To the Graduate Council:

I am submitting herewith a thesis written by Nayan Dinesh Patel entitled "Development and Automation of a Procedure for Improved Timing Analysis of Field-Programmable Gate Arrays." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin

Dr. Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Roberta Bodenheimer

Daniel B. Koch

Accepted for the Council:

Lewinkel

Associate Vice Chancellor
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Requests for permission for extensive quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature Nayan D. Patel

Date May 1, 1992

DEVELOPMENT AND AUTOMATION OF A PROCEDURE FOR IMPROVED TIMING ANALYSIS OF FIELD-PROGRAMMABLE GATE ARRAYS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

NAYAN DINESH PATEL

May 1992

DEDICATION

This thesis is dedicated to the members of my family

Mr. Chaganlal Lalubhai Patel

Mrs. Kasiba Chaganlal Patel

Mr. Dineshchandra Chaganlal Patel

Mrs. Ansuyaben Dineshchandra Patel

Mr. Rajeshbhai Dineshchandra Patel

and

Mr. Amit Dineshchandra Patel

ABSTRACT

The timing delay of a net connecting logic cells in field-programmable logic devices varies considerably (sometimes by a factor of two) depending on the specific routing resources used. The designer must analyze these delays carefully to minimize the time period required for the system clock and, hence, to maximize the speed of the system. This thesis describes a procedure, which has been automated, for improved timing analysis of these devices. This procedure will perform timing analysis with precise results after the automatic placement and routing has been completed for a design. The software was written in the C programming language specifically for the Xilinx family of devices. However, the procedure has been generalized to be readily adopted to other devices. The developed software, titled the "PATHFINDER," can be executed on an IBM-compatible personal computer.

ACKNOWLEDGEMENT

I would like to extend my most sincere gratitude to Dr. Donald W. Bouldin for his guidance, patience, advice and assistance throughout the preparation of this thesis. His encyclopedic knowledge of microelectronic systems has greatly influenced this thesis.

I am thankful to Dr. Donald W. Bouldin for serving as the Major Professor on my committee. Thanks also goes to my committee members, Dr. Daniel B. Koch and Dr. Robert E. Bodenheimer, for their comments and suggestions in the final stages of the thesis.

I am very thankful to my father, Mr. Dinesh Chaganlal Patel, a civil engineer, for his scholarly inspirations since my early childhood. I would also like to thank both of my parents, Dinesh and Ansuya Patel, for their financial assistance and advice which enabled me to continue my education.

TABLE OF CONTENTS

CHAPTER		PAGE
I.	INTRODUCTION	1
	Introduction	1
	The Problem	2
II.	FIELD PROGRAMMABLE GATE ARRAYS	5
	Xilinx	6
	Actel	8
	Altera and Plus Logic	8
	Rationale for Simulation	10
III.	THE XILINX 3000 SERIES	11
	Overview of the Architecture	11
	Configurable Logic Block	12
	Input/Output Block	14
IV.	THE DEVELOPMENT OF THE PATHFINDER PROGRAM	16
	Concept and Design Goals of the PATHFINDER	16
	Algorithm for the PATHFINDER	16
	Destinations Nodes	24
	Source Nodes	25
V.	RESULTS	30
	Test Case #1 -- N5R	30
	Results from Test Case #1	34
	Test Case #2 -- NEW2	36
	Results from Test Case #2	36
	Test Case #3 -- LCA1	39
	Results from Test Case #3	39
VI.	DISCUSSION AND CONCLUSION	41
	Comparison with the BAX Software	41
	Contributions of the PATHFINDER Program	43
	Summary	43

BIBLIOGRAPHY 44

APPENDIX 46
APPENDIX A -- PATHFINDER PROGRAM 47
APPENDIX B -- TEST CASE #1 72
Test Case #1 XNF File 72
Test Case #1 LCA File 77
Output File (DAT) For Test Case #1 79
APPENDIX C -- TEST CASE #2 80
Test Case #2 XNF File 80
Test Case #2 LCA File 84
Output File (DAT) For Test Case #2 86
APPENDIX D -- TEST CASE #3 87
Partial Listing of Test Case #3 XNF File 87
Partial Listing of Test Case #3 LCA File 89
Partial Listing of Output File (DAT) For Test Case #3 90

VITA 91

LIST OF FIGURES

Figure		Page
1.1	Logic-level Representation for a Sample Design	2
1.2	Physical-level Representation for a Sample Design	2
1.3	Unit-delay Logic Description	3
1.4	Back-annotated Full-Timing Description	3
2.1	FPGAs - The Best of Both Worlds	7
3.1	Structure of the Logic Cell Array of the Xilinx 3000 Series FPGA	11
3.2	Configurable Logic Block	13
3.3	Input/Output Block	15
4.1	Design Entry and XNF Translation	17
4.2	Initial Flowchart for the PATHFINDER Algorithm	19
4.3	Structured Flowchart of Algorithm for PATHFINDER	20
4.4	Block Diagram of All Types of Maximum Delays	23
4.5	CLB with Highlighted Paths for F and G Generators and X and Y flip-flops	26
4.6	IOB with Highlighted Paths for OUTFF and OBUF	27
4.7	IOB with Highlighted Paths for INFF and IBUF	28
5.1	Test Case # 1 Logic Design - N5R	31
5.2	Block Layout for N5R Routed - Actual Delays (not maximum)	32
5.3	Test Case # 2 Logic Design - NEW2	37
5.4	Output of the PATHFINDER for the NEW2 Design - NEW2.DAT	38
5.5	The Technique of Tracing Delay Paths from the LCA1.XNF File	40
6.1	Full-Timing Simulation Flow	42

LIST OF TABLES

Table		Page
4.1	Maximum Propagation Delays of Possible Paths	22
4.2	Summary of the Signal Propagation Delay Paths	29
5.1	Delay Paths for DX or DY's Traced Back to QX or QY	34
5.2	Delay Paths for DX or DY's Traced Back to IOBs (input)	35
5.3	Delay Path for IOBs (output) Traced Back to QY or QX	35

CHAPTER I

INTRODUCTION

Introduction

Steady advances in the level of integration in electronic circuits have improved performance and reliability, while reducing system size, power consumption, and costs. Increasing levels of integration are most evident in microprocessors and memory integrated circuits. With the need for higher speeds, higher densities, and lower costs, Field-Programmable Gate Arrays (FPGAs) have become an important alternative in the market for Application-Specific Integrated Circuits (ASICs) [1]. FPGAs are also very convenient because of their user programmability.

When a design is initially entered into a logic-level layout, no net delays are taken into account. After the design has been placed and routed, the physical-level layout embeds many of the logic gates into Configurable Logic Blocks (CLBs) or some type of logic modules.

In the Actel FPGA for example, the Actel Logic Module combines four inputs with four control lines to produce one output. The Xilinx FPGA uses a CLB, which consists of a 32-by-1 look-up table to implement Boolean functions. A CLB can take any combination of five inputs and produce two outputs using Boolean functions. This method can also generate two independent logic functions from four input variables if one variable is common between the two function generators F and G [1].

For any given design, there exists two representations for FPGAs. Figures 1.1 and 1.2 show the logic-level and physical-level representations of a sample design. As shown, most of the delays in the logic-level representation are not needed because in the physical layout many of the gate delays are lumped together. In the physical-level layout, the delays outside the logic modules are the critical delays that need to be analyzed. More detailed information about these types of examples will be presented in later chapters.

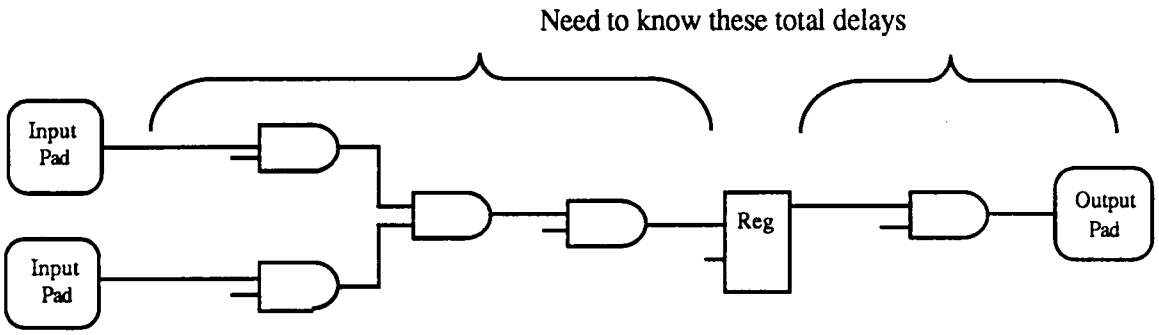


Figure 1.1 Logic-level Representation for a Sample Design.

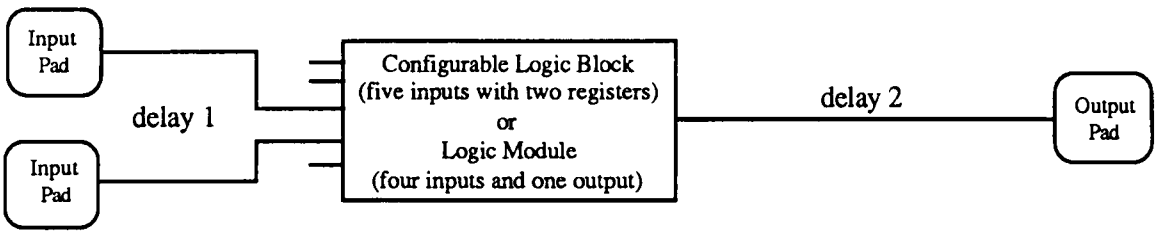


Figure 1.2 Physical-level Representation for a Sample Design.

The Problem

In particular, the timing characteristics of Xilinx Logic Cell Arrays (LCAs) are determined primarily by the routing delays that are introduced from the Automatic Placement and Routing (APR) for the design. The majority of the delay is incurred from APR because Xilinx LCAs use a programmable interconnect which consists of active devices [5]. These active devices (transistors or pass gates) coupled with the relatively large impedance of the LCA metal lines means that "no accurate estimate of timing can be made prior to APR" [5]. This problem is just the opposite of the typical timing problem encountered in most ASIC designs. In a typical ASIC, the majority of the typical timing delays are accounted for by the intrinsic delays of the cells and by predictable lumped delays on the nets. Accordingly, ASIC timing analysis can be performed with relatively precise results prior to layout by simply summing intrinsic delays and lumping net capacitance delays within data paths [5].

Xilinx LCA designs are difficult to back-annotate because of two reasons. The first is because many nodes are absorbed into the CLB look-up tables such that most nodes have zero delay and only the CLB output has a real delay [5]. This is a problem because there is no specific node on which to place the CLB output delay back on the original design. An appropriate place would be at a node which contains a flip-flop. The second reason that back-annotation is difficult is because most of the LCA timing is expressed as "skew delay" (distortion) on the nets. A given net "X" may have an intrinsic delay of 7.5 nanoseconds (ns) from the cell, while the routing delay along segment A to one input is one value (e.g. 5 ns) and the delay along segment B is a different value (e.g. 12 ns). This is exemplified in Figures 1.3 and 1.4. The delay for the net "X" cannot be added as a delay to the cell driving nodes. The Xilinx library should allow for input delay specification to account for skew on such nets.

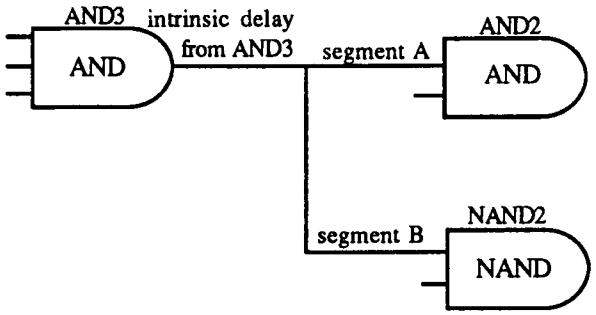


Figure 1.3. Unit-delay Logic Description

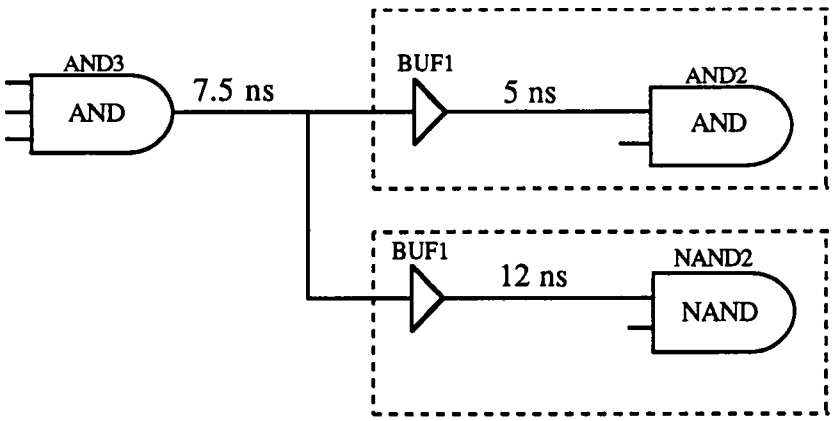


Figure 1.4. Back-annotated Full-Timing Description

This thesis is divided into six chapters, beginning with the introduction. Chapter II outlined FPGAs in general and briefly covers the Xilinx, Actel, Altera, and the Plus Logic FPGA products. This chapter also shows why simulation is necessary for designing FPGAs. The third chapter presents information on the Xilinx 3000 Series with detailed information on CLBs and IOBs.

In Chapter IV, the development of the PATHFINDER program is discussed. Included in this chapter are the concept and design goals of the PATHFINDER as well as the algorithm for the PATHFINDER. Also, the destination and source nodes of delay paths are explained. The fifth chapter analyzes the results for three test cases in which the PATHFINDER program was applied.

Chapter VI summarizes the automated procedure for improved timing analysis of FPGAs. This chapter also presents a comparison of the BAX program now offered by Xilinx and the unique contributions of the PATHFINDER program. The Appendix contains the PATHFINDER program with the test case .XNF and .LCA files.

CHAPTER II

FIELD-PROGRAMMABLE GATE ARRAYS

A Field-Programmable Gate Array is an integrated circuit (IC) which can be programmed by the user using electronic signals. These ICs are fabricated with the connections between transistors "missing"[4]. After entering a schematic and simulating the design, the designer executes specific software that maps the connections of the schematic into a string of bits, called a configuration file [4]. Then, this file is downloaded to the FPGA to "program" the connections of the IC. FPGA technology has made a significant impact on the design of digital logic ICs because the user can control the interconnection of a significant number of logic gates (up to 9000 in early 1992).

FPGAs have been used in many applications and the number of FPGA users is growing rapidly. FPGAs combine the logic integration benefits of custom Very Large Scale Integration (VLSI) circuits with the design, production, and time-to-market benefits of a user-programmable device. Logic functions previously performed by multiple TTL and programmable logic devices (PLDs) can be integrated into a single FPGA device while avoiding high cost, high risk, and long turnaround times for manufacturing that are normally associated with custom ASIC designs. FPGAs can hold approximately 1000 to 9000 two-input gates.

FPGAs are particularly appropriate for those applications between traditional simple and inexpensive TTL-based designs and modern, complex but expensive ASICs [4]. As shown in Figure 2.1, the FPGA provides the logic designer the best of both worlds - the flexibility, performance, and high integration levels of mask gate arrays, with the convenience and simplicity of PLDs.

The logic building blocks in FPGAs are programmable and can be configured to perform a wider variety of logic and storage functions than that is possible using PLDs. In addition, FPGAs, which are fabricated in CMOS technology, offer lower power consumption and more I/O resources than most PLDs which are generally manufactured

using bipolar technology. There are some FPGAs available that have an abundance of flip-flops that can be used to pipeline data for a design. The products offered by Actel, Xilinx, Altera and Plus Logic are discussed briefly at the end of this chapter.

Actel and Xilinx are solely FPGA companies, whereas Altera and Plus Logic manufacture both PLDs and FPGAs. Xilinx is one of the first companies to support the programmable array concept in the early mid-80's. Actel is a more recent start-up company, and Altera and Plus Logic are companies which started in the area of programmable logic devices and have since moved towards the FPGA technology.

Xilinx

The information that will be presented for the Xilinx family of devices will cover the XC3000 series. The programming technology used by Xilinx contains a static RAM cell constructed from two cross-coupled inverters [1]. This allows Xilinx to use a standard CMOS process with no modifications, which is an advantage. Xilinx currently uses a 1.2-micron CMOS technology for their XC3000 series, although the 0.8-micron technology has been announced for the XC4000 series [2]. The advantage of using static RAM is that the user can reuse the chips many times, and even reprogram the IC while it is in the system. Xilinx has introduced its third generation of products (XC4000 series), but is still shipping the second generation, XC3000 series.

The Xilinx XC3000 logic cell, labeled the Configurable Logic Block, contains both combinatorial logic and flip-flops. There are five different versions of the XC3000 series with between 64 and 320 CLBs on each chip. The CLBs have five logic inputs, an asynchronous direct reset input, an enable clock, and two logic outputs from the two internal flip-flops. The Xilinx CLB uses a 32-bit look-up table with coefficients stored in the static RAM to handle the combinatorial logic of the CLBs.

The Xilinx Input/Output Block (IOB) can have the input buffer portion programmed to be compatible with either TTL or CMOS levels. Configuration program bits for each IOB control features of an optional output register and provide tri-state and slew-rate control of the output.

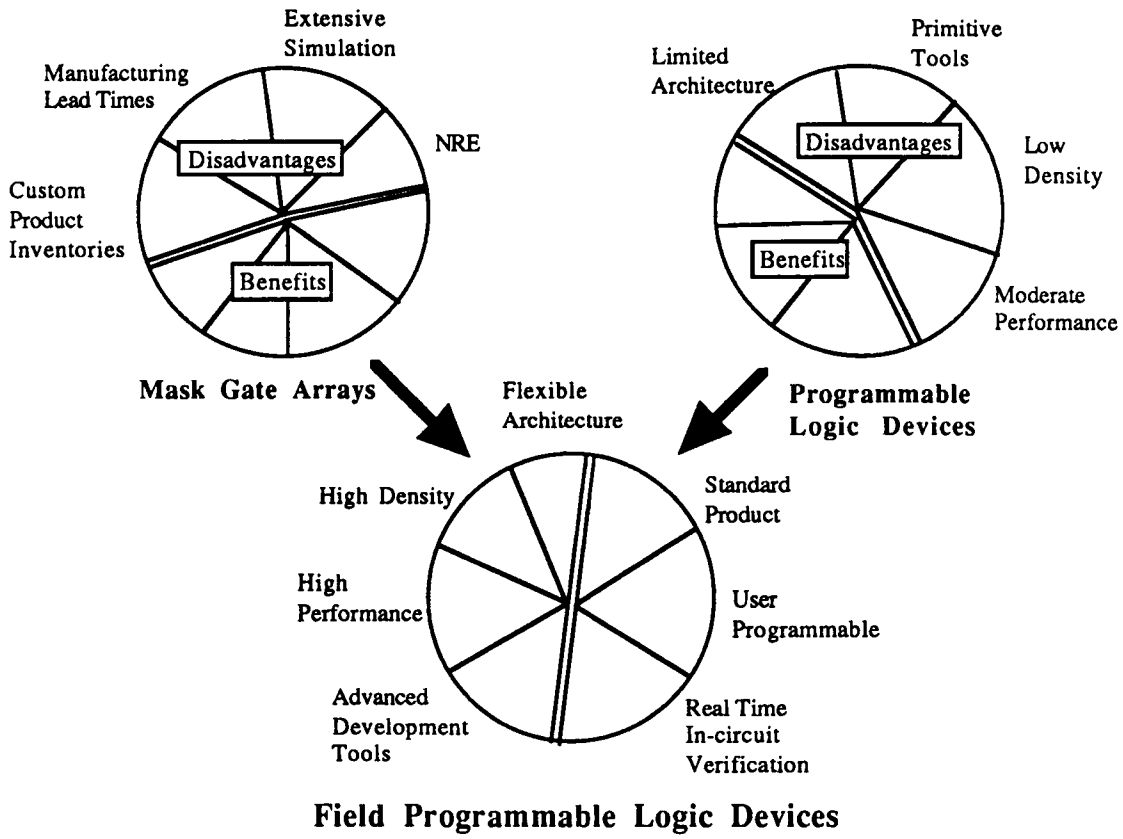


Figure 2.1 FPGAs - The Best of Both Worlds

The interconnect resources for the Xilinx products consists of five vertical and five horizontal lines that run in a matrix to "switch boxes" between the rows and columns of the CLBs. These "switch boxes" connect the CLBs using pass transistors at specific points on the chip, known as Programmable Interconnection Points (PIPs).

Actel

Actel has announced its second generation products (ACT 2), but the information presented will discuss the ACT 1 architecture. The Actel programming technology uses an "anti-fuse," which is the opposite of a regular fuse -- it is normally an open-circuit until a large current is passed through it [4]. Once the fuse is blown, it is permanent. The Actel 1010/1020 are the first production devices that use a 2-micron process, while the 1010A/1020A use a 1.2-micron process. The Actel ACT 1 logic cell consists of three two-input MUXes and an OR gate. Actel does not reveal how the cell is actually implemented. The way the logic cell is set up, each cell can become an equivalent gate that ranges from a simple 2-input AND gate to a complex 4-input NOR gate depending on its inputs and control lines. The Actel modules are highly symmetric, which simplifies the placement and routing.

The Actel 1010 contains 352 modules: 295 logic and 57 I/O modules in 8 rows by 44 columns The Actel 1020 contains 616 modules: 547 logic and 69 I/O modules in 14 rows by 44 columns. The Actel I/O module implements a bidirectional and tri-state I/O [4]. Placed around the edge of the die are I/O modules. Some of these I/O cells are reserved to interface with the core of the array [4]. The Actel interconnect uses segmented routing for interconnecting the logic cells, and has a variable interconnect delay.

Altera and Plus Logic

Recently, Altera announced their newest family of devices (EPM7000 series) that places their product line into the FPGA arena. Plus Logic is a 1988 start-up company which uses an architecture similar to that of Altera, but also tries to resemble that of Xilinx. The first product of Plus Logic is the FPGA2020.

Altera and Plus Logic both use Ultra-Violet-erasable EPROM (Electrically Programmable Read-Only Memory) cells [4]. Altera's EPROM cell uses 0.8-micron CMOS technology. The EPROM cell for Plus Logic is similar to that of Altera's. Plus Logic's EPROM cell uses 1.5-micron CMOS technology. For programming the EPROM cells, a high voltage (above 12V) must be applied to the drain of the n-channel EPROM transistor. This causes electrons under the gate moving towards the drain to move so fast they "jump" to the floating gate [4]. Once programmed, the n-channel EPROM devices will remain "off" even with a +5V applied to the top gate. This way, any of the unprogrammed n-channel devices will turn ON with a gate voltage of +5V. The EPROM cells can be erased by applying an ultra-violet light. This application of UV light gives the stuck electrons enough energy to "jump" back out of the gate.

Altera and Plus Logic are very similar in that they both have fixed pre-layout delays. Also, the pre-layout timing is equal to the post-layout timing. Therefore, most of the simulation can be done before the layout since the timing changes only if a second logic module must be utilized. Because the fixed delays are usually longer for each cell in comparison to Xilinx or Actel, Altera has fabricated their FPGAs using an intrinsically faster process (with smaller feature size) than the other companies in order to be competitive.

The Altera logic cell, called a macrocell, is more complex in size than the Xilinx CLB but is simpler in architecture [4]. The Altera macrocell is derived from PLA (programmable AND-OR) and PAL (programmable AND, fixed OR) architectures [4]. The Plus Logic cell is referred to as a Functional Block (FB). Each FB contains an array of programmable logic.

The Altera I/O Block uses a tri-state driver that is controlled by a dedicated enable signal: a macrocell product term [4]. This I/O Block for the Plus Logic FPGA also contains tri-state drivers with an enable from the FBs. The FPGA2020 contains 36 I/O Blocks (12 Input Blocks, 22 Output Blocks, and 2 CLOCK/Output blocks). The Altera devices use a Programmable Interconnect Array (PIA) to control many basic logic array blocks (LABs). The PIA is a crosspoint switch for logic signals traveling between LABs [4]. Plus Logic also uses a crosspoint bus to distribute signals within its FPGA [4]. The

fixed bus architecture, like Altera's PIA, provides an advantage of fixed routing delays and improved speed of routing.

Rationale for Simulation

Because the pre-layout and post-layout timing delays are equal for Altera and Plus Logic, timing analysis can be achieved with few complications. On the other hand, for Xilinx and Actel, the assumed timing usually changes after placement and routing. This is where the PATHFINDER program can be used. The PATHFINDER can improve timing analysis after layout for those FPGAs which do not have constant routing delays.

Thus, the design methodology consists of the following steps. The designer performs functional simulation after the design has been entered to verify that the circuit logic is correct before partitioning, placing, or routing the design [6]. Once the design has been placed and routed, a detailed timing simulation is performed to assess critical paths in the design. Thus, the purpose of the PATHFINDER is to aid in full-timing simulation.

The following chapter will discuss in detail the Xilinx XC3000 Series because this the IC for which the PATHFINDER program was designed. The Xilinx software generates a report (RPT) file, but it does not contain any information regarding summing of delay paths. Consequently, the PATHFINDER program can be used to collect this information in a specified format that enhances the timing analysis which the designer must perform.

CHAPTER III

THE XILINX 3000 SERIES

Overview of the Architecture

Xilinx's proprietary Logic Cell Array architecture is similar to that of other gate arrays, with an interior matrix of logic blocks and a surrounding ring of I/O interface blocks [1]. Interconnect resources occupy the channels between the rows and columns of logic blocks, and between the logic blocks and I/O blocks. Similar to a microprocessor, the Logic Cell Array (LCA) is a program-driven logic device. The functions of the LCA's Configurable Logic Blocks (CLBs), I/O blocks and their interconnections are controlled by a configuration program stored in an on-chip memory [1]. An overhead view of the top left section of the Xilinx 3000 Series chip can be seen in Figure 3.1.

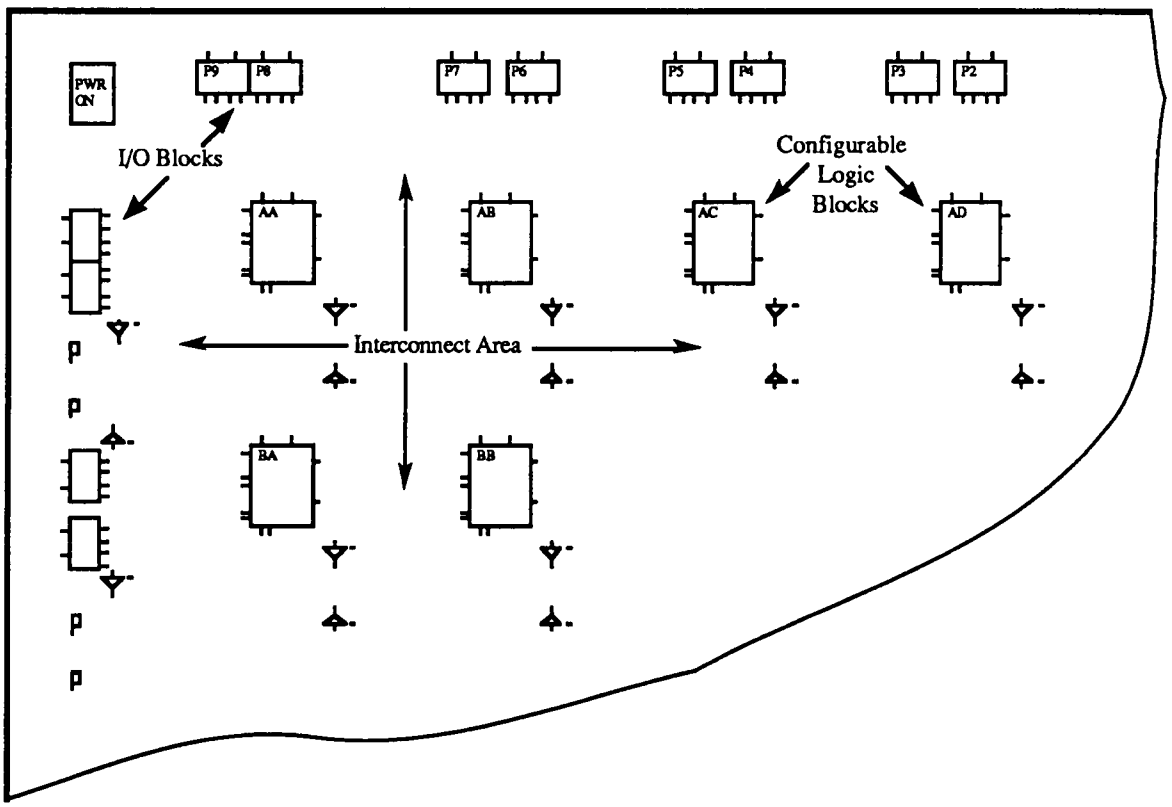


Figure 3.1 Structure of the Logic Cell Array of a Xilinx 3000 Series FPGA

The PATHFINDER program used the Xilinx 3042 FPGA for timing analysis. The Xilinx 3042 FPGA is a 12 x 12 matrix which has 4200 gates available for use, 288 combinatorial logic functions (144 CLBs), 480 flip-flops, and 96 Input/Output Buffers (IOBs). The configuration program is loaded automatically from an external memory on power-up.

The core of the LCA is a matrix of identical CLBs. For typical applications, system clock rates are one-third to one-half the maximum flip-flop toggle rate.

Configurable Logic Block

The array of CLBs provides the functional elements from which the user's logic is constructed [1]. The logic blocks are arranged in a matrix within the perimeter of I/O Blocks. The blocks' logic functions are implemented by programmable look-up tables. The XC3042 has 144 such blocks arranged in 12 rows and 12 columns. The user can define the CLBs and their interconnecting nets by an automatic translation from a schematic capture logic diagram.

Each configurable logic block has a combinatorial logic section, two flip-flops and an internal control section as shown in Figure 3.2 [1]. The inputs include: five logic inputs [.a, .b, .c, .d, .e], a common clock input [.k], an asynchronous direct reset input [.rd] and an enable clock [.ec]. All of the inputs can be driven from the interconnect resources adjacent to the blocks. Each CLB also has two outputs [.x and .y] which may drive interconnect networks [1].

In the combinatorial logic section of the Configurable Logic Block, a 32-by-1 look-up table is used to implement Boolean functions. This section takes any combination of five inputs and produces two outputs using the Boolean functions. This section can also produce two independent logic functions from four input variables if one variable is common between the two function generators F and G.

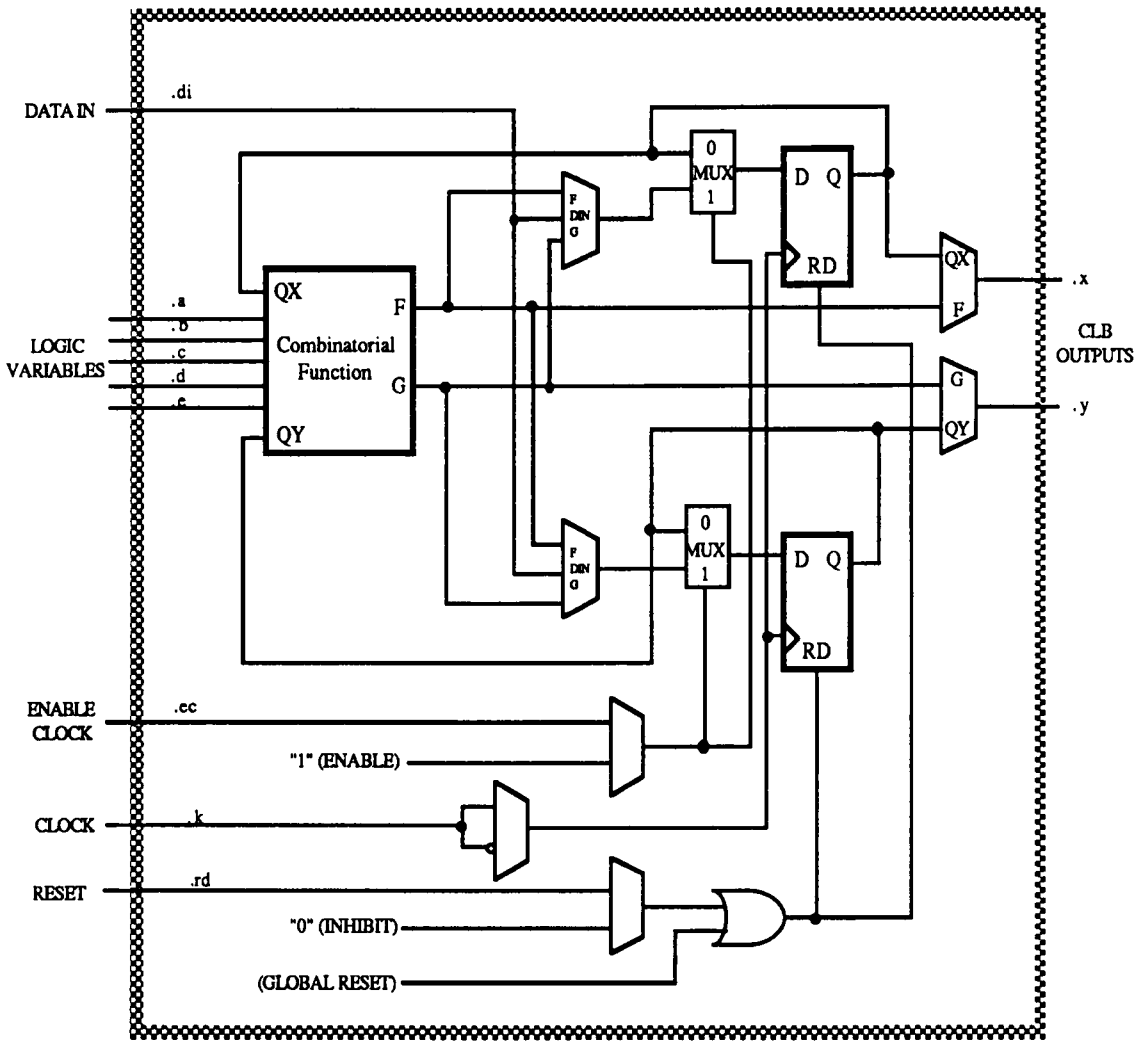


Figure 3.2 Configurable Logic Block

There are also two flip-flops in the CLB which share the enable clock [.ec.] which, when LOW, refreshes the flip-flops' present states. The user may enable the control inputs and select their sources [1]. The CLB uses a programmable inversion which eliminates the need to route both phases of the clock signal throughout the device [1].

Input/Output Block

There are 96 Input/Output Blocks (IOBs) in the XC3042, which provide interfacing between the internal user logic and the external package pins. The IOB includes input and output storage elements and input/output options selected by configuration memory cells as shown in Figure 3.3. These blocks are used to input and output signals from the Xilinx chip. Each IOB includes both registered and direct input paths [1]. There is also a programmable three-state output buffer that can be driven by a registered or direct output signal. The input circuits also have input clamping diodes to prevent electrostatic discharge.

The input buffer portion of each Input/Output Block provides threshold detection to translate external signals applied to the package pin to internal logic levels [1]. This global input-buffer threshold can be programmed to be compatible with either CMOS or TTL logic levels. The sense of the clock can be inverted as long as all IOBs on the same clock net use the same clock sense [1]. The clock signals [.ik and .ok] may be chosen from either of the two metal lines in the die. Each Input/Output Block may have interconnect to the direct input [.i] or the registered input [.q].

Output buffers of the IOBs provide CMOS-compatible 4 mA (milliamps) source-or-sink drive for high fan-out CMOS- or TTL-compatible signal levels [1]. The registered or direct data source for the output buffer is controlled by the OUT pin [.o] of the Input/Output Block. The output activity is controlled by the three-state control signal [.t]. Flip-flop loop delays for the IOB and logic block flip-flops are about 3 nanoseconds [1].

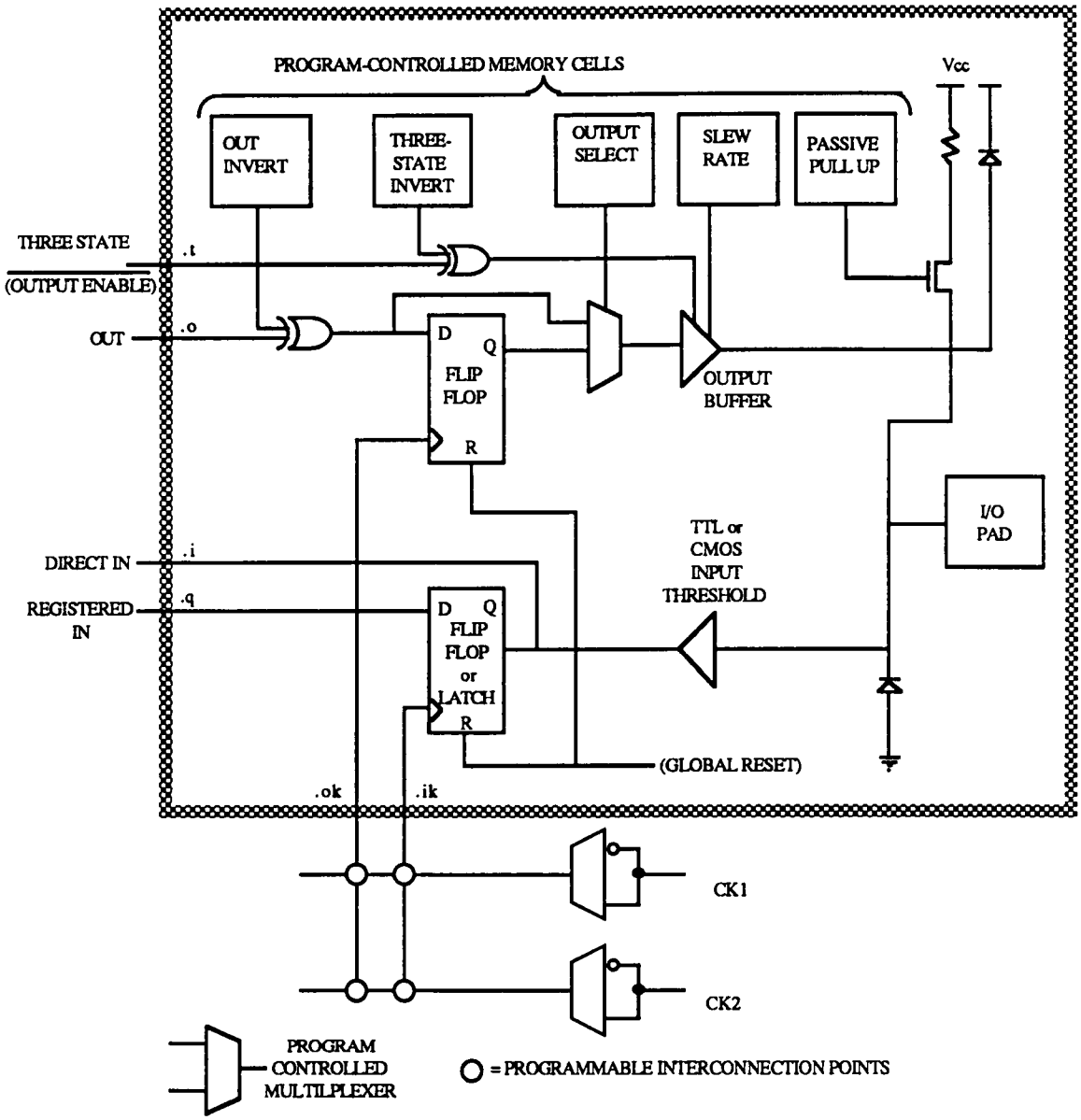


Figure 3.3 Input/Output Block

CHAPTER IV

THE DEVELOPMENT OF THE PATHFINDER PROGRAM

Concept and Design Goals of the PATHFINDER

The design entry and XNF translation steps can be observed in Figure 4.1. These steps outline where a design begins and where the PATHFINDER program takes effect. The PATHFINDER program analyzes the timing characteristics from the .xnf file which contains all the routing information and design logic after the place and routing has been completed. A speed grade of -100 is used when the place and routing is done.

A goal for the PATHFINDER was to analyze all possible signal delay paths and determine the maximum clock speed for which the Xilinx FPGA can run. To accomplish this, several types of signal paths must be found. These signal paths could begin at any of these blocks: INFF, IBUF, or the input of a D flip-flop. Then the signal paths propagate through Configurable Logic Blocks (CLBs), until they eventually come to an ending node. These ending nodes include: OUTFF, OBUF, and DX or DY into flip-flops. From the entire set of signal paths encountered, PATHFINDER must determine the longest delay path and then calculate the maximum system clock speed for the IC.

Algorithm for the PATHFINDER

A stand-alone static-timing analysis program was created to determine timing performance for a logic configurable array on the delays provided in the routed Xilinx Netlist Format (.xnf) files. The Xilinx 3042 FPGA is the integrated circuit that was used for performing the timing analysis. In the beginning of the PATHFINDER program the structures, arrays, and all of the variables are defined. The next step is to set all the initial values. After all the initial values are set, the PATHFINDER requests the name of the routed .xnf file, the routed Logic Cell Array (.lca) file, and the name of the data output file. An initial high-level flowchart is given in Figure 4.2. The final structured flowchart of the algorithm for the PATHFINDER is presented in Figure 4.3. This algorithm represents the heart and soul of the PATHFINDER program.

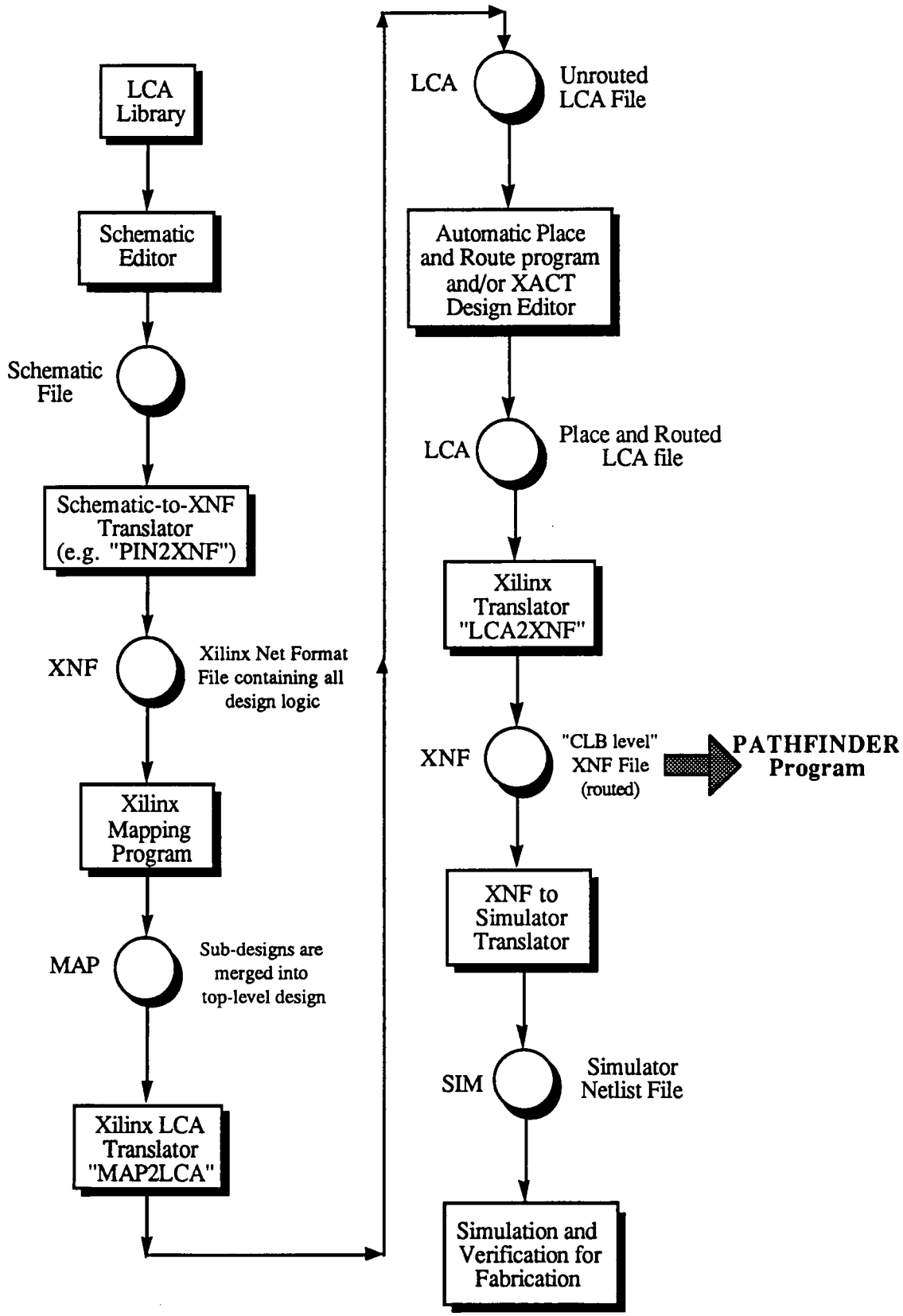


Figure 4.1 Design Entry and XNF Translation

Once execution of the initial routine is completed, the PATHFINDER opens the .xnf file and begins searching for destination nodes. The first type of destination node includes internal D flip-flops. After a destination node is found, the delay path is examined and analyzed. The program will continue to search for "DFF" until the .xnf file pointer reaches an end-of-file (EOF) position. When the current .xnf file pointer reaches the EOF position, the PATHFINDER will then reuse the same file pointer to begin another search for the second type of destination nodes: Xilinx output blocks.

If a D flip-flop is located, its block name will be stored for reference. Next, the PATHFINDER will determine whether the signal flow is through the F or G generator, and if it is an X or Y internal flip-flop. The generators are portions of the CLB which contain a portion of the combinational logic of the original design.

After the destination nodes have been located, the PATHFINDER will begin searching for possible source nodes. This is where the PATHFINDER actually does its most work. The PATHFINDER will trace back from here looking for specific source nodes. In many cases, the final source node will not be found. Instead, a temporary source node will be found. These temporary nodes, when traced back, will not lead to any internal flip-flops or input buffers but instead to F or G generators which then have more inputs to trace. The extra inputs obviously increase the number of possible delay paths. The PATHFINDER will continue searching all of the source nodes until all delay paths have been traced back to a final source node. After any source node is found, the delay associated with that source is compared to the longest possible path. By calculating the longest possible path, the maximum clock speed for the FPGA can be determined. The routine that is used in the PATHFINDER to locate the source nodes is also used to locate both types of destination possibilities (D flip-flops and Xilinx outputs blocks).

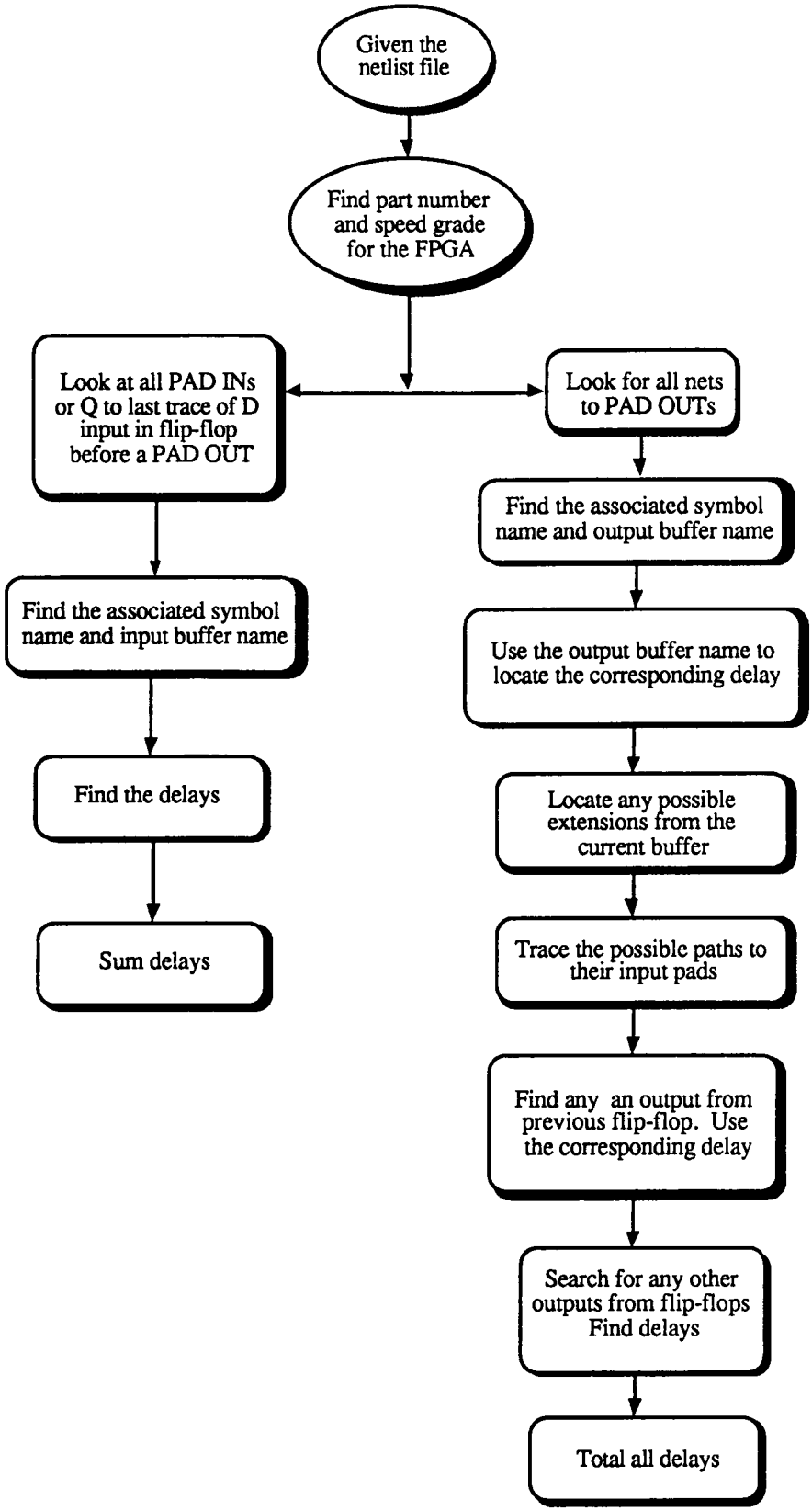


Figure 4.2 Initial Flowchart for the PATHFINDER Algorithm.

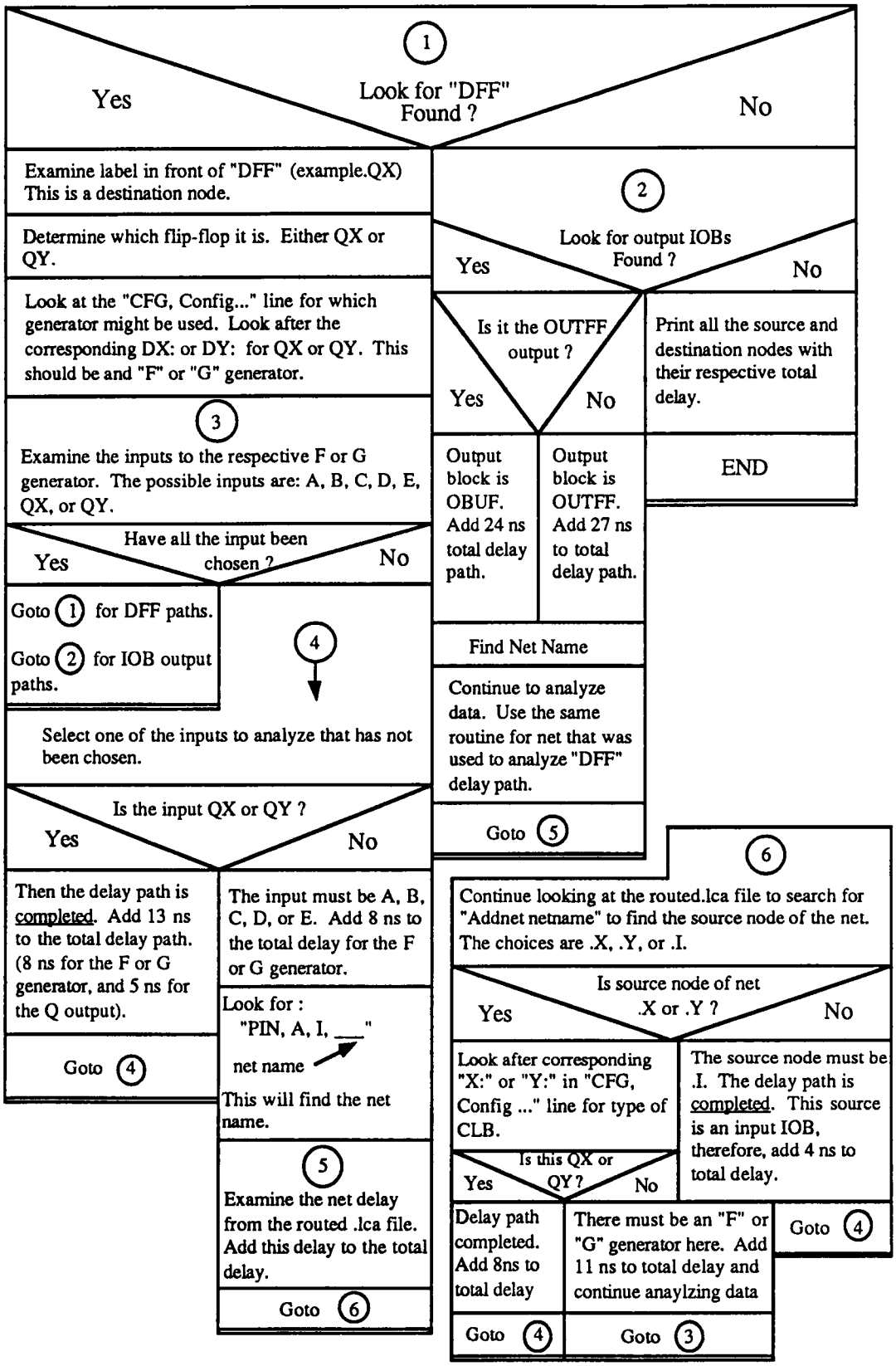


Figure 4.3 Structured Flowchart of Algorithm for PATHFINDER

Table 4.1 shows all the types of maximum delays that may occur in a Xilinx 3000 Series FPGA running at a speed grade of -100. These delays are the maximum of those encountered for each segment since the purpose of the PATHFINDER is to find the longest delay path. These delays are also displayed in Figure 4.4. Each delay will be explained throughout this chapter. The next two sections will discuss the types and characteristics of destination nodes and source nodes. Listed below is an overview of the possible types of delay paths that can occur.

1. Input Pad to D-input of flip-flop.
2. Input Pad to Output Pad.
3. Q output from flip-flop to D-input of flip-flop.
4. Q output from flip-flop to Output Pad.

The maximum propagation delays in the above table were established using the IOB and CLB switching characteristic guidelines found in the Xilinx Programmable Gate Array Data Book [1]. These delays will be referred to as 'standard' delays for future references. Some of these 'standard' delays were calculated from data test files using sample designs. Note that these are not fixed delays, but they are the maximum delays that can occur for such possible paths. The actual delays may vary in the design, but the purpose of the PATHFINDER program is to find the longest (i.e. maximum) delay path.

Signal Propagation Delay Paths	Delay (ns)	Segment
'IBUF' input in IOB	4	A
'INFF' input in IOB	17	B
Propagation through CLB with no flip-flops	11	C
Output QX or QY from CLB flip-flop	8	D
Internal output QX or QY back to F or G generator	13	E
Propagation through just F or G generator	8	F
'OBUF' ouput from IOB	24	G
'OUTFF' output from IOB	27	H

Table 4.1 Maximum Propagation Delays of Possible Paths

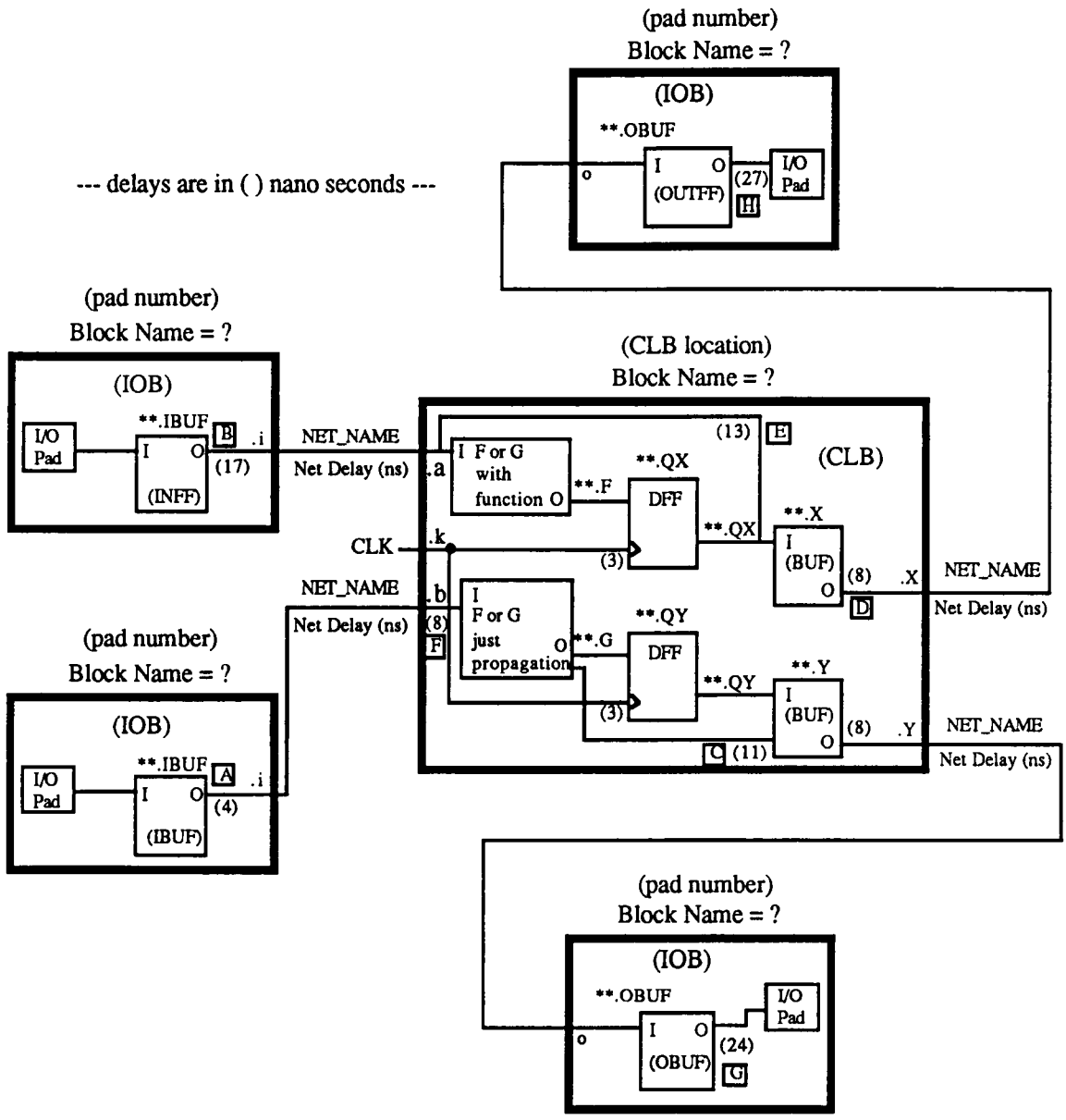


Figure 4.4 Block Diagram of All Types of Maximum Delays

Destination Nodes

A destination node is the ending point of a signal path but is the starting point for tracing a delay path. It is the point where a signal can no longer go further unless it is clocked to its next stage.

The first type of destination node is that which is associated with an internal flip-flop. This node is used as a reference point because the signal must be held internally by a flip-flop. The PATHFINDER first searches for this type of destination node ('DFF'). There is an 8 ns maximum delay assigned for this type of destination node that will be the initial delay for delay paths beginning here. Each internal flip-flop has a possibility of five inputs from which such delay paths may begin. As presented below, there are actually four types of flip-flop destination nodes possible.

1. Signal propagation through an F generator to an X flip-flop.
2. Signal propagation through an G generator to an X flip-flop.
3. Signal propagation through an F generator to an Y flip-flop.
4. Signal propagation through an G generator to an Y flip-flop.

These signal types are highlighted in Figure 4.5, which shows how the configurable logic block (CLB) is formed inside a Xilinx 3000 Series FPGA. Each of these types of signals also has a block name associated with it. The block name is used as a reference to locate this destination node. In the PATHFINDER, a second file pointer is used to pinpoint and locate the block names for these delays.

The second type of destination node is a signal path that ends at an output block. These output blocks are selected as part of the design requirements which restrict the speed of the Xilinx IC. There are two types of output blocks that will be analyzed: output buffers (OBUF) and output flip-flops (OUTFF). Both of these types of destination nodes can be seen in Figures 4.4 and 4.6 of the Input/Output Block. The OBUFs have no flip-flops. Therefore, the signal passing through an OBUF just simply goes external to the IC without encountering a stopping point. Because an OBUF node goes external to the IC, it is considered to be a destination node. With this in mind, the signal path may be continued on another Xilinx FPGA. The maximum delay for an OBUF is 24 ns. A flip-flop is

contained in an OUTFF output. Therefore, this is an obvious destination node and also a "stopping" point. The maximum delay for an OUTFF is 27 nanoseconds.

Source Nodes

For the external input source nodes (I), there is only one possible type of straight input. This is the "IBUF" input buffer. The source nodes of IBUF can be seen on Figures 4.4 and 4.7 of the Input/Output Block. This is where the external input is being brought into the Xilinx 3000 Series FPGA through the IBUF. A maximum delay of 4 ns will be used to account for into the total delay of a signal path.

The next type of source node is one that contains a flip-flop. There are two kinds of flip-flops that may be used in the Xilinx 3000 Series FPGA. The first of which is an input flip-flop through the IOB. This type of flip-flop, 'INFF', brings in an external signal input into the Xilinx FPGA through a flip-flop. A maximum delay of 17 ns is calculated for this type of signal flow. The second type of flip-flop is an internal flip-flop located inside a CLB that has an output of QX or QY. One of the two possible internal flip-flops (either X or Y) is available for each specific CLB in the Xilinx 3000 Series FPGA. Each QX or QY output arrives from a corresponding X or Y flip-flop. These types of signal paths have a maximum delay of 8 ns.

The remaining type of source node in which the net (signal path) is coming from a CLB is one which has no flip-flops inside it. Therefore, the signal passing through the logic block goes through the F or G generator. These generators are used to implement one logic function out of many gates. This type of signal flow greatly increases the number of possibilities from which the signal path can be sourced. This occurs because the signal paths being analyzed are those related to flip-flops, of which this type of source has no flip-flops. Therefore, this type of source node will never be the final source node for a delay path. It can always be traced further back. These types of source paths have a maximum delay of 11 ns. The labels for these paths have .F or .G suffixes. The possible source names are labeled as:

***.IBUF, ***.INFF, ***.QX, ***.QY, ***.F, ***.G

Note : the *** is the block name from which the source is located.

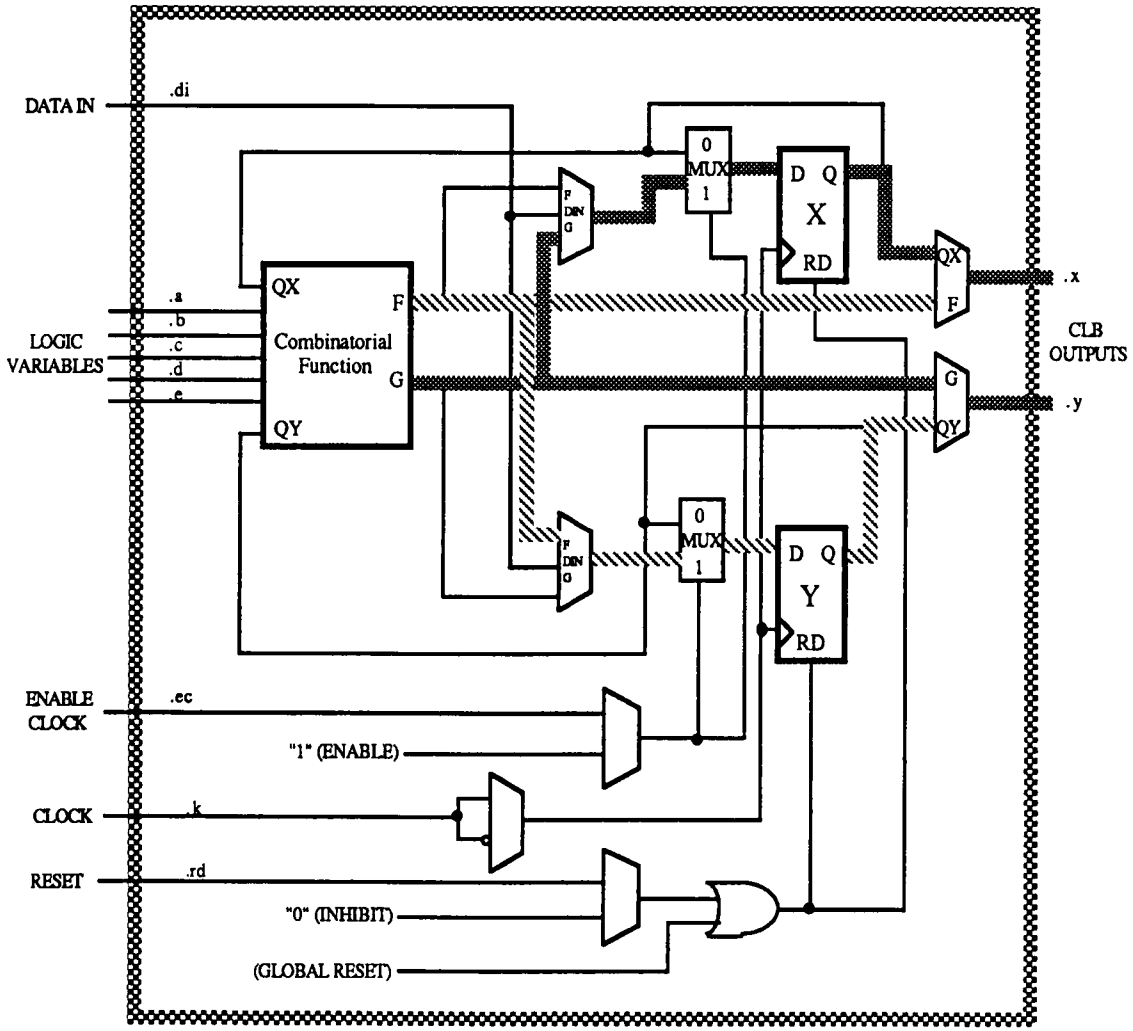


Figure 4.5 CLB with Highlighted Paths for F and G Generators and X and Y flip-flops

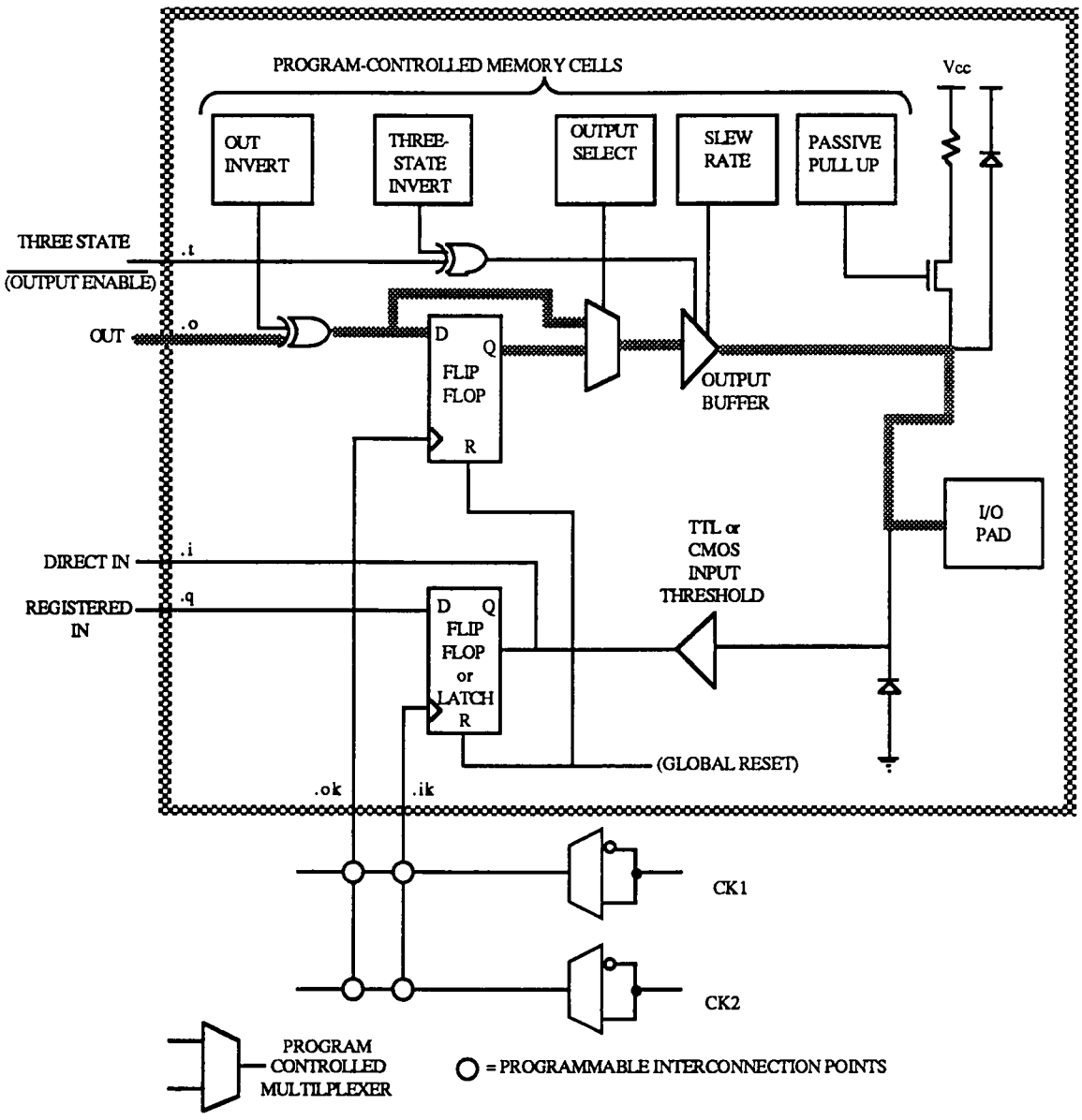


Figure 4.6 IOB with Highlighted Paths for OUTFF and OBUF

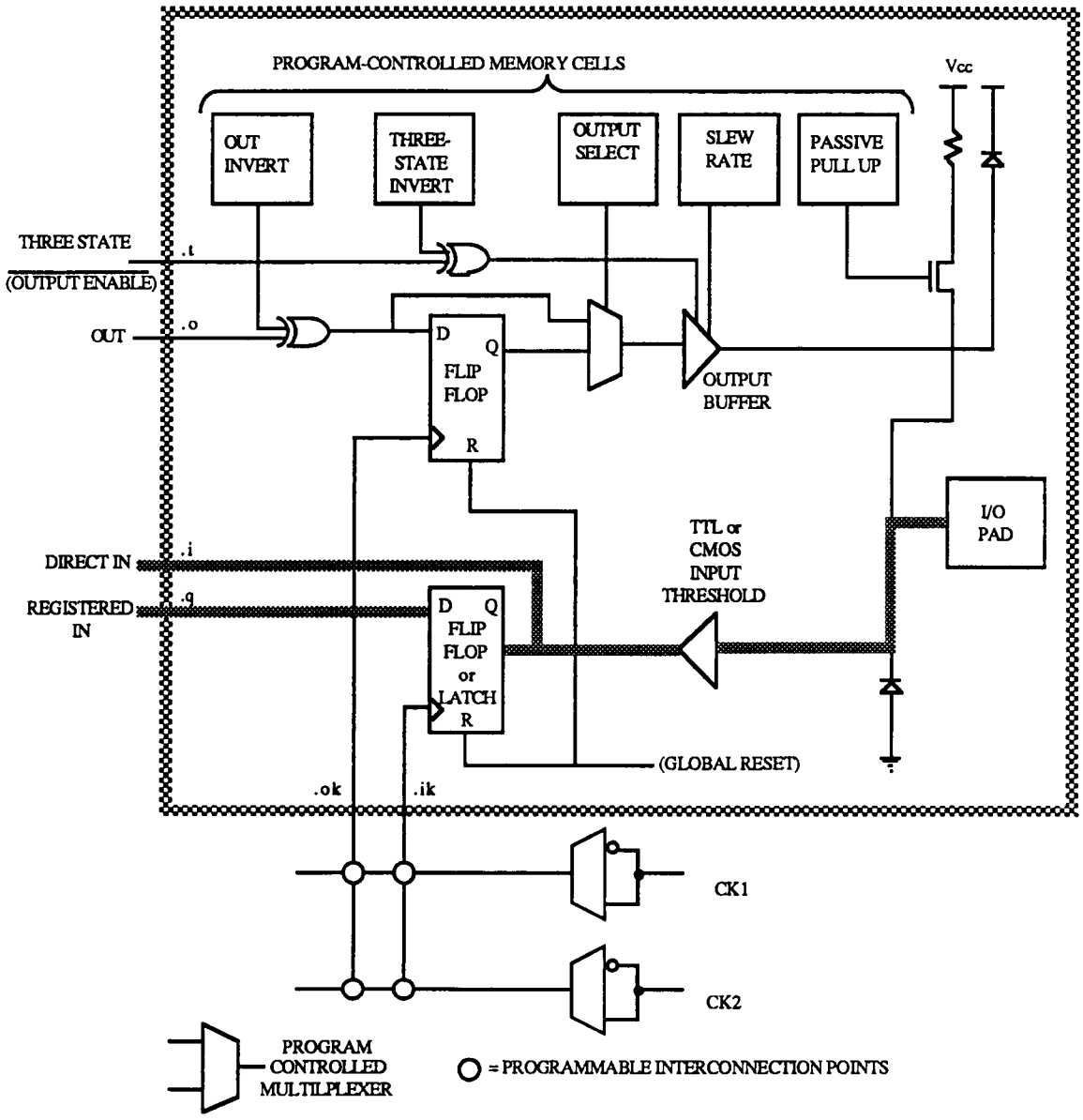


Figure 4.7 IOB with Highlighted Paths for INFF and IBUF

For the sources of IBUF, INFF, and QX or QY, the signal delay path has reached its final source node. That is, the beginning node of the delay path has been found. What is meant here is that for the original concept of the PATHFINDER, the purpose was to find all possible delay paths from flip-flop to flip-flop, or input to flip-flop, or flip-flop to output. Therefore, after these sources have been located, the source name is registered as the final source name, and a complete delay path has been found. The total delay path is now completed after locating any of these sources. After any one of these sources has been found, the next delay path can be analyzed. In the PATHFINDER, the path number would be incremented here and the array which contained the final path number, the source name, the destination name and the total path delay would be destroyed here. Thus, the the data storage can be recycled for future use. The purpose of the "dff" structure in the PATHFINDER program is not only to keep the data in an organized manner, but also to compare it to the longest path encountered at that time. If the current path being analyzed is the longest path, than the previous one (structure "longdff") is replaced by the current "dff" structure (source name, destination name, total delay, and path number).

To summarize all of the signal propagation delay paths, Table 4.2 is included again. These signal paths will also be more evident in the test cases discussed in the next chapter.

Signal Propagation Delay Paths	Delay (ns)	Segment
'IBUF' input in IOB	4	A
'INFF' input in IOB	17	B
Propagation through CLB with no flip-flops	11	C
Output QX or QY from CLB flip-flop	8	D
Internal output QX or QY back to F or G generator	13	E
Propagation through just F or G generator	8	F
'OBUF' ouput from IOB	24	G
'OUTFF' output from IOB	27	H

Table 4.2 Summary of the Signal Propagation Delay Path

CHAPTER V

RESULTS

In this chapter, three test cases will be discussed concerning the application of the PATHFINDER program. Note that many smaller scale test cases were created to establish the information needed for the PATHFINDER program, but only the three most relevant ones will be analyzed in this chapter.

Test Case #1 -- N5R

Test Case #1 is a sample design created by the author to simulate and examine the design verification methods of the Xilinx system. Figure 5.1 shows the circuit design, which was completed using the NET ED (Net Editor) program available from Mentor Graphics. The name given to this design was N5R. The circuitry for this design after placement and routing is located in files N5R.XNF and N5R.LCA. Both of these files are included in the APPENDIX.

As shown in Figure 5.1, this design includes five flip-flops, six exclusive-OR gates, and six input/output buffers, of which only one is used for output. The design for this circuit is simple in nature, but its purpose of its design was to make use of more than one CLB in the Xilinx FPGA. Originally, this design did not contain any flip-flops and after the placement and routing was completed, the design logic was all placed into one CLB. Therefore, with the addition of five flip-flops, the Xilinx 3000 Series part was required to use at least three CLBs since each one can only hold a maximum of two registers each.

This N5R design, after placement and routing, uses three CLBs and six IOBs in the Xilinx 3000 Series FPGA. Figures 5.2a and 5.2b show the block layout of how each CLB and IOB is used on the Xilinx 3042 FPGA for the N5R design. All the delays and labels are included in these figures.

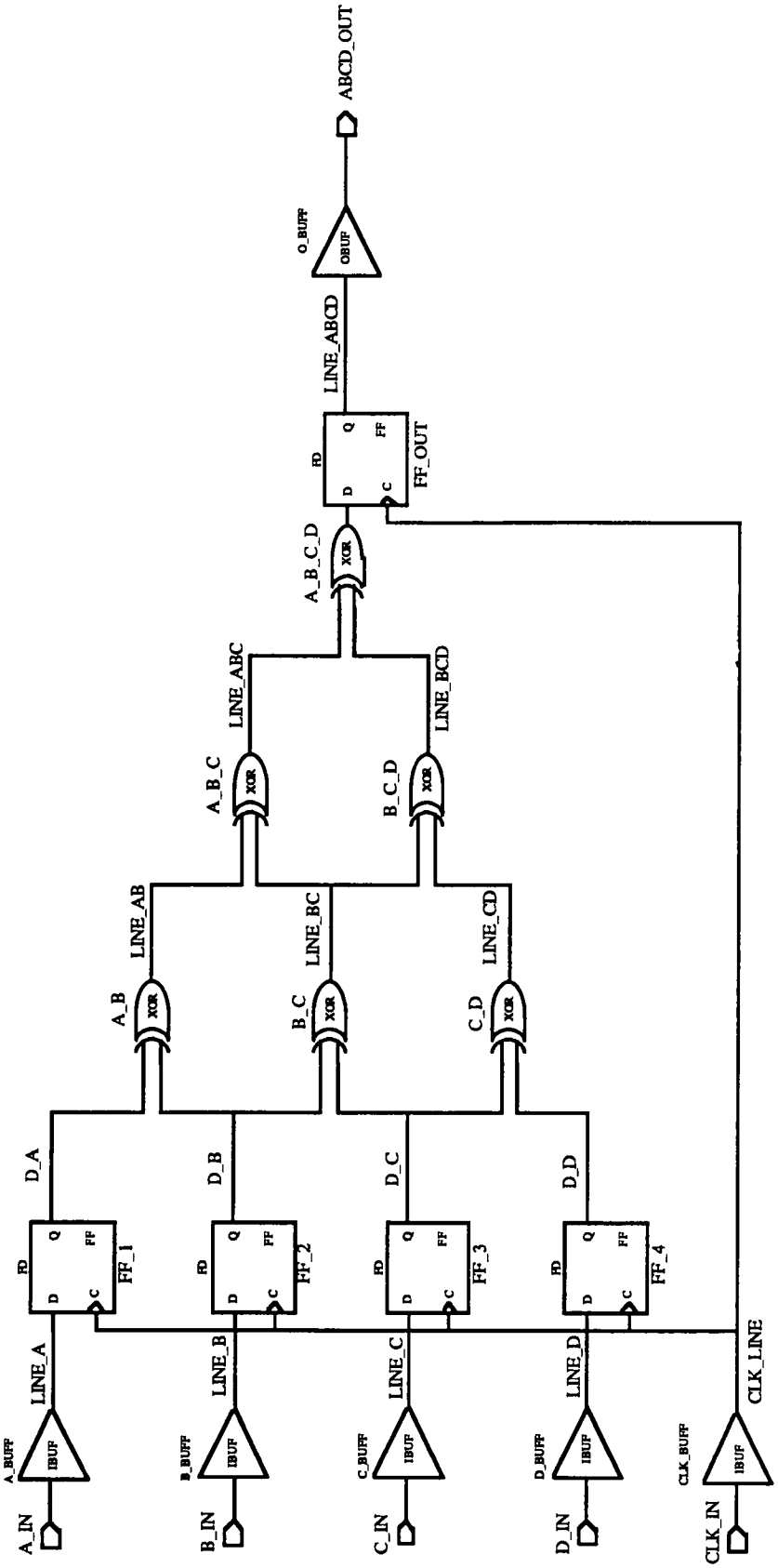
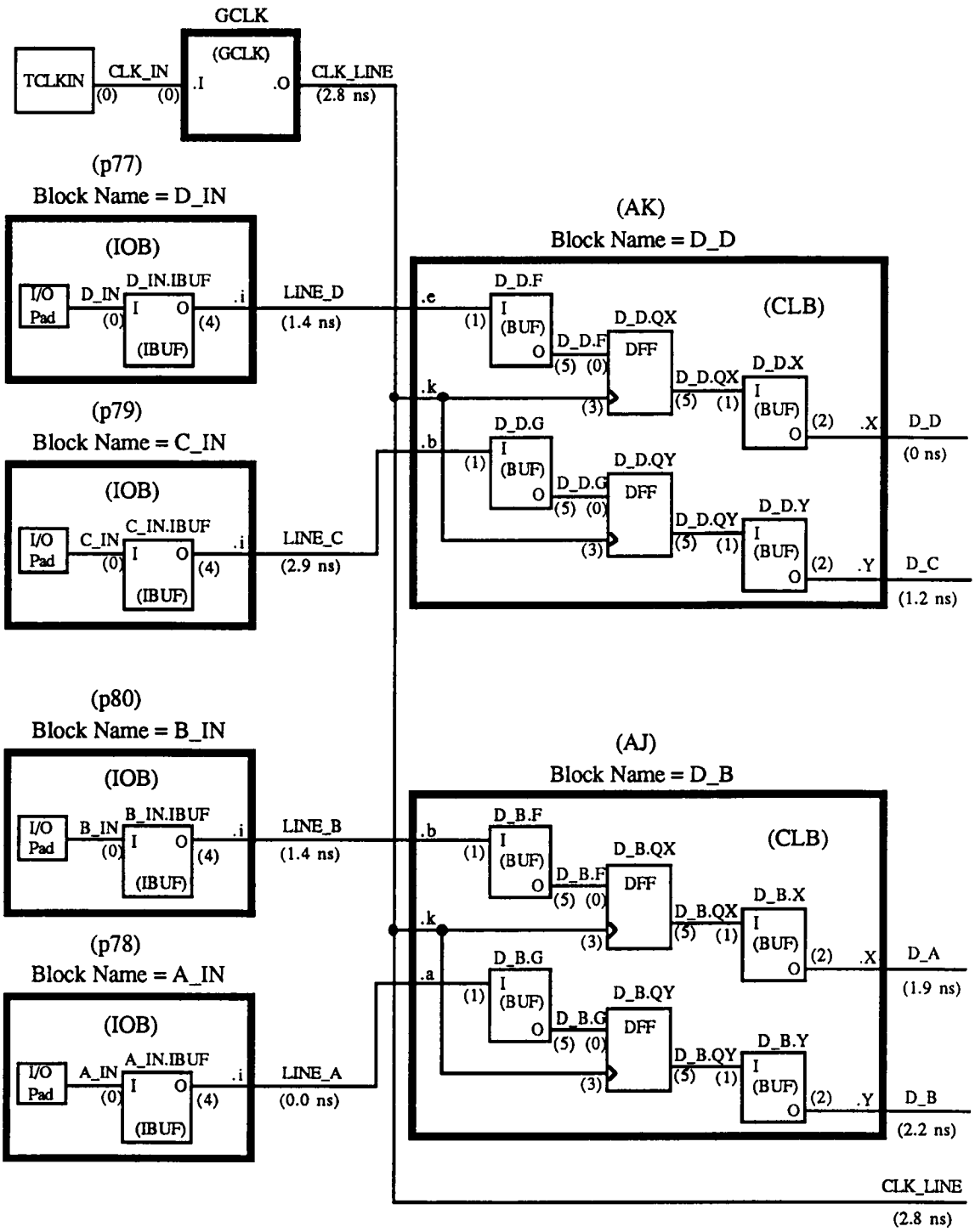
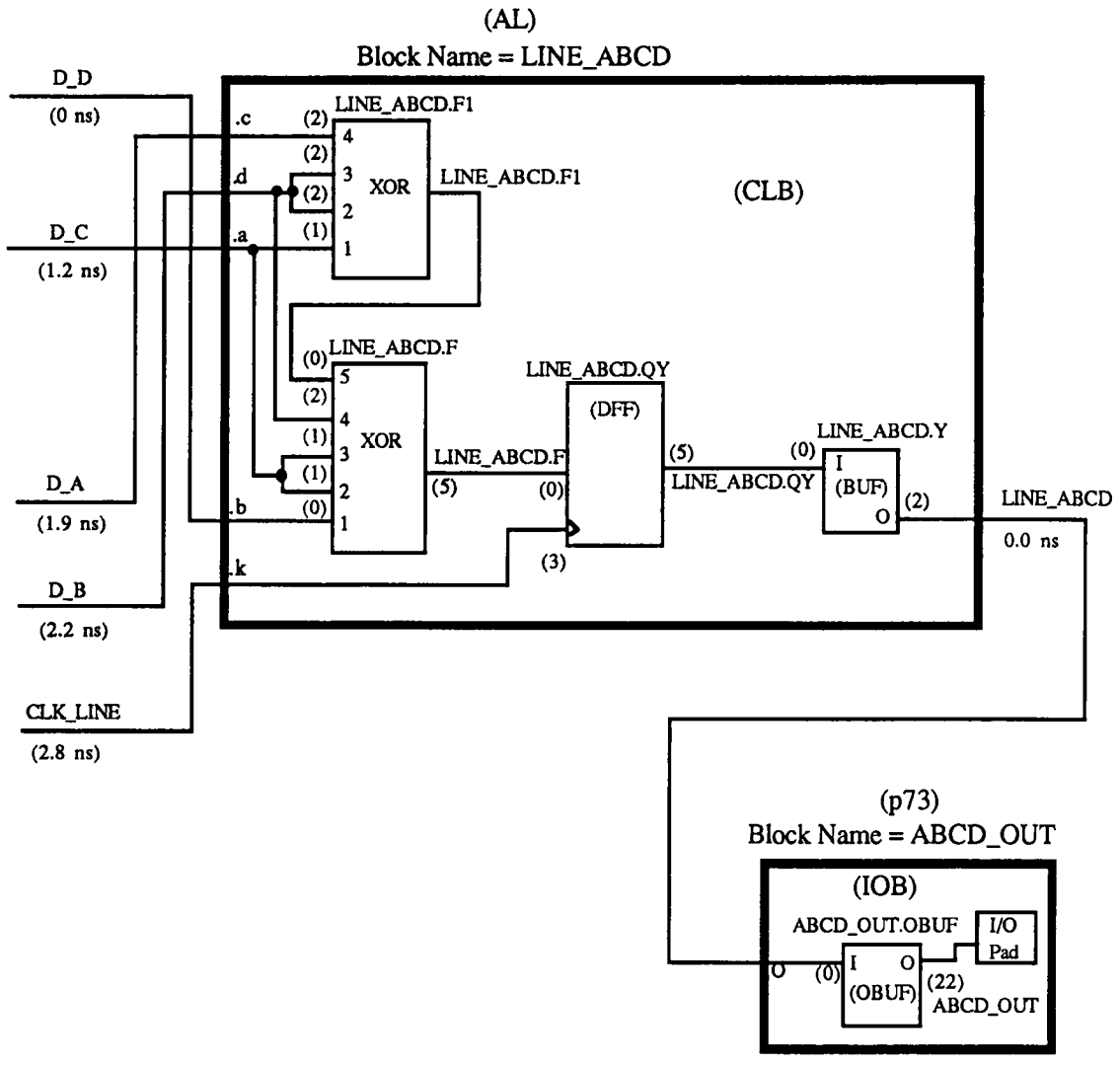


Figure 5.1 Test Case #1 logic design - N5R



a. Part 1

Figure 5.2 Block layout for N5R Routed -- Actual Delays (not maximum)



b. Part 2

Figure 5.2 (continued)

Results from Test Case #1

When a comparison is made between the original N5R logic design and the placed and routed version of the same design, it can be noticed that part of the logic from the original design is "missing." This is because some the logic in the original N5R design has been programmed into the function generators of the CLBs using Boolean functions resulting in a more efficient use of the resources inside the FPGA. An observation that should also be made is that every component (block) of the Xilinx 3000 Series FPGA is labeled automatically through the layout process by the Xilinx software.

The PATHFINDER program took approximately 2 minutes to execute using an IBM-compatible 386 - 20 MHz computer. Tables 5.1, 5.2 and 5.3 include the delay paths totaled using maximum delays. The order of the tables is respective to how the PATHFINDER program operates. The output data was placed into the N5R.DAT file.

Table 5.1 Delay Paths for DX or DY's Traced Back to QX or QY

		DX-->F.B NET X-->QX	
(1)	AL.DY from AK.QX	8 + 0 + 8	= 16.0 ns
		DX-->F.A NET Y-->QY	
(2)	AL.DY from AK.QY	8 + 1.2 + 8	= 17.2 ns
		DX-->F.B NET X-->QX	
(3)	AL.DY from AJ.QY	8 + 1.9 + 8	= 17.9 ns
		DX-->F.A NET Y-->QY	
(4)	AL.DY from AJ.QX	8 + 2.2 + 8	= 18.2 ns

(5)	AK.DX from P77.I	DX-->F.E NET P77 I -->PAD 8 + 1.4 + 4	= 13.4 ns
(6)	AK.DY from P79.I	DX-->G.B NET P79 I -->PAD 8 + 2.9 + 4	= 14.9 ns
(7)	AJ.DX from P78.I	DX-->F.A NET P78 I -->PAD 8 + 0.0 + 4	= 12.0 ns
(8)	AJ.DY from P80.I	DX-->G.B NET P80 I -->PAD 8 + 2.1 + 4	= 14.1 ns

Table 5.2 Delay Paths for DX or DY's traced back to IOBs (input)

(9)	P73.O from AL.QY	O PAD-->.0 NET Y --> QY 24 + 0.0 + 8	= 32.0 ns

Table 5.3 Delay Path for IOBs (output) traced back to QY or QX

By observing the above tables, the longest delay path can be found as the delay path of 32 ns for IOB P73 traced back from CLB AL.QY. The longest delay path in the output of the PATHFINDER program, N5R.DAT (given in APPENDIX B) matches this longest delay path of 32.0 ns. The remainder of the delay paths in the N5R.DAT also match the delay paths in the corresponding tables above. By performing a simple calculation, the maximum clock frequency can be found :

$$\text{max clock frequency} = \frac{1}{\text{longest delay path}} = \text{units of MHz}$$

$$\text{max clock frequency} = \frac{1}{\text{32.0 nanoseconds}} = 31.25 \text{ MHz}$$

Therefore, the maximum clock frequency for which the N5R design can run is 31.25 MHz.

Test Case #2 -- NEW2

Test Case #2 is a sample design created by the author to simulate and examine the design verification methods of the Xilinx system. This design differs from Test Case #1 in that it uses only one flip-flop and there is an addition of a Global Clock input. Figure 5.3 shows the circuit design, which was completed using the NET ED (Net Editor) program available for the Mentor Graphics workstation. The name given to this design was NEW2. The circuitry for this design, after placement and routing, is located in files NEW2.XNF and NEW2.LCA. Both of these files are included in the APPENDIX.

As shown in Figure 5.3, this design includes only one flip-flop, six NOR gates and eight input/output buffers. Of the eight IOBs, one of them is used for output, and one for a global clock input. The purpose of this design, NEW2, was to include different types of delays from those tested in the N5R design. Specifically, this design contains a path in which a signal to flow through only an F or G generator and completely through a CLB without encountering a flip-flop in that CLB. This type of path existed for the NEW2 design because there are a total of six inputs from the IOBs and each CLB has a maximum of five inputs. Therefore, one of the CLBs was used to pass through two signals, LINE_A and LINE_B, then to a NOR gate and subsequently to the next CLB.

This NEW2 design, after placement and routing, uses two CLBs and eight IOBs of the Xilinx 3000 Series FPGA. There is no block layout diagram included for the NEW2 routed design because it would be very similar to that of the N5R design and hence, was considered redundant.

Results from Test Case #2

The PATHFINDER program took approximately 2 minutes to execute the analysis for the NEW2 design using an IBM-compatible 386 - 20 MHz computer. Because of the nature of the design of NEW2, there were two particular signal paths that were much longer than other related signal paths. As shown below in Figure 5.4, the output of the PATHFINDER program, the NEW2.DAT file, lists paths 0 through 3 which have very similar delays. However, paths 4 and 5 are much larger.

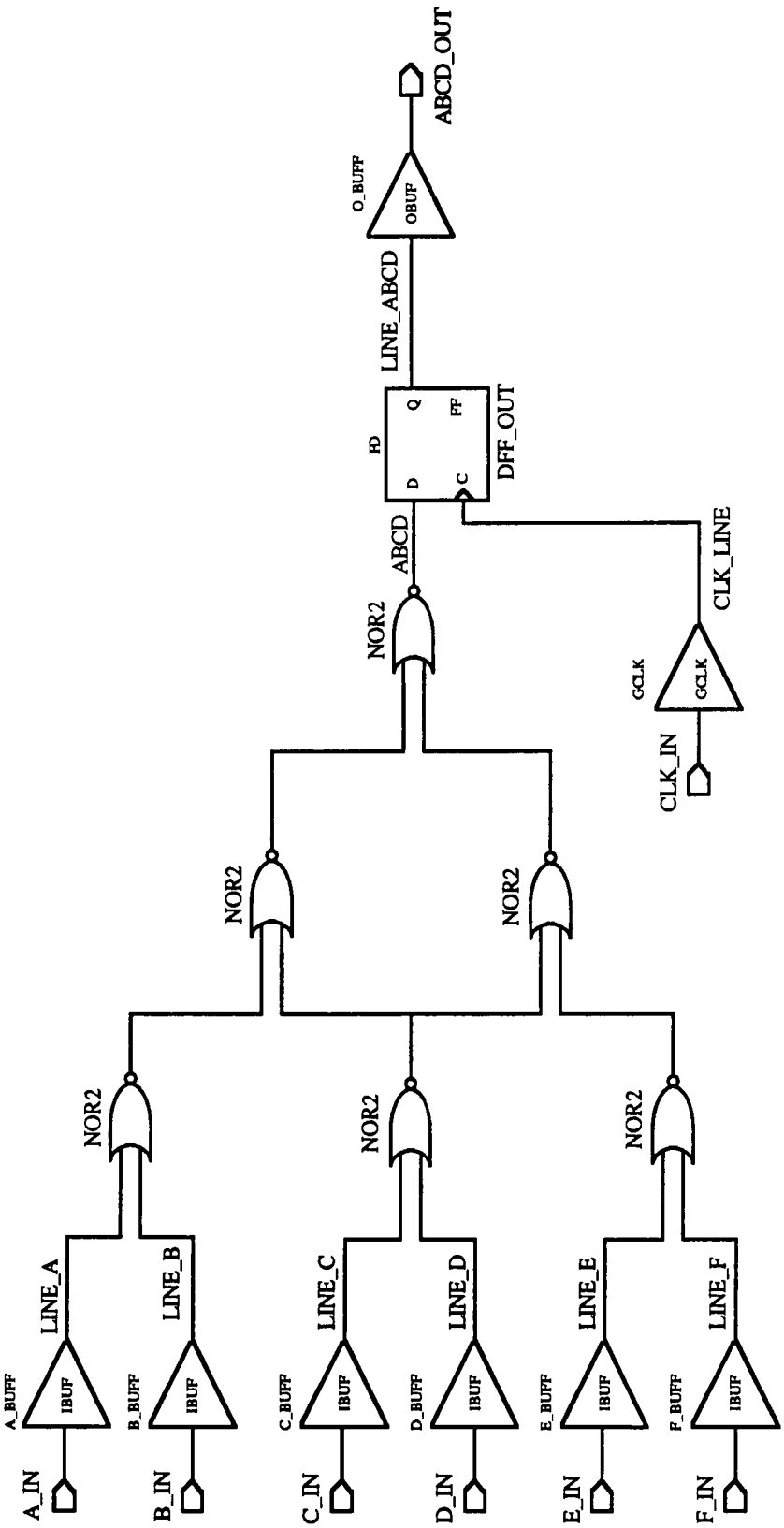


Figure 5.3 Test Case #2 Logic Design - NEW2

By observing Figure 5.3, a designer would expect paths 0 through 5 to all have relatively the same total delay. The reason that they do not is that two of the inputs have been placed into a second CLB during the placement operation. This change was introduced because the limit on the number of inputs in each CLB is five. The longest delay path that was discovered was path 6 which goes from the Y flip-flop output of the LINE_ABCD CLB to the output buffer pad ABCD_OUT. The longest delay was of 34.2 ns.

```
FILES ARE :  new2.xnf      new2.lca      new2.dat

PATH  0 :     F_IN.IBUF  ---> LINE_ABCD.DY    == 13.4 nsec
PATH  1 :     E_IN.IBUF  ---> LINE_ABCD.DY    == 14.9 nsec
PATH  2 :     D_IN.IBUF  ---> LINE_ABCD.DY    == 13.4 nsec
PATH  3 :     C_IN.IBUF  ---> LINE_ABCD.DY    == 12.0 nsec
PATH  4 :     B_IN.IBUF  ---> LINE_ABCD.DY    == 27.6 nsec
PATH  5 :     A_IN.IBUF  ---> LINE_ABCD.DY    == 26.2 nsec
PATH  6 :  LINE_ABCD.QY  ---> ABCD_OUT.OBUF   == 34.2 nsec

LONGEST
PATH  6 :  LINE_ABCD.QY  ---> ABCD_OUT.OBUF   == 34.2 nsec
```

Figure 5.4 Output of the PATHFINDER for the NEW2 Design -- NEW2.DAT

By executing the same calculation as before, the maximum clock frequency for the NEW2 design can be computed :

$$\text{max clock frequency} = \frac{1}{34.2 \text{ nanoseconds}} = 29.24 \text{ MHz}$$

Therefore, the maximum clock frequency for which the NEW2 design can run is 29.24 MHz.

Test Case #3 -- LCA1

Test Case #3 is a design created by Reid Wender of Philips Consumer Electronics Company in Knoxville, Tennessee. Since this design had proprietary information included in it, only the .XNF (Xilinx Netlist Format) and the .LCA (Logic Cell Array) files were given to the author to analyze. This design was also completed on a Mentor Graphics system using the NET ED program. The name given to this design was LCA1. The placed and routed design for LCA1 is given in the files: LCA1.XNF and LCA1.LCA. Due to the large size of these files, only partial listings of them are included in the APPENDIX.

The LCA1 design has 118 blocks (CLBs and IOBs) and 136 nets inside the Xilinx 3000 Series FPGA. The speed was executed at - 100 for the LCA1 design. This design included every type of delay path possible (refer to Table 4.1). There is no block diagram for the LCA1 design included in the report because, as before, it would be unnecessary.

Results from Test Case #3

Note that every block and net for the LCA1 design is labeled in the LCA1.XNF and LCA1.LCA files, even if some of the blocks were not labeled initially. The automatic placement and routing program assigns some variable names to the unassigned blocks or nets. These can be observed in the LCA1.XNF file. Since no block diagrams or schematic were given for the LCA1 design, many delay paths had to be searched for using the technique given in Figure 5.5. This only shows a small portion of the delays to exemplify the method. This was a long process because of the complexity of the design.

The PATHFINDER program took approximately 5 hours and 15 minutes to execute using an IBM-compatible 386 - 20 MHz computer due to the complexity of the LCA1 design. To speed up the process, an IBM-compatible 486 - 33 MHz computer with a math coprocessor was attained. When the PATHFINDER program was executed again, the total execution time was only about 3 hours. Even though 3 hours was required, vital information was obtained by using the PATHFINDER program. All of the delay paths found in the LCA1 design were totaled and placed into the LCA1.DAT file.

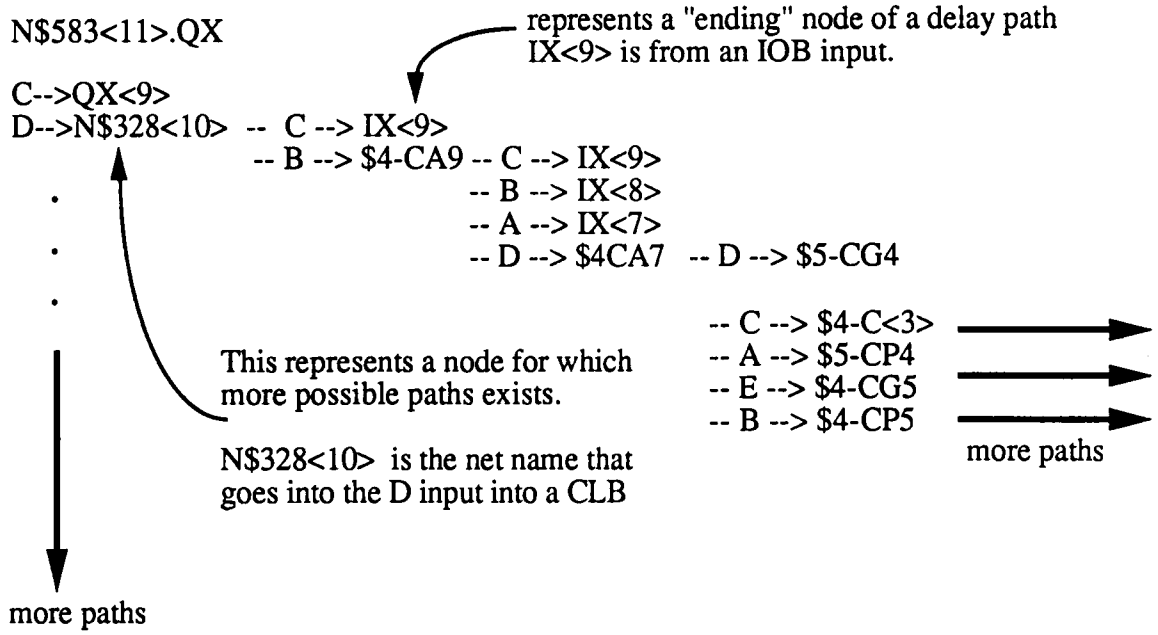


Figure 5.5 The technique of tracing delay paths from the LCA1.XNF file

There were a total of 1,861 delay paths, of which path 630 was the longest. Path 630 began at the I<2> input flip-flop and worked its way to the D-input flip-flop of CLB N$583<10>. Its total delay was 138.6 nanoseconds. By executing the same calculation as before, the maximum clock frequency for the NEW2 design can be computed.

$$\text{max clock frequency} = \frac{1}{138.6 \text{ nanoseconds}} = 7.22 \text{ MHz}$$

Therefore, the maximum clock frequency for which the NEW2 design can run is 7.22 MHz. This is very vital information for such a large design.

CHAPTER VI

DISCUSSION AND CONCLUSION

Comparison with the BAX Software

As previously noted, the purpose of the PATHFINDER was to aid in full-timing simulation. One should perform functional simulation immediately after the design has been entered. This would verify that the circuit logic is correct before partitioning, placing, or routing the design [6]. After the design has been placed and routed, timing simulation should be performed to assess critical paths in the design and thereby verify the functionality of the design under certain time constraints.

The development of the PATHFINDER program begin in May, 1991. In January of 1992, Xilinx released a new simulation tool for the XC3000 series called the "BAX" program [6]. The BAX program creates a routed XNF file using the same net names as the original schematic. BAX works by first reading in the routed XNF file, the unrouted MAP file, and the AKA file, if one exits. The AKA file created by MAP2LCA, contains the hierarchical path names for symbols and signals and the associated abbreviations. BAX then reads the AKA file to map net names in the routed XNF file back to the MAP file. It then matches signals and components inside matched pairs of CLBs and IOBs. Next it adds delays from the XNF file to the matching signal from the MAP file. Finally, the BAX program generates an XNF file that contains delays corresponding to the original schematic representation. This XNF file can be translated into an EREL file, which in turn can be used for performing timing analysis using QuickSim and QuickPath in the Mentor Graphics environment. An overview of a full-timing simulation flow is shown in Figure 6.1.

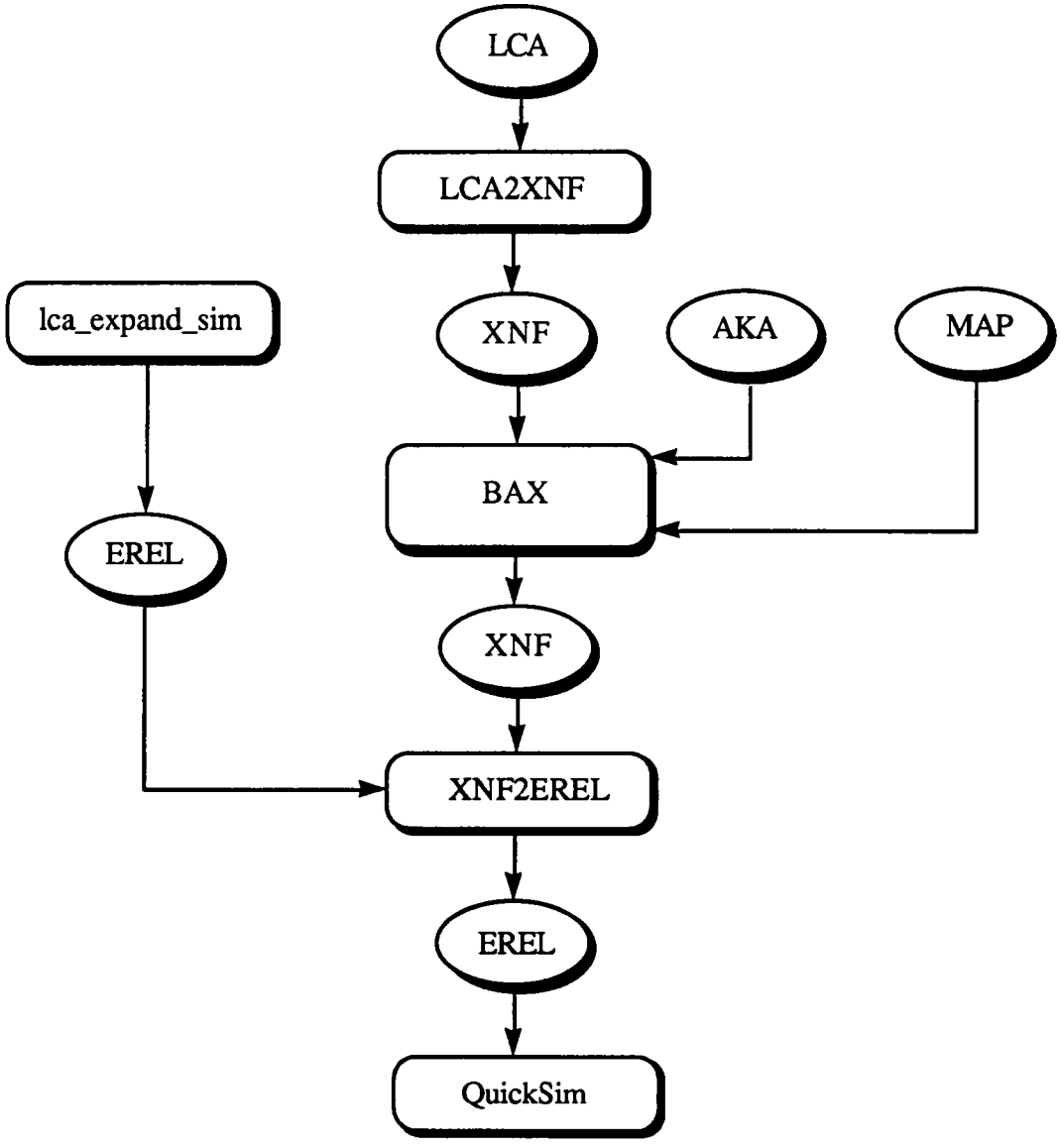


Figure 6.1 Full-Timing Simulation Flow

Contributions of the PATHFINDER Program

The BAX program improves the simulation interface for the XC3000 family by placing delay information in the original MAP (logic partitioned by XNFMAP) file which allows the user to use the original schematic for timing simulation. However, it does not perform the functions as the PATHFINDER program. Thus the BAX program is suitable for relaying routed information back to the original schematic, but it does not analyze the data. The PATHFINDER program performs the required timing analysis to find the longest delay path through an IC after placement and routing has occurred. Hence, the user can then calculate the maximum clock frequency at which the design can operate. These are the unique contributions of the PATHFINDER program.

Summary

An algorithm to determine the path delays and to calculate the longest delay path was developed and automated. For every design that needs to be analyzed, a Xilinx Netlist Format file and a Logic Cell Array file is required for performing the timing analysis. All of the delay paths found with their corresponding total delays are placed into a data file named by the user. While analyzing the whole design, the PATHFINDER program keeps track of the longest total delay path encountered. This longest delay path for the design is placed at the tail end of the data file. This vital information can be used to calculate the maximum clock frequency at which the FPGA design can execute. Overall, this project met its goals and is a success.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Alfke, Peter, Keath Armstrong, et. al. XILINX The Programmable Gate Array Data Book. San Jose, CA : Xilinx, Inc., 1989.

[2] Bouldin, Donald W. Designing Field-Programmable Gate Arrays. ECE 552 Class Notes, University of Tennessee, 1992.

[3] Read, John W. Gate Arrays. New York, NY : Kingsport Press, 1985.

[4] Smith, M.J.S. Field Programmable Gate Arrays -- Getting Started. Draft Manuscript, University of Hawaii, March 1991.

[5] Wender, Reid. Rapid Prototyping Development. Internal Report. Philips Consumer Electronics Company, 1991.

[6] Xilinx, Inc. XACT Development System. "Design Interface User Guide : Design Verification." San Jose, CA : Xilinx, Inc., January 1992.

APPENDIX

APPENDIX A

PATHFINDER PROGRAM

```
/*      FOR : Philips Consumer Electronics Company - Dept 676
              (Digital Systems)

PROGRAMMER : Nayan D. Patel

DATE : October 30, 1991

PATHFIND.C : This program reads the routed.xnf and routed.lca files
             to find delay paths from flip-flop to flip-flop, from
             flip-flop from input iob's, or from output iob's back
             from flip-flops. These paths are used to determine the
             maximum delay path to calculate the maximum clock speed
             for the Xilinx ASIC. The delay paths are placed into
             a structure where they can be stored for later use.  */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct path
{  char dest[20];     /*  xxxx.DX, xxxx.OUTFF, xxxx.INFF */
   char source[20];   /*  xxxx.QX, xxxx.I  xxxx.O       */
   float delay;       /*  in ns (nano sec)              */
   int num;           /*  pathnumber                    */
};

void display( struct path show, FILE *fpdata );
void display2( struct path show );

/************************** START MAIN ROUTINE
*******************************/

char savepath[1500][25], dffpaths[2500][2];
float actualnum, totaldelay[1500];

main ()

{  /************* initialize all variables ******************/

   struct path dff, longdff;

   char file_xnf[50], file_lca[50], file_out[50], file_dat[50];

   char chkstr[80], chkstr2[80], chkstr3[80], chkstr4[80];
```

```
  char chkstr5[80], chkstr6[80], chkstr7[80], chkstr8[80];
  char tempstr[80];

  char dffname[30], dfftype[30], iobname[30];

  char lookfor[80], lookfordff[80], lookforblk[80], lookforpin[80];
  char lookforblkname[80], lookforblkname2[80], lookforobuf[80];
  char lookfornet[30], lookfornet2[30], lookforx[30], lookfory[30];
  char lookforoutff[80];

  char blkname[30], blockname[30], check[3], sourcename[30];
  char netname[30], destname[30], temp[5], ch, file3[3], file4[3];

  int i, k, j, l, m, n, p, q, longpath, xnum, xnumber;
  int linenumber, dffnumber, blknumber, count, qnum, iobnumber;
  int dxlocation, dylocation, pathnum, pathnumber;

  FILE *fp, *fp2, *fp3, *fp4, *fp5, *fpdata, *fpout;

  count = 0;
  dffnumber = 0;
  pathnum = 0;
  pathnumber = 0;
  linenumber = 0;
  strcpy (file4, "off");

/********************** GET DATA FILE NAMES FROM USER
**********************/
  while ( count < 24 )   /*** clear screen ***/
  {  puts ("");
     count++;
  }
  puts( "\nFILENAME of routed.xnf file (ex. d:\\test_rt.xnf ) ?\n" );
  gets( file_xnf );
  puts( "\nFILENAME of routed.lca file ( ex. d:\\test_rt.lca ) ?\n" );
  gets( file_lca );
  puts( "\nFILENAME of desired output data ( ex. output.dat ) ?\n" );
  gets( file_out );
  fpout = fopen (file_out, "w" );

/* strcpy(file_dat,"&");
  strcat(file_dat, file_out);
  fpdata = fopen (file_dat, "w" );  */

  fprintf ( fpout, "\n\nFILES ARE :  %s\t%s\t%s\n", file_xnf, file_lca, file_out);
/*************************************************************************
*****/

  dff.delay = 0.0;
  dff.num = 0;
  strcpy( dff.source, '\0' );
  strcpy( dff.dest, '\0' );
```

```
  longdff.delay = 0.0;
  longdff.num = 0;
  strcpy( longdff.source, '\0' );
  strcpy( longdff.dest, '\0' );

/********** OPEN .xnf file and look for "DFF" in the file ******************/

  if ( fp = fopen ( file_xnf, "r" ) )
  { /*** look at one line in the .xnf file and place it into the string
         that is labeled 'chkstr' ***/

start:if ( ( fgets( chkstr, 80, fp )) != NULL )
    { linenumber++;
        strcpy (check, "DFF");
        strcpy (lookfordff , "DFF" );
        if ( strstr( chkstr, lookfordff ) != NULL )
        { dffnumber++;

          /* fprintf
(fpdata,"\n==========================================================
================"); */
          /* printf
("\n============================================================
============");*/
          /* fprintf
(fpdata,"\n==========================================================
================"); */
          printf
("\n============================================================
============");
          /* fprintf ( fpdata,"\n\t\t NEW checkstring = %s\n", chkstr ); */
          printf ( "\n\t\t NEW checkstring = %s\n", chkstr );

          /*** clear previous dff names ***/
          strcpy (dffname, "                    " );

          for (i = 0; i < 1500; i++ )
          { savepath[i][0] = '\0';
            totaldelay[i] = 0;
          }

          for (i = 0; i < 2500; i++ )
            dffpaths[i][0] = '\0';

          for (i = 5; i < 30; i++ )
          { ch = chkstr[i];    /*** extract dffname from checkstring ***/
            if ( ch == ',' )
                 break;
            dffname [ (i-5) ] = chkstr[i];
          }

          strcpy ( destname, dffname ); /*** change label of dest. name ***/
```

```
            for (i = 0; i < 30; i++ )
            {  if (( destname[i] == 'Q' ) & ( destname[i+1] == 'X' ))
                    destname[i] = 'D';
               if (( destname[i] == 'Q' ) & ( destname[i+1] == 'Y' ))
                    destname[i] = 'D';
            }
            /*** clear other previous data strings ***/
            strcpy (blkname, "                          " );
            strcpy (sourcename, "                          " );
            strcpy ( lookforblk, "SYM, ");

            for (j = 5; j < 30; j++ )
            {  ch = chkstr[j];
               if ( ch == '.' )
                    break;        /*** exstract part of the block string ***/
               lookforblk[j] = ch;
               lookforblk[j+1] = '\0';
               blkname [ (j-5) ] = chkstr[j];
            }
            strcat (lookforblk, ", CLB");
            /* fprintf( fpdata, "dffname ===> %s at line %i  dff#= %i", dffname,
linenumber, dffnumber );*/
            printf("dffname ===> %s at line %i  dff#= %i", dffname, linenumber,
dffnumber );
            /* fprintf( fpdata, "blkname ===> %s", blkname );*/

/*************************************************************************
*****/
            /*** Use another file pointer to find the block name of dff ***/

            if ( fp2 = fopen ( file_xnf, "r" ) )
            {  blknumber = 0;
    /*      fprintf ( fpdata, "lookforblk ===> %s\n\n", lookforblk );   */

    /* ******** Need this while loop to find chkstr3 ********* */

              while ( ( fgets( chkstr2, 80, fp2 )) != NULL )
              {  blknumber++;
                   if ( strstr( chkstr2, lookforblk ) != NULL )
                   {  /*  fprintf ( fpdata, "checkstring 2 = %s", chkstr2 );   */
                     break;
                   }
              }
              fgets( chkstr3, 80, fp2 );  /*** need to step two lines ***/
              fgets( chkstr3, 80, fp2 );
              /* fprintf ( fpdata, "\nCONFIG  ===> %s ", chkstr3 );*/

/*************************************************************************
*****/
/***************************** look at DX if QX
*****************************/
/*************************************************************************
```

```
*****/
            strcpy ( lookfor, ".QX" );
            if ( strstr( chkstr, lookfor ) != NULL )
            {  for (k = 12; k < 80; k++ )
                 {  if (( chkstr3[k] == 'D' ) & ( chkstr3[k+1] == 'X' ))
                    {  dxlocation = k+3; /* LOOK 3 CHAR'S AFTER 'D' of 'DX' */
                       break;
                    }
                 }
/*************************  find F: after DX  ***********************/

                if ( chkstr3[dxlocation] == 'F' )
                {  for (j = 0; j < 5; j++ )
                   strcpy ( dffpaths[j] , "  ");
                   for (k = 12; k < 30; k++ )
                   {  if (( chkstr3[k] == 'F' ) & ( chkstr3[k+1] == ':' ))
                        {  n=0;
                           for (m = k+2; m < k+12; m++ )
                           {  if ( chkstr3[m] != ':' )
                              {  /* printf ("\nchkstr3[%i] ==> %c", m, chkstr3[m] ); */
                                 dffpaths[n][0] = chkstr3[m];
                                 if (chkstr3[m] == 'Q')
                                 {  /* printf ("%c",chkstr3[m+1]); */
                                    dffpaths[n][1] = chkstr3[m+1];
                                    m++;
                                 }
                                 n++;
                                 if ( chkstr3[m] != ' ' )
                                 {  /* printf (" --> path %i", pathnum ); */
                                    pathnum++;
                                 }
                              }
                              if ( chkstr3[m] == ' ')
                                 break;
                           }
                        }
                   }
                }
/**********************  find G:  after DX  ***************************/
                if ( chkstr3[dxlocation] == 'G' )
                {  for (j = 0; j < 5; j++ )
                      strcpy ( dffpaths[j] , "  ");
                   for (k = 12; k < 30; k++ )
                   {  if (( chkstr3[k] == 'G' ) & ( chkstr3[k+1] == ':' ))
                        {  n=0;
                           for (m = k+2; m < k+12; m++ )
                           {  if ( chkstr3[m] != ':' )
                              {  /* printf ("\nchkstr3[%i] ==> %c", m, chkstr3[m] ); */
                                 dffpaths[n][0] = chkstr3[m];
                                 if (chkstr3[m] == 'Q')
                                 {  /* printf ("%c",chkstr3[m+1]); */
                                    dffpaths[n][1] = chkstr3[m+1];
```

```
                        m++;
                      }
                      n++;
                      if ( chkstr3[m] != ' ' )
                      { /* printf (" --> path %i", pathnum ); */
                        pathnum++;
                      }
                    }
                    if ( chkstr3[m] == ' ')
                      break;
                  }
                }
              }
            }
/*************************************************************************
*****/
/*************************** look at DY if QY
***************************/
/*************************************************************************
*****/
          strcpy ( lookfor, ".QY" );
          if ( strstr( chkstr, lookfor ) != NULL )
          { for (k = 12; k < 80; k++ )
              { if (( chkstr3[k] == 'D' ) & ( chkstr3[k+1] == 'Y' ))
                { dylocation = k+3; /* LOOK 3 CHAR'S AFTER 'D' of 'DY' */
                  break;
                }
              }
/************************** find F: after DY **********************/
            if ( chkstr3[dylocation] == 'F' )
            { for (j = 0; j < 5; j++ )
                strcpy ( dffpaths[j] , "  ");
              for (k = 12; k < 30; k++ )
              { if (( chkstr3[k] == 'F' ) & ( chkstr3[k+1] == ':' ))
                  { n=0;
                    for (m = k+2; m < k+12; m++ )
                    { if ( chkstr3[m] != ':' )
                      { /* printf ("\nchkstr3[%i] ==> %c", m, chkstr3[m] ); */
                        dffpaths[n][0] = chkstr3[m];
                        if (chkstr3[m] == 'Q')
                        { /* printf ("%c",chkstr3[m+1]); */
                          dffpaths[n][1] = chkstr3[m+1];
                          m++;
                        }
                        n++;
                        if ( chkstr3[m] != ' ' )
                        { /* printf (" --> path %i", pathnum ); */
                          pathnum++;
                        }
                      }
                      if ( chkstr3[m] == ' ')
```

```
                        break;
                    }
                }
            }
        }
/*********************  find G: after DY  ***************************/
        if ( chkstr3[dylocation] == 'G' )
        { for (j = 0; j < 5; j++ )
              strcpy ( dffpaths[j] , "  ");
          for (k = 12; k < 30; k++ )
          { if (( chkstr3[k] == 'G' ) & ( chkstr3[k+1] == ':' ))
              { n=0;
                for (m = k+2; m < k+12; m++ )
                { if ( chkstr3[m] != ':' )
                  { /* printf ("\nchkstr3[%i] ==> %c", m, chkstr3[m] ); */
                    dffpaths[n][0] = chkstr3[m];
                    if (chkstr3[m] == 'Q')
                    { /* printf ("%c",chkstr3[m+1]); */
                      dffpaths[n][1] = chkstr3[m+1];
                      m++;
                    }
                    n++;
                    if ( chkstr3[m] != ' ' )
                    { /* printf (" --> path %i", pathnum ); */
                      pathnum++;
                    }
                  }
                  if ( chkstr3[m] == ' ')
                    break;
                }
              }
          }
        }
    }
/*********************  analyze the possible dff paths  ******************/

/*  right now you know the following :
        dffname, dfflinenumber, blockname, destination of path (i.e. dffname)
        either DX or DY
        destname point xxxx.DX
        maximum of 5 possible paths from the one DX or DY in "dffpaths"
        labels of checkstrings

        Begin for loop for 5 possible choices of one dff.  If it may be
        the case that more than five inputs lead to one dff, that will be
        taken care of by 'qnum'.  'qnum' is incremented when that may occur */

        qnum = 5;
        for ( q = 0; q < qnum; q++ )
        { if (q>=1500)
            xnum = q -1490;
          if ( dffpaths[q][0]=='A' || dffpaths[q][0]=='B' ||
```

```
                dffpaths[q][0]=='C' || dffpaths[q][0]=='D' ||
                dffpaths[q][0]=='E' )
continueout:    { /* fprintf (fpdata,"\n----------------------------------------------------------------
---------"); */
                /* printf ("\n----------------------------------------------------------------------
"); */
                /* fprintf (fpdata,"\ndffpaths[q=%i] = %c", q, dffpaths[q][0]);*/
                /* printf ("\ndffpaths[q=%i] = %c", q, dffpaths[q][0]); */

                if ( q >= 5 )    /*** this is where 'qnum' is used ***/
                {  strcpy( lookforpin, "PIN, A, I, ");
                   lookforpin[5] = dffpaths[q][0];

                   /*  fprintf (fpdata, "\nLOOKFORPIN ===> %s", lookforpin);
                   printf ("\nLOOKFORPIN ===> %s", lookforpin);  */

                   i = 0;
findpath:          if (q>=1500)
                   {  if (savepath[xnum][0] == '\0')
                      { /* printf("\nsavepath SWITCHED"); */
                        i++;
                        strcpy ( savepath[xnum], savepath[xnum-i]);
                        totaldelay[xnum] = totaldelay[xnum-i];
                      }
                   }
                   else
                   {  if (savepath[q][0] == '\0')
                      { /* printf("\nsavepath SWITCHED"); */
                        i++;
                        strcpy (savepath[q], savepath[q-i]);
                        totaldelay[q] = totaldelay[q-i];
                      }
                   }

                   if (q>=1500)
                   {  if (savepath[xnum][0] == '\0')
                     goto findpath;
                   }
                   else
                   {  if (savepath[q][0] == '\0')
                     goto findpath;
                   }

                   if (q>=1500)
                   {  dff.delay = totaldelay[xnum];
                   }
                   else
                   {  dff.delay = totaldelay[q];
                   }

                   strcpy( dff.dest, destname );
```

```
                    /*** Use fseek to move the file pointer back 150
                       characters. This is approximately 5 to 8 lines
                       depending on the number of characters in the
                       checkstring4 line ***/

                    rewind(fp5);

                    /* printf("\nSEARCH savepath[q=%i] ===> %s",q,savepath[q]);*/
                    while ( ( fgets( chkstr4, 80, fp5)) != NULL)
                    { if (q>=1500)
                      { if ( strstr( chkstr4, savepath[xnum] ) != NULL)
                        { /* printf("FOUND savepath[xnum=%i] ===>
%s",xnum,chkstr4); */
                              break;
                        }
                      }
                      else
                      { if ( strstr( chkstr4, savepath[q] ) != NULL)
                        { /* printf("FOUND savepath[q=%i] ===>  %s",q,chkstr4); */
                          if (q>10) strcpy(savepath[q-5],"\0");
                          break;
                        }
                      }
                    }

                    fseek (fp5, -150, SEEK_CUR );
      qlook1:     while (( fgets( chkstr4, 80, fp5 )) != NULL )
                    { if ( strstr( chkstr4, lookforpin ) != NULL )
                      { /* fprintf ( fpdata,"\ncheckstring 4 = %s", chkstr4 ); */
                        /* printf ( "\nLOOPING checkstring 4 = %s", chkstr4 ); */
                        break; /*** break when the chkstr4 is found ***/
                      }
                    }
                    goto continueq; /*** skip part for q < 5 ***/
                }

                strcpy( lookforpin, "PIN, A, I, ");
                lookforpin[5] = dffpaths[q][0];
                /*  fprintf ( fpdata,"\nlookforpin ===> %s", lookforpin );
                printf ( "\nlookforpin ===> %s", lookforpin );  */

                dff.delay = 8.0 ;
                strcpy( dff.dest, destname );

                fseek (fp2, -150, SEEK_CUR );
      qlook2:    while ( ( fgets( chkstr4, 80, fp2 )) != NULL )
                { if ( strstr( chkstr4, lookforpin ) != NULL )
                    { /* fprintf ( fpdata,"\ncheckstring 4 = %s", chkstr4 ); */
                      /* printf ( "\nREGULAR checkstring 4 = %s", chkstr4 ); */
                      break;
                    }
                }
```

```
continueq:    strcpy (lookfornet, "\0");
              strcpy( lookfornet,"Netdelay \0" );

              for (i = 11; i < 30; i++ )
              {  if (( chkstr4[i] == ',') & ( q >= 5 ))
                    goto qlook1;
                 if (( chkstr4[i] == ',') & ( q < 5 ))
                    goto qlook2;
                 if (( chkstr4[i] == ' ' ) || ( chkstr4[i] == '\0' ))
                 {  lookfornet[(i-3)]=' ';
                    break;
                 }
                 else
                 {  netname[(i-11)] = chkstr4[i];
                    netname[(i-10)] = '\0';
                    lookfornet[(i-2)] = chkstr4[i];
                    lookfornet[(i-1)] = '\0';
                 }
               }

              /* fprintf( fpdata,"\nnetname =====> %s", netname);
              printf( "\nnetname =====> %s", netname);
              printf( "\nlookfornet ==> %s--ENDCHK\n", lookfornet ); */

              if ( strstr( file3, "on" ) != NULL )
                   goto file3open;

loopforiob:   if ( fp3 = fopen ( file_lca, "r" ) )

    file3open: {  /*** Begin looping finding NET DELAYS ***/
                  strcpy (file3, "on");
                  rewind(fp3);
    backif:       if ( ( fgets( chkstr5, 80, fp3 )) != NULL )
                  {  if ( strstr( chkstr5, lookfornet ) != NULL )
                     {  j=11;
          backtoj:    if (j < 30 )
                      {  if ( chkstr5[j] == ' ' )
                         {  for (k = j+1; k < 80; k++ )
                            {  if ( chkstr5[k] == ' ' )
                               {  temp[0] = chkstr5[(k+1)] ;
                                  temp[1] = chkstr5[(k+2)] ;
                                  temp[2] = chkstr5[(k+3)] ;
                                  temp[3] = chkstr5[(k+4)] ;
                                  temp[4] = chkstr5[(k+5)] ;

                                  /*** use 'atof' to convert the ascii
                                      number to a float number ***/
                                  actualnum = atof(temp);

                                  strcpy ( temp, "     ");
                                  /* fprintf (fpdata,"Net delay = %4.1f", actualnum );
```

```
*/
                                        /* printf ("Net delay = %4.1f", actualnum ); */
                                        break;
                                    }
                                   /* assuming net delays are < 100ns */
                                 }
                                 j=30;
                               }
                           }
                           j++;
                           if (j<30) goto backtoj;

                           dff.delay = dff.delay + actualnum;
                           strcpy( dff.dest, destname );
                           strcpy( dff.source, "GOING BACK TO CLB" );

                         }
                       }
                       else goto contif;
                       goto backif;
          contif: /* fprintf (fpdata,"\nfile4 = %s\n", file4); */

                       if ( strstr( file4, "on" ) != NULL )
                         goto file4open;

                       /* FIND WHERE THE SOURCE COULD BE FROM I,Y,X,G or
F */

                       if (( fp4 = fopen ( file_xnf, "r" )) && (fp5 = fopen (file_xnf, "r")) )

        file4open: { rewind(fp4);
                     strcpy (file4, "on");
                     strcpy (lookfornet2, ", O, ");
                     strcat (lookfornet2, netname);
                     /* printf ("\nlfn2 ==> %s--ENDCHK\n",lookfornet2); */
                     while ( ( fgets( chkstr6, 80, fp4 )) != NULL )
                     { if ( strstr( chkstr6, lookfornet2 ) != NULL )
                       { /* fprintf ( fpdata,"\ncheckstring 6 = %s", chkstr6 ); */
                         /* printf ("\ncheckstring 6 == %s", chkstr6); */
                         break;
                       }
                     }
                     if ( chkstr6[5] == 'I')
                     { /* puts ("\nSOURCE destination found !"); */
                       actualnum = 4.0;
                       fgets( chkstr6, 80, fp4 );
                       fgets( chkstr6, 80, fp4 );
                       for (j = 5; j < 30; j++ )
                       { sourcename [ (j-5) ] = chkstr6[j];
                         sourcename [ (j-4) ] = '\0';
                         if ( chkstr6[j] == '.' )
                           break;
```

```
  }
  strcat (sourcename, "IBUF");

  dff.delay = dff.delay + actualnum;
  dff.num = pathnumber;
  strcpy( dff.source, sourcename );
  /* display ( dff, fpdata ); */
  display ( dff, fpout );
  display2( dff);
  if (dff.delay >= longdff.delay)
  { longdff.delay = dff.delay;
      longdff.num = dff.num;
      strcpy (longdff.source, dff.source);
      strcpy (longdff.dest, dff.dest);
  }

  pathnumber++;
}

if ( chkstr6[5] == 'Q')
{ /* puts ("\nSOURCE destination found !"); */
  actualnum = 17.0;
  fgets( chkstr6, 80, fp4 );
  fgets( chkstr6, 80, fp4 );
  fgets( chkstr6, 80, fp4 );
  for (j = 5; j < 30; j++ )
  { sourcename [ (j-5) ] = chkstr6[j];
      sourcename [ (j-4) ] = '\0';
      if ( chkstr6[j] == '.' )
        break;
  }
  strcat (sourcename, "INFF");

  dff.delay = dff.delay + actualnum;
  dff.num = pathnumber;
  strcpy( dff.source, sourcename );
  /* display ( dff, fpdata );*/
  display ( dff, fpout );
  display2( dff );
  if (dff.delay >= longdff.delay)
  { longdff.delay = dff.delay;
      longdff.num = dff.num;
      strcpy (longdff.source, dff.source);
      strcpy (longdff.dest, dff.dest);
  }

  pathnumber++;
}

if ( chkstr6[5] == 'X')
{ /* puts ("\nSOURCE destination found !"); */
  strcpy (lookforblkname, "SYM, ");
```

```
while ( ( fgets( chkstr7, 80, fp4 )) != NULL )
{  if ( strstr( chkstr7, lookforblkname ) != NULL )
      {  /* printf ( "\ncheckstring 7 = %s", chkstr7 ); */
        break;
      }
}
strcpy (lookforblkname2, "SYM, ");
for (j = 5; j < 30; j++ )
{  if ( chkstr7[j] == '.' )
        break;
     lookforblkname2 [j] = chkstr7[j];
     lookforblkname2 [(j+1)] = '\0';
     blockname [ (j-5) ] = chkstr7[j];
     blockname [ (j-4) ] = '\0';
}
strcat (lookforblkname2, ", CLB");
/* printf ("\nlfbn2 ==> %s--ENDCHK",lookforblkname2); */
/* printf ("\nblockname ==> %s--ENDCHK",blockname); */
rewind(fp4);

while ( ( fgets( chkstr8, 80, fp4 )) != NULL )
{  if ( strstr( chkstr8, lookforblkname2 ) != NULL )
      {  /* printf ( "\ncheckstring 8 = %s", chkstr8 ); */
        break;
      }
}
fgets( chkstr8, 80, fp4 );
fgets( chkstr8, 80, fp4 );
/* printf ( "\ncheckstring 8#2 = %s", chkstr8 ); */

strcpy(lookforx," X:QX");
if ( strstr( chkstr8, lookforx ) != NULL )
{  strcpy (sourcename, blockname);
     strcat (sourcename, ".QX");
     actualnum = 8.0;

     dff.delay = dff.delay + actualnum;
     dff.num = pathnumber;
     strcpy( dff.source, sourcename );
     /* display ( dff, fpdata );*/
     display ( dff, fpout );
     display2( dff );
     if (dff.delay >= longdff.delay)
     {  longdff.delay = dff.delay;
        longdff.num = dff.num;
        strcpy (longdff.source, dff.source);
        strcpy (longdff.dest, dff.dest);
     }

     pathnumber++;
}
```

```
                    strcpy(lookforx," X:QY");
                    if ( strstr( chkstr8, lookforx ) != NULL )
                    { strcpy (sourcename, blockname);
                        strcat (sourcename, ".QY");
                        actualnum = 8.0;

                        dff.delay = dff.delay + actualnum;
                        dff.num = pathnumber;
                        strcpy( dff.source, sourcename );
                        /* display ( dff, fpdata );*/
                        display ( dff, fpout );
                        display2( dff );
                        if (dff.delay >= longdff.delay)
                        { longdff.delay = dff.delay;
                          longdff.num = dff.num;
                          strcpy (longdff.source, dff.source);
                          strcpy (longdff.dest, dff.dest);
                        }

                        pathnumber++;
                    }

                    strcpy(lookforx," X:F");
                    if ( strstr( chkstr8, lookforx ) != NULL )
                    { strcpy (sourcename, blockname);
                        strcat (sourcename, ".F");
                        actualnum = 11.0;
                        rewind(fp5);
                        while ( ( fgets( tempstr, 80, fp5)) != NULL)
                        { if ( strstr( tempstr, lookfornet2 ) != NULL)
                          { /* printf ("       TEMPSTR == %s", tempstr);*/
                            if (qnum>=1500)
                            { xnumber = qnum - 1490;
                                strcpy (savepath[xnumber], tempstr);
                                strcpy ( savepath[xnumber+1], "\0" );
                                /* printf ("\nsavepath[xnumber=%i qnum=%i] ==
%s",xnumber,qnum,savepath[xnumber]);*/
                                break;
                            }
                            else
                            { strcpy (savepath[qnum], tempstr);
                                /* printf ("\nsavepath[qnum=%i] ==
%s",qnum,savepath[qnum]);*/
                                break;
                            }
                          }
                        }

                        dff.delay = dff.delay + actualnum;
                        dff.num = pathnumber;
                        strcpy( dff.source, sourcename );
                        if (qnum>=1500)
```

```
                    {  totaldelay[xnumber] = dff.delay;
                       totaldelay[xnumber+1] = 0;
                       totaldelay[9] = totaldelay[10];
                    }
                    else
                       totaldelay[qnum] = dff.delay;

                    /* display ( dff, fpdata );*/
                    display2( dff);
                    if (dff.delay >= longdff.delay)
                    {  longdff.delay = dff.delay;
                       longdff.num = dff.num;
                       strcpy (longdff.source, dff.source);
                       strcpy (longdff.dest, dff.dest);
                    }

                    k = 12;
                    pathnum = 1;
backtok1:           if (( chkstr8[k] == 'F' ) & ( chkstr8[k+1] == ':' ))
                    {  n = qnum;
                       m = k+2;
backtom1:              if ( chkstr8[m] != ':' )
                       {  /* fprintf (fpdata,"\nchkstr8[%i] ==> %c", m,
chkstr8[m] );*/
                          /* printf ("\nchkstr8[%i] ==> %c", m, chkstr8[m] ); */
                          dffpaths[n][0] = chkstr8[m];
                          if (chkstr8[m] == 'Q')
                          {  /* fprintf (fpdata,"%c",chkstr8[m+1]); */
                             /* printf ("%c",chkstr8[m+1]);*/
                             dffpaths[n][1] = chkstr8[m+1];
                             m++;
                          }
                          n++;
                          if ( chkstr8[m] != ' ' )
                          {  /* fprintf (fpdata," --> path %i", pathnum-1 ); */
                             /* printf (" --> path %i", pathnum-1 );  */
                             pathnum++;
                          }
                          qnum++;
                       }
                       m++;
                       if ( chkstr8[m] == ' ')  m = 50;
                       if ( m <= 34 ) goto backtom1;
                    }
                    k++;
                    if (k <= 29 ) goto backtok1;
                    if ( strstr( check, "IOB" ) != NULL )
                    {  q++;
                       goto continueout;
                    }
                 }
```

```
                    strcpy(lookforx," X:G");
                    if ( strstr( chkstr8, lookforx ) != NULL )
                    { strcpy (sourcename, blockname);
                      strcat (sourcename, ".G");
                      actualnum = 11.0;
                      rewind(fp5);
                      while ( ( fgets( tempstr, 80, fp5)) != NULL)
                      { if ( strstr( tempstr, lookfornet2 ) != NULL)
                        { /* printf ("       TEMPSTR == %s", tempstr); */
                          if (qnum>=1500)
                          { xnumber = qnum - 1490;
                            strcpy (savepath[xnumber], tempstr);
                            strcpy ( savepath[xnumber+1], '\0' );
                            /* printf ("\nsavepath[xnumber=%i qnum=%i] ==
%s",xnumber,qnum,savepath[xnumber]);*/
                            break;
                          }
                          else
                          { strcpy (savepath[qnum], tempstr);
                            /* printf ("\nsavepath[qnum=%i] ==
%s",qnum,savepath[qnum]);*/
                            break;
                          }
                        }
                      }

                      dff.delay = dff.delay + actualnum;
                      dff.num = pathnumber;
                      strcpy( dff.source, sourcename );
                      if (qnum>=1500)
                      { totaldelay[xnumber] = dff.delay;
                        totaldelay[xnumber+1] = 0;
                        totaldelay[9] = totaldelay[10];
                      }
                      else
                        totaldelay[qnum] = dff.delay;

                      /* display ( dff, fpdata );*/
                      display2( dff );
                      if (dff.delay >= longdff.delay)
                      { longdff.delay = dff.delay;
                        longdff.num = dff.num;
                        strcpy (longdff.source, dff.source);
                        strcpy (longdff.dest, dff.dest);
                      }

                      k = 12;
backtok2:             if (( chkstr8[k] == 'G' ) & ( chkstr8[k+1] == ':' ))
                      { n = qnum;
                        m = k+2;
backtom2:               if ( chkstr8[m] != ':' )
                        { /* fprintf (fpdata,"\nchkstr8[%i] ==> %c", m,
```

```
chkstr8[m] );*/
                        /* printf ("\nchkstr8[%i] ==> %c", m, chkstr8[m] );*/
                        dffpaths[n][0] = chkstr8[m];
                        if (chkstr8[m] == 'Q')
                        { /* fprintf (fpdata,"%c",chkstr8[m+1]);*/
                            /* printf ("%c",chkstr8[m+1]);  */
                            dffpaths[n][1] = chkstr8[m+1];
                            m++;
                        }
                        n++;
                        if ( chkstr8[m] != ' ' )
                        { /* fprintf (fpdata," --> path %i", pathnum-1 );*/
                            /* printf (" --> path %i", pathnum-1 ); */
                            pathnum++;
                        }
                        qnum++;
                      }
                      m++;
                      if ( chkstr8[m] == ' ')  m = 50;
                      if ( m <= 34 ) goto backtom2;
                    }
                    k++;
                    if (k <= 29 ) goto backtok2;
                    if ( strstr( check, "IOB" ) != NULL )
                    { q++;
                      goto continueout;
                    }
                }
            }

            if ( chkstr6[5] == 'Y')
            { /* puts ("\nSOURCE destination found !"); */
              strcpy (lookforblkname, "SYM, ");
              while ( ( fgets( chkstr7, 80, fp4 )) != NULL )
              { if ( strstr( chkstr7, lookforblkname ) != NULL )
                    { /* printf ( "\ncheckstring 7 = %s", chkstr7 ); */
                      break;
                    }
              }
              strcpy (lookforblkname2, "SYM, ");
              for (j = 5; j < 30; j++ )
              { if ( chkstr7[j] == '.' )
                      break;
                    lookforblkname2 [j] = chkstr7[j];
                    lookforblkname2 [(j+1)] = '\0';
                    blockname [ (j-5) ] = chkstr7[j];
                    blockname [ (j-4) ] = '\0';
              }
              strcat (lookforblkname2, ", CLB");
              /* printf ("\nlfbn2 ==> %s--ENDCHK",lookforblkname2); */
              /* printf ("\nblockname ==> %s--ENDCHK",blockname); */
              rewind(fp4);
```

```
while ( ( fgets( chkstr8, 80, fp4 )) != NULL )
{  if ( strstr( chkstr8, lookforblkname2 ) != NULL )
      {  /* printf ( "\ncheckstring 8 = %s", chkstr8 ); */
         break;
      }
}
fgets( chkstr8, 80, fp4 );
fgets( chkstr8, 80, fp4 );
/* printf ( "\ncheckstring 8#2 = %s", chkstr8 ); */

strcpy(lookfory," Y:QY");
if ( strstr( chkstr8, lookfory ) != NULL )
{  strcpy (sourcename, blockname);
      strcat (sourcename, ".QY");
      actualnum = 8.0;

      dff.delay = dff.delay + actualnum;
      dff.num = pathnumber;
      strcpy( dff.source, sourcename );
      /* display ( dff, fpdata );*/
      display ( dff, fpout );
      display2( dff );
      if (dff.delay >= longdff.delay)
      {  longdff.delay = dff.delay;
         longdff.num = dff.num;
         strcpy (longdff.source, dff.source);
         strcpy (longdff.dest, dff.dest);
      }

      pathnumber++;
}

strcpy(lookfory," Y:QX");
if ( strstr( chkstr8, lookfory ) != NULL )
{  strcpy (sourcename, blockname);
      strcat (sourcename, ".QX");
      actualnum = 8.0;

      dff.delay = dff.delay + actualnum;
      dff.num = pathnumber;
      strcpy( dff.source, sourcename );
      /* display ( dff, fpdata );*/
      display ( dff, fpout );
      display2( dff );
      if (dff.delay >= longdff.delay)
      {  longdff.delay = dff.delay;
         longdff.num = dff.num;
         strcpy (longdff.source, dff.source);
         strcpy (longdff.dest, dff.dest);
      }
```

```
                    pathnumber++;
                }
                strcpy(lookfory,'\0');
                strcpy(lookfory," Y:F");
                if ( strstr( chkstr8, lookfory ) != NULL )
                { strcpy (sourcename, blockname);
                    strcat (sourcename, ".F");
                    actualnum = 11.0;
                    rewind(fp5);
                    while ( ( fgets( tempstr, 80, fp5)) != NULL)
                    { if ( strstr( tempstr, lookfornet2 ) != NULL)
                      { /* printf ("       TEMPSTR == %s", tempstr); */
                        if (qnum>=1500)
                        { xnumber = qnum - 1490;
                            strcpy (savepath[xnumber], tempstr);
                            strcpy ( savepath[xnumber+1], '\0' );
                            /* printf ("\nsavepath[xnumber=%i qnum=%i] ==
%s",xnumber,qnum,savepath[xnumber]);*/
                            break;
                        }
                        else
                        { strcpy (savepath[qnum], tempstr);
                            /* printf ("\nsavepath[qnum=%i] ==
%s",qnum,savepath[qnum]);*/
                            break;
                        }
                      }
                    }

                    dff.delay = dff.delay + actualnum;
                    dff.num = pathnumber;
                    strcpy( dff.source, sourcename );
                    if (qnum>=1500)
                    { totaldelay[xnumber] = dff.delay;
                      totaldelay[xnumber+1] = 0;
                      totaldelay[9] = totaldelay[10];
                    }
                    else
                      totaldelay[qnum] = dff.delay;

                    /* display ( dff, fpdata );*/
                    display2( dff );
                    if (dff.delay >= longdff.delay)
                    { longdff.delay = dff.delay;
                      longdff.num = dff.num;
                      strcpy (longdff.source, dff.source);
                      strcpy (longdff.dest, dff.dest);
                    }

                    k = 12;
backtok3:           if (( chkstr8[k] == 'F' ) & ( chkstr8[k+1] == ':' ))
                    { n = qnum;
```

```
                        m = k+2;
backtom3:               if ( chkstr8[m] != ':' )
                        {  /* fprintf (fpdata,"\nchkstr8[%i] ==> %c", m,
chkstr8[m] ); */
                          /* printf ("\nchkstr8[%i] ==> %c", m, chkstr8[m] );*/
                          dffpaths[n][0] = chkstr8[m];
                          if (chkstr8[m] == 'Q')
                          {  /* fprintf (fpdata,"%c",chkstr8[m+1]); */
                              /* printf ("%c",chkstr8[m+1]);*/
                              dffpaths[n][1] = chkstr8[m+1];
                              m++;
                          }
                          n++;
                          if ( chkstr8[m] != ' ' )
                          {  /* fprintf (fpdata," --> path %i", pathnum-1 );*/
                              /* printf (" --> path %i", pathnum-1 );*/
                              pathnum++;
                          }
                          qnum++;
                        }
                        m++;
                        if ( chkstr8[m] == ' ')  m = 50;
                        if ( m <= 34 ) goto backtom3;
                      }
                      k++;
                      if (k <= 29 ) goto backtok3;
                      if ( strstr( check, "IOB" ) != NULL )
                      {  q++;
                        goto continueout;
                      }
                    }

                    strcpy(lookfory," Y:G");
                    if ( strstr( chkstr8, lookfory ) != NULL )
                    {  strcpy (sourcename, blockname);
                        strcat (sourcename, ".G");
                        actualnum = 11.0;
                        rewind(fp5);
                        while ( ( fgets( tempstr, 80, fp5)) != NULL)
                        {  if ( strstr( tempstr, lookfornet2 ) != NULL)
                          {  /* printf ("       TEMPSTR == %s", tempstr); */
                            if (qnum>=1500)
                            {  xnumber = qnum - 1490;
                                strcpy (savepath[xnumber], tempstr);
                                strcpy ( savepath[xnumber+1], '\0' );
                                /* printf ("\nsavepath[xnumber=%i qnum=%i] ==
%s",xnumber,qnum,savepath[xnumber]);*/
                                break;
                            }
                            else
                            {  strcpy (savepath[qnum], tempstr);
                                /* printf ("\nsavepath[qnum=%i] ==
```

```
%s",qnum,savepath[qnum]);*/
                                    break;
                                }
                              }
                            }

                            dff.delay = dff.delay + actualnum;
                            dff.num = pathnumber;
                            strcpy( dff.source, sourcename );
                            if (qnum>=1500)
                            { totaldelay[xnumber] = dff.delay;
                              totaldelay[xnumber+1] = 0;
                              totaldelay[9] = totaldelay[10];
                            }
                            else
                              totaldelay[qnum] = dff.delay;

                            /* display ( dff, fpdata );*/
                            display2( dff );
                            if (dff.delay >= longdff.delay)
                            { longdff.delay = dff.delay;
                              longdff.num = dff.num;
                              strcpy (longdff.source, dff.source);
                              strcpy (longdff.dest, dff.dest);
                            }

                            k = 12;
                backtok4:   if (( chkstr8[k] == 'G' ) & ( chkstr8[k+1] == ':' ))
                            { n = qnum;
                              m = k+2;
                backtom4:     if ( chkstr8[m] != ':' )
                              { /* fprintf (fpdata,"\nchkstr8[%i] ==> %c", m,
chkstr8[m] );*/
                                /* printf ("\nchkstr8[%i] ==> %c", m, chkstr8[m] );*/
                                dffpaths[n][0] = chkstr8[m];
                                if (chkstr8[m] == 'Q')
                                { /* fprintf (fpdata, "%c",chkstr8[m+1]);*/
                                    /* printf ("%c",chkstr8[m+1]);*/
                                    dffpaths[n][1] = chkstr8[m+1];
                                    m++;
                                }
                                n++;
                                if ( chkstr8[m] != ' ' )
                                { /* fprintf (fpdata," --> path %i", pathnum-1 );*/
                                    /* printf (" --> path %i", pathnum-1 );*/
                                    pathnum++;
                                }
                                qnum++;
                              }
                              m++;
                              if ( chkstr8[m] == ' ')  m = 50;
                              if ( m <= 34 ) goto backtom4;
```

```
                        }
                        k++;
                        if (k <= 29 ) goto backtok4;
                        if ( strstr( check, "IOB" ) != NULL )
                        {  q++;
                           goto continueout;
                        }
                     }
                  }
               }
               else printf ("\nERROR OPENING FILE  %s\n", file_xnf);
               /* fclose(fp4); */
               strcpy (file4, "on");
            }
            else printf ( "\n\nEND OF FILE OR ERROR OPENING FILE :
%s!!\n\n", file_lca );

            fclose (fp3);  /*** REQUIRED FOR LOOPING TO WORK ***/
            strcpy (file3, "off");
            if (( strstr(check,"IOB") != NULL) & ( q>=qnum ))
               goto returnloop;
            if (( strstr(check,"IOB") != NULL) & ( q!=qnum ))
            {  q++;
               goto continueout;
            }
            if ( strstr (check, "IOB" ) != NULL )
               goto returnloop; /*** if actually iob loop, return ***/
         }

         /************** end of path if QX or QY here ***************/

         if (dffpaths[q][0] == 'Q')
         {  strcpy (sourcename, blockname);
            if (dffpaths[q][1] == 'X')
               strcat (sourcename, ".QX");
            if (dffpaths[q][1] == 'Y')
               strcat (sourcename, ".QY");
            actualnum = 8.0;

            dff.delay = dff.delay + actualnum;
            dff.num = pathnumber;
            strcpy( dff.source, sourcename );
            /* display ( dff, fpdata );*/
            display ( dff, fpout );
            display2( dff );
            if (dff.delay >= longdff.delay)
            {  longdff.delay = dff.delay;
               longdff.num = dff.num;
               strcpy (longdff.source, dff.source);
               strcpy (longdff.dest, dff.dest);
            }
```

```
                    pathnumber++;
                  }
              }
            }
            fclose ( fp2 );
          }
    }
    else goto strt2;  /* else and goto are used because they */
    goto start;       /* would require less space on the stack */

/*************** Begin looping for finding output buffer paths **************/
strt2:rewind(fp);

    while ( ( fgets( chkstr, 80, fp )) != NULL )
    { linenumber++;
        q = qnum;
        if (q>=1500)
          xnum = q - 1490;

        strcpy (check, "IOB");
        strcpy (lookforobuf , ", OBUF" );
        strcpy (lookforoutff , ", OUTFF");
        if ( ( strstr( chkstr, lookforobuf ) != NULL ) ||
             ( strstr( chkstr, lookforoutff) != NULL ) )
        { iobnumber++;
          q++;
          if (q>=1500)
            xnum = q - 1490;

          /* fprintf
(fpdata,"\n==========================================================
================");*/
          printf
("\n==========================================================
============");
          /* fprintf ( fpdata,"\n\n\n\t\t NEW checkstring = %s\n", chkstr );*/
          printf ( "\n\n\n\t\t NEW checkstring = %s\n", chkstr );
          strcpy (iobname, "                    " );
          strcpy (blkname, "                    " );
          strcpy (sourcename, "                      " );

          strcpy (lookforpin, "PIN, O, I,");
          strcpy (lookfornet, "\0");
          strcpy (lookfornet,"Netdelay \0" );
          for (i = 5; i < 30; i++ )
          { ch = chkstr[i];
            if ( ch == ',' )
                 break;
            iobname [ (i-5) ] = chkstr[i];
          }
          strcpy ( destname, iobname );
          /* fprintf( fpdata,"iobname ===> %s at line %d  iob#= %d\n", iobname,
```

```
linenumber, iobnumber );*/

            if ( strstr( chkstr, lookforobuf ) != NULL )
              dff.delay = 24.0;

            if ( strstr( chkstr, lookforoutff ) != NULL )
              dff.delay = 27.0;

            fseek (fp, -150, SEEK_CUR );
            while ( ( fgets( chkstr, 80, fp )) != NULL )
            {  if ( strstr( chkstr, lookforpin ) != NULL )
               {  /* fprintf ( fpdata,"\ncheckstring  = %s", chkstr );*/
                    /* printf ( "\ncheckstring  = %s", chkstr ); */
                    break;
               }
            }
            for (j = 11; j < 30; j++ )
            {  if (( chkstr[j] == '\0' ) || ( chkstr[j] == ',' ))
               {  lookfornet[(j-3)] = ' ';
                    j=30;
               }
               else
               {  netname[j-11] = chkstr[j];
                    netname[j-10] = '\0';
                    lookfornet[(j-2)] = chkstr[j];
                    lookfornet[(j-1)] = '\0';
               }
            }
            /* fprintf( fpdata,"\nnetname =====> %s", netname);*/
            /* printf( "\nnetname =====> %s", netname);*/
            goto loopforiob;

returnloop: /* puts ("\nMADE RETURNLOOP");    */
            /* Use fseek to move file pointer back past found iob position */
            fseek (fp, +155, SEEK_CUR );
          }
      }
   }
   else printf ( "\n\nERROR IN OPENING FILE : %s!!\n\n", file_xnf );

end:
   /* fprintf ( fpdata,"\nEND OF FILE !! :  %s",chkstr );*/
   fclose ( fp );
   /* fputs
("\n\n**********************************************************************
****\n",fpdata);*/
   /* fputs
("**********************************************************************
*\n",fpdata);*/
   puts
("\n\n**********************************************************************
****\n");
```

```
  puts
("*************************************************************************
*\n");

  /* fprintf (fpdata,"\nLONGEST ");*/
  fprintf (fpout,"\n\nLONGEST ");
  printf ( "\nLONGEST ");
  /* display ( longdff, fpdata );*/
  display ( longdff, fpout );
  display2 ( longdff );

/*  fclose(fpdata);
  fclose(fpout);        */
}
/************************** END MAIN  ******************************/

void display( struct path show, FILE *fpout )
{  fprintf( fpout,"\nPATH %3i : %15s  ---> %16.15s == %4.1f nsec", show.num,
show.source, show.dest, show.delay );
}
void display2( struct path show )
{  printf( "\nPATH %3i : %15s  ---> %16.15s == %4.1f nsec", show.num, show.source,
show.dest, show.delay );
}
```

APPENDIX B

TEST CASE # 1

N5R.XNF

```
LCANET, 1
PROG, LCA2XNF, Ver. 3.01: File=nayan5_routed.lca Date=Tue Jun 18 07:42:26 1991 -
PART,3042pc84-100
PWR, 0, GND
PWR, 1, VCC
SYM, D_IN, IOB, LOC=P77, BLKNM=D_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_D
MODEL
SYM, D_IN.IBUF, IBUF
PIN, O, O, LINE_D, 4
PIN, I, I, D_IN, 0
END
ENDMOD
END
EXT, D_IN, I, 77
SYM, C_IN, IOB, LOC=P79, BLKNM=C_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_C
MODEL
SYM, C_IN.IBUF, IBUF
PIN, O, O, LINE_C, 4
PIN, I, I, C_IN, 0
END
ENDMOD
END
EXT, C_IN, I, 79
SYM, B_IN, IOB, LOC=P80, BLKNM=B_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_B
MODEL
SYM, B_IN.IBUF, IBUF
PIN, O, O, LINE_B, 4
PIN, I, I, B_IN, 0
END
ENDMOD
END
EXT, B_IN, I, 80
```

```
SYM, ABCD_OUT, IOB, LOC=P73, BLKNM=ABCD_OUT
CFG, Base IO
CFG, Config IN: OUT:O TRI:
PIN, O, I, LINE_ABCD
MODEL
SYM, ABCD_OUT.OBUF, OBUF
PIN, O, O, ABCD_OUT, 22
PIN, I, I, LINE_ABCD, 0
END
ENDMOD
END
EXT, ABCD_OUT, O, 73
SYM, A_IN, IOB, LOC=P78, BLKNM=A_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_A
MODEL
SYM, A_IN.IBUF, IBUF
PIN, O, O, LINE_A, 4
PIN, I, I, A_IN, 0
END
ENDMOD
END
EXT, A_IN, I, 78
SYM, LINE_ABCD, CLB, LOC=AL, BLKNM=LINE_ABCD
CFG, Base F
CFG, Config F:B:A:D:C Y:QY X: DY:F DX: RSTDIR: CLK:K ENCLK:
CFG, Equate F=(((B@A)@(A@D))@((A@D)@(D@C)))
PIN, Y, O, LINE_ABCD
PIN, B, I, D_D
PIN, A, I, D_C
PIN, D, I, D_B
PIN, C, I, D_A
PIN, K, I, CLK_LINE
MODEL
SYM, LINE_ABCD.Y, BUF
PIN, I, I, LINE_ABCD.QY, 0
PIN, O, O, LINE_ABCD, 2
END
SYM, LINE_ABCD.QY, DFF
PIN, Q, O, LINE_ABCD.QY, 5
PIN, D, I, LINE_ABCD.F, 0
PIN, C, I, CLK_LINE, 3
SETUP, D, C, +, 2, 5
PULSE, C, +, 5
PIN, GR, I, GLOBALRESET-, 24
PULSE, GR, -, 25
END
SYM, LINE_ABCD.F1, XOR
PIN, O, O, LINE_ABCD.F1, 0
PIN, 1, I, D_C, 1
PIN, 2, I, D_B, 2
```

```
PIN, 3, I, D_B, 2
PIN, 4, I, D_A, 2
END
SYM, LINE_ABCD.F, XOR
PIN, O, O, LINE_ABCD.F, 5
PIN, 1, I, D_D, 0
PIN, 2, I, D_C, 1
PIN, 3, I, D_C, 1
PIN, 4, I, D_B, 2
PIN, 5, I, LINE_ABCD.F1, 0
END
ENDMOD
END
SYM, D_B, CLB, LOC=AJ, BLKNM=D_B
CFG, Base FG
CFG, Config G:B F:A Y:QY X:QX DY:G DX:F RSTDIR: CLK:K ENCLK:
CFG, Equate F=A
CFG, Equate G=B
PIN, B, I, LINE_B
PIN, A, I, LINE_A
PIN, Y, O, D_B
PIN, X, O, D_A
PIN, K, I, CLK_LINE
MODEL
SYM, D_B.X, BUF
PIN, I, I, D_B.QX, 0
PIN, O, O, D_A, 2
END
SYM, D_B.Y, BUF
PIN, I, I, D_B.QY, 0
PIN, O, O, D_B, 2
END
SYM, D_B.F, BUF
PIN, I, I, LINE_A, 0
PIN, O, O, D_B.F, 5
END
SYM, D_B.G, BUF
PIN, I, I, LINE_B, 2
PIN, O, O, D_B.G, 5
END
SYM, D_B.QX, DFF
PIN, Q, O, D_B.QX, 5
PIN, D, I, D_B.F, 0
PIN, C, I, CLK_LINE, 3
SETUP, D, C, +, 2, 5
PULSE, C, +, 5
PIN, GR, I, GLOBALRESET-, 24
PULSE, GR, -, 25
END
SYM, D_B.QY, DFF
PIN, Q, O, D_B.QY, 5
PIN, D, I, D_B.G, 0
```

```
PIN, C, I, CLK_LINE, 3
SETUP, D, C, +, 2, 5
PULSE, C, +, 5
PIN, GR, I, GLOBALRESET-, 24
PULSE, GR, -, 25
END
ENDMOD
END
SYM, D_D, CLB, LOC=AK, BLKNM=D_D
CFG, Base FG
CFG, Config F:E G:B X:QX Y:QY DX:F DY:G RSTDIR: CLK:K ENCLK:
CFG, Equate F=E
CFG, Equate G=B
PIN, E, I, LINE_D
PIN, B, I, LINE_C
PIN, X, O, D_D
PIN, Y, O, D_C
PIN, K, I, CLK_LINE
MODEL
SYM, D_D.X, BUF
PIN, I, I, D_D.QX, 0
PIN, O, O, D_D, 2
END
SYM, D_D.Y, BUF
PIN, I, I, D_D.QY, 0
PIN, O, O, D_C, 2
END
SYM, D_D.F, BUF
PIN, I, I, LINE_D, 1
PIN, O, O, D_D.F, 5
END
SYM, D_D.G, BUF
PIN, I, I, LINE_C, 3
PIN, O, O, D_D.G, 5
END
SYM, D_D.QX, DFF
PIN, Q, O, D_D.QX, 5
PIN, D, I, D_D.F, 0
PIN, C, I, CLK_LINE, 3
SETUP, D, C, +, 2, 5
PULSE, C, +, 5
PIN, GR, I, GLOBALRESET-, 24
PULSE, GR, -, 25
END
SYM, D_D.QY, DFF
PIN, Q, O, D_D.QY, 5
PIN, D, I, D_D.G, 0
PIN, C, I, CLK_LINE, 3
SETUP, D, C, +, 2, 5
PULSE, C, +, 5
PIN, GR, I, GLOBALRESET-, 24
PULSE, GR, -, 25
```

```
END
ENDMOD
END
SYM, GCLK, GCLK
PIN, I, I, CLK_IN, 0
PIN, O, O, CLK_LINE, 1
END
EXT, CLK_IN, I
EXT, GLOBALRESET-, I, 54
EOF
```

N5R.LCA

```
; LCA Design=nayan5 Part=3042pc84 -- Blocks=10 Nets=11.
; Program=APR Version=3.02 Date=Tue Jun 18 07:42:15 1991.
Design 3042pc84
Speed -100
Addnet CLK_IN TCLKIN.I GCLK.I
Netdelay CLK_IN GCLK.I 0.0
Addnet CLK_LINE GCLK.O AK.K AJ.K AL.K
Netdelay CLK_LINE AK.K 2.8 AJ.K 2.8 AL.K 2.8
Addnet D_A AJ.X AL.C
Netdelay D_A AL.C 1.9
Addnet D_B AJ.Y AL.D
Netdelay D_B AL.D 2.2
Addnet D_C AK.Y AL.A
Netdelay D_C AL.A 1.2
Addnet D_D AK.X AL.B
Netdelay D_D AL.B 0.0
Addnet LINE_A P78.I AJ.A
Netdelay LINE_A AJ.A 0.0
Addnet LINE_ABCD AL.Y P73.O
Netdelay LINE_ABCD P73.O 0.0
Addnet LINE_B P80.I AJ.B
Netdelay LINE_B AJ.B 2.1
Addnet LINE_C P79.I AK.B
Netdelay LINE_C AK.B 2.9
Addnet LINE_D P77.I AK.E
Netdelay LINE_D AK.E 1.4
NProgram AK.X:AL.B AJ.A:PAD19.I PAD25.O:AL.Y AJ.20.1.8 AJ.20.1.16 AJ.20.1.5
AJ.20.1.19 AK.20.1.5 AK.20.1.19 BL.20.1.9 BL.20.1.15 GCLK.I:TCLKIN.I
AL.20.1.11 AL.20.1.16 col.J.long.0:AJ.K col.K.long.0:AK.K col.L.long.0:AL.K
col.J.long.0:GCLK.O col.K.long.0:GCLK.O col.L.long.0:GCLK.O row.A.local.1:AK.B
col.L.local.3:AL.A row.A.local.4:AJ.B col.L.local.4:AL.D row.B.local.5:AL.C
col.K.local.5:AK.E col.L.local.3:AK.Y row.A.local.4:AJ.Y row.B.local.5:AJ.X
row.A.local.1:PAD18.I col.K.local.5:PAD20.I
NProgram row.A.local.4:PAD17.I
Nameblk P78 A_IN
Editblk P78
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk P73 ABCD_OUT
Editblk P73
Base IO
Config IN: OUT:O TRI:
Endblk
Nameblk P80 B_IN
Editblk P80
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk P79 C_IN
```

```
Editblk P79
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk AJ D_B
Editblk AJ
Base FG
Config G:B F:A Y:QY X:QX DY:G DX:F RSTDIR: CLK:K ENCLK:
Equate G = B
Equate F = A
Endblk
Nameblk AK D_D
Editblk AK
Base FG
Config F:E G:B X:QX Y:QY DX:F DY:G RSTDIR: CLK:K ENCLK:
Equate F = E
Equate G = B
Endblk
Nameblk P77 D_IN
Editblk P77
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk AL LINE_ABCD
Editblk AL
Base F
Config F:B:A:D:C Y:QY X: DY:F DX: RSTDIR: CLK:K ENCLK:
Equate F = (((B@A)@(A@D))@((A@D)@(D@C)))
Endblk
```

N5R.DAT

```
FILES ARE :  n5r.xnf        n5r.lca n5r.dat

PATH  0 :      D_D.QX ---> LINE_ABCD.DY   == 16.0 nsec
PATH  1 :      D_D.QY ---> LINE_ABCD.DY   == 17.2 nsec
PATH  2 :      D_B.QY ---> LINE_ABCD.DY   == 18.2 nsec
PATH  3 :      D_B.QX ---> LINE_ABCD.DY   == 17.9 nsec
PATH  4 :    A_IN.IBUF ---> D_B.DX        == 12.0 nsec
PATH  5 :    B_IN.IBUF ---> D_B.DY        == 14.1 nsec
PATH  6 :    D_IN.IBUF ---> D_D.DX        == 13.4 nsec
PATH  7 :    C_IN.IBUF ---> D_D.DY        == 14.9 nsec
PATH  8 :  LINE_ABCD.QY ---> ABCD_OUT.OBUF  == 32.0 nsec

LONGEST
PATH  8 :  LINE_ABCD.QY ---> ABCD_OUT.OBUF  == 32.0 nsec
```

APPENDIX C

TEST CASE # 2

NEW2.XNF

```
LCANET, 1
PROG, LCA2XNF, Ver. 3.01: File=nayan5_routed.lca Date=Mon Sep  9 08:57:57 1991 -
PART,3042pc84-100
PWR, 0, GND
PWR, 1, VCC
SYM, F_IN, IOB, LOC=P69, BLKNM=F_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_F
MODEL
SYM, F_IN.IBUF, IBUF
PIN, O, O, LINE_F, 4
PIN, I, I, F_IN, 0
END
ENDMOD
END
EXT, F_IN, I, 69
SYM, E_IN, IOB, LOC=P70, BLKNM=E_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_E
MODEL
SYM, E_IN.IBUF, IBUF
PIN, O, O, LINE_E, 4
PIN, I, I, E_IN, 0
END
ENDMOD
END
EXT, E_IN, I, 70
SYM, D_IN, IOB, LOC=P67, BLKNM=D_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_D
MODEL
SYM, D_IN.IBUF, IBUF
PIN, O, O, LINE_D, 4
PIN, I, I, D_IN, 0
END
ENDMOD
END
EXT, D_IN, I, 67
```

```
SYM, C_IN, IOB, LOC=P68, BLKNM=C_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_C
MODEL
SYM, C_IN.IBUF, IBUF
PIN, O, O, LINE_C, 4
PIN, I, I, C_IN, 0
END
ENDMOD
END
EXT, C_IN, I, 68
SYM, B_IN, IOB, LOC=P65, BLKNM=B_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_B
MODEL
SYM, B_IN.IBUF, IBUF
PIN, O, O, LINE_B, 4
PIN, I, I, B_IN, 0
END
ENDMOD
END
EXT, B_IN, I, 65
SYM, ABCD_OUT, IOB, LOC=P66, BLKNM=ABCD_OUT
CFG, Base IO
CFG, Config IN: OUT:O TRI:
PIN, O, I, LINE_ABCD
MODEL
SYM, ABCD_OUT.OBUF, OBUF
PIN, O, O, ABCD_OUT, 24
PIN, I, I, LINE_ABCD, 0
END
ENDMOD
END
EXT, ABCD_OUT, O, 66
SYM, N$571, CLB, LOC=GL, BLKNM=N$571
CFG, Base F
CFG, Config F:B:C Y:F X: DY: DX: RSTDIR: CLK: ENCLK:
CFG, Equate F=~(B+C)
PIN, Y, O, N$571
PIN, B, I, LINE_B
PIN, C, I, LINE_A
MODEL
SYM, N$571.Y, BUF
PIN, I, I, N$571.F, 0
PIN, O, O, N$571, 2
END
SYM, N$571.F, NOR
PIN, O, O, N$571.F, 5
PIN, 1, I, LINE_B, 1
PIN, 2, I, LINE_A, 0
```

```
END
ENDMOD
END
SYM, A_IN, IOB, LOC=P63, BLKNM=A_IN
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, LINE_A
MODEL
SYM, A_IN.IBUF, IBUF
PIN, O, O, LINE_A, 4
PIN, I, I, A_IN, 0
END
ENDMOD
END
EXT, A_IN, I, 63
SYM, LINE_ABCD, CLB, LOC=EL, BLKNM=LINE_ABCD
CFG, Base F
CFG, Config F:B:A:E:D:C Y:QY X: DY:F DX: RSTDIR: CLK:K ENCLK:
CFG, Equate F=~(~(~(B+A)+~(D+C))+~(~(D+C)+E))
PIN, E, I, N$571
PIN, B, I, LINE_F
PIN, A, I, LINE_E
PIN, D, I, LINE_D
PIN, C, I, LINE_C
PIN, Y, O, LINE_ABCD
PIN, K, I, CLK_LINE
MODEL
SYM, LINE_ABCD.Y, BUF
PIN, I, I, LINE_ABCD.QY, 0
PIN, O, O, LINE_ABCD, 2
END
SYM, LINE_ABCD.QY, DFF
PIN, Q, O, LINE_ABCD.QY, 5
PIN, D, I, LINE_ABCD.F, 0
PIN, C, I, CLK_LINE, 3
SETUP, D, C, +, 2, 5
PULSE, C, +, 5
PIN, GR, I, GLOBALRESET-, 24
PULSE, GR, -, 25
END
SYM, LINE_ABCD.F1, NOR
PIN, O, O, LINE_ABCD.F1, 0
PIN, 1, I, LINE_F, 1
PIN, 2, I, LINE_E, 3
END
SYM, LINE_ABCD.F2, NOR
PIN, O, O, LINE_ABCD.F2, 0
PIN, 1, I, LINE_D, 1
PIN, 2, I, LINE_C, 0
END
SYM, LINE_ABCD.F3, NOR
PIN, O, O, LINE_ABCD.F3, 0
```

```
PIN, 1, I, LINE_ABCD.F1, 0
PIN, 2, I, LINE_ABCD.F2, 0
END
SYM, LINE_ABCD.F4, NOR
PIN, O, O, LINE_ABCD.F4, 0
PIN, 1, I, LINE_D, 1
PIN, 2, I, LINE_C, 0
END
SYM, LINE_ABCD.F5, NOR
PIN, O, O, LINE_ABCD.F5, 0
PIN, 1, I, LINE_ABCD.F4, 0
PIN, 2, I, N$571, 3
END
SYM, LINE_ABCD.F, NOR
PIN, O, O, LINE_ABCD.F, 5
PIN, 1, I, LINE_ABCD.F3, 0
PIN, 2, I, LINE_ABCD.F5, 0
END
ENDMOD
END
SYM, GCLK, GCLK
PIN, I, I, CLK_IN, 0
PIN, O, O, CLK_LINE, 1
END
EXT, CLK_IN, I
EXT, GLOBALRESET-, I, 54
EOF
```

NEW2.LCA

```
; LCA Design=nayan5 Part=3042pc84 -- Blocks=11 Nets=10.
; Program=APR Version=3.02 Date=Mon Sep 9 08:57:56 1991.
Design 3042pc84
Speed -100
Addnet CLK_IN TCLKIN.I GCLK.I
Netdelay CLK_IN GCLK.I 0.0
Addnet CLK_LINE GCLK.O EL.K
Netdelay CLK_LINE EL.K 2.8
Addnet LINE_A P63.I GL.C
Netdelay LINE_A GL.C 0.0
Addnet LINE_ABCD EL.Y P66.O
Netdelay LINE_ABCD P66.O 2.2
Addnet LINE_B P65.I GL.B
Netdelay LINE_B GL.B 1.4
Addnet LINE_C P68.I EL.C
Netdelay LINE_C EL.C 0.0
Addnet LINE_D P67.I EL.D
Netdelay LINE_D EL.D 1.4
Addnet LINE_E P70.I EL.A
Netdelay LINE_E EL.A 2.9
Addnet LINE_F P69.I EL.B
Netdelay LINE_F EL.B 1.4
Addnet N$571 GL.Y EL.E
Netdelay N$571 EL.E 3.2
NProgram PAD33.I:EL.C PAD37.I:GL.C GM.20.1.0 GM.20.1.13 DM.20.1.1
DM.20.1.14 FM.20.1.4 FM.20.1.11 EM.20.1.0 EM.20.1.19 GCLK.I:TCLKIN.I
FM.20.1.14 FM.20.1.19 col.L.long.0:EL.K col.L.long.0:GCLK.O row.E.local.1:EL.A
row.F.local.1:EL.E row.E.local.4:EL.B row.G.local.4:GL.B row.F.local.4:EL.D
col.M.local.2:GL.Y col.M.local.5:EL.Y col.M.local.2:PAD30.I row.E.local.4:PAD32.I
row.F.local.4:PAD34.I row.G.local.4:PAD36.I col.M.local.4:PAD35.O
Nameblk P63 A_IN
Editblk P63
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk P66 ABCD_OUT
Editblk P66
Base IO
Config IN: OUT:O TRI:
Endblk
Nameblk P65 B_IN
Editblk P65
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk P68 C_IN
Editblk P68
Base IO
Config IN:I OUT: TRI:
Endblk
```

```
Nameblk P67 D_IN
Editblk P67
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk P70 E_IN
Editblk P70
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk P69 F_IN
Editblk P69
Base IO
Config IN:I OUT: TRI:
Endblk
Nameblk EL LINE_ABCD
Editblk EL
Base F
Config F:B:A:E:D:C Y:QY X: DY:F DX: RSTDIR: CLK:K ENCLK:
Equate F = ~(~(~(B+A)+~(D+C))+~(~(D+C)+E))
Endblk
Nameblk GL N$571
Editblk GL
Base F
Config F:B:C Y:F X: DY: DX: RSTDIR: CLK: ENCLK:
Equate F = ~(B+C)
Endblk
```

NEW2.DAT

```
FILES ARE :  new2.xnf      new2.lca      new2.dat

PATH  0 :     F_IN.IBUF  ---> LINE_ABCD.DY   == 13.4 nsec
PATH  1 :     E_IN.IBUF  ---> LINE_ABCD.DY   == 14.9 nsec
PATH  2 :     D_IN.IBUF  ---> LINE_ABCD.DY   == 13.4 nsec
PATH  3 :     C_IN.IBUF  ---> LINE_ABCD.DY   == 12.0 nsec
PATH  4 :     B_IN.IBUF  ---> LINE_ABCD.DY   == 27.6 nsec
PATH  5 :     A_IN.IBUF  ---> LINE_ABCD.DY   == 26.2 nsec
PATH  6 :  LINE_ABCD.QY  ---> ABCD_OUT.OBUF  == 34.2 nsec

LONGEST
PATH  6 :  LINE_ABCD.QY  ---> ABCD_OUT.OBUF  == 34.2 nsec
```

APPENDIX D

TEST CASE # 3

Partial Listing of LCA1.XNF

```
LCANET, 1
PROG, LCA2XNF, Ver. 3.01: File=lca1.lca Date=Wed Sep 18 05:38:31 1991 -
PART,3042pc84-100
PWR, 0, GND
PWR, 1, VCC
SYM, MATRIX, IOB, LOC=P66, BLKNM=MATRIX
CFG, Base IO
CFG, Config IN:I OUT: TRI:
PIN, I, O, N$454
MODEL
SYM, MATRIX.IBUF, IBUF
PIN, O, O, N$454, 4
PIN, I, I, MATRIX, 0
END
ENDMOD
END
EXT, MATRIX, I, 66
SYM, N$367<0>, CLB, LOC=GB, BLKNM=N$367<0>
CFG, Base F
CFG, Config F:B:D:E:A:C Y:F X: DY: DX: RSTDIR: CLK: ENCLK:
CFG, Equate
F=((B*~D)+(D*~((~(~E*~A*C)*~(~E*C)*~(~E*~A*C))*(~(E*~A*~C)*~(E*~C)*~(E*~
A*~C)))))
PIN, C, I, N$583<2>
PIN, A, I, N$583<1>
PIN, D, I, N$454
PIN, B, I, N$451<0>
PIN, Y, O, N$367<0>
PIN, E, I, N$359<0>
MODEL
SYM, N$367<0>.Y, BUF
PIN, I, I, N$367<0>.F, 0
.
.
.
.
.
.
.
.
.
```

```
.
.
.
.
.
.
.
.
.
SYM, $2-C<1>.F5, NAND
PIN, O, O, $2-C<1>.F5, 0
PIN, 1, I, QX<3>, 4, INV
PIN, 2, I, QX<4>, 4, INV
END
SYM, $2-C<1>.F6, NAND
PIN, O, O, $2-C<1>.F6, 0
PIN, 1, I, QX<3>, 4, INV
PIN, 2, I, IX<1>, 5
PIN, 3, I, N$328<1>, 2
END
SYM, $2-C<1>.F7, NAND
PIN, O, O, $2-C<1>.F7, 0
PIN, 1, I, QX<3>, 4, INV
PIN, 2, I, QX<4>, 4, INV
PIN, 3, I, IX<1>, 5
END
SYM, $2-C<1>.F8, AND
PIN, O, O, $2-C<1>.F8, 0
PIN, 1, I, $2-C<1>.F5, 0
PIN, 2, I, $2-C<1>.F6, 0
PIN, 3, I, $2-C<1>.F7, 0
END
SYM, $2-C<1>.F, NAND
PIN, O, O, $2-C<1>.F, 5
PIN, 1, I, $2-C<1>.F1, 0
PIN, 2, I, $2-C<1>.F2, 0
PIN, 3, I, $2-C<1>.F3, 0
PIN, 4, I, $2-C<1>.F4, 0
PIN, 5, I, $2-C<1>.F8, 0
END
ENDMOD
END
EXT, GLOBALRESET-, I, 54
EOF
```

Partial Listing of LCA1.LCA

```
; LCA Design=iqmatrix Part=3042pc84 -- Blocks=118 Nets=136.
; Program=APR Version=3.02 Date=Fri Jun 21 08:40:50 1991.
Design 3042pc84
Speed -100
Addnet $2-C<1> FB.X GC.B FC.B EC.C
Netdelay $2-C<1> GC.B 1.6 FC.B 0.3 EC.C 2.2
Addnet $2-C<3> EC.X ED.B DD.D DE.C
Netdelay $2-C<3> ED.B 0.3 DD.D 2.8 DE.C 3.2
Addnet $2-CA5 DE.X DF.B
Netdelay $2-CA5 DF.B 0.0
Addnet $2-CA7 CF.X BH.E CH.C
Netdelay $2-CA7 BH.E 2.7 CH.C 2.4
Addnet $2-CA9 BH.X CI.B BI.B
Netdelay $2-CA9 CI.B 1.4 BI.B 0.1
Addnet $2-CG5 CD.Y CE.E DE.B
Netdelay $2-CG5 CE.E 1.2 DE.B 4.1
Addnet $2-CP5 CD.X CE.B DE.A
Netdelay $2-CP5 CE.B 0.1 DE.A 1.3
Addnet $3-N$126 BG.Y BI.A
Netdelay $3-N$126 BI.A 2.2
Addnet $3-N$134 AH.X BI.E
Netdelay $3-N$134 BI.E 2.0
.
.
.
.
.
.
.
.
.
.
Config IN:FF:IQ OUT: TRI:
Endblk
Nameblk P3 Q<7>
Editblk P3
Base IO
Config IN:FF:IQ OUT: TRI:
Endblk
Nameblk P6 Q<8>
Editblk P6
Base IO
Config IN:FF:IQ OUT: TRI:
Endblk
Nameblk P80 Q<9>
Editblk P80
Base IO
Config IN:FF:IQ OUT: TRI:
Endblk
```

Partial Listing of LCA1.DAT

```
FILES ARE :  lca1.xnf        lca1.lca        lca1.dat

PATH  0 :     I<9>.INFF ---> N$451<9>.DX    == 33.8 nsec
PATH  1 :     I<8>.INFF ---> N$451<9>.DY    == 38.6 nsec
PATH  2 :     Q<0>.INFF ---> N$359<1>.DX    == 26.5 nsec
PATH  3 :     Q<1>.INFF ---> N$359<1>.DY    == 26.4 nsec
PATH  4 :     I<4>.INFF ---> N$451<5>.DX    == 33.1 nsec
PATH  5 :     I<5>.INFF ---> N$451<5>.DY    == 39.2 nsec
PATH  6 :     Q<6>.INFF ---> N$359<7>.DX    == 35.8 nsec
PATH  7 :     Q<7>.INFF ---> N$359<7>.DY    == 37.4 nsec
PATH  8 :     I<0>.INFF ---> N$451<1>.DX    == 27.9 nsec
PATH  9 :     I<1>.INFF ---> N$451<1>.DY    == 30.4 nsec
PATH  10 :    Q<4>.INFF ---> N$583<1>.DX    == 29.0 nsec
PATH  11 :    I<1>.INFF ---> N$583<1>.DX    == 30.4 nsec
PATH  12 :    Q<3>.INFF ---> N$583<1>.DX    == 30.1 nsec
PATH  13 :    I<2>.INFF ---> N$583<1>.DX    == 45.9 nsec
PATH  14 :    I<0>.INFF ---> N$583<1>.DX    == 40.7 nsec
PATH  15 :    I<1>.INFF ---> N$583<1>.DX    == 43.2 nsec
PATH  16 :    I<7>.INFF ---> N$451<7>.DX    == 40.1 nsec
PATH  17 :    I<6>.INFF ---> N$451<7>.DY    == 35.7 nsec
PATH  18 :    Q<4>.INFF ---> N$359<5>.DX    == 29.0 nsec
PATH  19 :    Q<5>.INFF ---> N$359<5>.DY    == 30.4 nsec
PATH  20 :    Q<8>.INFF ---> N$359<9>.DX    == 35.5 nsec
.
.
.
.
.
.
.
PATH 1847 :   N$583<5>.QY ---> IRY<5>.OUTFF   == 82.7 nsec
PATH 1848 :   N$359<3>.QX ---> IRY<5>.OUTFF   == 80.3 nsec
PATH 1849 :   N$359<1>.QY ---> IRY<5>.OUTFF   == 86.0 nsec
PATH 1850 :   N$583<4>.QY ---> IRY<5>.OUTFF   == 87.3 nsec
PATH 1851 :   N$583<3>.QY ---> IRY<5>.OUTFF   == 90.9 nsec
PATH 1852 :   N$359<3>.QX ---> IRY<5>.OUTFF   == 79.0 nsec
PATH 1853 :   N$359<1>.QY ---> IRY<5>.OUTFF   == 84.7 nsec
PATH 1854 :   N$583<4>.QY ---> IRY<5>.OUTFF   == 86.0 nsec
PATH 1855 :   N$583<3>.QY ---> IRY<5>.OUTFF   == 89.6 nsec
PATH 1856 :   N$583<4>.QY ---> IRY<5>.OUTFF   == 86.0 nsec
PATH 1857 :   N$359<1>.QY ---> IRY<1>.OUTFF   == 59.6 nsec
PATH 1858 :   N$583<3>.QY ---> IRY<1>.OUTFF   == 64.5 nsec
PATH 1859 :   N$359<1>.QX ---> IRY<1>.OUTFF   == 70.1 nsec
PATH 1860 :   N$583<2>.QX ---> IRY<1>.OUTFF   == 67.3 nsec
PATH 1861 :   N$583<2>.QX ---> IRY<1>.OUTFF   == 67.3 nsec

LONGEST
PATH 630 :    I<2>.INFF ---> N$583<10>.DX    == 138.6 nsec
```

VITA

Nayankumar Dineshchandra Patel was born in Jambusar, India on June 10, 1968. After graduating from Farragut High School in 1986 with honors, he entered the University of Tennessee at Knoxville for his undergraduate studies. He was awarded a Bachelor of Science degree in Electrical Engineering in May of 1990. In August of 1990 he rejoined the University of Tennessee for the graduate program in Electrical Engineering. He was awarded a Master of Science degree in Electrical Engineering in May of 1992.

His technical work experience includes working for approximately a year and a half for Philips Consumer Electronics Company (PCEC) in Knoxville as an Electrical Engineering intern. He was responsible for trouble shooting television chassis as well as testing televisions for electromagnetic interference (EMI), line radiation, and power supply problems. He also developed automated data acquisition software for voltage, current, and temperature readings. His final work at PCEC included the development of automated software for improved timing analysis of Field-Programmable Gate Arrays.

The author's permanent mailing address is 1724 Dunraven Drive, Knoxville, Tennessee 37922.

This thesis was typed by the author using the MacWrite and SuperPaint software available for Macintosh computers.