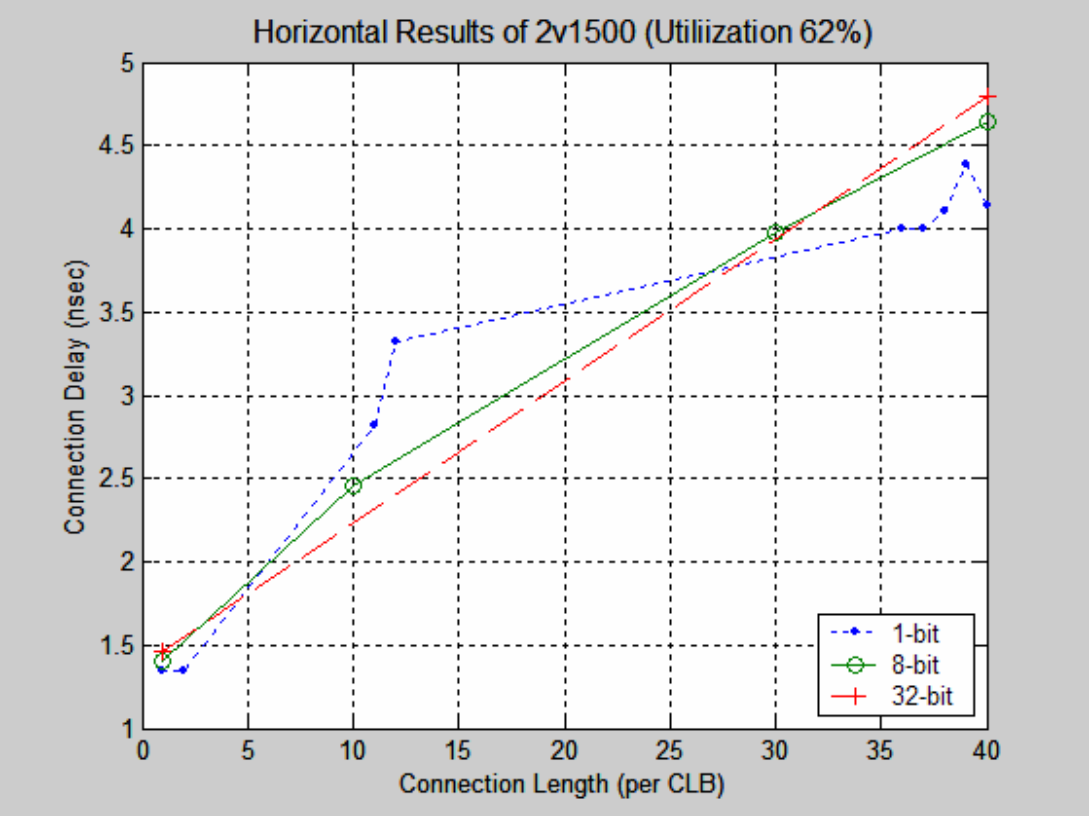
These charts are presented in total in Appendix A (Figures 7-44). There are two charts for each device for each value of utilization, the chart presenting the horizontal dimension results and the one presenting the vertical dimension results. Each chart consists of three graphs (1, 8 and 32 bit). In this section only few examples are presented in Figures 31-36.

Following the same road as in the first phase, we present below some interesting remarks and comments that emerged through our result analysis. In this phase we have to take in mind the additional factor of logic utilization of the FPGA.

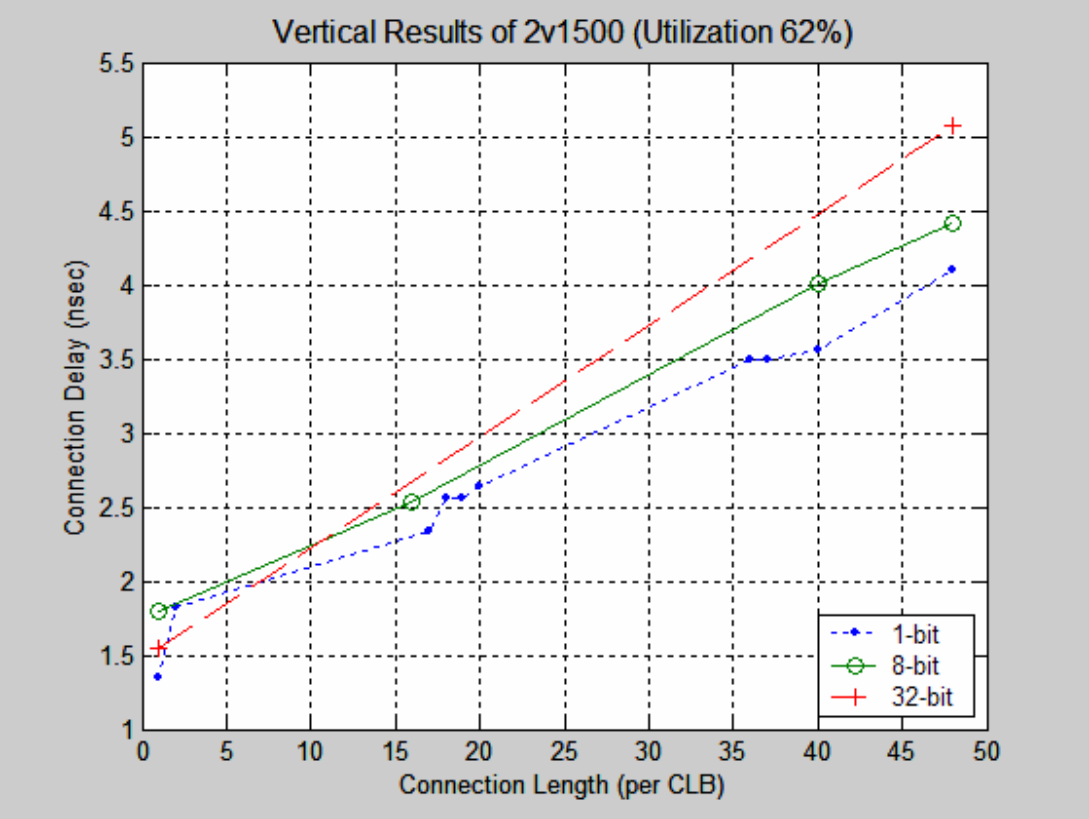
1. Generally, as in phase one, the connection delay increases as the connection length increases. Specifically, we observe that there is an increment of about 250 to 350 % of the connection delay value between the lowest and the highest connection length value. This applies for all the devices, utilization values, bus widths and dimensions with few exceptions.

2. Comparing the graphs of designs with the bus width as varying factor we come up with the remark that they have similarities. But this time (high logic utilization), the delay results for the 32 and 8 bit designs are higher in a rate of 10% from these of the 1-bit design. This happens mostly for large connection lengths. This is the general case of course. In some results this rate is even greater and in some others it doesn't exist.

3. In a few figures we observe some "extraordinary" results. This means results that logically should not appear and were not expected. Typical examples are in Figure 18 and Figure 39 (see Appendix). These results are "damaged" due to noise. We note this fact when it appears by making a comment on the figure.



Figures 31 – 32 : Virtex2-1500



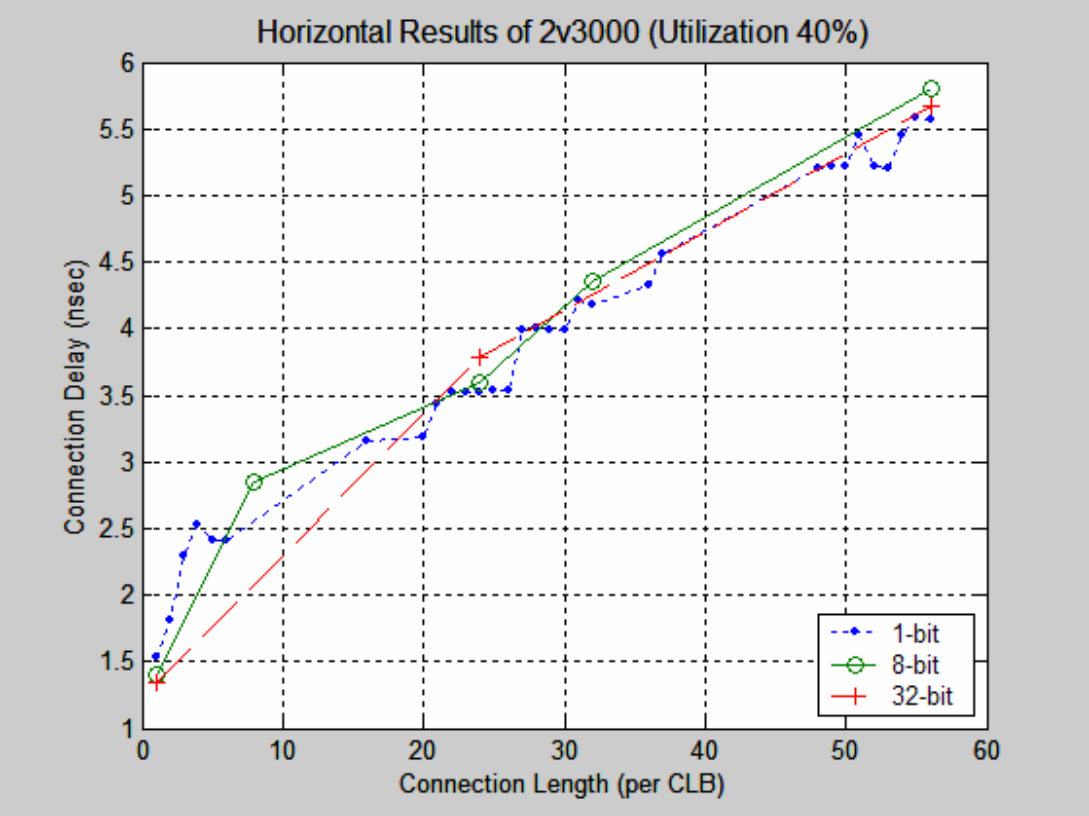
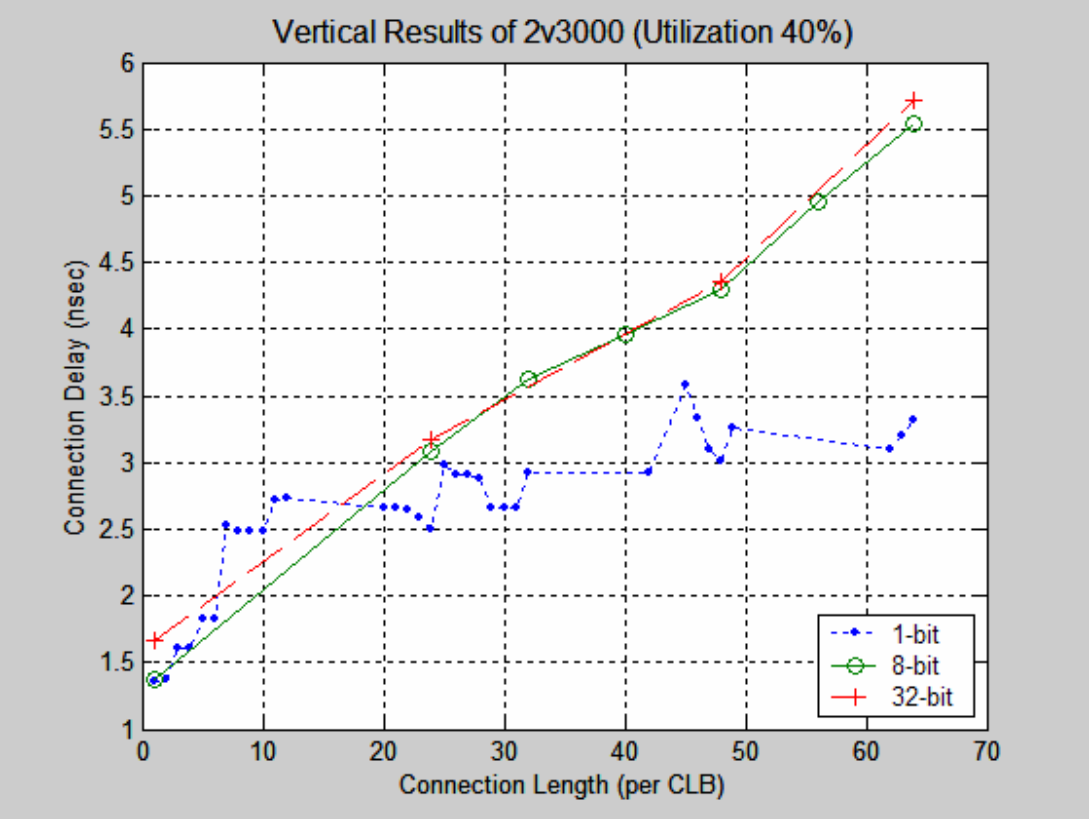
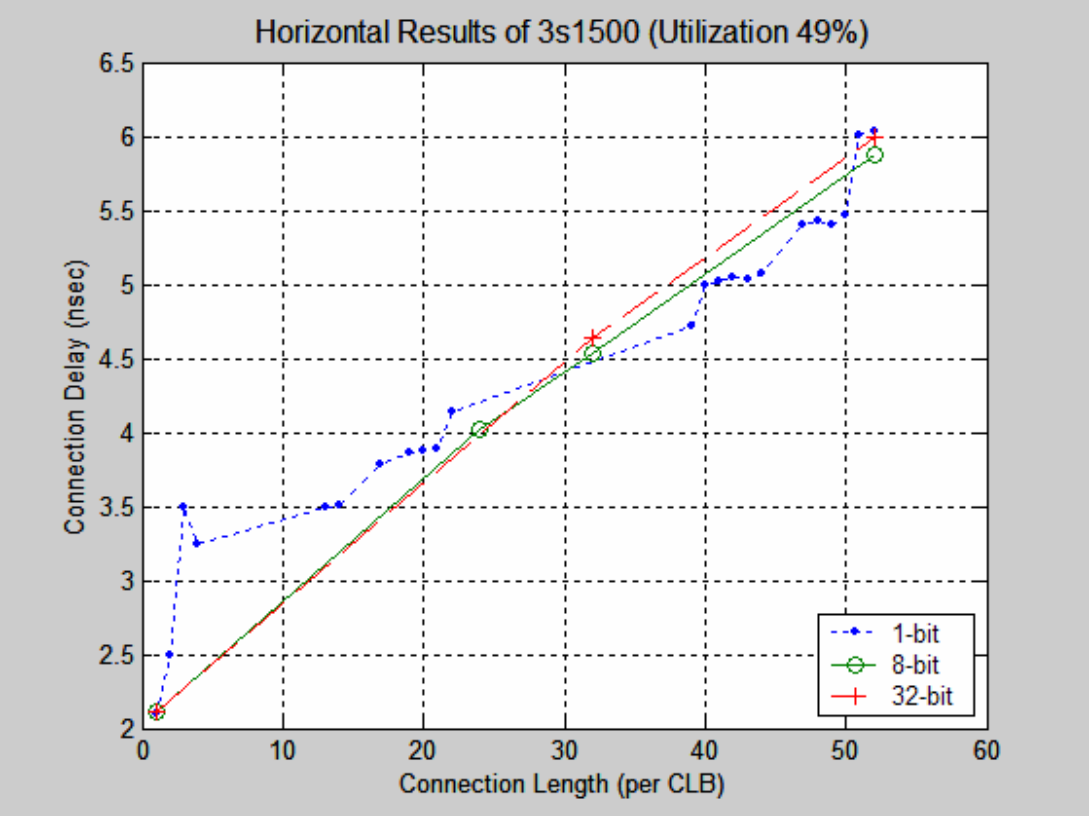
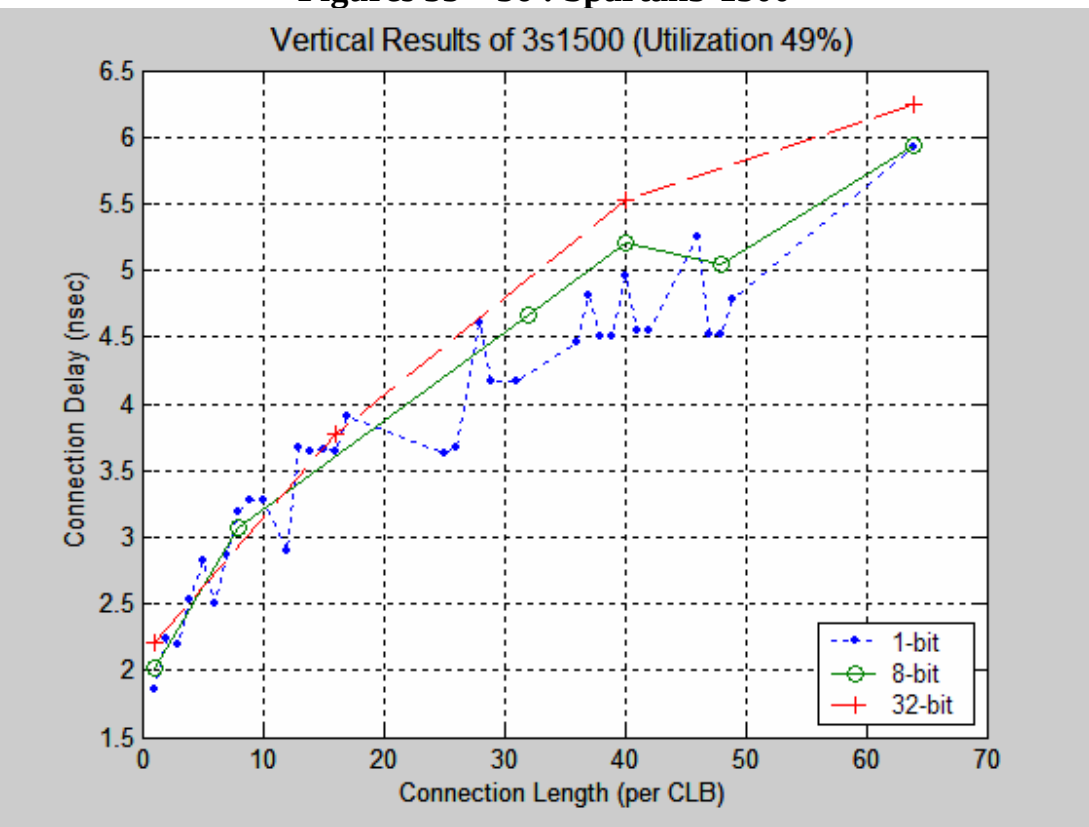


Figure 33 – 34 : Virtex2-3000





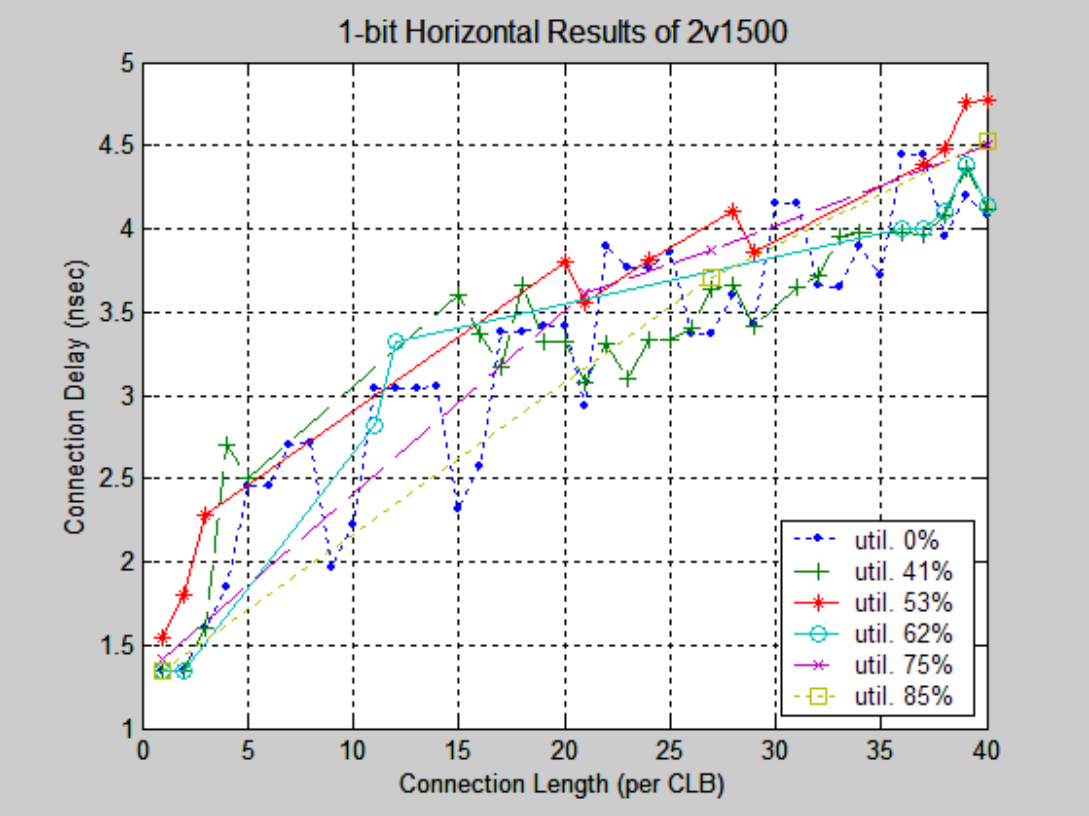
Figures 35 – 36 : Spartan3-1500



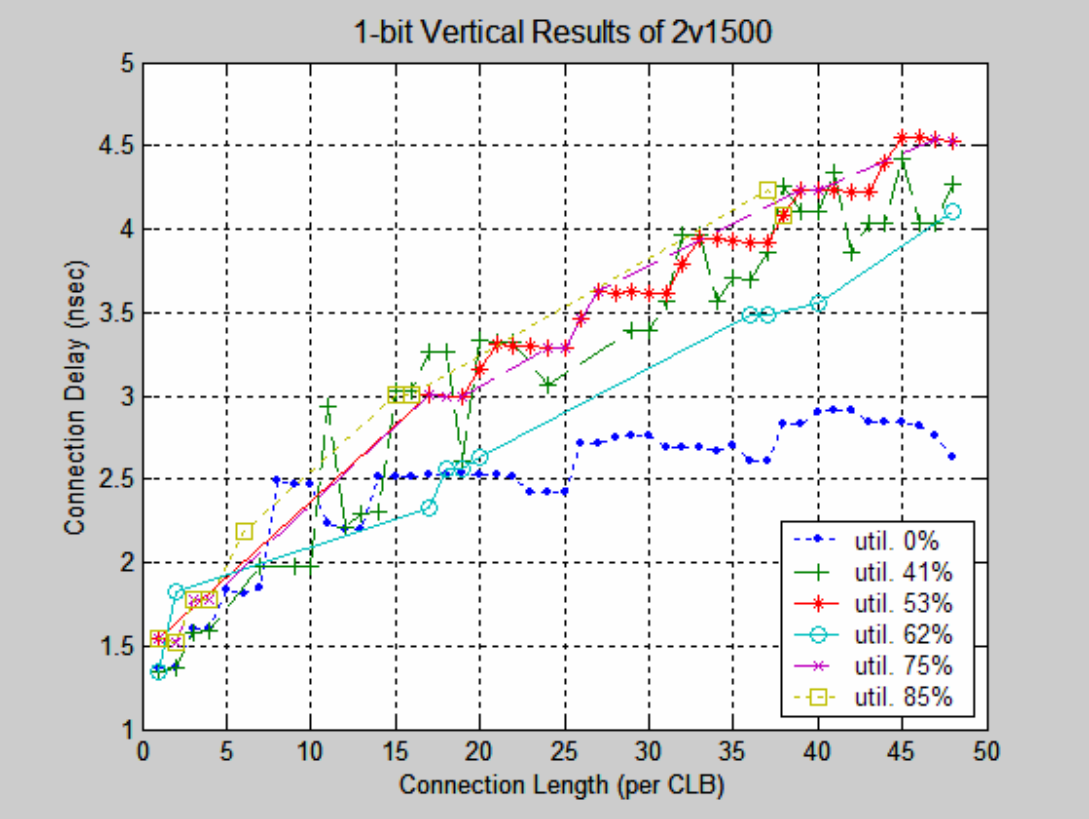
The charts presented and examined so far focus on the comparison between 1, 8 and 32 bit results. But the comparison between different values of utilization should also be studied. This comparison is presented through Figures 37-42 for the 1 bit design. The chart in these figures consists of one graph for each utilization value. Two for each device (horizontal and vertical). In these figures the heading reveals the bus width of the design, while the legend label shows the logic utilization value of each graph. Below, we present a few remarks based on these figures.

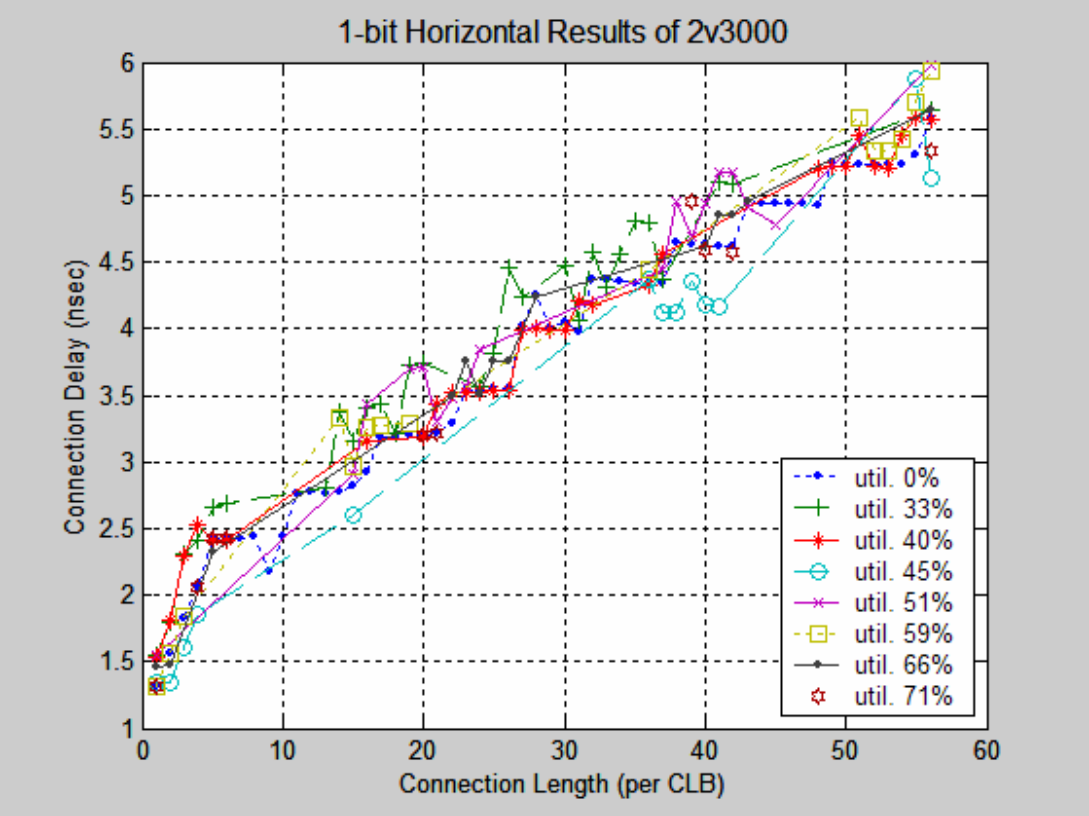
We expected that as logic utilization increased the connection delay would show a plain tendency to increase too. But this does not happen. We cannot notice a rule or a clear overall conclusion. For example, in Figure 37 we see that the greater delay values occur for 53% utilization. None the less, we can say that the graphs corresponding to 0% utilization tend to have one of the lower delays. This is really clear in Figures 38, 40 and 41.

Finally, we can mention that in general all graphs are close to each other. We mean that the range of the results is small, especially for small connection length values. Of course there is an increment of this range for bigger connection length values (e.g. Figure 40).

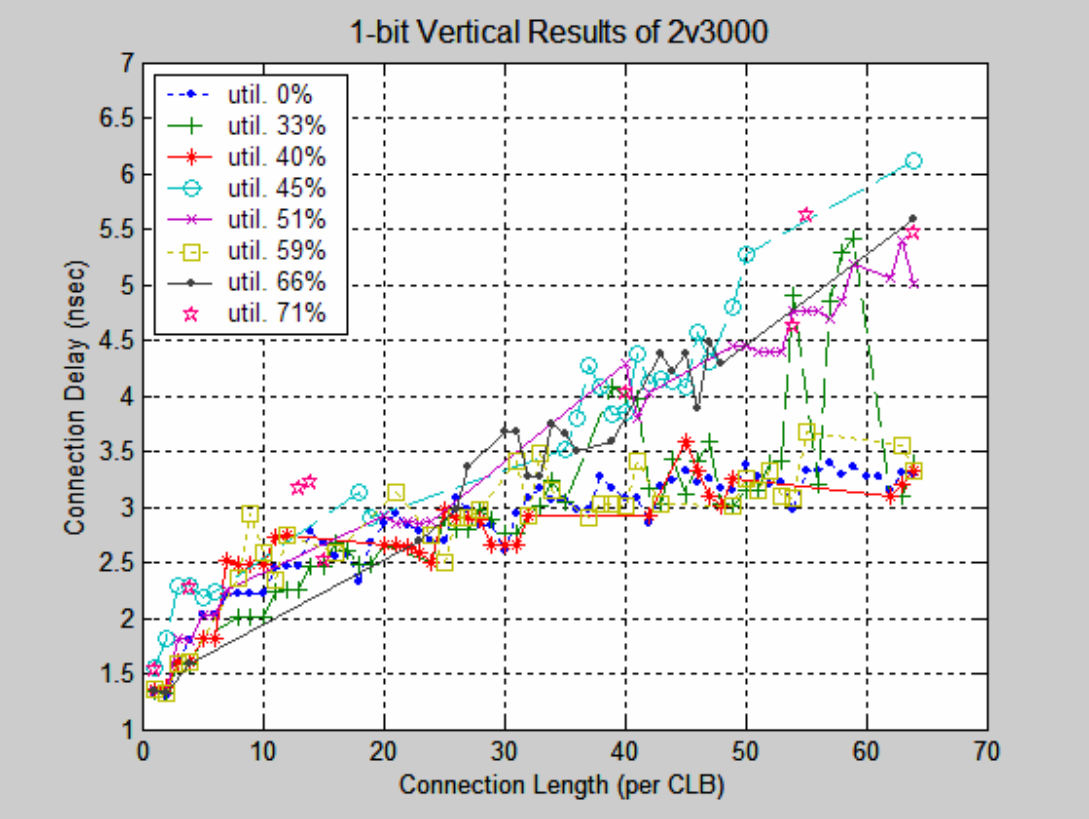


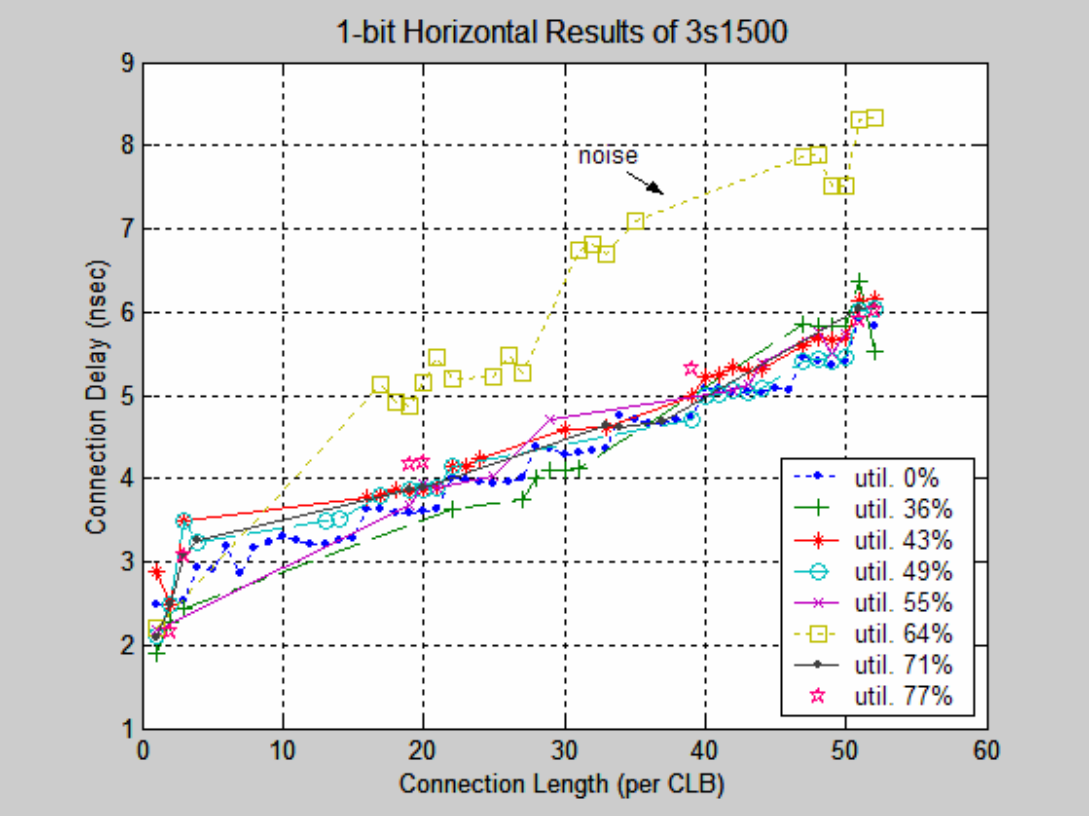
Figures 37 – 38 : Virtex2-1500



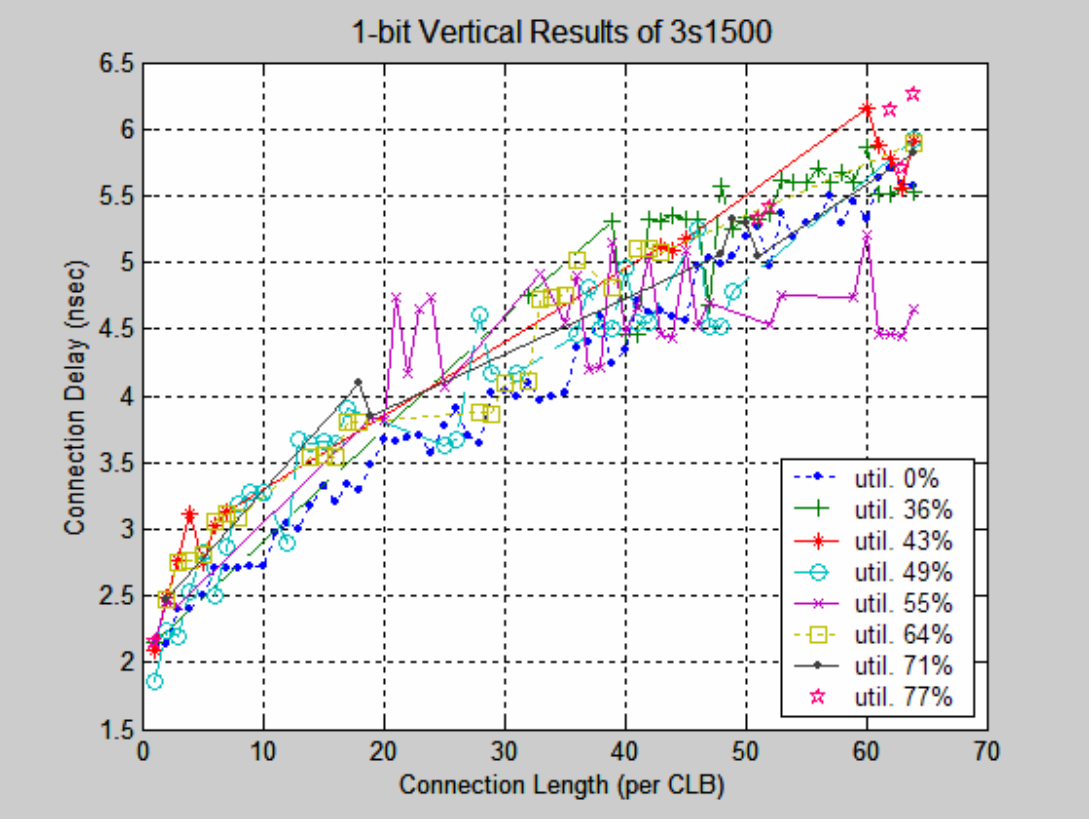


Figures 39 – 40 : Virtex2-3000





Figures 41 - 42 : Spartan3-1500



4 Conclusions

In the previous chapter we gave the objects of this study and presented a detailed overview of the experimental procedure followed and of the experimental results acquired. In this last chapter, we cite the conclusions we have worked out during this study. We, also, refer to the contributions of this study and make suggestions for future work.

4.1 Summary - Contributions - Suggestions for Future Work

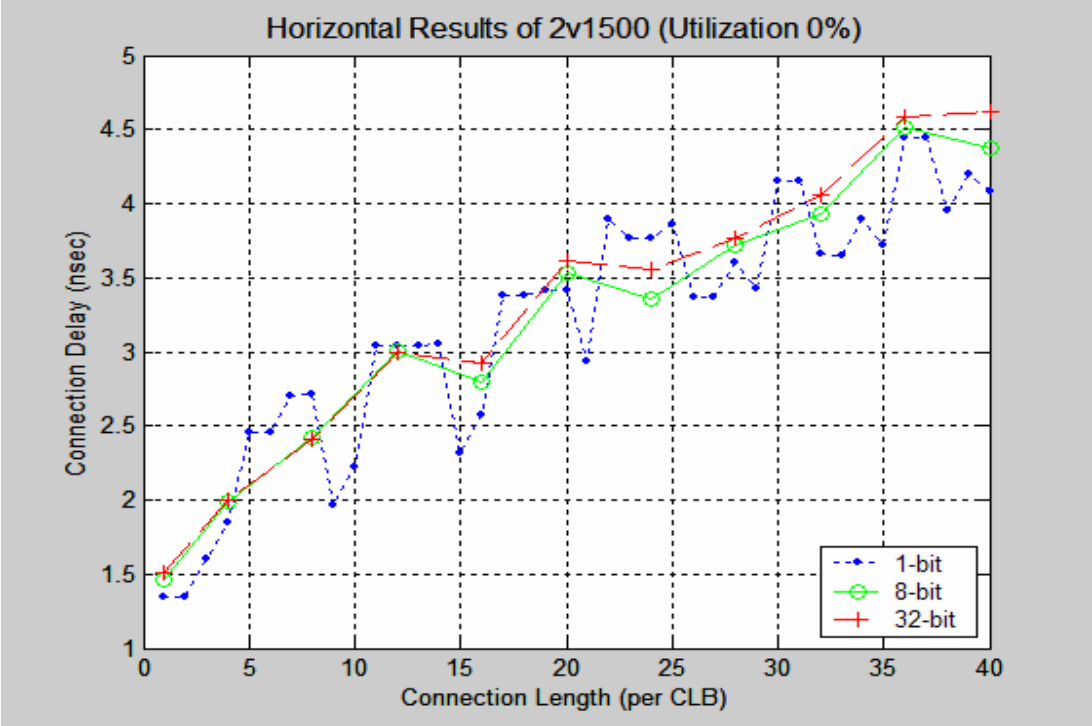
Throughout this study we have emphasized on the importance of routing architecture, routing algorithm and routing resources for FPGAs. So after having presented the necessary background for this field, we conducted a detailed experimental procedure, focusing on commercially available FPGAs, in order to reach useful overall conclusions. These conclusions are stated here.

As an overall conclusion we must mention that FPGA routing resources manage to meet up to our expectations. This happens both in terms of quantity and of quality. The fact that the delay did not increase radically in designs that utilized a percentage of 70 – 80 % of the FPGA logic resources compared to designs that utilized only 1%, shows that a good tradeoff between the quantity of routing resources and logic resources has been achieved. Moreover, the reasonable increment in the delay we had as a result in empty FPGA compared to the connection length shows the ability of routing resources to handle various lengths satisfying. This is due mainly to routing architecture which uses different types of wires according to each case. These conclusions are reinforced from the fact that even the results for designs with 32-bit bus width (and high utilization) did not present great increment compared to designs with 1-bit bus width.

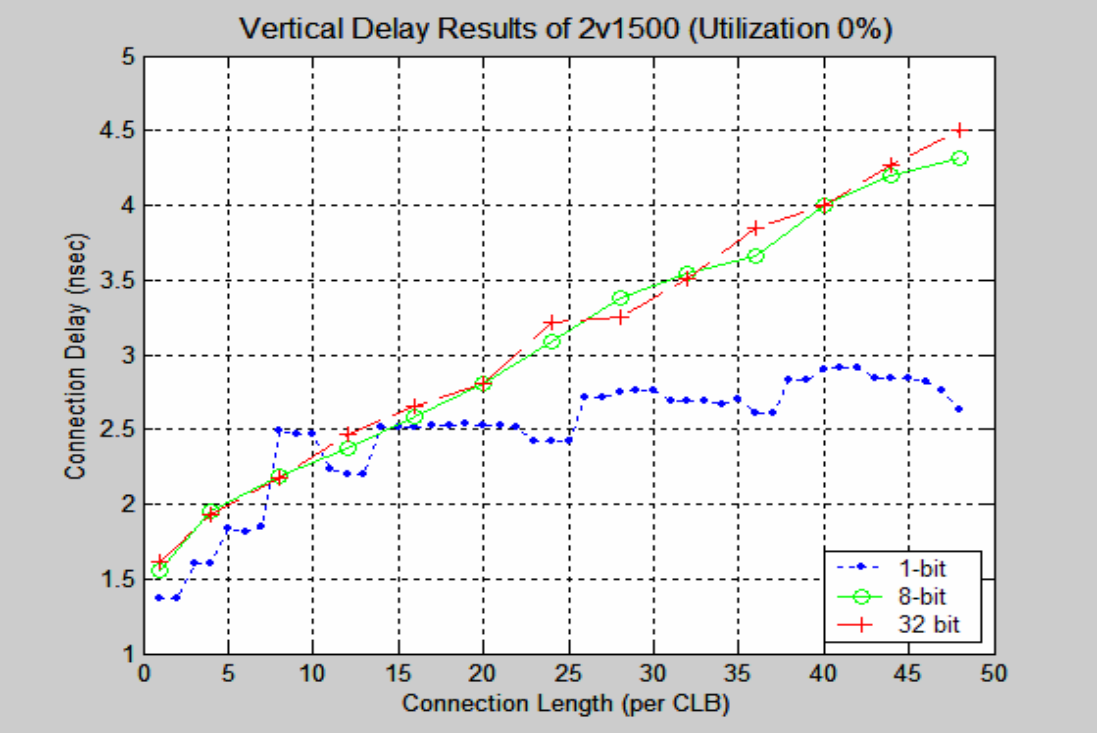
But there is a point we can support that the results spotted a weakness. This point is the routing algorithm. This conclusion begins from the fact that some results were very satisfying compared to some others in the same graph. This difference of results cannot be attributed to the connection length, which means that the routing algorithm reaches a result but not the best in some cases. Generally, there is an uncertainty as far as the routing algorithm is concerned.

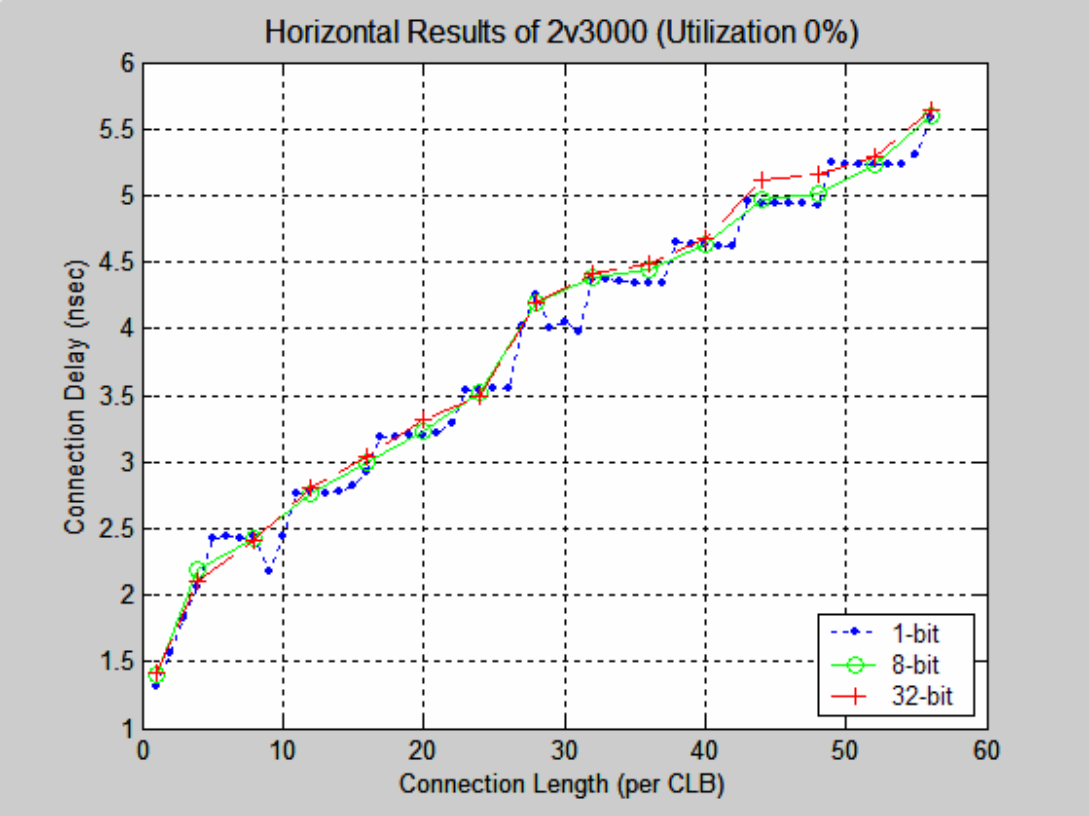
To summarize, today FPGAs are a leading choice of technology for the implementation of many digital circuits and systems. A key field of research for the FPGAs is routing resources. This study, following an experimental approach and focusing mainly on commercially available FPGAs, shed some light in this field and stated some interesting remarks and useful conclusions. There is still a lot of study needed, concerning routing, in order to accomplish an even more satisfying level and manage to keep up to the fast development of technology. There is a great variety of designs of various characteristics that could be used in a similar experimental approach. This would expand the research and possibly reveal interesting conclusions. Finally, the subject of routing algorithm should be re-examined (but that is not new data).

APPENDIX: ALL GRAPHS PRODUCED

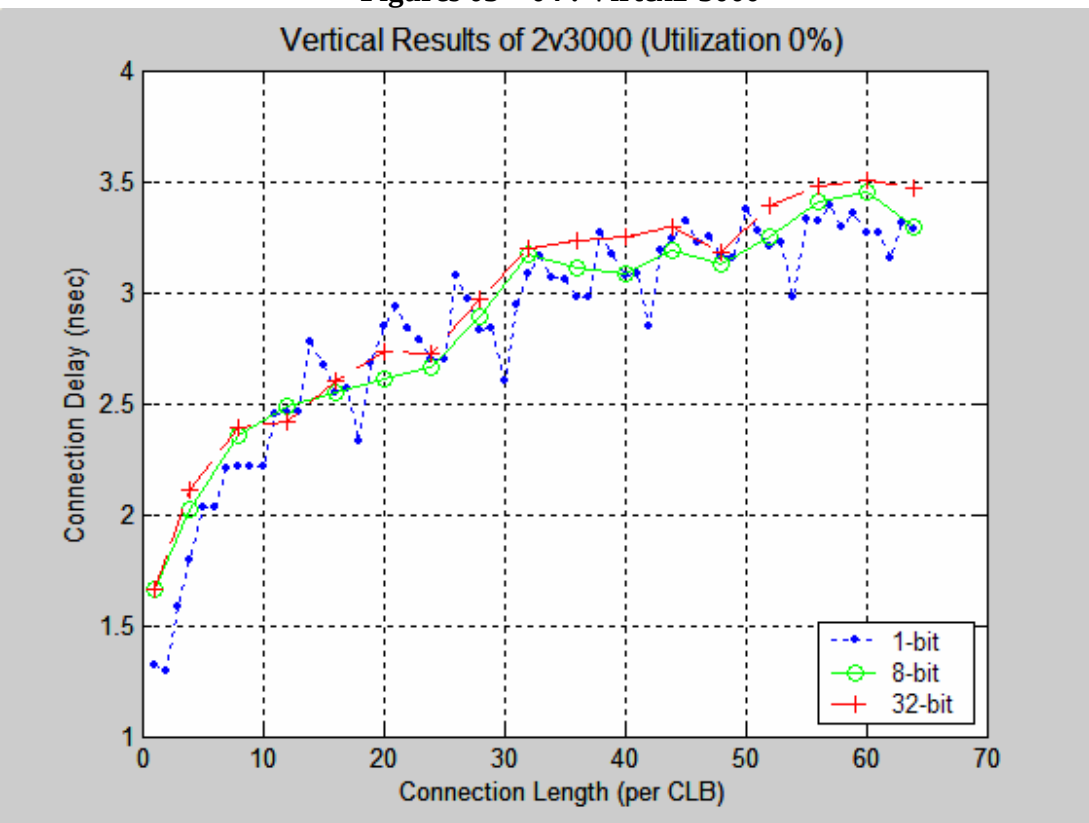


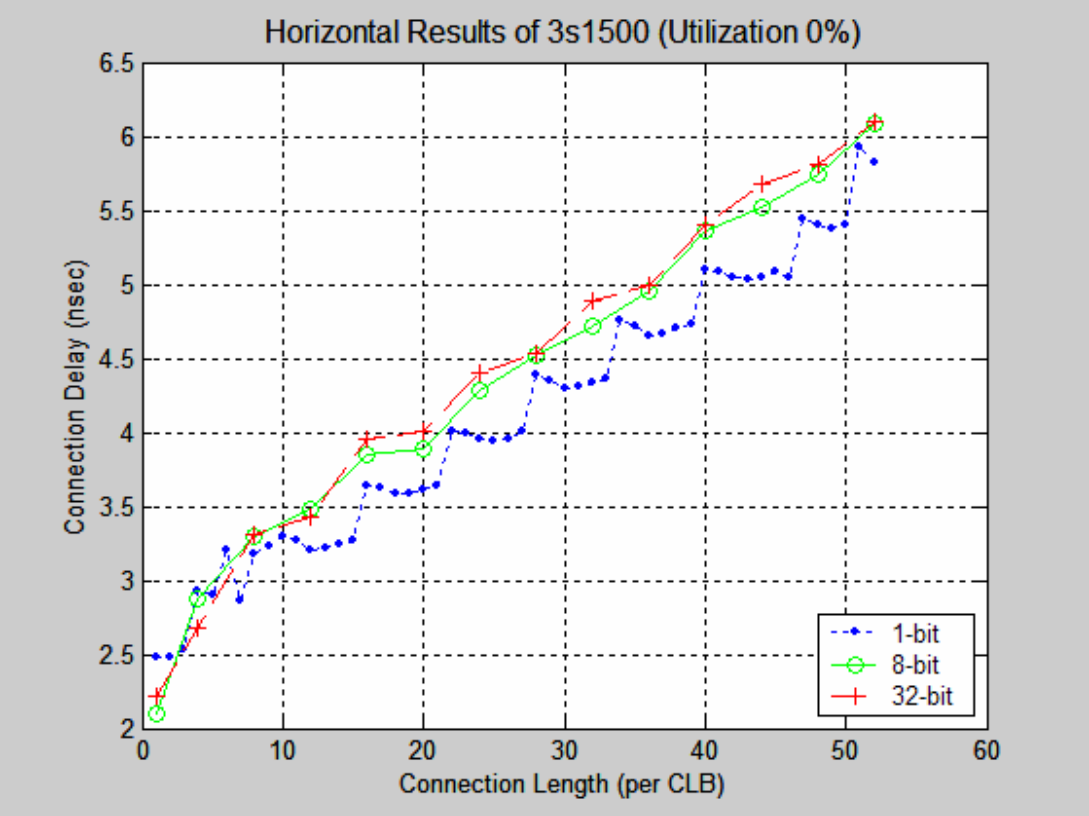
Figures 01 – 02 : Virtex2-1500



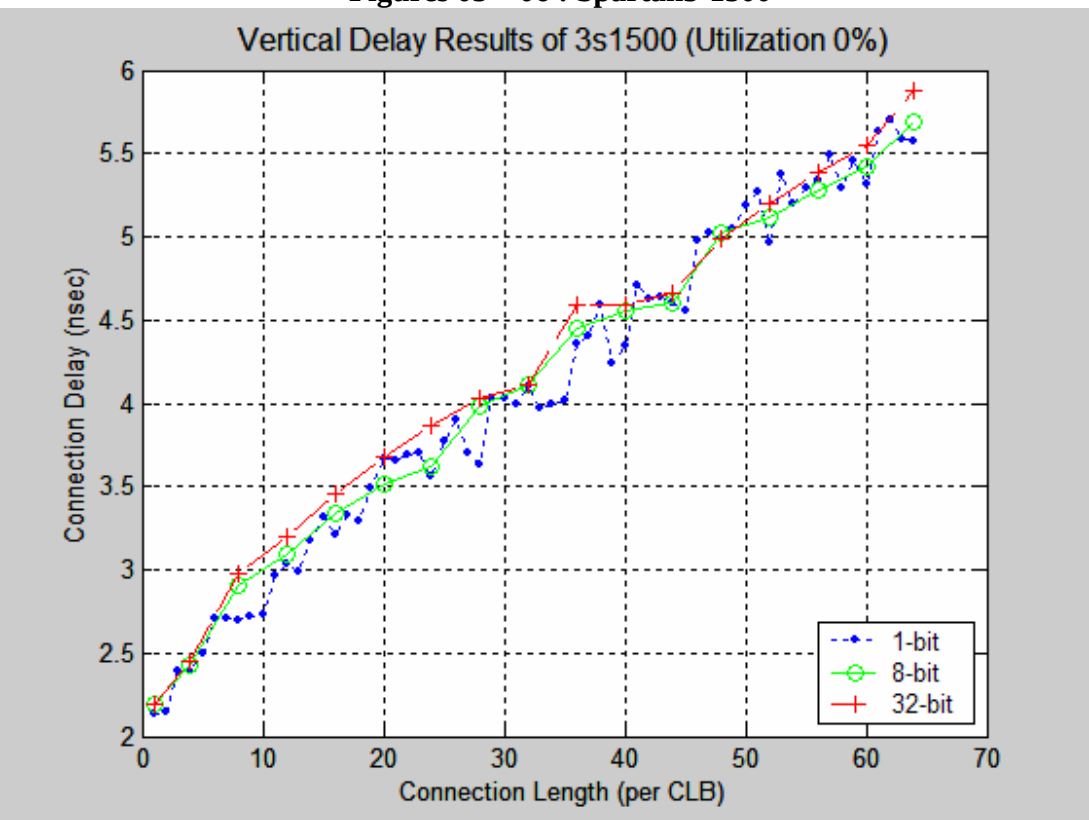


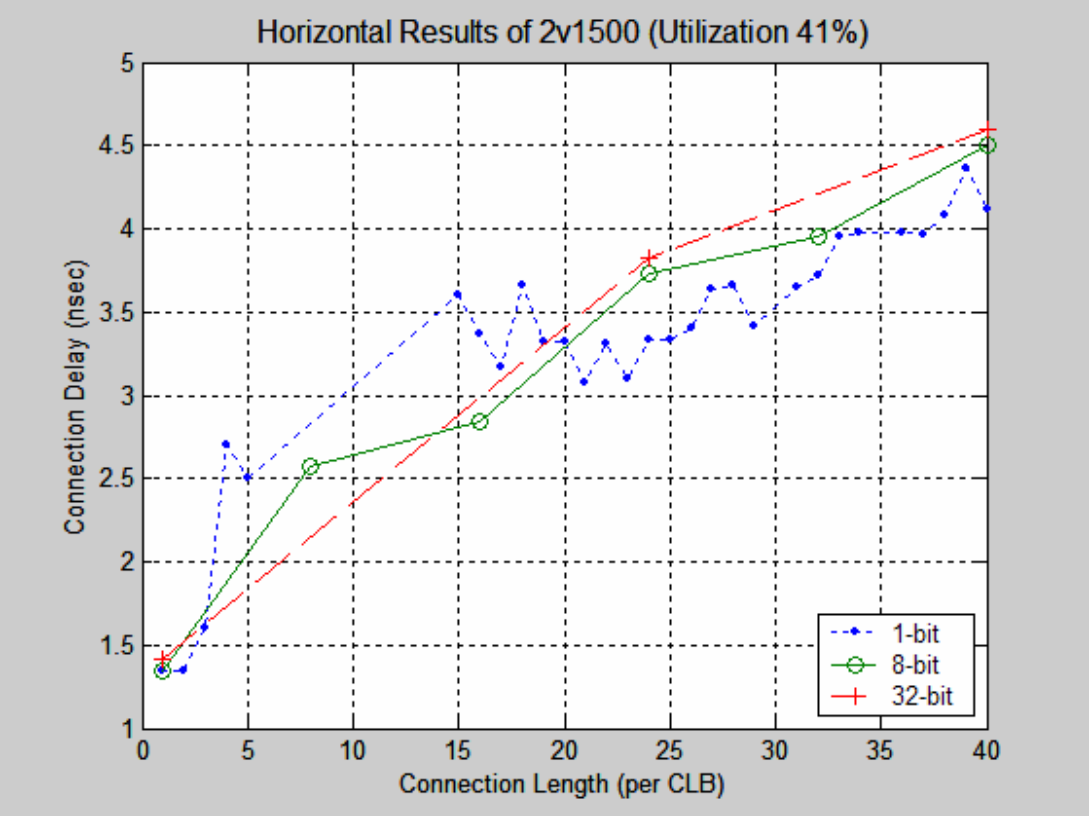
Figures 03 – 04 : Virtex2-3000



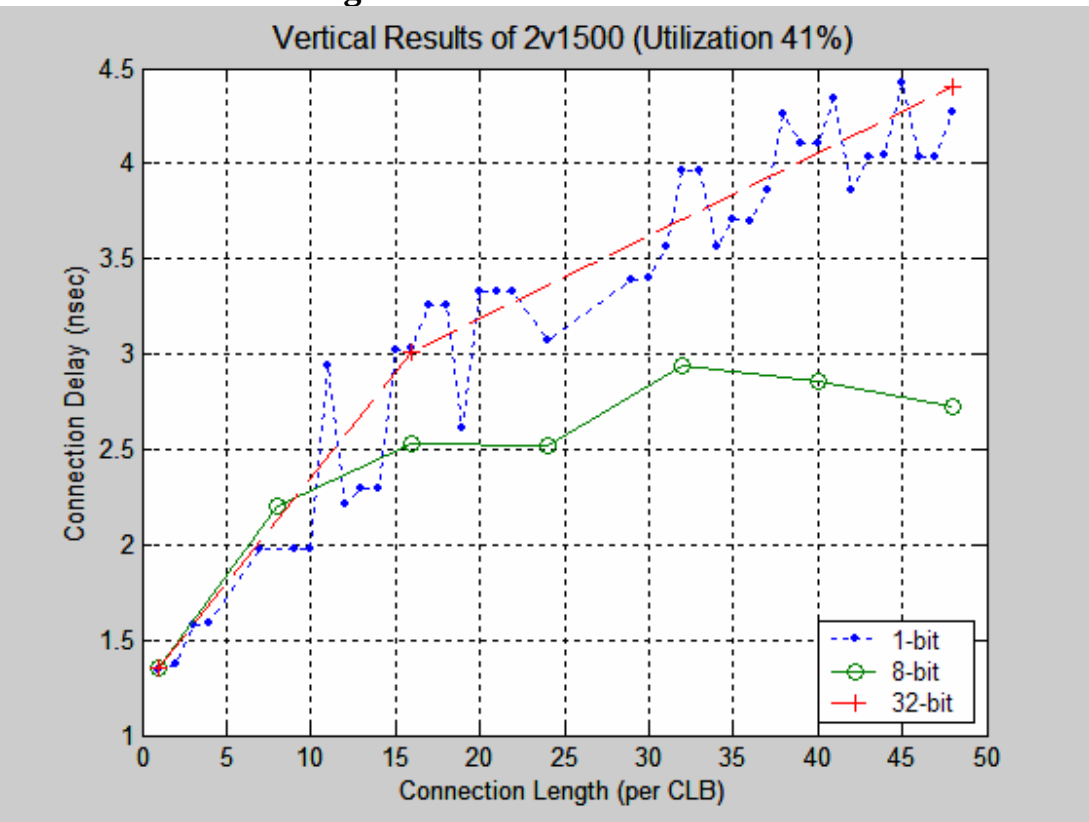


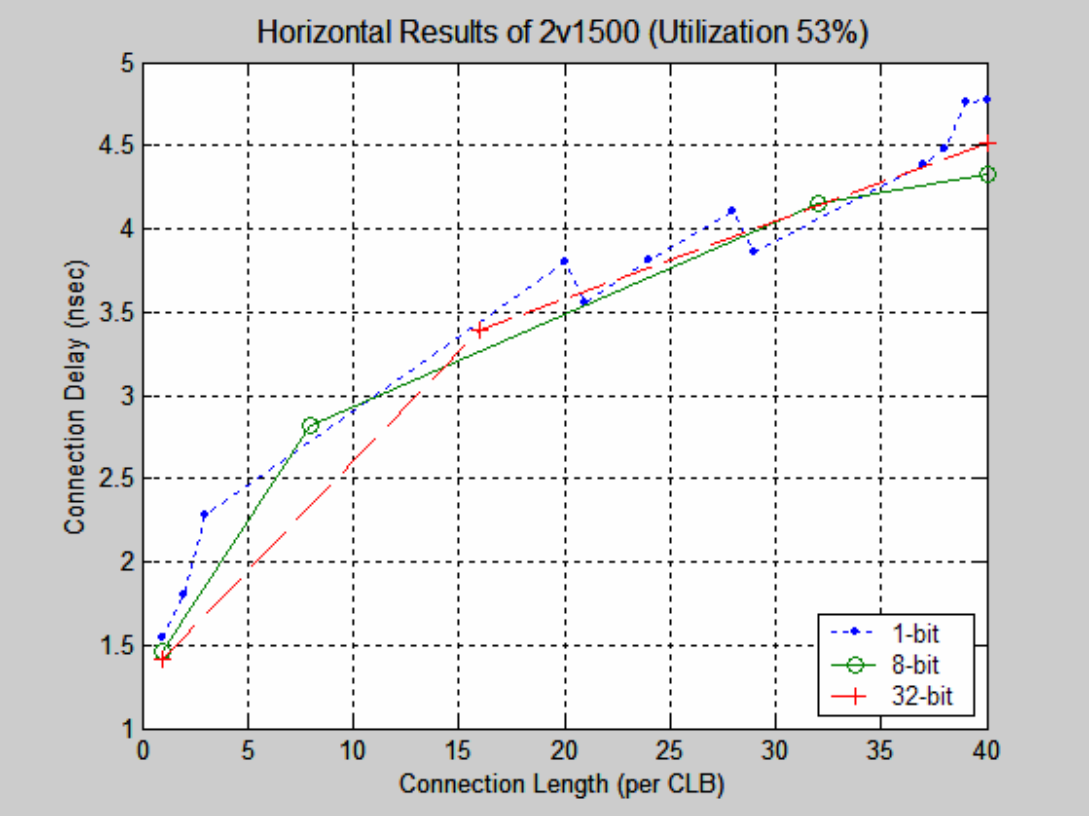
Figures 05 – 06 : Spartan3-1500



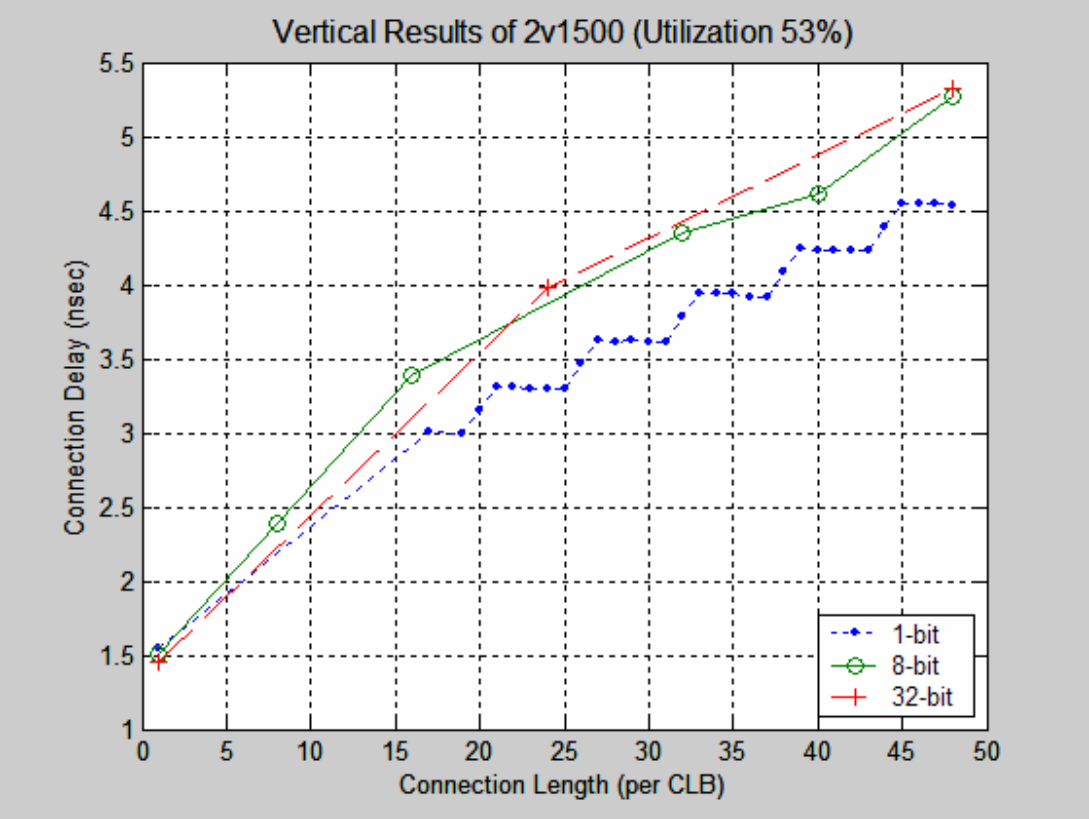


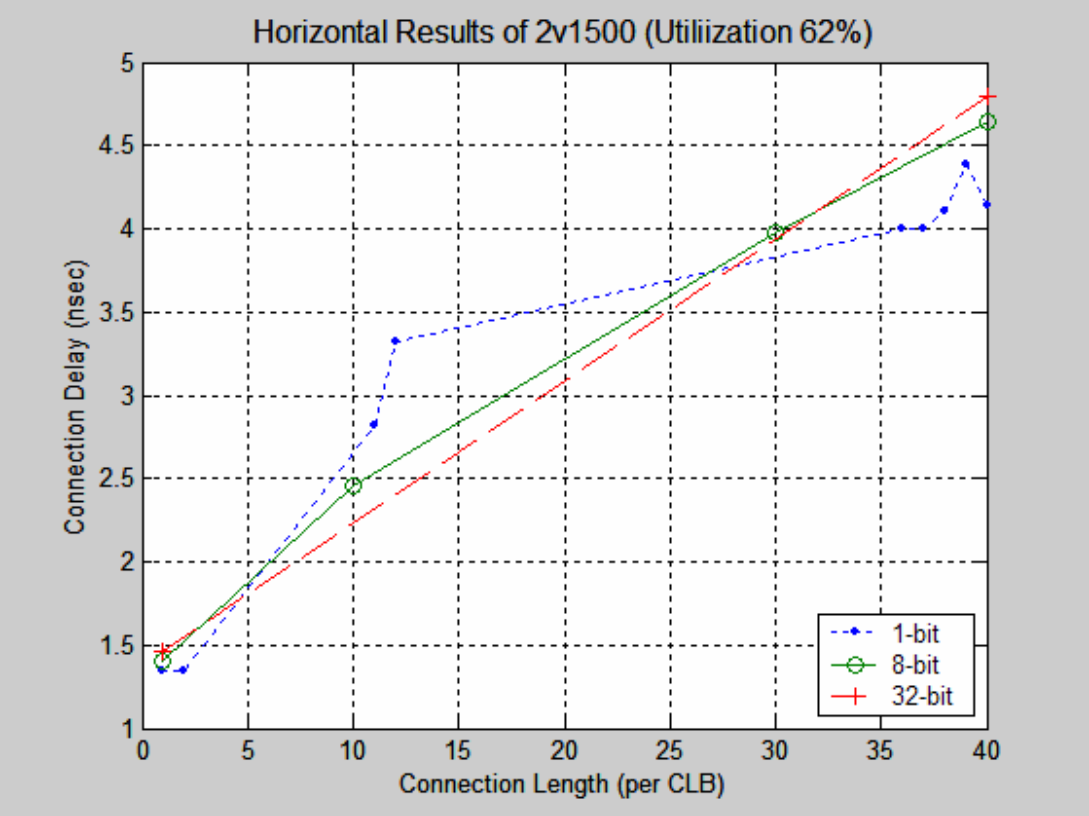
Figures 07 – 08 : Virtex2-1500



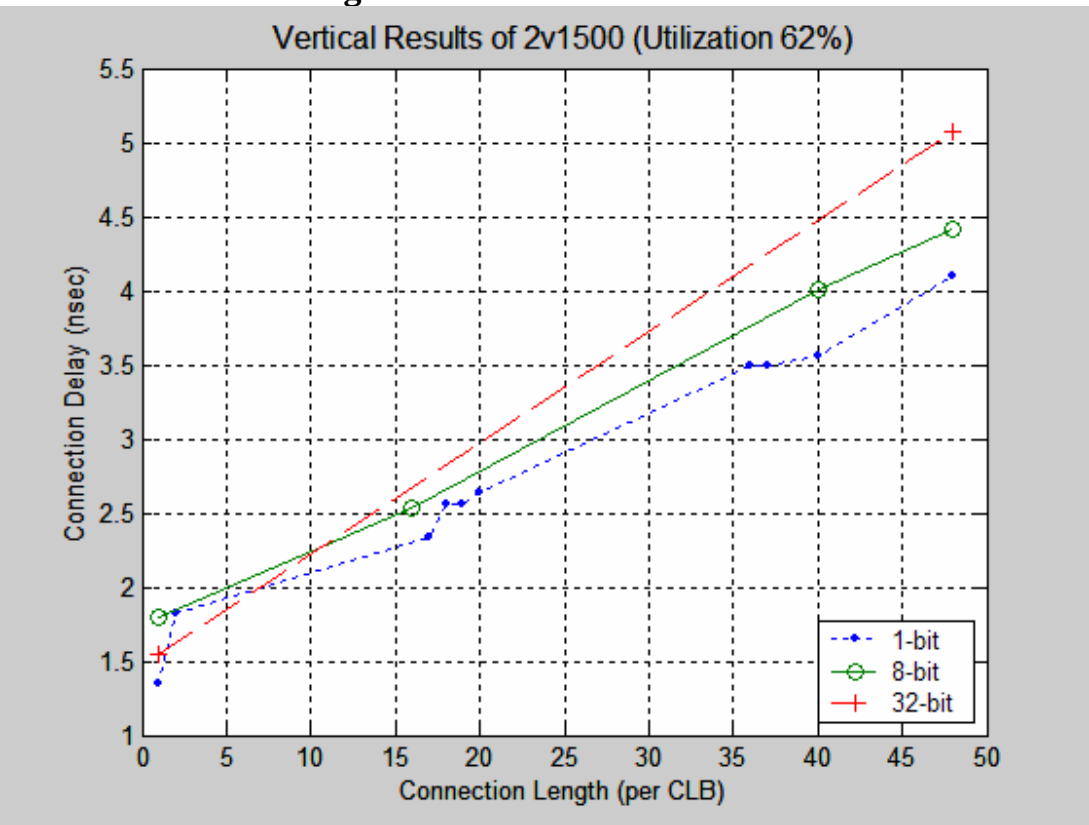


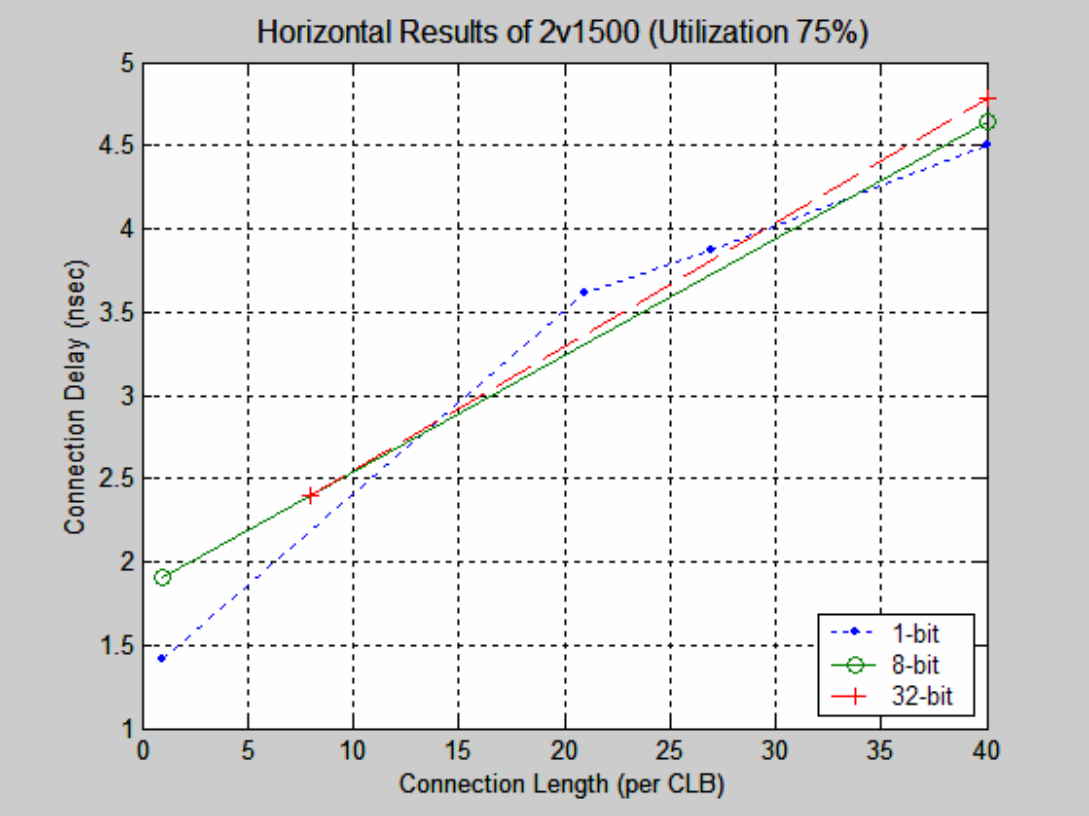
Figures 09 – 10 : Virtex2-1500



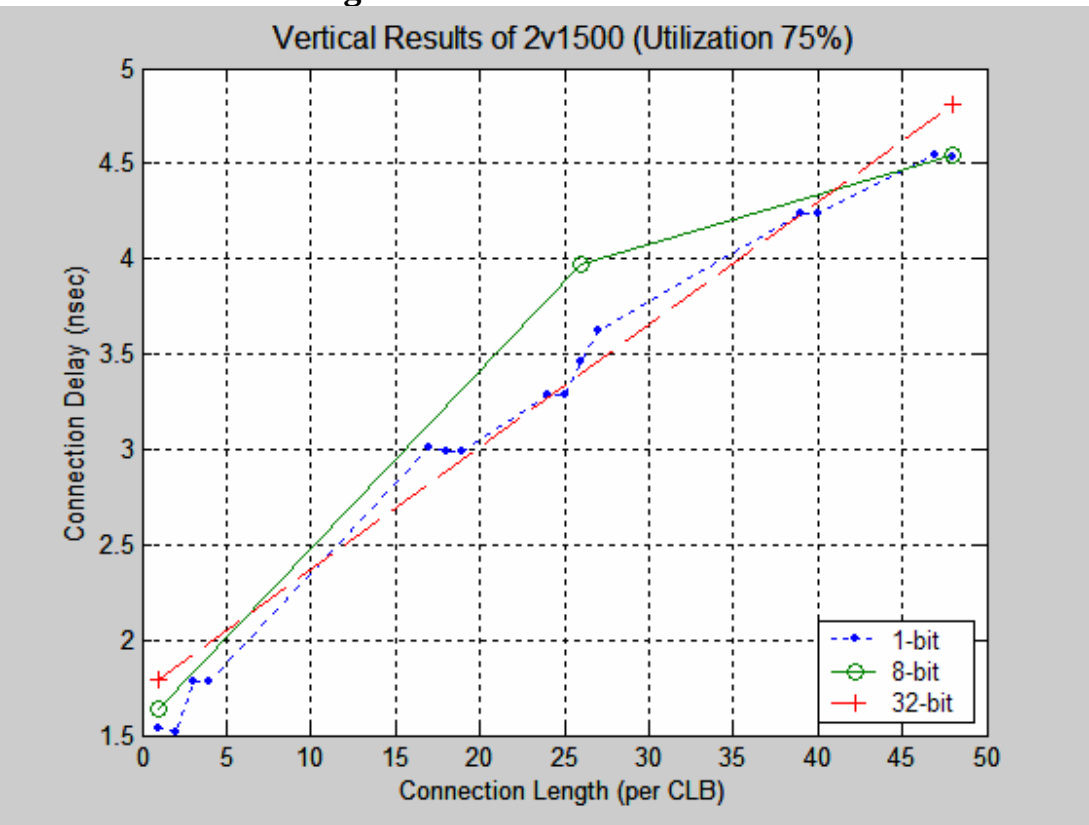


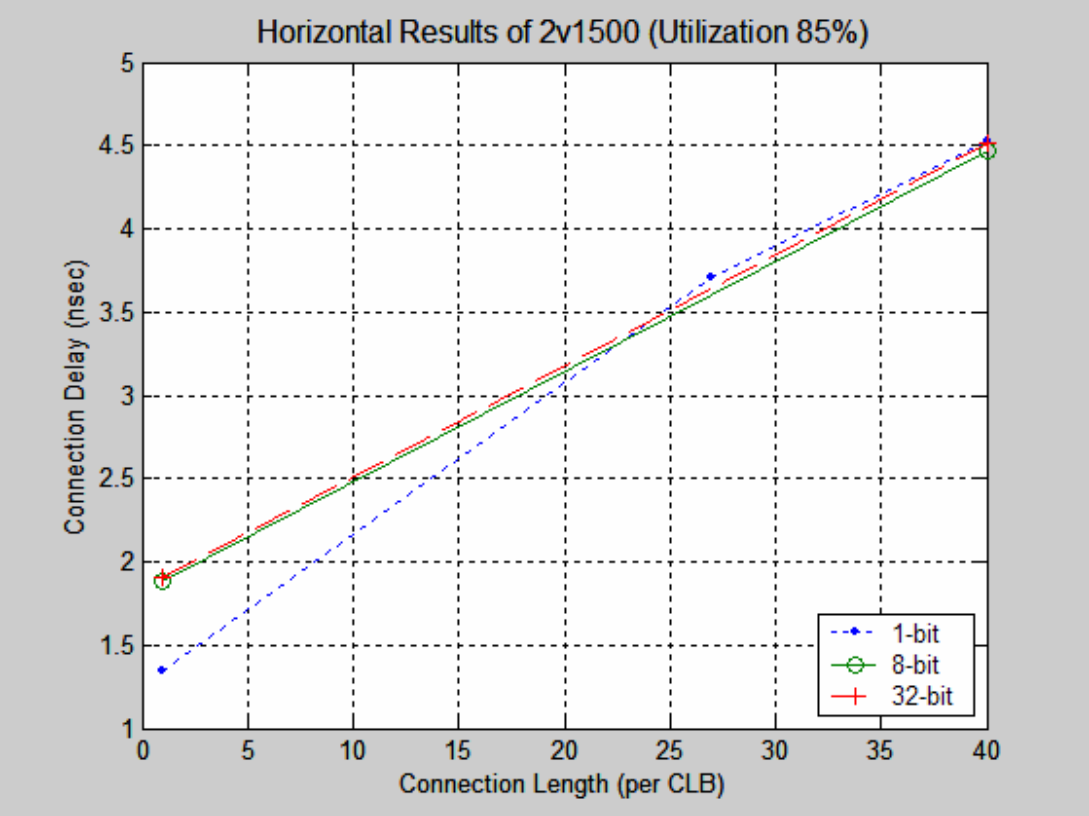
Figures 11 – 12 : Virtex2-1500



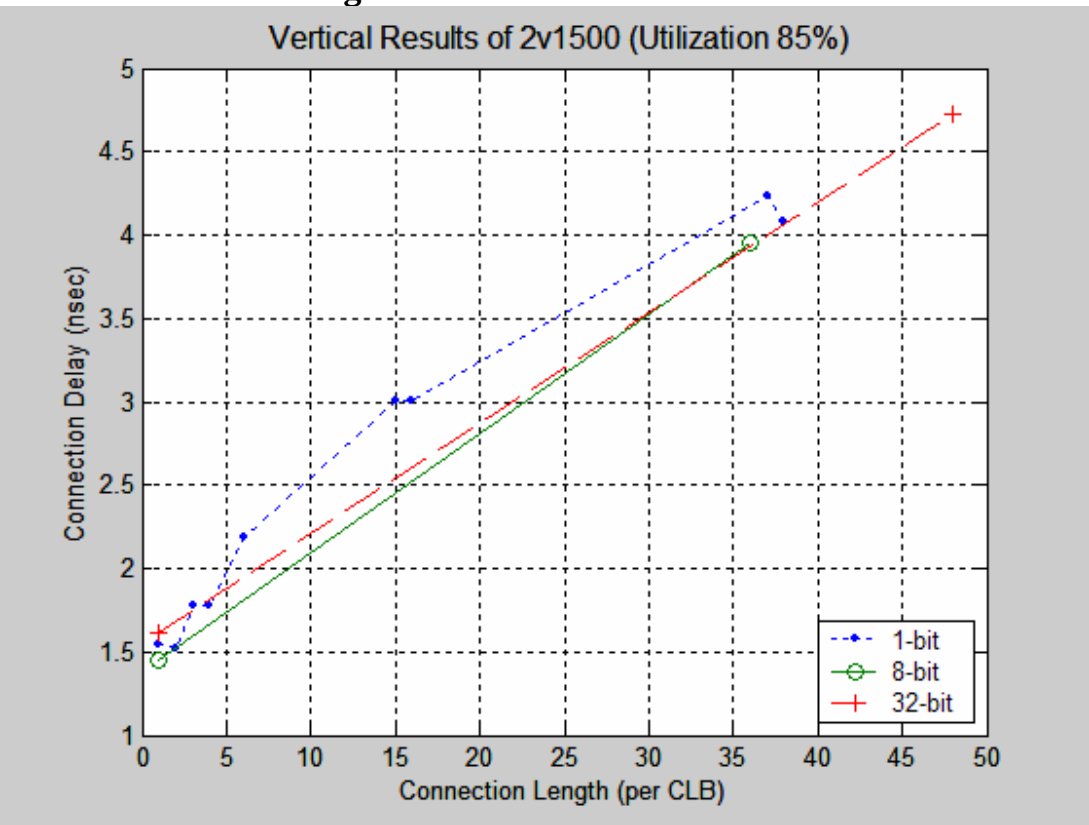


Figures 13 – 14 : Virtex2-1500





Figures 15 – 16 : Virtex2-1500



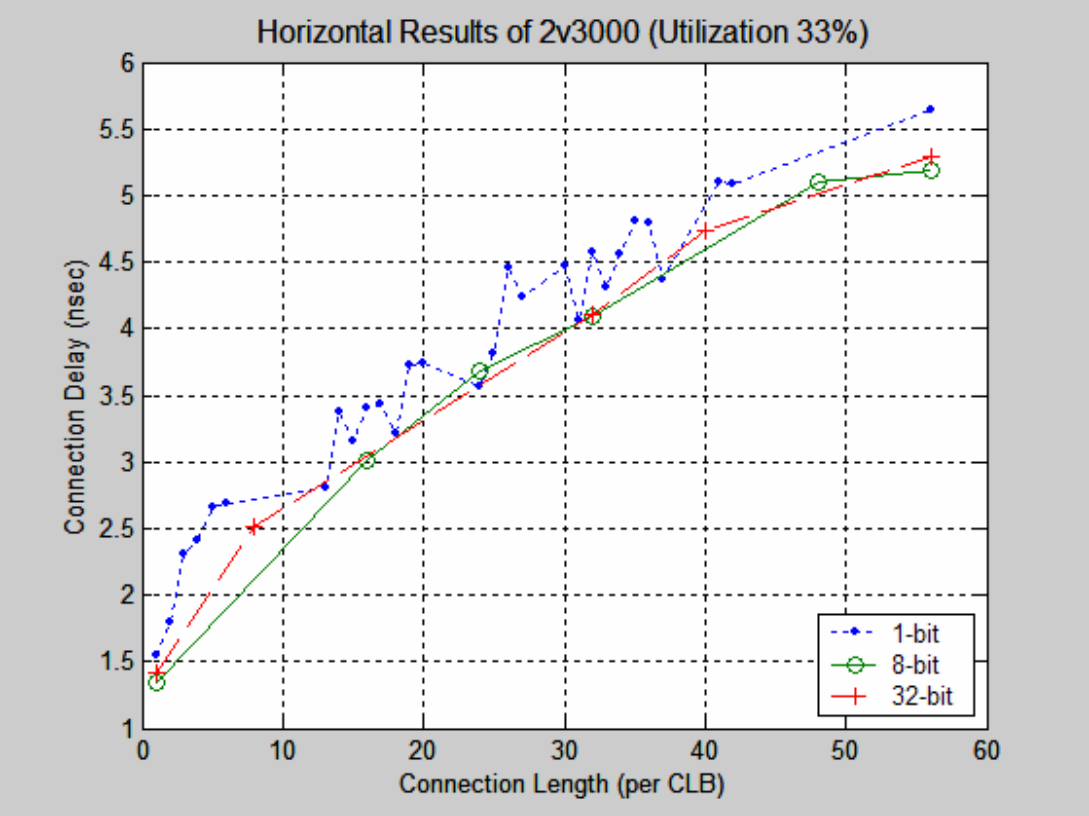
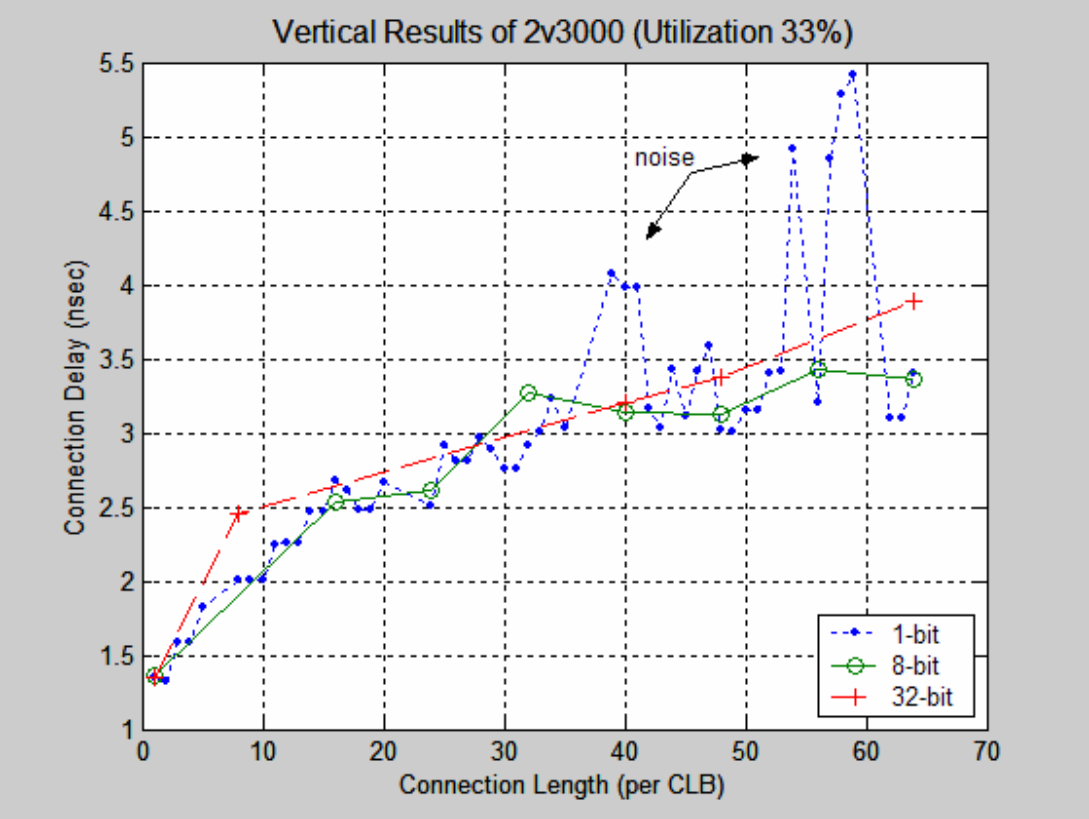


Figure 17 – 18 : Virtex2-3000



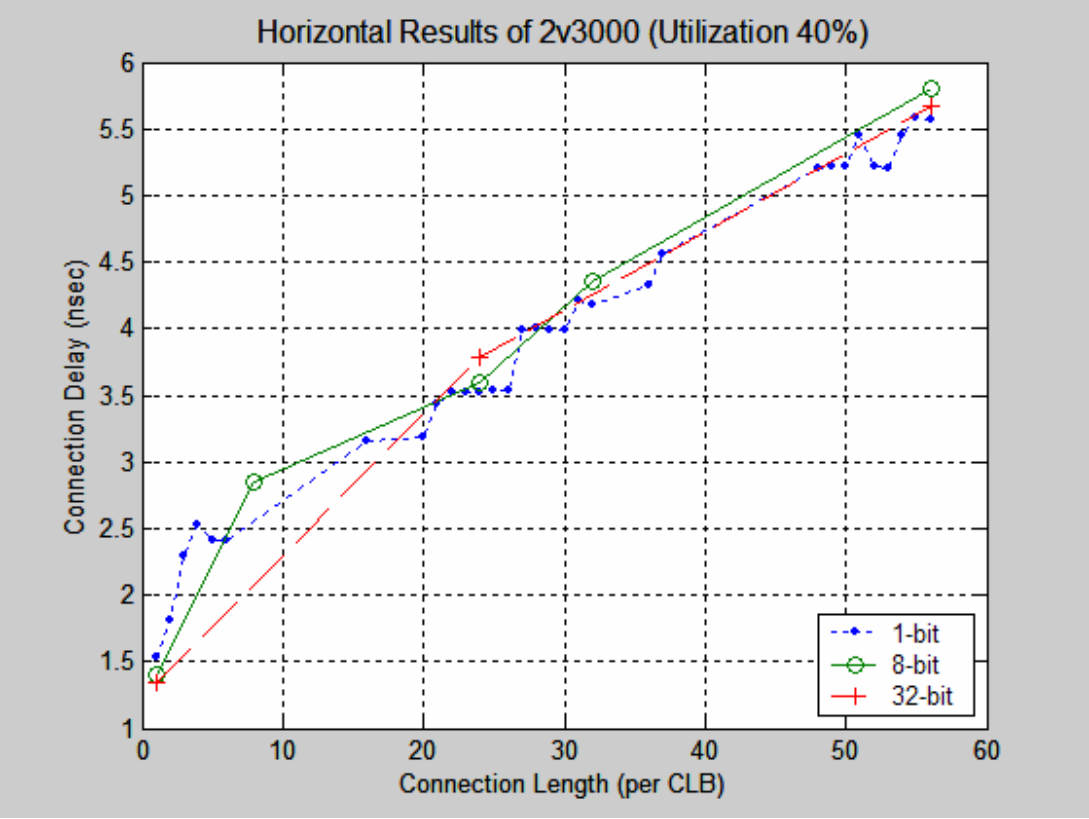
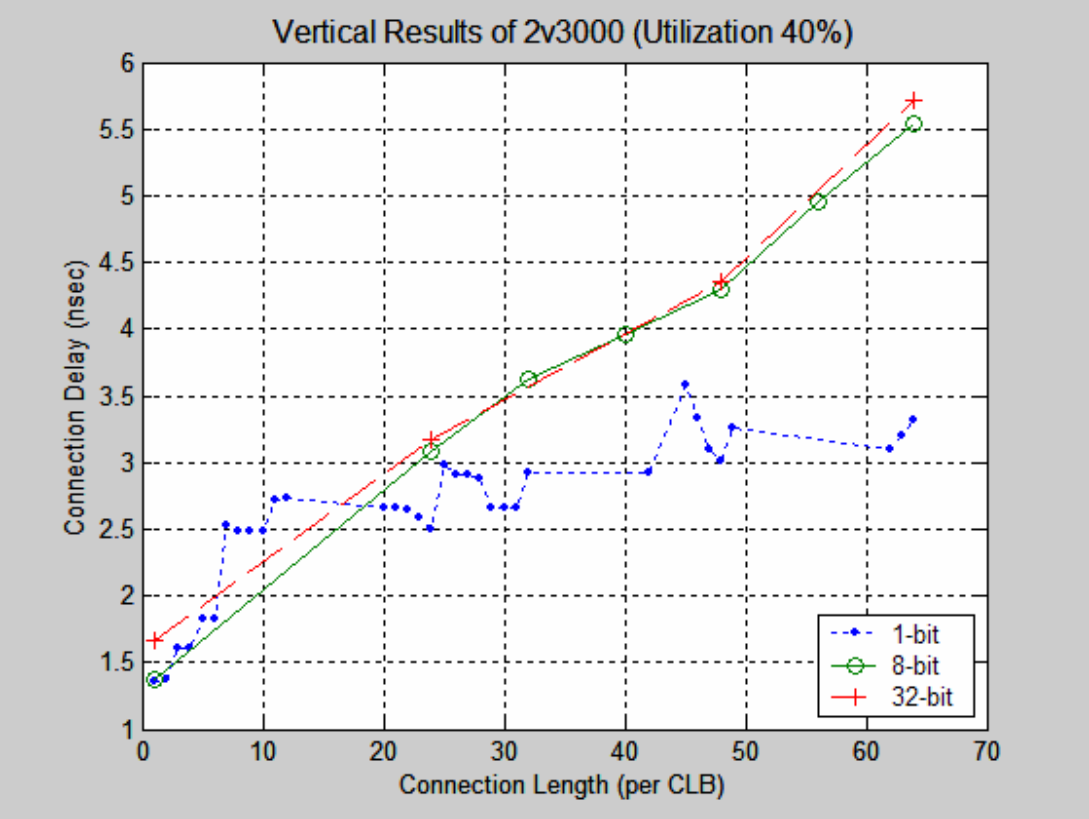


Figure 19 – 20 : Virtex2-3000



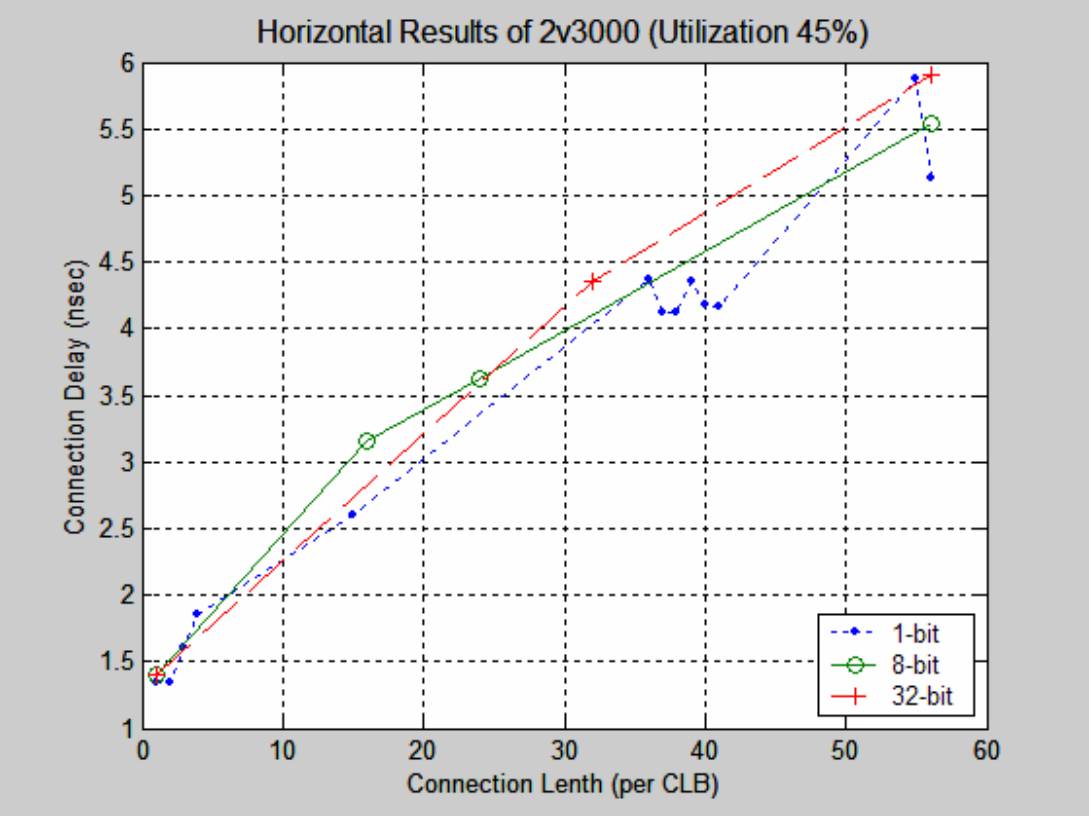
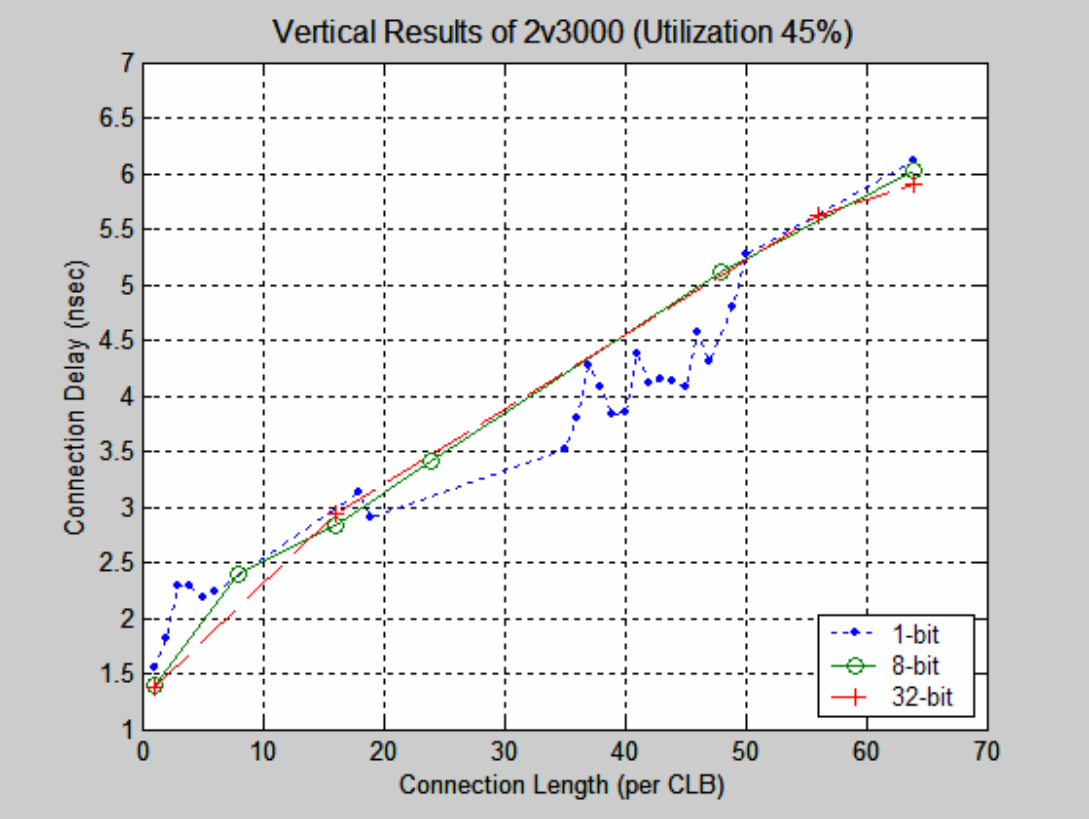


Figure 21 – 22 : Virtex2-3000



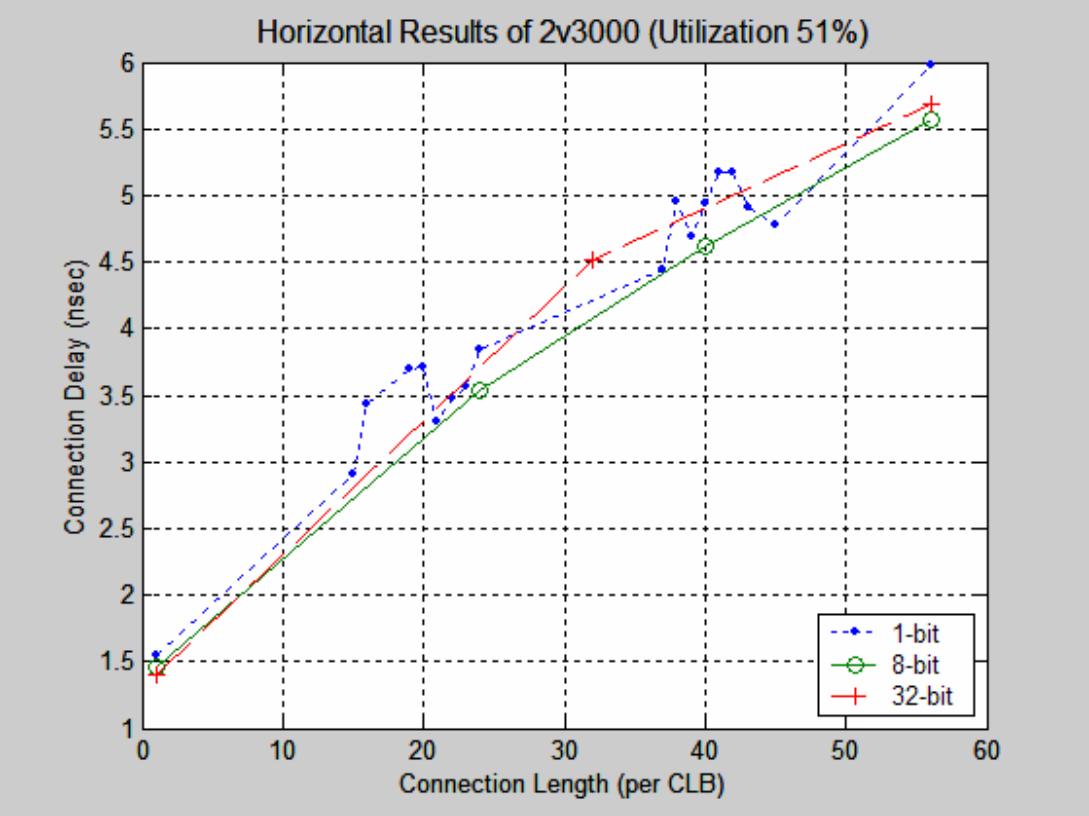
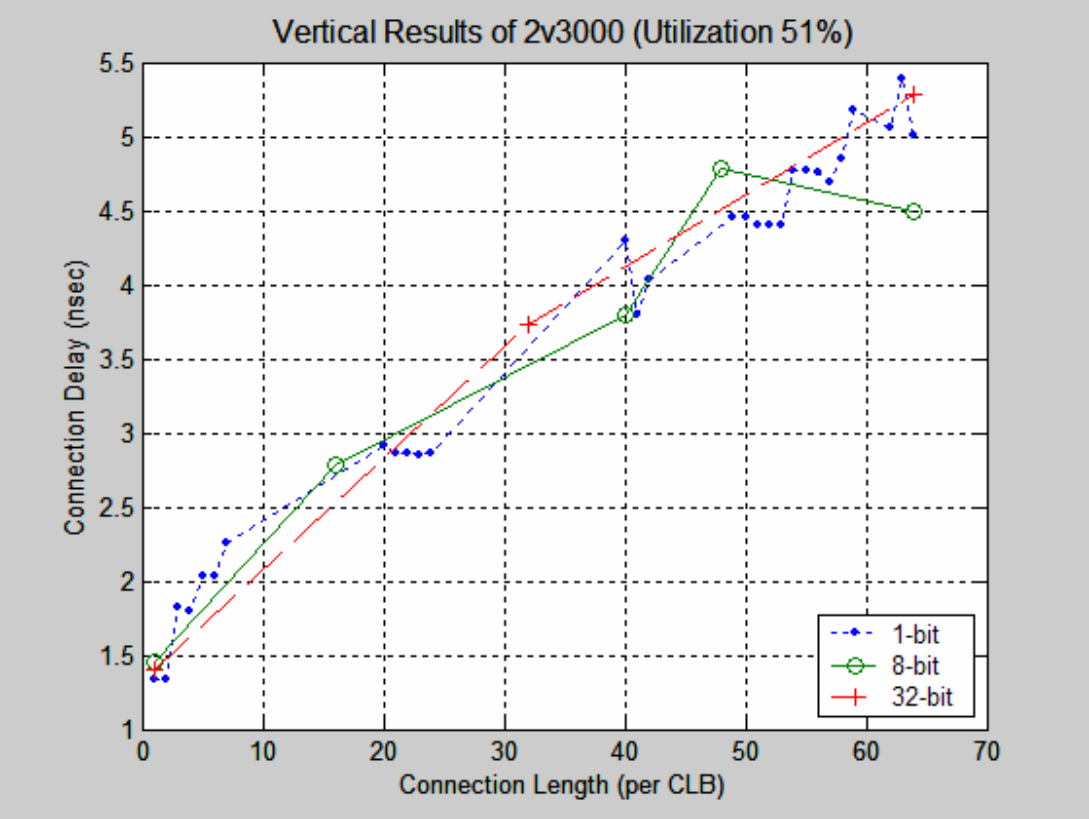


Figure 23 – 24 : Virtex2-3000



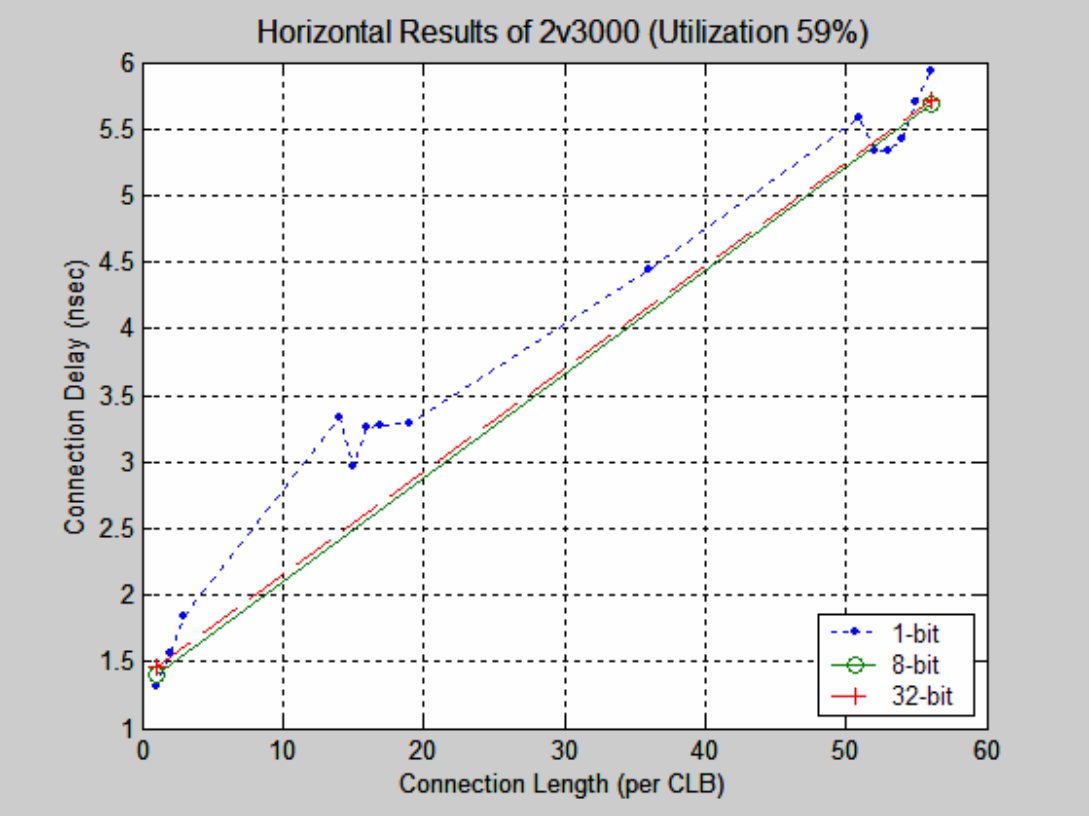
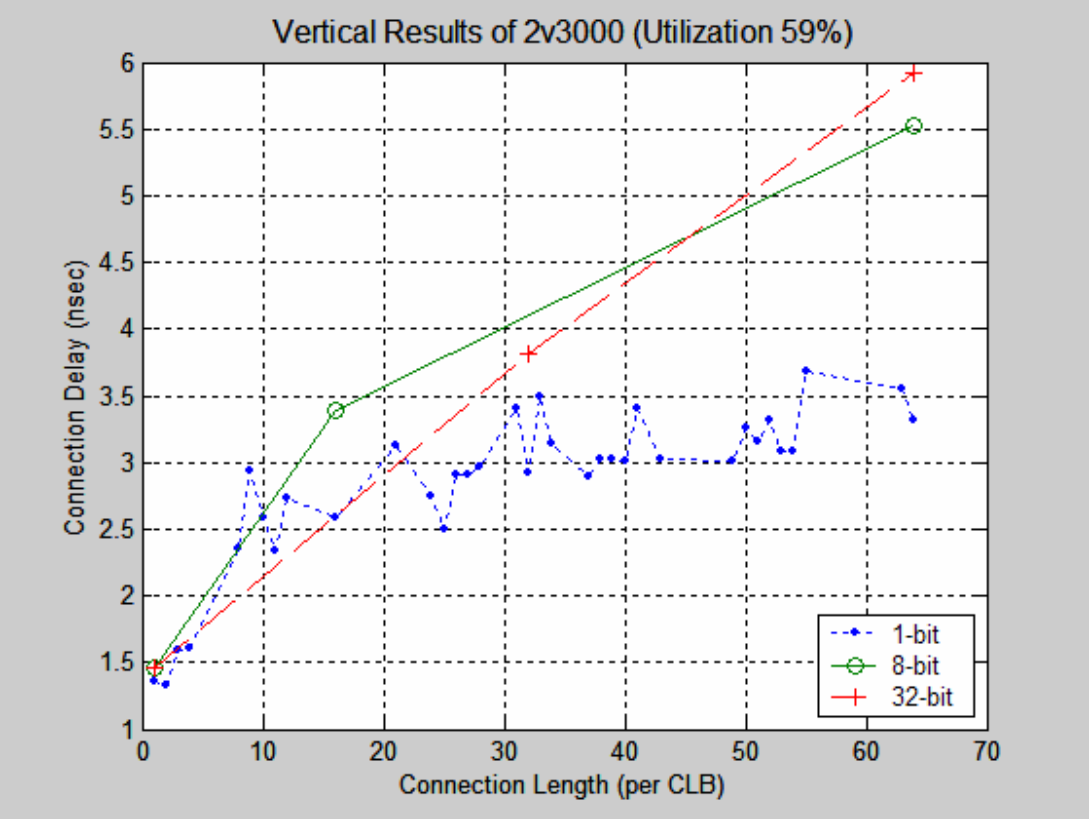


Figure 25 – 26 : Virtex2-3000



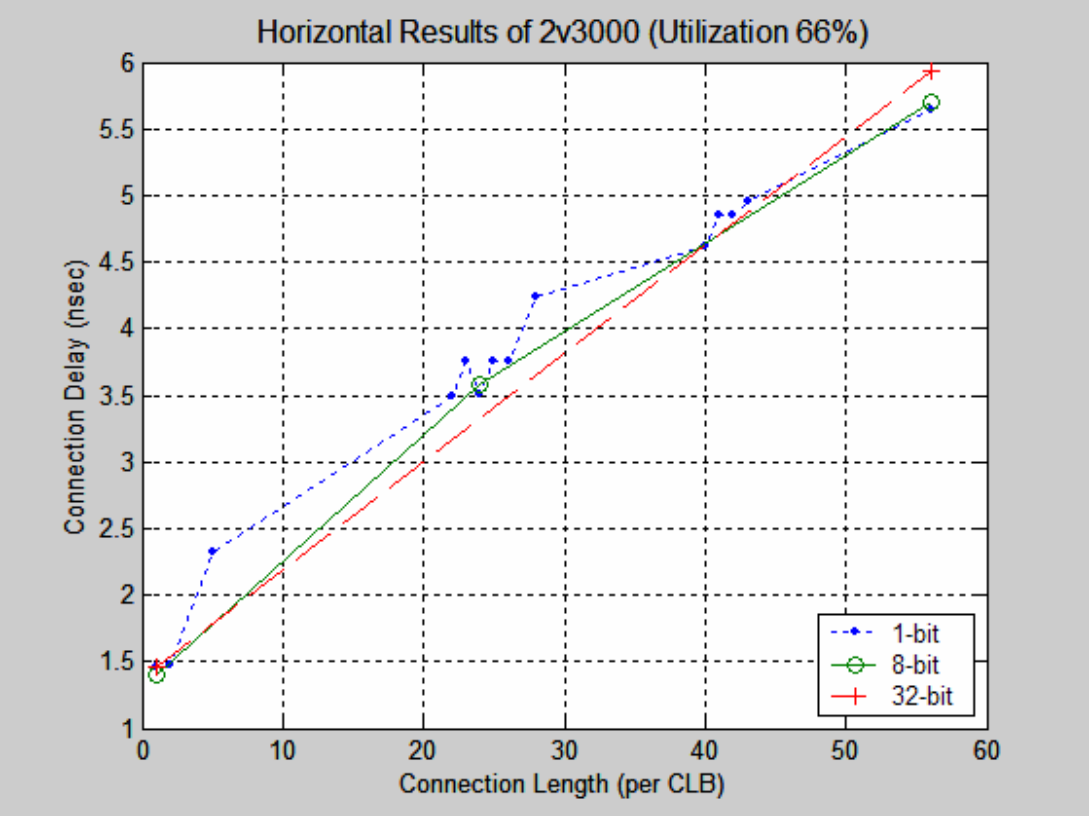
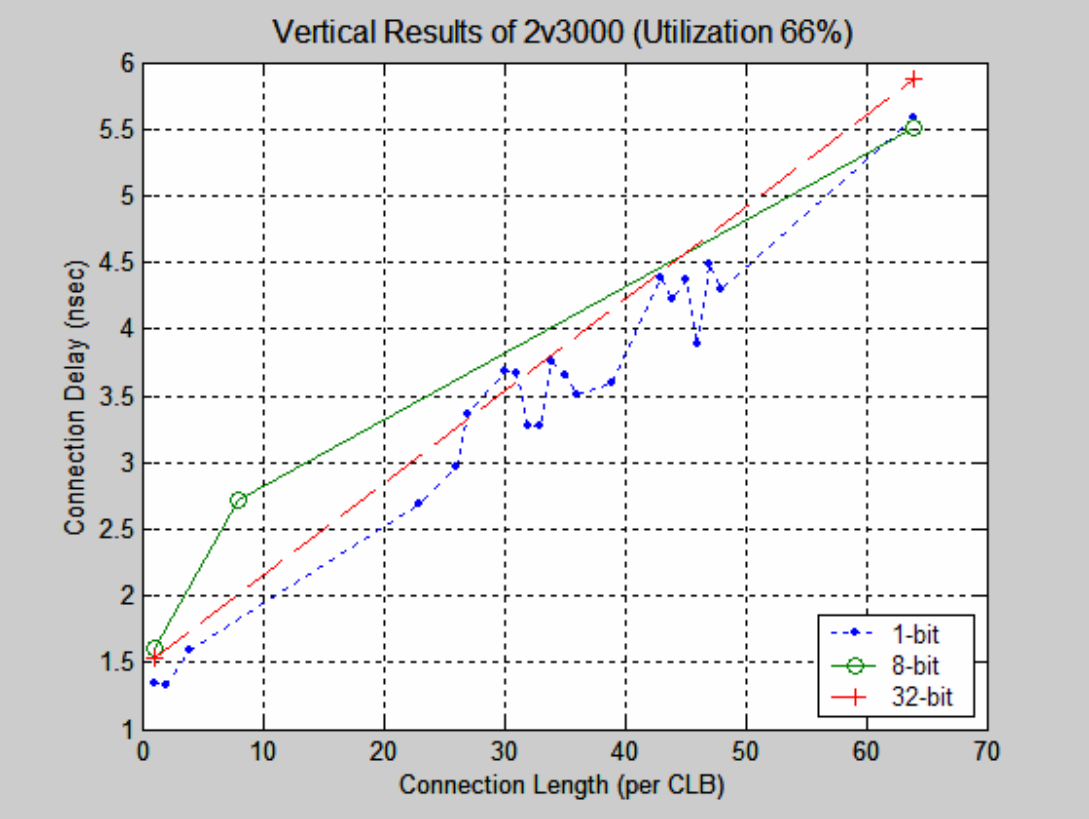


Figure 27 – 28 : Virtex2-3000



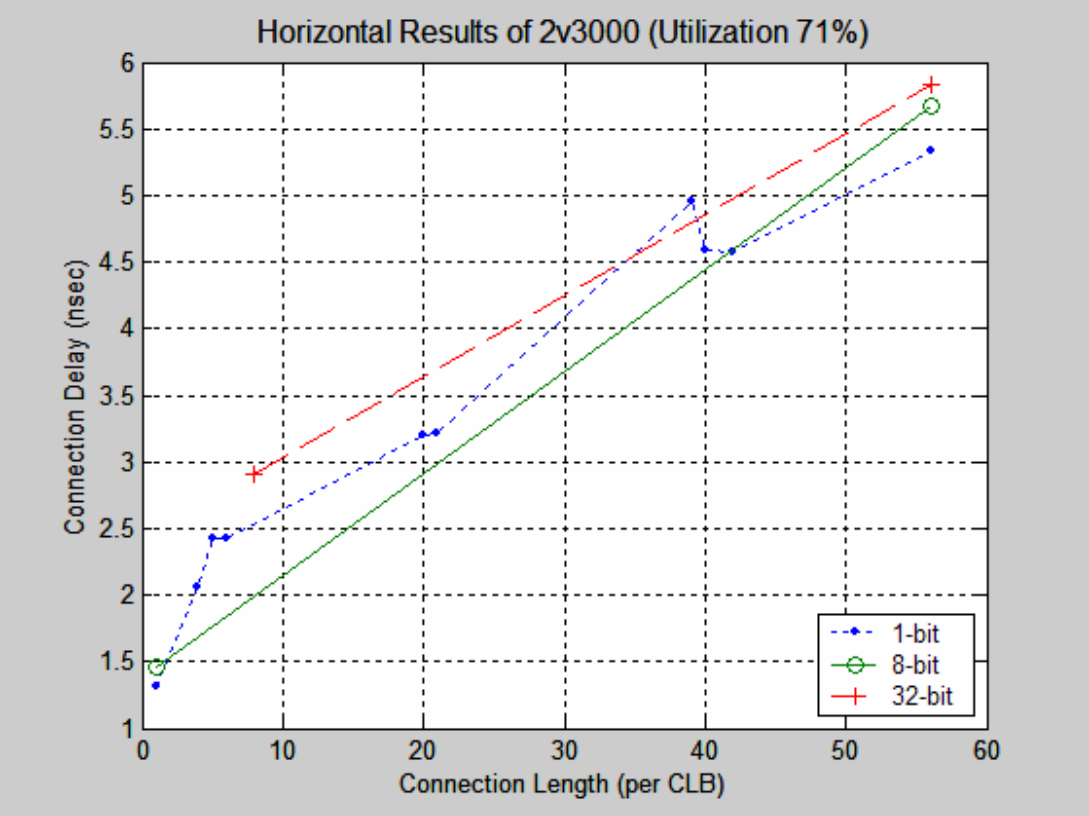
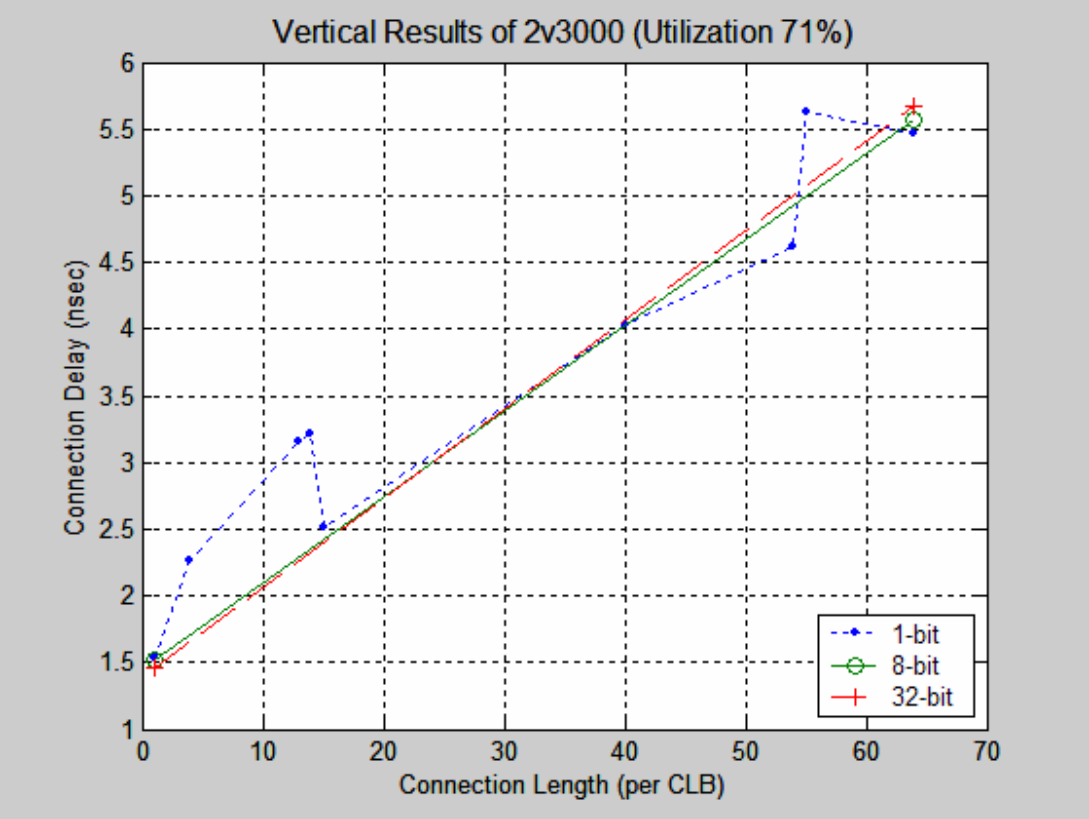
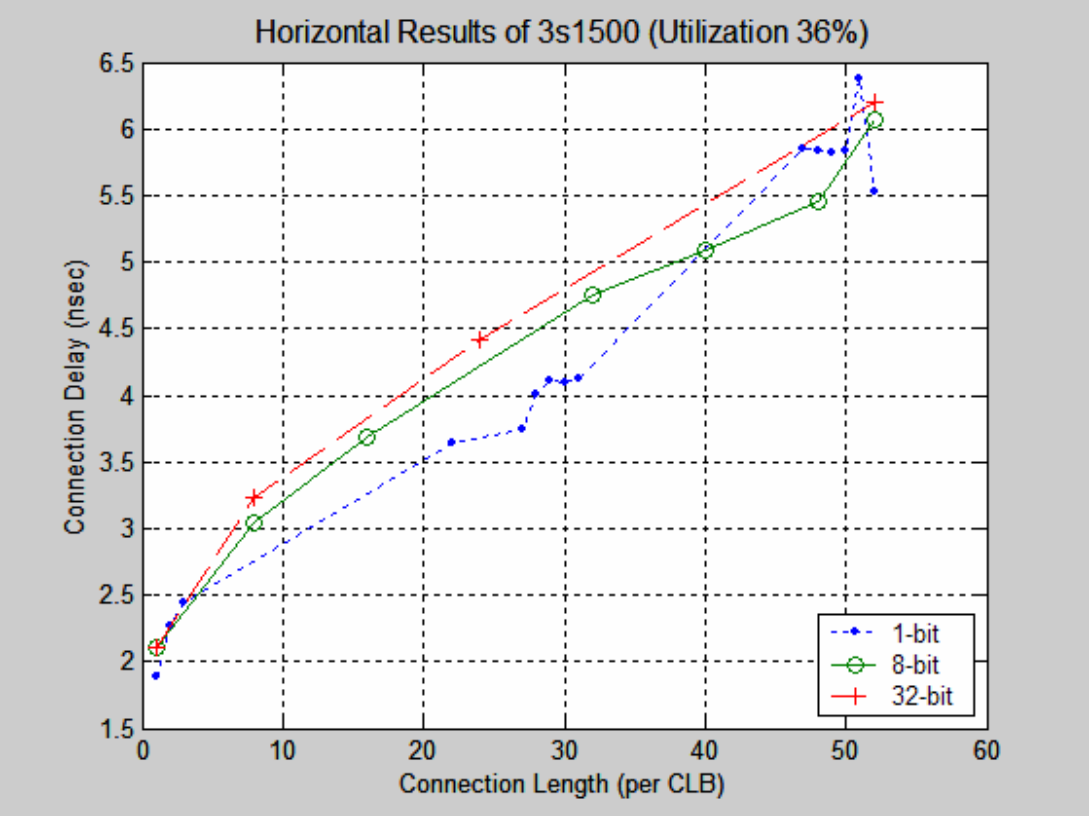
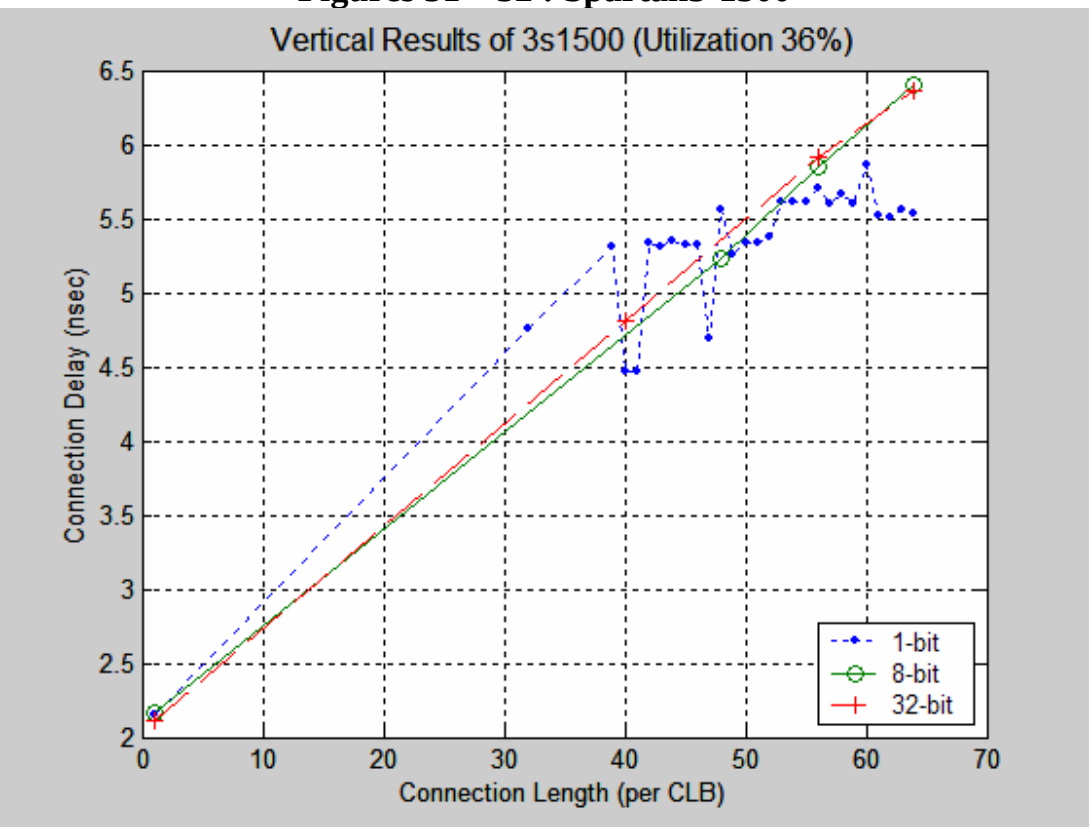


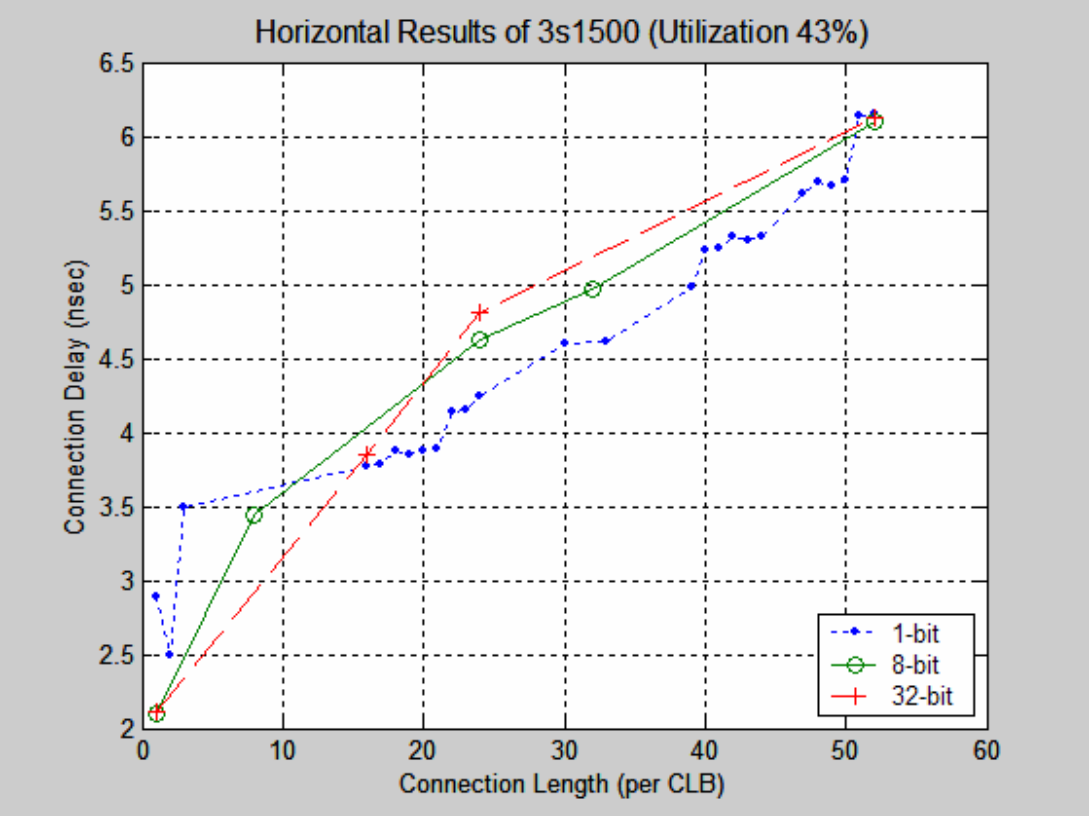
Figure 29 – 30 : Virtex2-3000



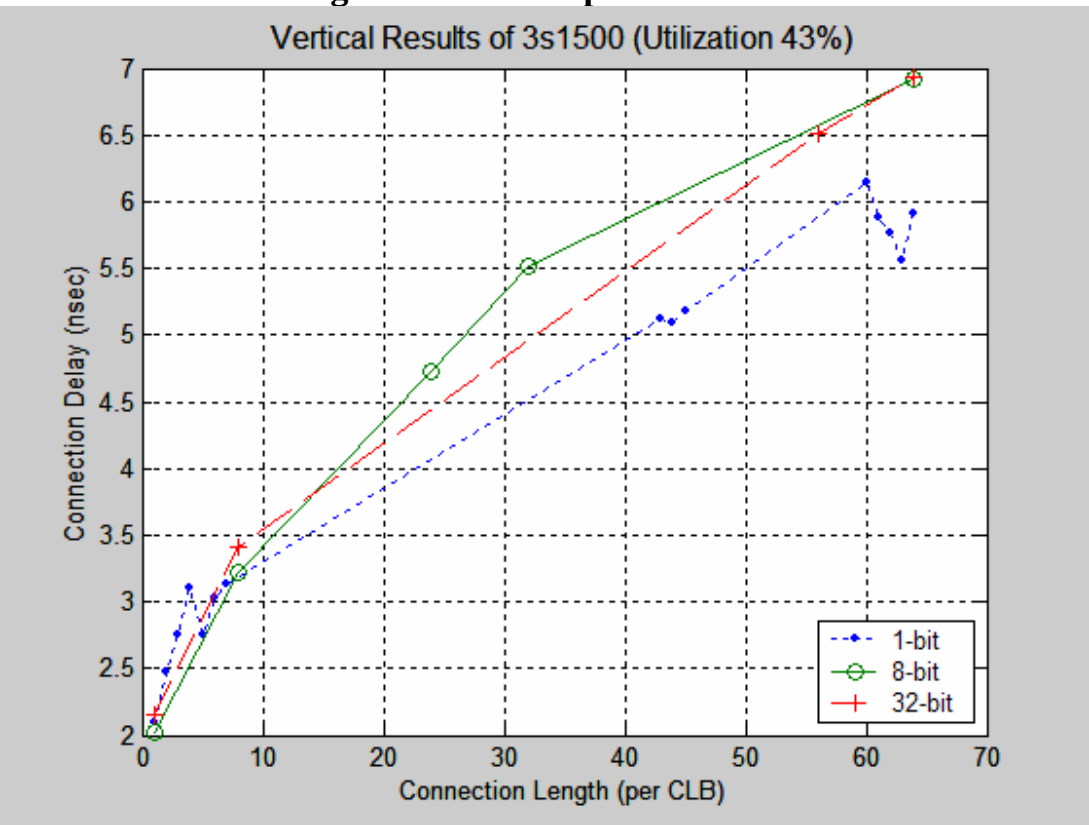


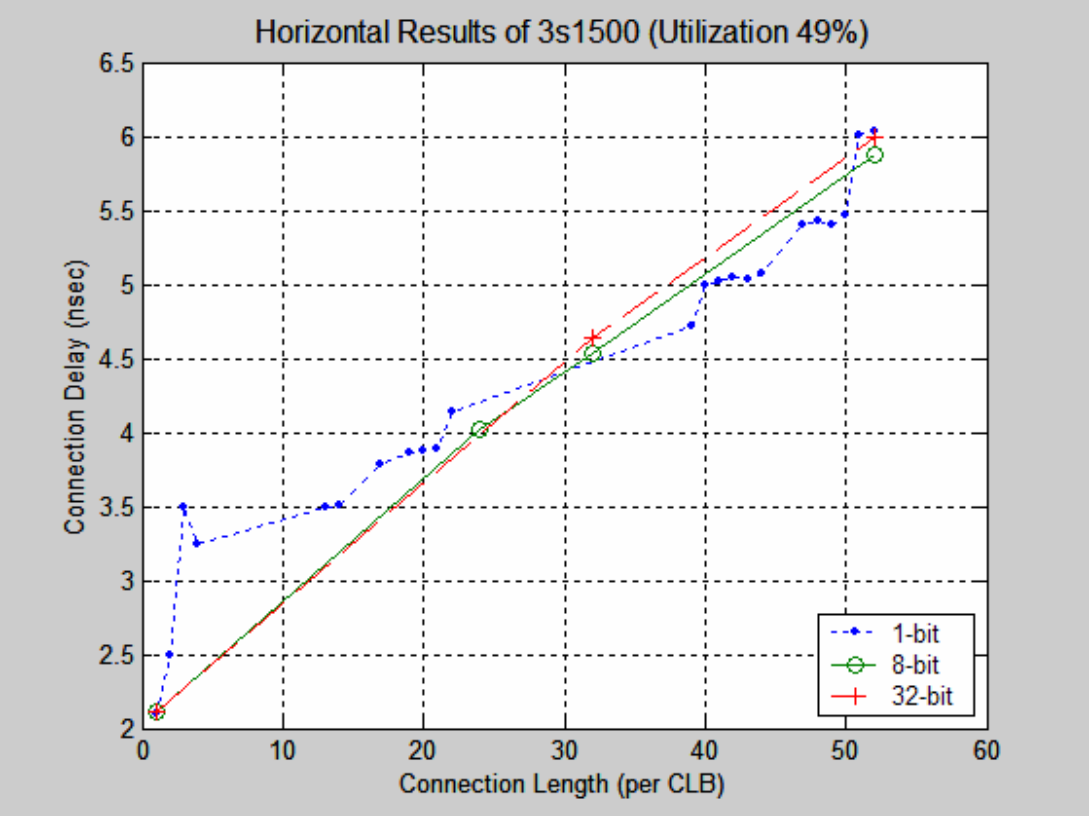
Figures 31 – 32 : Spartan3-1500



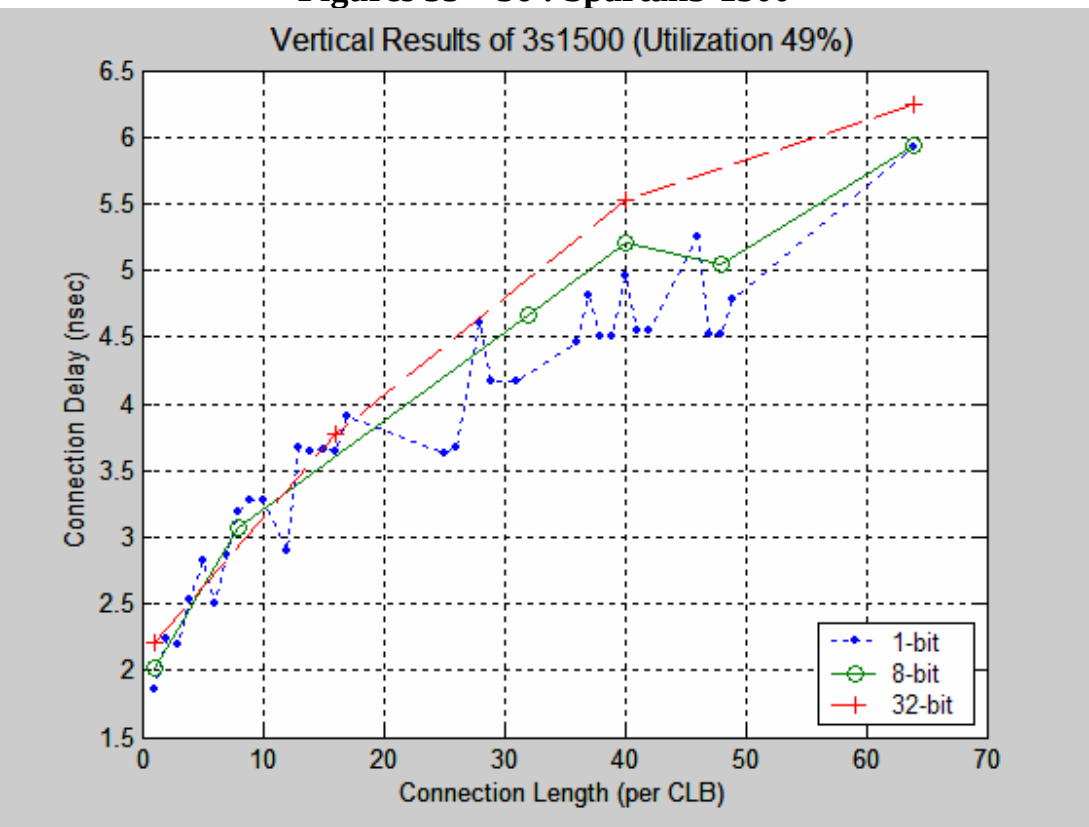


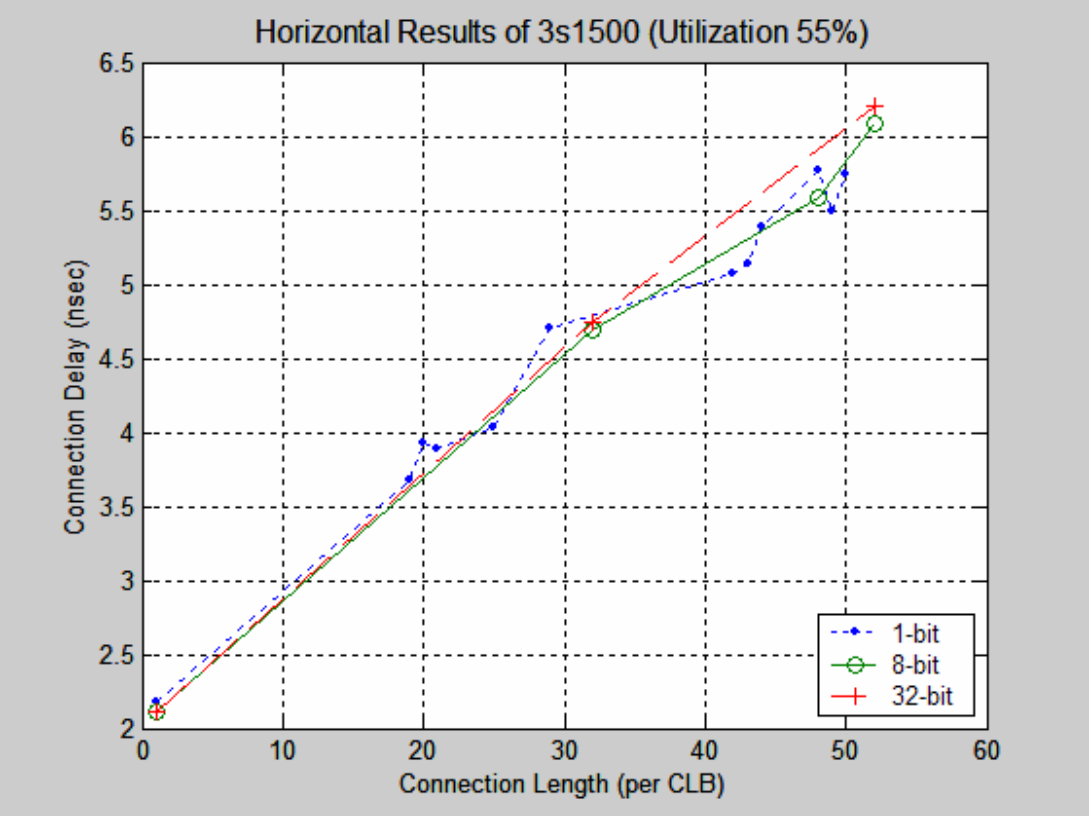
Figures 33 – 34 : Spartan3-1500



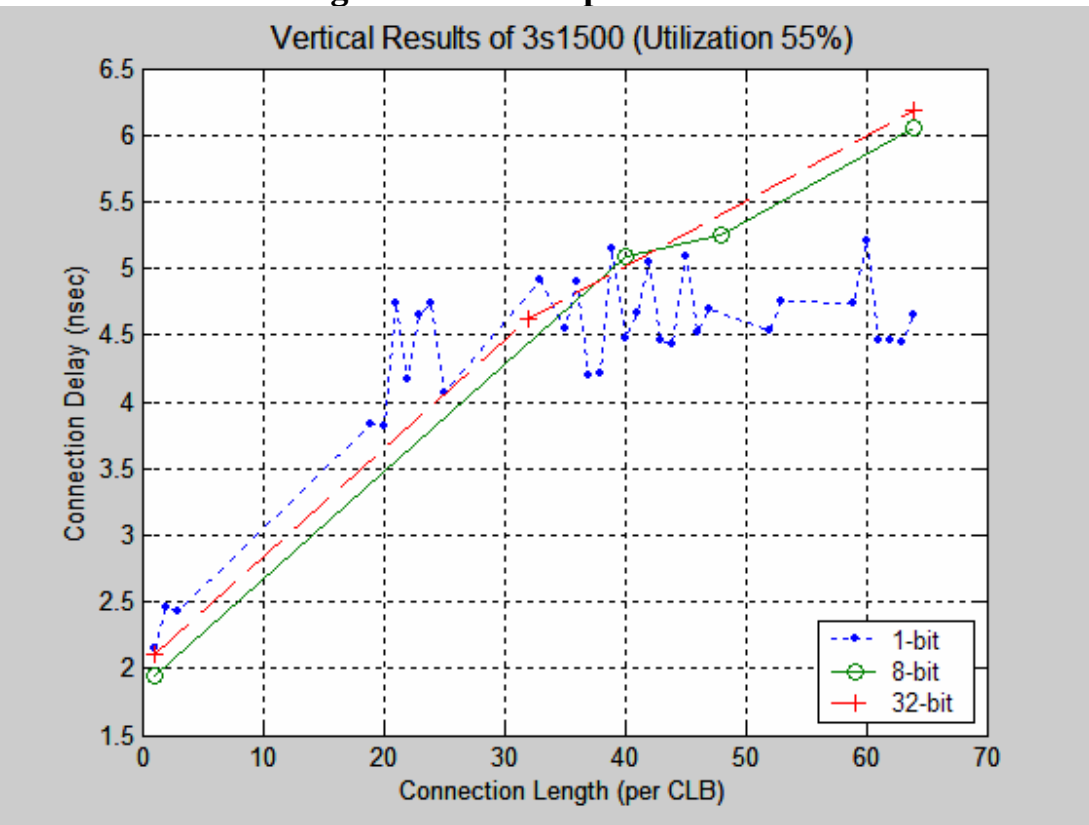


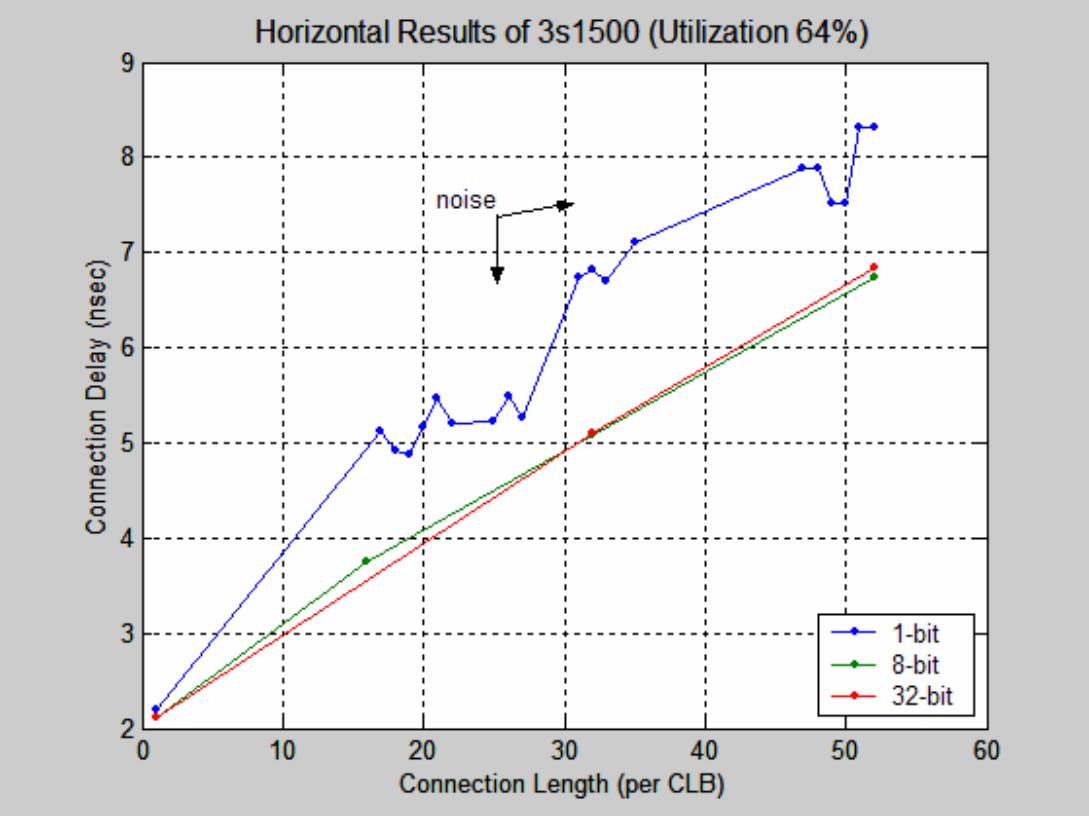
Figures 35 – 36 : Spartan3-1500



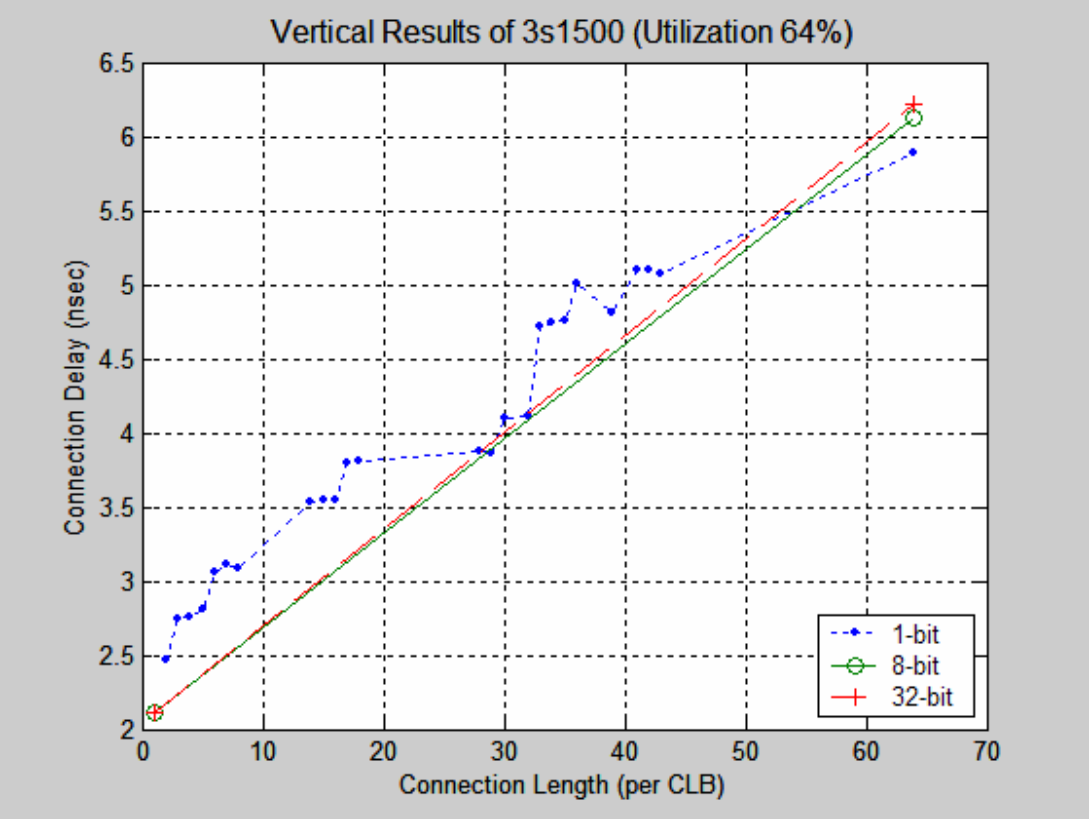


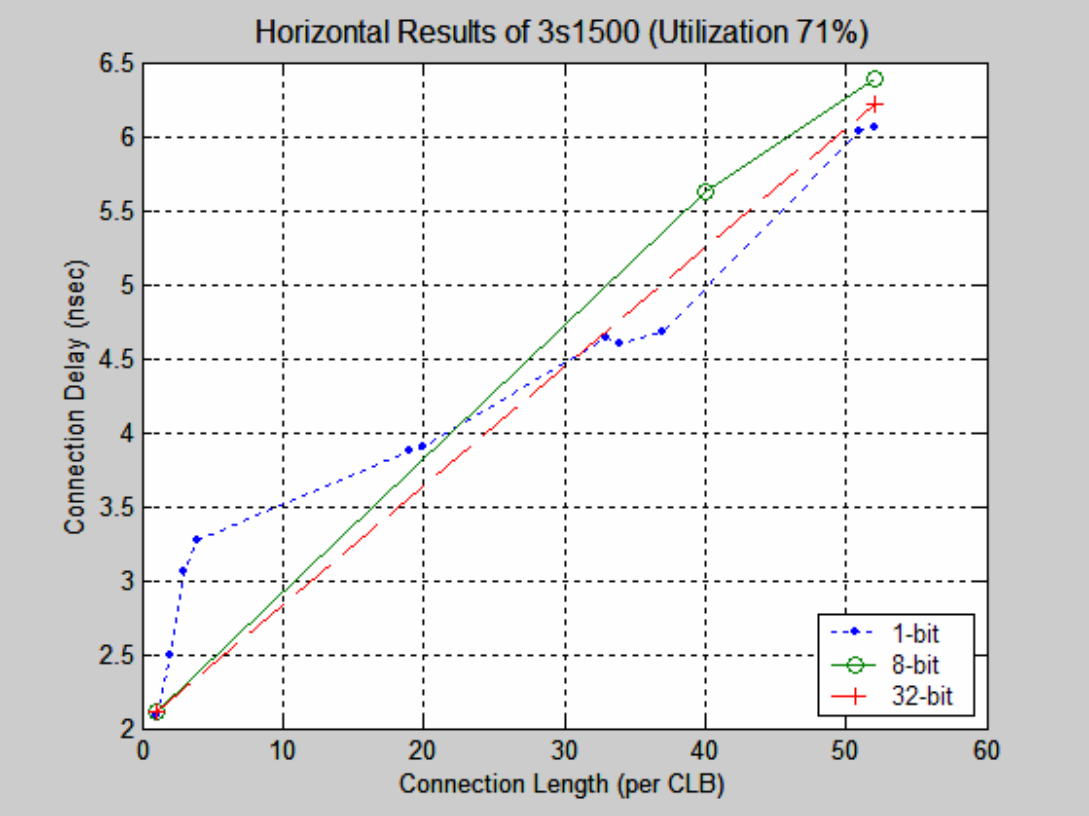
Figures 37 – 38 : Spartan3-1500



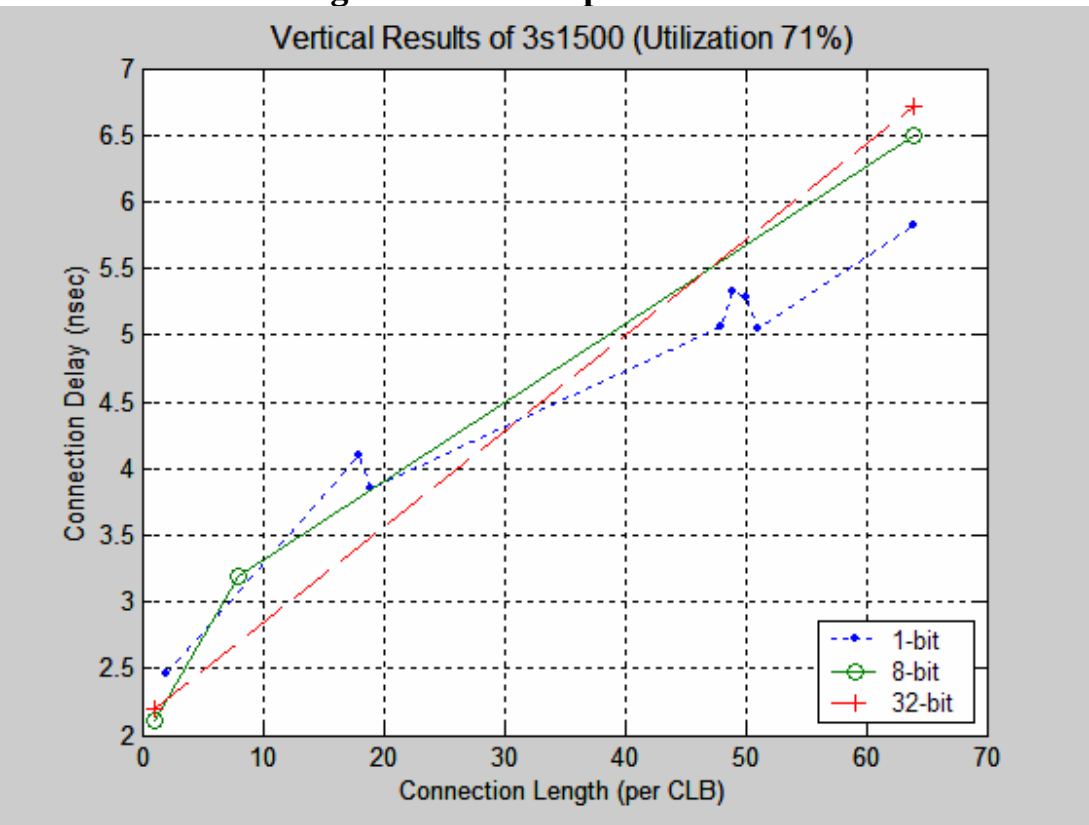


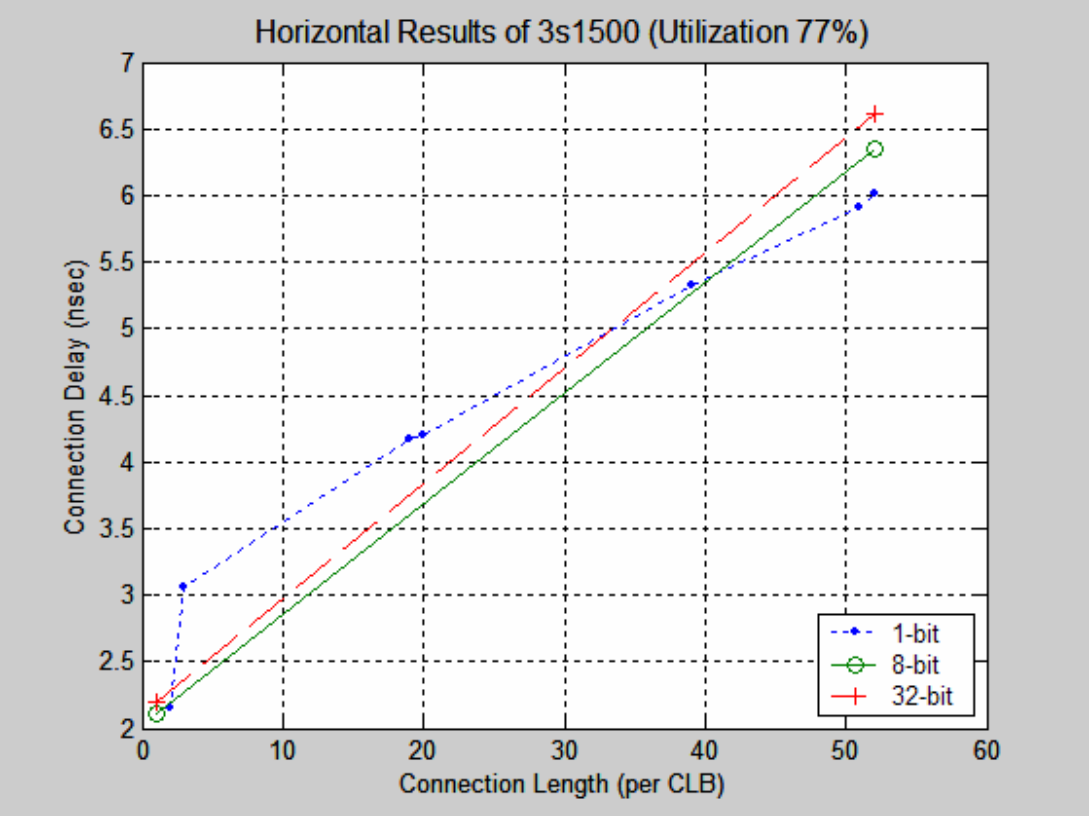
Figures 39 – 40 : Spartan3-1500



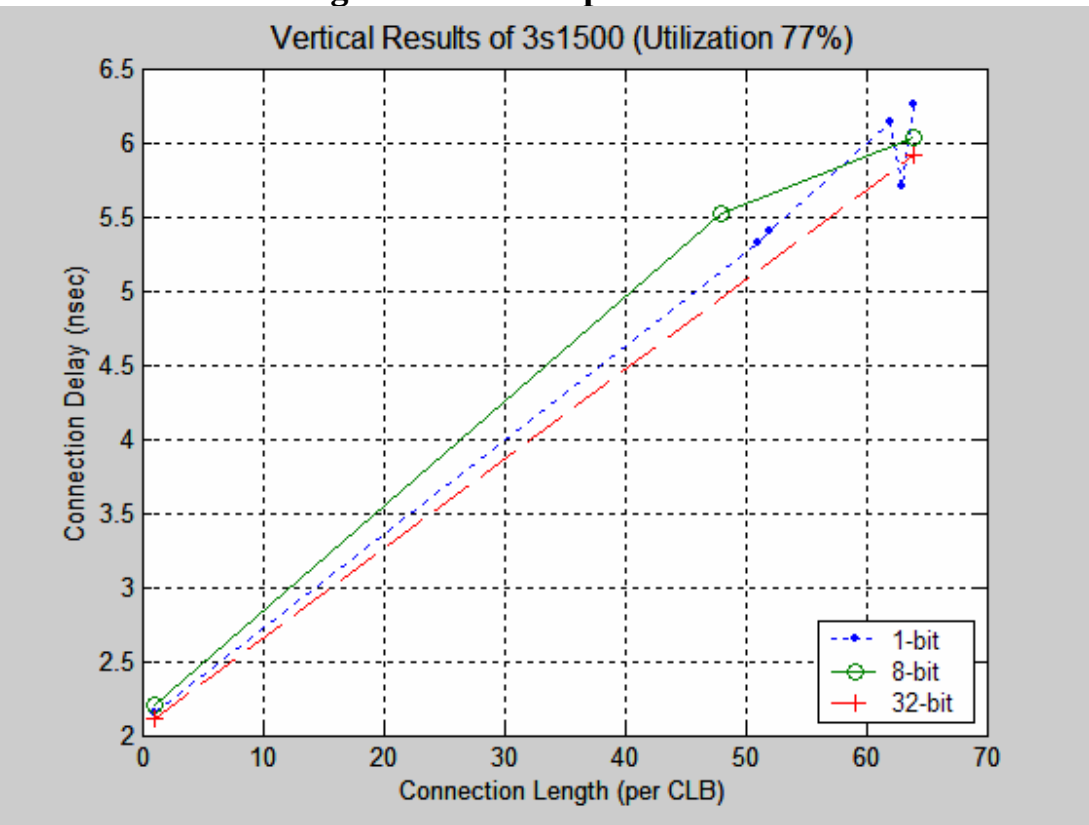


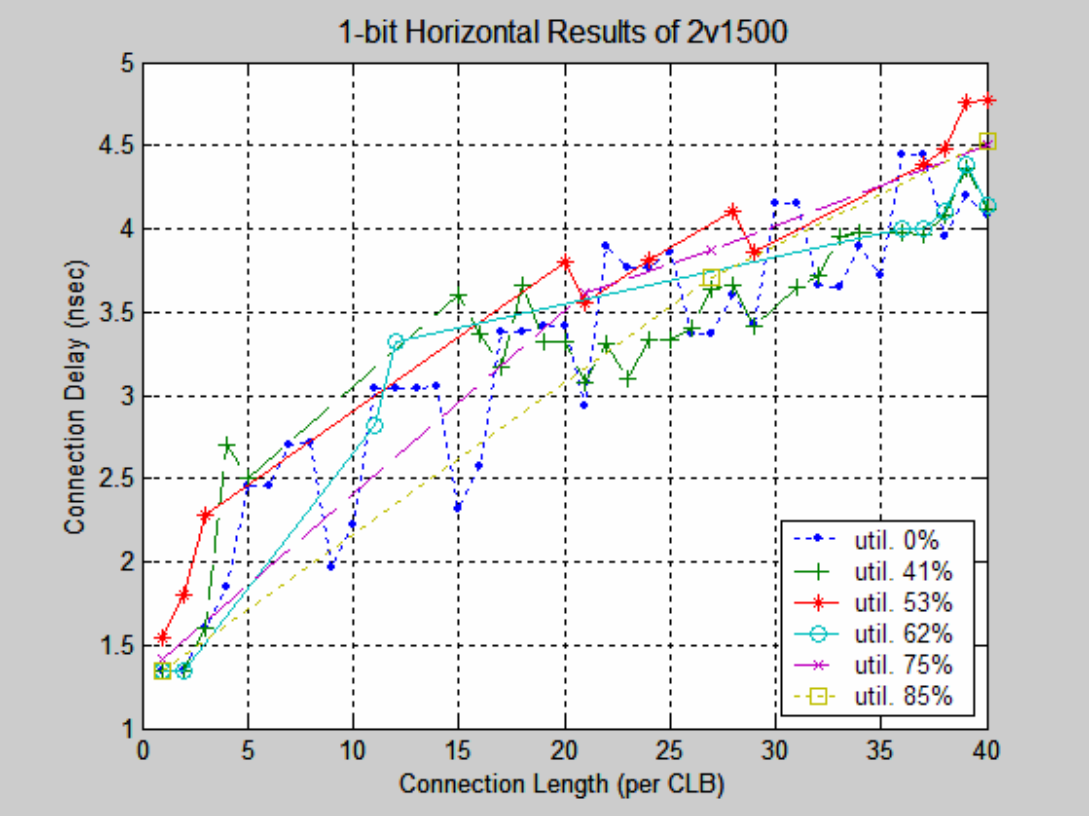
Figures 41 – 42 : Spartan3-1500



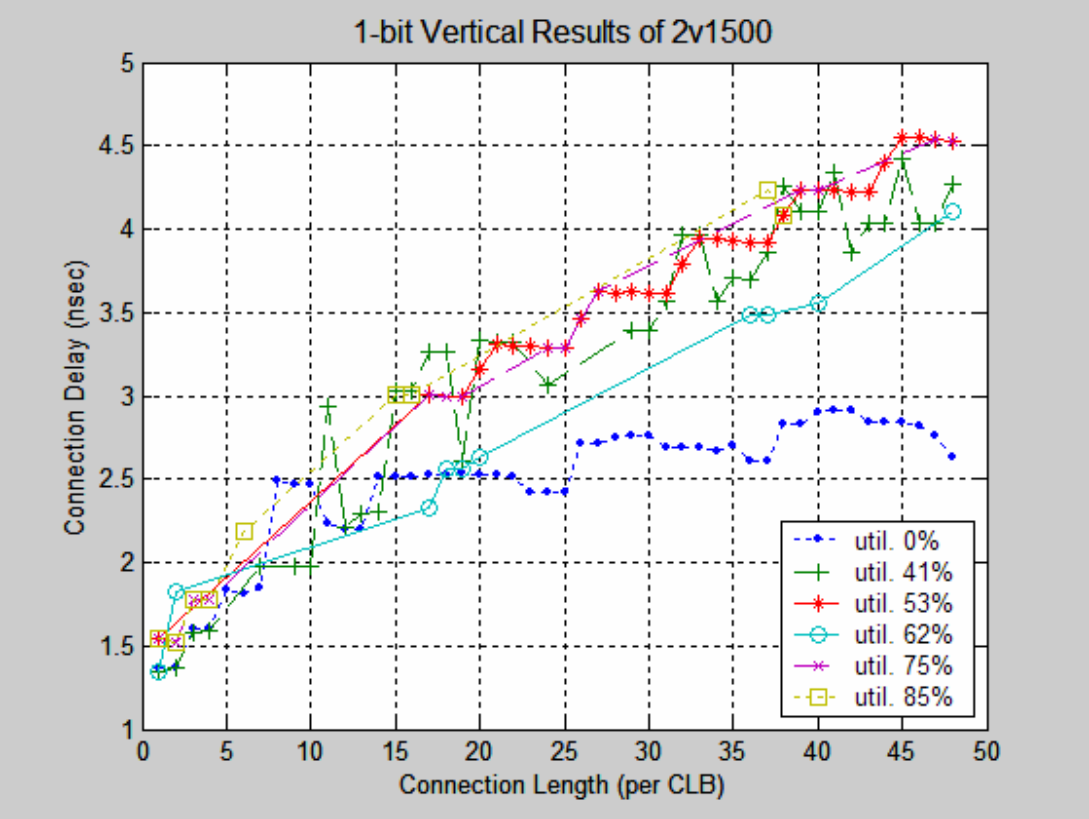


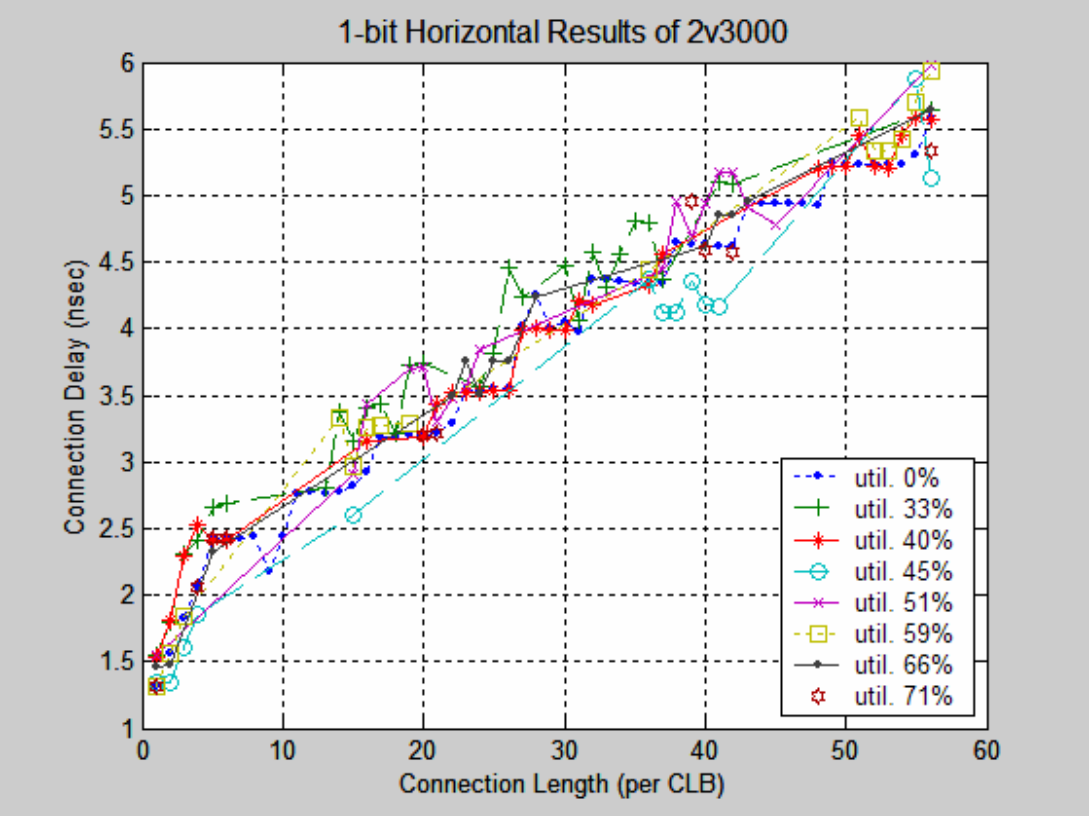
Figures 43 – 44 : Spartan3-1500



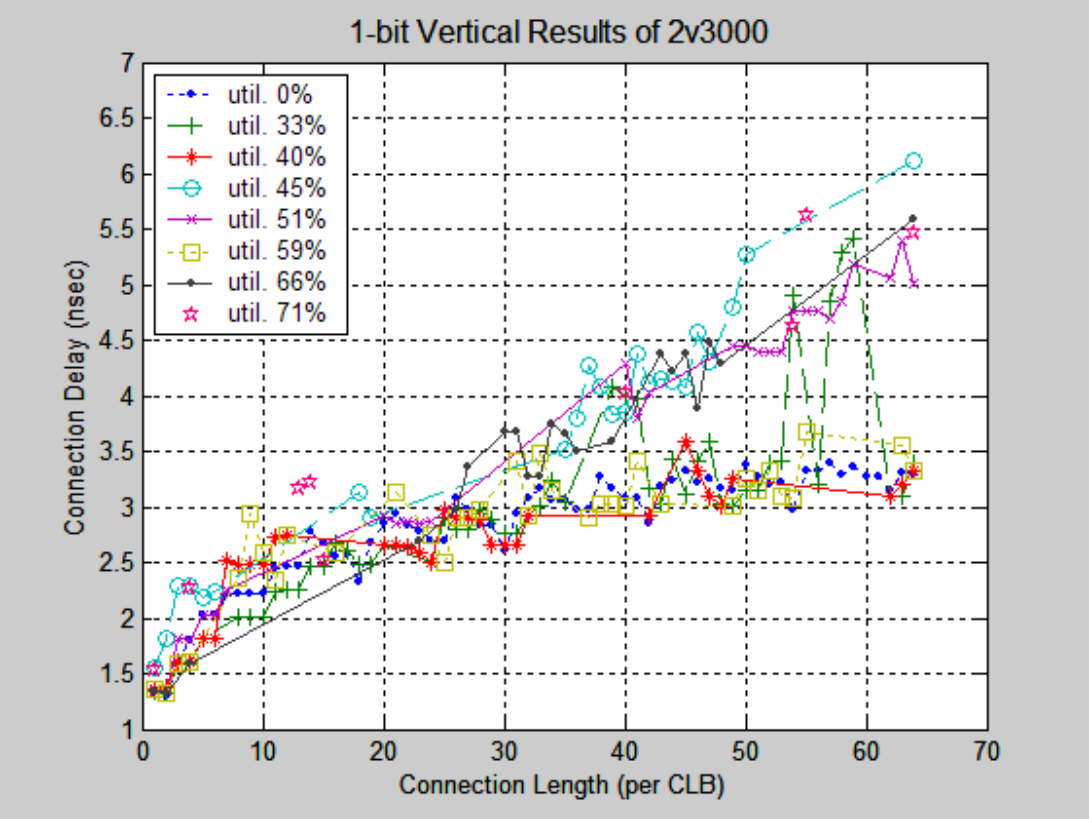


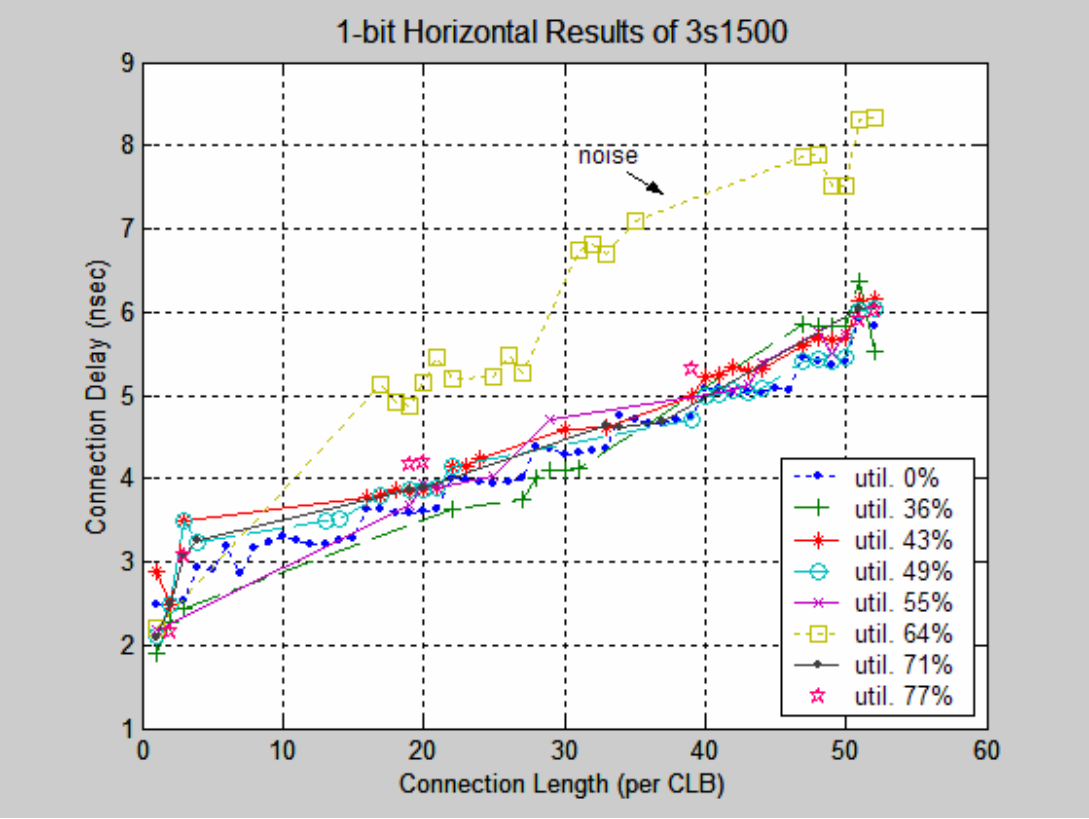
Figures 45 – 46 : Virtex2-1500



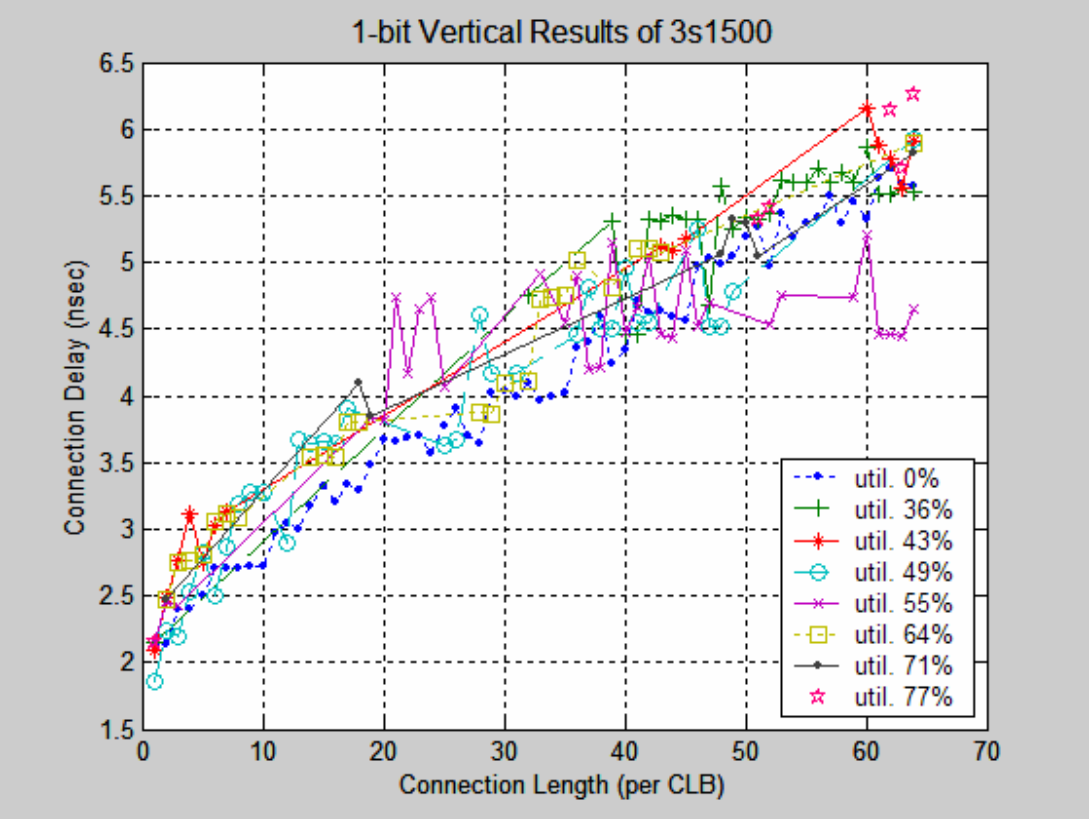


Figures 47 – 48 : Virtex2-3000





Figures 49 - 50 : Spartan3-1500



REFERENCES

1. Xilinx Incorporation, Data Book, 2004.

2. Altera Incorporation, Data Book, 2004.

3. Actel Corporation, Data Book, 2004.

4. “Programmable Logic News and Views”, Volume VIII, Number 10, October 1999, pages 3-4.

5. J. Rose and D. Hill, “Architectural and physical design challenges for one-million gate FPGAs and beyond,” in Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Feb. 1997, pp. 129-132.

6. Lucent Technologies, FPGA Data Book, 1999.

7. QuickLogic, Data Book, 1999.

8. Cypress Semiconductor, The Programmable Logic Data Book, 1999.

9. R. Brayton, E. Detjens, S. Krishna, T. Ma, P. McGeer, L.Pei, N. Phillips, R. Rudell, R. Segal, A. Wang, R. Yung and A. Sangiovanni-Vincentelli, "Multiple-Level Logic Optimization System," *Proc. IEEE International Conference on Computer Aided Design*, pp. 356-359, Nov. 1986.

10. D. Gregory, K. Bartlett, A. de Geus and G. Hachtel, "Socrates: a system for automatically synthesizing and optimizing combinational logic," *Proc. 23rd Design Automation Conference*, June 1986, pp. 79-85.

11. M. Kahrs, "Matching a parts library in a silicon compiler," *Proc. IEEE International Conference on Computer Aided Design*, pp. 169-172, Nov. 1986.

12. K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," *Proc. 24th Design Automation Conference*, June 1987, pp. 341-347.

13. R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. 28th Design Automation Conference, June 1991, pp. .*

14. M. Hanan and J.M. Kurtzberg, "Placement Techniques," Chapter 4 of *Design Automation of Digital Systems; Theory and Techniques*, M.A. Breuer, Ed., NJ, Prentice-Hall, 1972.

15. J. Rose, Z. Vranesic and W.M. Snelgrove, "ALTOR: An Automatic Standard Cell Layout Program," *Proc. Canadian Conference on VLSI*, Nov. 1985, pp. 168-173.

16. C. Sechen and K. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," *Proc. IEEE International Conference on Computer Aided Design*, Nov. 1987, pp. 478-481.

17. W. Dees and R. Smith, "Performance of Interconnection Rip-Up and Reroute Strategies," in *Proc. 18th Design Automation Conference*, June 1981, pp. 382-390.

18. R. Linsker, "An Iterative-Improvement Penalty-Function-Driven Wire Routing System," *IBM Journal of Research and Development*, vol. 28, Sept. 1984, pp. 613-624.

19. J. Cohn, D. Garrod, R. Rutenbar, and L. Carley, "KOAN/ANAGRAM II: New Tools for Device-Level Analog Placement and Routing," *IEEE Journal of Solid-State Circuits*, vol. 26, March 1991, pp. 330-342.

20. L. McMurchie and C. Ebeling, "Pathfinder: A Negotiation Based Performance Driven Router for FPGAs, "

21. R. Nair, "A Simple Yet Effective Technique for Global Wiring," *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, no. 6, March 1987, pp. 165-172.

22. “Layout driven decomposition with congestion consideration,” in *Proc. Design Test Eur.*, Mar. 2002, pp. 672–676.

23. I. Sourdis, "Efficient and High-Speed FPGA-based String Matching for Packet Inspection," Ms thesis, Technical University of Crete, 2004.