

To the Graduate Council:

I am submitting herewith a master thesis written by Akhil Bagri entitled "AUTOMATED EVALUATION AND FPGA IMPLEMENTATION OF VHDL-BASED DESIGNS." 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:

Robert R. Bouldin

Mohan M. Trivedi

Accepted for the Council:

C. W. Mink

Associate Vice Chancellor
and Dean of The Graduate School

**AUTOMATED EVALUATION AND FPGA
IMPLEMENTATION OF VHDL-BASED DESIGNS**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Akhil Bagri

August 1994

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. I am thankful to him for serving as the Major Professor on my committee.

Thanks also goes to my committee members, Dr. Robert E. Bodenheimer and Dr. Mohan M. Trivedi, for their comments and suggestions in the final stages of the thesis.

I am grateful to the University of Tennessee Computing Center for providing financial support during my graduate studies. Acknowledgements are due to my supervisors at UTCC, Mr. Bruce Delaney and Ms. Sue Smith for their support and understanding. Thanks goes to Mr. Alan Luchuk at UTCC, for his consistent support in writing this thesis.

A very special thanks to my family members for providing me with constant and unshakable support throughout my life.

ABSTRACT

Field Programmable Gate Arrays (FPGAs) offer the benefits of both programmable logic arrays and gate arrays. This has resulted in tremendous growth in their market, leading to the evolution of various series of FPGAs. The variety of FPGAs has created a need for software to find the best available FPGA for a particular application. This thesis describes a procedure for determining the FPGA best suited for a design in terms of utilization. The procedure automatically evaluates and implements VHDL-based designs on the FPGAs available from different vendors. The results of the evaluation are summarized in a file. This procedure enhances the productivity of the designer. The software was written using C-shell scripts and the C programming language. It uses Viewlogic toolsets and runs on a Sun workstation.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
1.1 Overview	1
1.1.1 What is an FPGA?	1
1.2 The Problem	3
1.3 Chapter Contents	6
2. COMMERCIAL FPGAs	8
2.1 Programming Technologies	8
2.2 Commercially Available FPGAs	10
2.3 The Xilinx FPGAs	13
2.3.1 The XC3000 Series	13
2.3.2 The XC4000 Series	16
2.4 The Actel FPGAs	19
2.4.1 The ACT 1 Series	19
3. THE DEVELOPMENT OF THE SOFTWARE	21
3.1 Concept and Design Goals of the software	21
3.2 Generic Algorithm for the Software	21
3.3 Software Requirements	24
3.4 Xilinx Series	25

CHAPTER	PAGE
3.5 Actel Series	27
4. USING THE SOFTWARE AND RESULTS	31
4.1 Tutorial	31
4.2 Results	32
4.2.1 Test Case # 1	32
4.2.2 Test Case # 2	37
4.2.3 Test Case # 3	41
5. SUMMARY, CONCLUSION AND FUTURE WORK	46
5.1 Summary and Conclusion	46
5.2 Future Work	47
BIBLIOGRAPHY	49
APPENDIXES	51
A. VARIOUS C-SHELL SCRIPTS	52
B. VARIOUS C PROGRAMS	67
C. VHDL code for test case # 1	92
VITA	110

LIST OF FIGURES

FIGURE	PAGE
1.1 <i>A Conceptual FPGA.</i>	2
2.1 <i>The Four Classes of Commercially Available FPGAs.</i>	12
2.2 <i>The X3000 CLB.</i>	15
2.3 <i>The X4000 CLB.</i>	18
2.4 <i>The ACT 1 Logic Module.</i>	20
3.1 <i>Design Flow for the Generic Algorithm.</i>	22
3.2 <i>Design Flow for the Xilinx X4000 Series.</i>	26
3.3 <i>Design Flow for the Xilinx X3000 Series.</i>	28
3.4 <i>Design Flow for the Actel Series.</i>	29
4.1 <i>Top-Level Schematic of Card-Dealer.</i>	33
4.2 <i>Report generated by the software for test case # 1.</i>	34
4.3 <i>Report generated by the software for test case # 2.</i>	37
4.4 <i>Top-Level Schematic of Huffman Encoding Algorithm.</i>	38
4.5 <i>Top-Level Schematic of Binary Image Segmenter.</i>	42
4.6 <i>Report generated by the software for test case # 3.</i>	43

LIST OF TABLES

TABLE	PAGE
1.1 <i>XC3000 Family Members</i>	5
1.2 <i>XC4000 Family Members</i>	5
1.3 <i>ACTEL Family Members</i>	6
2.1 <i>Characteristics of Programming Technologies</i>	10
2.2 <i>Summary of Commercially Available FPGAs</i>	11
4.1 <i>The X4000 Series Comparision for test case #1</i>	35
4.2 <i>The X3000 Series Comparision for test case #1</i>	36
4.3 <i>The Actel ACT1 Series Comparision for test case #1</i>	36
4.4 <i>The X4000 Series Comparision for test case #2</i>	39
4.5 <i>The X3000 Series Comparision for test case #2</i>	40
4.6 <i>The Actel ACT1 Series Comparision for test case #2</i>	40
4.7 <i>The X4000 Series Comparision for test case #3</i>	44
4.8 <i>The X3000 Series Comparision for test case #3</i>	44
4.9 <i>The Actel ACT1 Series Comparision for test case #3</i>	45

CHAPTER 1

INTRODUCTION

1.1 Overview

In the electronics industry it is vital to reach the market with new products in the shortest possible time, and so reduced development and production time is essential. Furthermore, it is important that the financial risk incurred in the development of a new product be limited so that more new ideas can be prototyped. Field-Programmable Gate Arrays (FPGAs) have emerged as a very attractive solution to these time-to-market and risk problems because they provide instant manufacturing and very low-cost prototypes. An FPGA can be configured in only minutes and prototype costs are on the order of \$100. A field-programmable device is a device in which the final logic structure can be directly configured by the end user without the use of an integrated circuit fabrication facility.

1.1.1 What is an FPGA?

Figure 1.1 shows a conceptual diagram of a typical FPGA which consists of a two-dimensional array of logic blocks that can be connected by general interconnection resources. The *interconnect* is comprised of segments of wire, where the

segments may be of various lengths. Present in the interconnect are the programmable switches that serve to connect the logic blocks to the wire segments, or one wire segment to the another. Logic circuits are implemented in the FPGA by partitioning the logic into the individual logic blocks and then interconnecting the blocks as required via the switches.

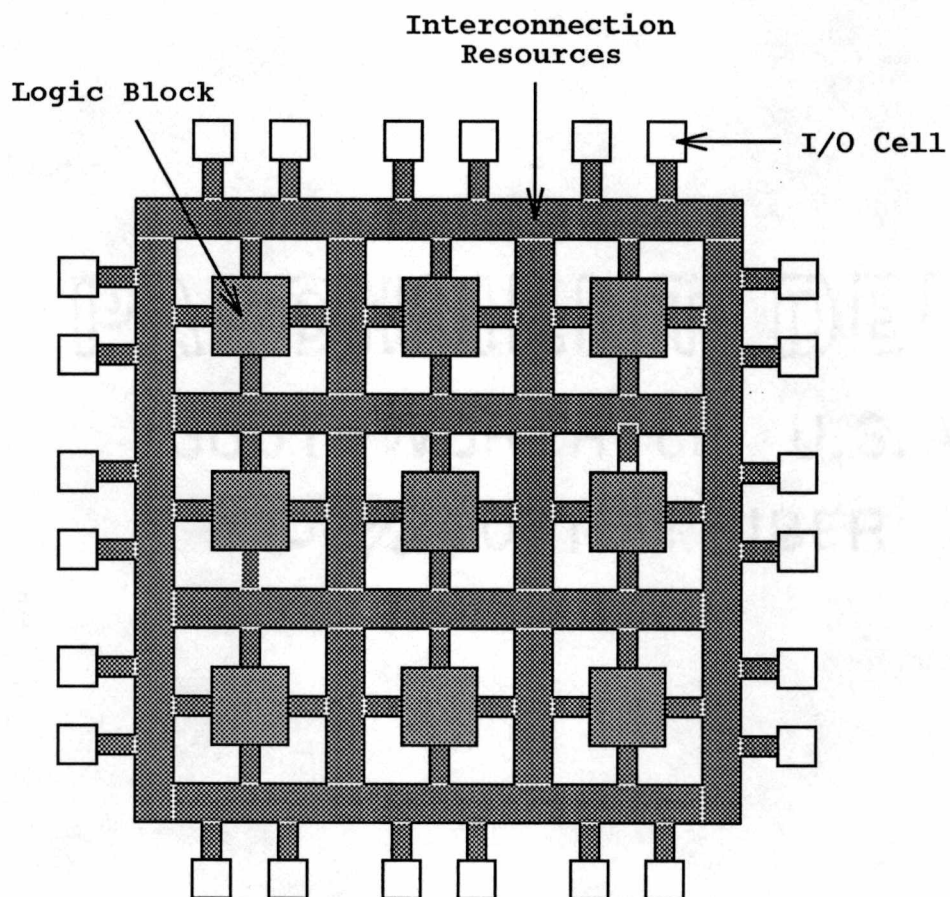


Figure 1.1: A Conceptual FPGA.

Logic Blocks

The structure and contents of the logic blocks are called its *architecture*. Logic block architectures can be designed in many different ways. Some FPGA logic blocks are as simple as 2-input NAND gates. Other blocks are more complex, such as multiplexers or lookup tables. There exists a myriad of possibilities for defining the logic block as a more complex circuit, consisting of several sub-circuits and having more than one output. Most logic blocks also contain some type of flip-flop to aid in the implementation of sequential circuits.

Interconnection Resources

The structure and content of the interconnect in an FPGA is called its *routing architecture*. The routing architecture consists of both wire segments and programmable switches. The programmable switches can be constructed in several ways, including: pass-transistors controlled by static RAM cells, anti-fuses, EPROM transistors, and EEPROM transistors. Similar to the logic blocks, there exist many different ways to design the structure of a routing architecture.

1.2 The Problem

Over the last few years, several companies have introduced a number of different types of FPGAs. While the features offered in each company's product differ somewhat, a user's logic circuit can generally be implemented in any class of FPGA by making use of a set of sophisticated CAD tools. Some tools are

developed specifically by the FPGA manufacturer, and others are offered through third-party vendors.

The last three years have seen FPGAs grow from a tiny market niche into a \$200 million business. It is expected that almost one billion dollars worth of FPGAs will be sold every year by 1996, representing a significant proportion of the IC market. This has led to the evolution of a number of FPGA companies. Xilinx, Actel, Altera, Advanced Micro Devices (AMD), QuickLogic, Algotronix, Plessey and Plus are some of them. The fierce competition among them to develop new products to capture the major share of the market has brought a variety of FPGAs to market.

Xilinx calls its XC3000 series a second generation family of CMOS Logic Cell Arrays that includes five compatible members with logic densities from 2000 to 9000 gates. Some other parameters of the various XC3000 family products are shown in the Table 1.1 [13].

Xilinx XC4000 series is a third generation family of CMOS logic cell arrays. It includes ten compatible members with logic densities from 2000 to 20000 gates. Some of the parameters of XC4000 family members are depicted in Table 1.2 [13].

ACTEL calls its ACT1 family as the first generation of field programmable gate arrays. Its members have logic densities of 1200 and 2000 gates. The ACT2 family represents ACTEL's second generation of field programmable gate arrays. It includes three compatible members with logic densities from 2500 to 8000 gates. Table 1.3 [1] illustrates the ACT1 and ACT2 family profiles.

Table 1.1: *XC3000 Family Members*

	<i>XC3020</i>	<i>XC3030</i>	<i>XC3042</i>	<i>XC3064</i>	<i>XC3090</i>
Equiv. Gates	2000	3000	4200	6400	9000
CLB Matrix	64	100	144	224	320
Latches and Flip-Flops	256	360	480	688	928
Input-Outputs	64	80	96	120	144

Table 1.2: *XC4000 Family Members*

	<i>XC4002</i>	<i>XC4003</i>	<i>XC4004</i>	<i>XC4005</i>	<i>XC4006</i>
Equiv. Gates	2000	3000	4000	5000	6000
CLB Matrix	8 x 8	10 x 10	12 x 12	14 x 14	16 x 16
CLB	64	100	144	196	256
Max. RAM Bits	2048	3200	4608	6272	8192
Input-Outputs	64	80	96	112	128

	<i>XC4008</i>	<i>XC4010</i>	<i>XC4013</i>	<i>XC4016</i>	<i>XC4020</i>
Equiv. Gates	8000	10,000	13,000	16,000	20,000
CLB Matrix	18 x 18	20 x 20	24 x 24	26 x 26	30 x 30
CLB	324	400	576	676	900
Max. RAM Bits	10,368	12,800	18,432	21,632	28,800
Input-Outputs	144	160	192	208	240

Table 1.3: *ACTEL Family Members*

	<i>ACT1</i>		<i>ACT2</i>		
	<i>A1010A</i>	<i>A1020A</i>	<i>A1225</i>	<i>A1240</i>	<i>A1280</i>
Equiv. Gates	1200	2000	2500	4000	8000
Logic Modules	295	547	451	684	1232
Max. Flip-Flops	147	273	382	565	998
Max. User I/Os	57	69	83	104	140

The variety of FPGAs has created a need for software which can evaluate a design and then find the best available part to suit a particular application. If this software can report the best available parts in different FPGA families of different vendor, for a given application, it would be an added advantage. This can help a user in comparing commercially available FPGAs. Some of the grounds of comparison can be costs involved, resource utilization, critical delays, etc. The software would also help in enhancing the productivity of the designer.

1.3 Chapter Contents

This thesis is divided into five chapters, beginning with the introduction. Chapter II outlines FPGAs in general and briefly covers the Xilinx and Actel FPGA products, their floorplan and placement and routing.

The third chapter discusses the development of the software. Included in this chapter are the concept and design goals of the software as well as the algorithm of the software. Chapter IV has a short tutorial on using the software. It also analyzes the results of three designs in which the software developed was applied.

Chapter V summarizes the FPGAs and automated procedure along with future work that can be done to make the software more sophisticated and user friendly. The Appendix contains the various C-shell scripts and the C programs that comprise this software.

CHAPTER 2

COMMERCIAL FPGAs

2.1 Programming Technologies

Field Programmable Gate Arrays have programming elements to implement the programmable connections among the logic blocks. There are a number of different ways of implementing programming elements. Programming technologies that are currently in use in commercial products are static RAM cells, anti-fuses, EPROM transistors and EEPROM transistors. While each of these technologies are quite different, the programming elements all share the property of being configurable in one of two states: ON or OFF. A typical FPGA may contain more than 100,000 programming elements. The elements should have the following properties:

- the programming element should consume as little chip area as possible.
- the programming element should have a low ON resistance and a very high OFF resistance.
- the programming element should contribute low parasitic capacitance to the wiring resources to which it is attached, and

- it should be possible to fabricate a large number of programming elements reliably on a single chip.

Depending on the application in which the FPGA is to be used, it may also be desirable for the programming element to possess other features. For example, a programming element that is non-volatile might be attractive, as well as an element that is re-programmable. Re-programmable elements make it possible to re-configure the FPGA, perhaps without even removing it from the circuit board. Finally, in terms of ease of manufacture, it might be desirable if the programming elements can be produced using a standard CMOS process technology.

Table 2.1 [4] lists some of the characteristics of the various programming technologies. The second column from the left gives an indication of whether or not the programming element is volatile, and the third column states whether the element is re-programmable. The amount of area consumed by the programming element is indicated, in relative terms, in the fourth column. While each company that produces FPGAs may use a different base CMOS process, the numbers shown in the table are all normalized to 1.2-micron CMOS. The fifth and sixth columns list the series resistance of the programming element (when in the ON state) and the capacitance that the element adds to each wire to which it is attached, respectively. The resistance and capacitance numbers are approximate and are meant to provide a relative measure for the various elements.

Table 2.1: *Characteristics of Programming Technologies*

<i>Programming Technology</i>	<i>Volatile</i>	<i>Re-Prog</i>	<i>Chip Area</i>	<i>R (ohm)</i>	<i>C (ff)</i>
Static RAM Cells	yes	in circuit	large	1-2 K	10-20 ff
PLICE Anti-fuse	no	no	small anti-fuse, large prog. trans.	300-500	3-5 ff
ViaLink Anti-fuse	no	no	small anti-fuse large prog. trans.	50-80	1.3 ff
EPROM	no	out of circuit	small	2-4 k	10-20 ff
EEPROM	no	in circuit	2 x EPROM	2-4 k	10-20 ff

2.2 Commercially Available FPGAs

Figure 2.1 [4] depicts the four main classes of FPGAs: symmetrical array, row-based, hierarchical PLD, and sea-of-gates. This section also provides descriptions of several commercially available FPGA families, including those from Xilinx, Actel, Altera, Plessey, Plus, Advanced Micro Devices (AMD), QuickLogic, Algotronix, Concurrent Logic and Crosspoint Solutions. Table 2.2 [4] provides a short summary of several key features for each FPGA. It gives an indication of the FPGA's architecture, the type of logic block used, the structure of the routing resources, and the type of programming technology.

Table 2.2: *Summary of Commercially Available FPGAs*

<i>Company</i>	<i>General Architecture</i>	<i>Logic Block Type</i>	<i>Programming Technology</i>
Xilinx	Symmetrical Array	Look-up Table	Static RAM
Actel	Row-based	Multiplexer- based	anti-fuse
Altera	Hierarchical-PLD	PLD Block	anti-fuse
Plessey	Sea-of-gates	NAND-gate	Static RAM
Plus	Hierarchical-PLD	PLD Block	EPROM
AMD	Hierarchical-PLD	PLD Block	EEPROM
QuickLogic	Symmetrical Array	Multiplexer- Based	anti-fuse
Algotronix	Sea-of-gates	Multiplexers & Basic Gates	Static RAM
Atmel	Sea-of-gates	Multiplexers & Basic Gates	Static RAM
Crosspoint	Row-based	Transistor Pairs & Multiplexers	anti-fuse

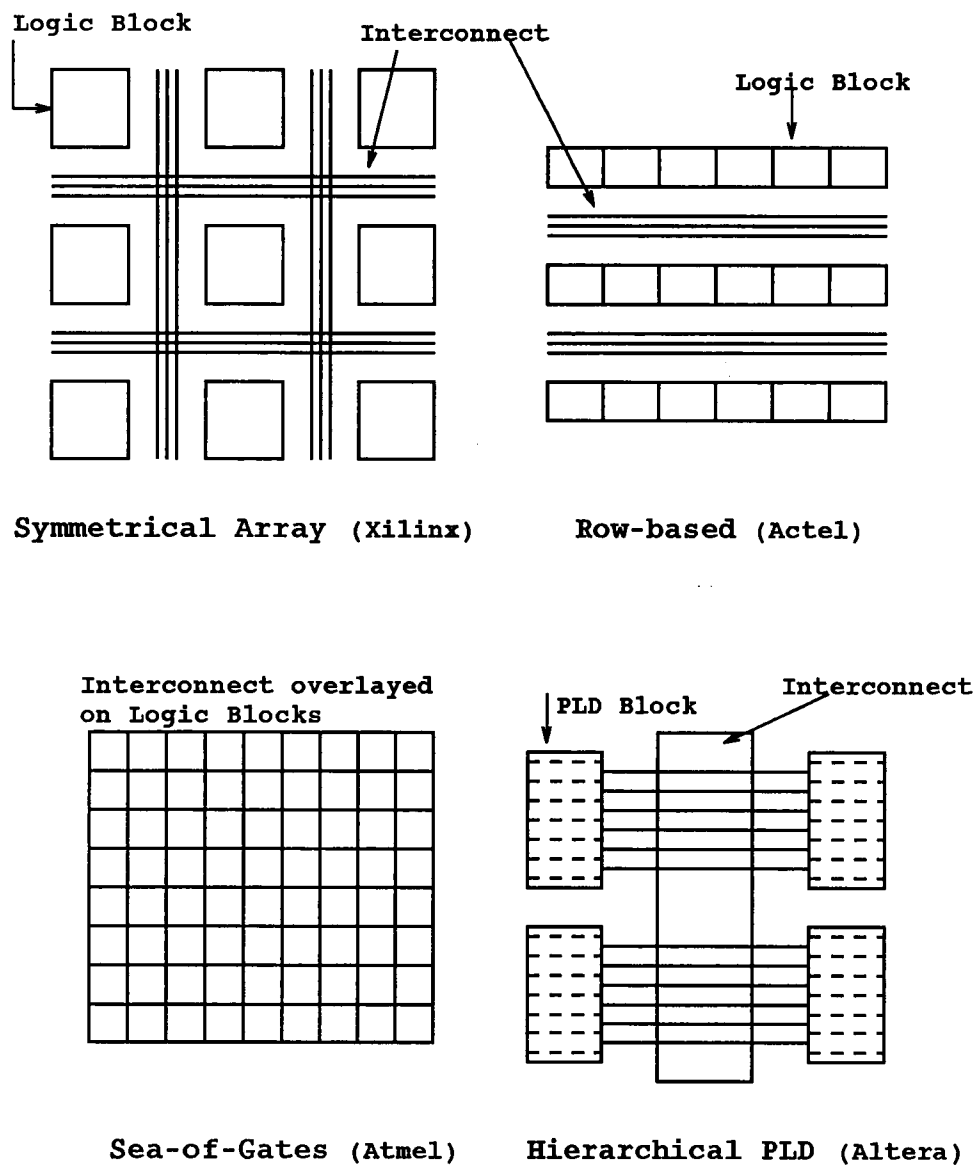


Figure 2.1: *The Four Classes of Commercially Available FPGAs.*

2.3 The Xilinx FPGAs

2.3.1 The XC3000 Series

The CMOS XC3000 Logic Cell Array family provides a group of high-performance, high-density, digital integrated circuits. Their regular, extendable, flexible, user-programmable array architecture is composed of a configuration program store plus three types of configurable elements: a perimeter of IOBs, a core array of CLBs and resources for interconnection. Some of its features are:

- High Performance 70-, 100- and 125-MHz Toggle Rates.
- Second Generation Field-Programmable Gate Array
 - I/O functions
 - Digital logic functions
 - Interconnections
- Flexible array architecture
 - Compatible arrays, 2000 to 9000 gate logic complexity
 - Extensive register and I/O capabilities
 - High fan-out signal distribution
 - Internal 3-state bus capabilities
 - TTL or CMOS input thresholds
 - On-chip oscillator amplifier

- Standard product availability
 - Low-power, CMOS, static memory technology
 - Performance equivalent to TTL SSI/MSI
 - 100% factory pre-tested
 - selectable configuration modes

As shown in Figure 2.2 [13], each configurable logic block has a combinatorial logic section, two flip-flops, and an internal control section. There are: five logic inputs [a, b, c, d and e]; a common clock input; an asynchronous direct reset input; and an enable clock. All may be driven from the interconnect resources adjacent to the blocks. Each CLB also has two outputs [x and y] which may drive interconnect networks.

The combinatorial-logic portion of the logic block uses a 32 by 1 look-up table to implement Boolean functions. Variables selected from the five logic inputs and two internal block flip-flops are used as table address inputs. The combinatorial propagation delay through the network is independent of the logic function generated and is spike free for the single input variable changes. This technique can generate two independent logic functions of up to four variables each, or a single function of five variables, or some functions of seven variables.

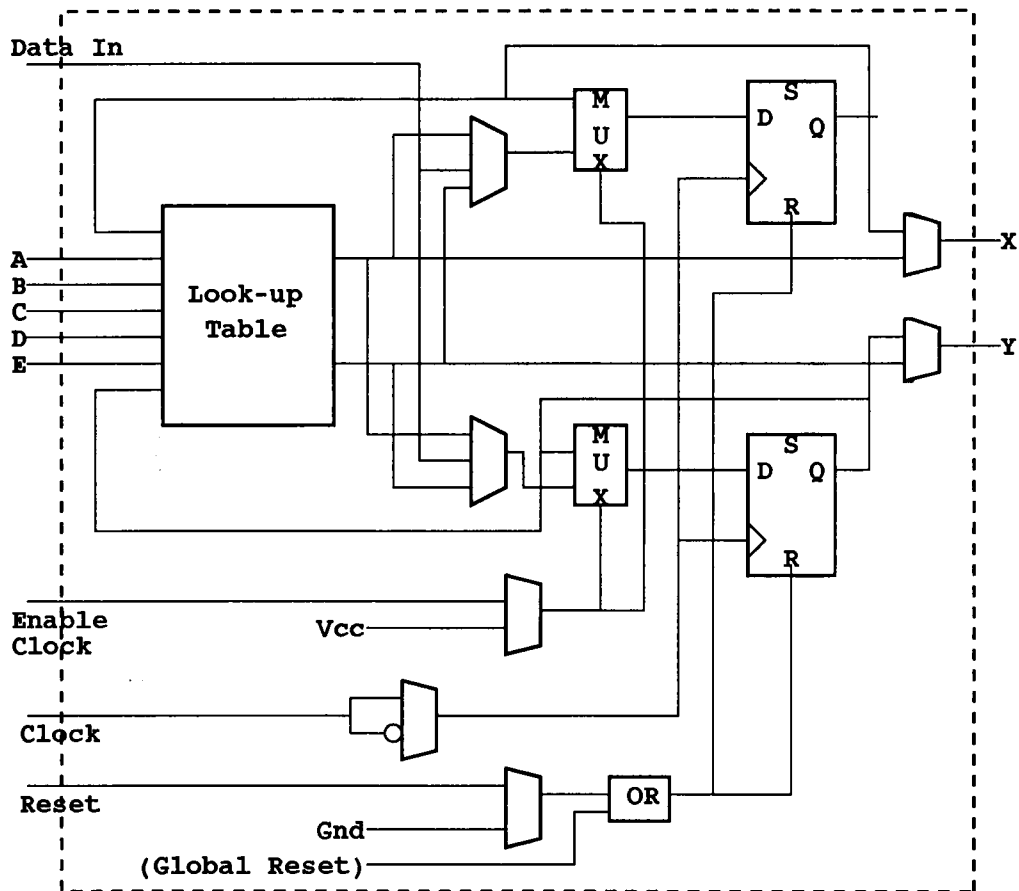


Figure 2.2: *The X3000 CLB.*

2.3.2 The XC4000 Series

The XC4000 family of Field-Programmable Gate Arrays provides the benefits of custom CMOS VLSI, while avoiding the initial cost, time delay, and inherent risk of a conventional masked gate array. The XC4000 family achieves high speed through advanced semiconductor technology and through improved architecture, and supports system clock rates of 60 to 70 MHz. Compared to older Xilinx FPGA families, the new family is more powerful, offering on-chip RAM and wide-input decoders. It is more versatile in its applications, and design cycles are faster due to a combination of increased routing resources and more sophisticated software. Also, it more than doubles the available complexity, up to the 20,000-gate level. Its features are:

- Third Generation Field-Programmable Gate Array
 - Abundant flip-flops
 - Flexible function generators
 - On-chip ultra-fast RAM
 - Dedicated high-speed carry propagation circuit
 - Fast, wide decoders
 - Efficient implementation of multi-level logic
 - Hierarchy of interconnect lines
 - Internal 3-state bus capability
- Flexible Array Architecture

- Programmable logic blocks and I/O blocks
 - Programmable interconnects and wide decoders
- Sub-micron CMOS Process
 - High-speed logic and interconnect
 - Low power consumption
- Configured by Loading Binary File
 - Unlimited re-programmability
 - Six programming modes

A number of architectural improvements contribute to the XC4000 family's increased logic density and performance levels. The most important one is a more powerful and flexible CLB surrounded by a versatile set of routing resources, resulting in more "effective gates per CLB." The principal CLB elements are shown in Figure 2.3 [13]. Each new CLB also packs a pair of flip-flops and two independent 4-input function generators. The two function generators offer designers plenty of flexibility because most combinatorial logic functions need less than four inputs. Consequently, the design software tools can deal with each function generator independently, thus improving cell usage.

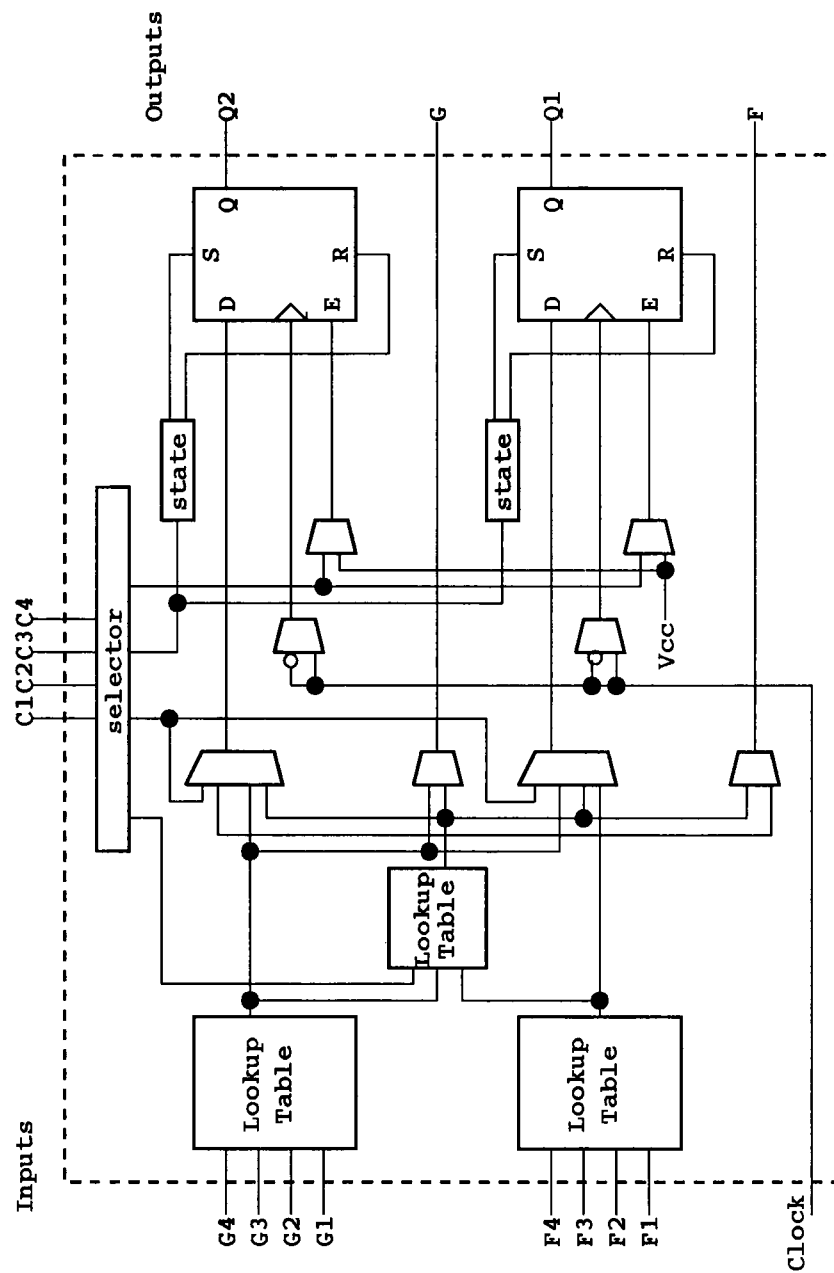


Figure 2.3: *The X4000 CLB.*

2.4 The Actel FPGAs

2.4.1 The ACT 1 Series

The ACT 1 family of field programmable gate arrays offers a variety of package, speed, and application combinations. Devices are implemented in silicon gate, 1-micron two-level metal CMOS, and they employ Actel's PLICE anti-fuse technology. The unique architecture offers gate array flexibility, high performance, and instant turnaround through user programming. Device utilization is typically 95 percent of available logic modules.

ACT 1 devices also provide system designers with unique on-chip diagnostic probe capabilities, allowing convenient testing and debugging. Additional features include an on-chip clock driver with a hardwired distribution network. The network provides efficient clock distribution with minimum skew.

The user-definable I/Os are capable of driving at both TTL and CMOS drive levels. Available packages include plastic and ceramic J-leaded chip carriers, ceramic and plastic quad flatpacks, and ceramic pin grid array.

A security fuse may be programmed to disable all further programming and to protect the design from being copied or reverse engineered.

Figure 2.4 [1] shows the ACT 1 logic module. It is an 8-input, one-output logic circuit chosen for the wide range of functions it implements and for its efficient use of interconnect routing resources.

The logic module can implement the four basic logic functions (NAND, AND, OR and NOR) in gates of two, three, or four inputs. Each function may have many

versions with different combinations of active-low inputs. The logic module can also implement a variety of D-latches, exclusivity functions, AND-ORs and OR-ANDs. No dedicated hardwired latches or flip-flops are required in the array, since latches and flip-flops may be constructed from logic modules whenever needed in the application.

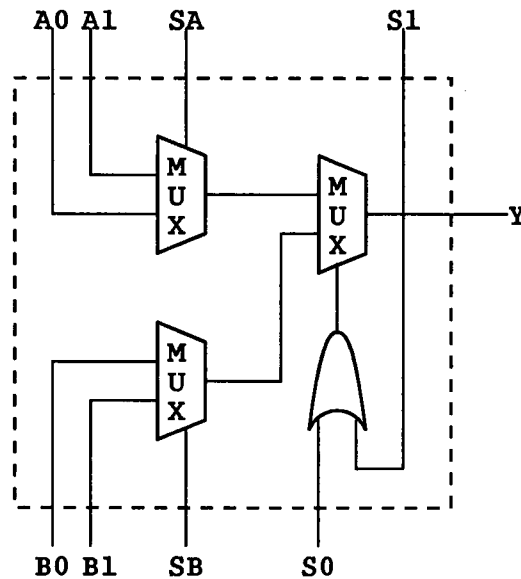


Figure 2.4: *The ACT 1 Logic Module.*

CHAPTER 3

THE DEVELOPMENT OF THE SOFTWARE

3.1 Concept and Design Goals of the software

The motivation for developing this software was to come up with an automated way of finding the best available part for different vendors for a given design. Also, the software should be such that it requires a minimum of human intervention. Once all the necessary files are supplied, it should furnish the designer with the final report. This is possible using shell scripts which are analogous to a user interface to UNIX. Scripts are useful for executing a series of individual commands. Also, a high-level programming language can be used to get into the file to make necessary changes and for comparisons and generating reports.

3.2 Generic Algorithm for the Software

The various steps involved in the design flow for the Generic Algorithm can be observed in Figure 3.1. The starting point for the design process is the initial logic entry of the circuit that is to be implemented. This step typically involves entering a VHDL description as it is independent of various technologies. This is usually translated into a standard form such as Boolean expressions. The Boolean expressions are then processed by a logic optimization tool, which manipulates the

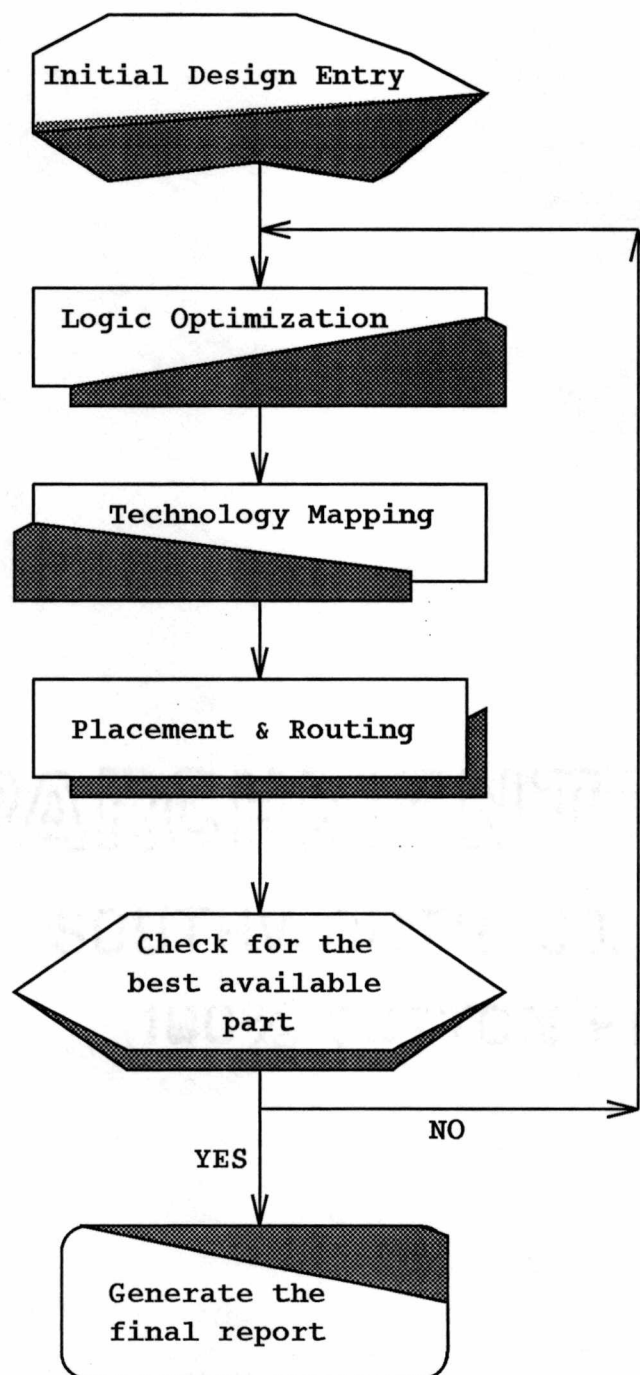


Figure 3.1: *Design Flow for the Generic Algorithm.*

expressions. The goal is to modify these expressions to optimize the area or speed of the final circuit. A combination of both area and delay requirements can also be considered. This optimization usually performs the equivalent of an algebraic minimization of the Boolean expressions and it is appropriate when implementing a logic circuitry.

The optimized Boolean expression must next be transformed into a circuit of FPGA logic blocks. This is done by a *technology mapping* program. The mapper may attempt to minimize the total number of blocks required, which is known as area optimization.

Having mapped the circuit into logic blocks, it is necessary to decide where to place each block in the FPGA's array. A *placement* program is used to solve this problem. The routing software assigns the FPGA's wire segments and chooses programmable switches to establish the required connections among the logic blocks. The routing software ensures that 100 percent of the required connections are formed.

Next, the software checks to see that it has selected the best available part in that technology. If not, then the best part is selected. The criteria for selecting the best available part is resource utilization. The one with the highest resource utilization is selected and the entire process from logic optimization is repeated. Finally, a report is generated to convey the findings to the designer.

3.3 Software Requirements

Listed below are the requirements to run the design to find out the best available part with respect to resource utilization in different technologies.

- The initial design entry should be in the form of a VHDL description only. This is because VHDL files can be technology independent and can be easily ported to different technologies. Drawing a schematic would involve components from the library of a particular vendor which might not be available in another vendor's library.
- The designer supplies the necessary VHDL files only. Care should be taken that the entity name is not the same in any two VHDL files. This normally happens when the designer copies the VHDL file to some other name but forgets to change the entity name.
- The software uses Viewlogic toolsets.
- The designer supplies the symbol for the top-level schematic. Various modules are connected in the top-level schematic with nets. All inputs/outputs have builtin:in.1 and builtin:out.1 components from the Viewlogic's builtin library.
- In the top-level schematic the external clock is labelled as "CLK". This is because the ACTEL series has a particular pin designated as the external clock in each of its parts. Keeping the same label helps in finding out the external clock pin in different designs.

3.4 Xilinx Series

Figure 3.2 shows the step-by-step procedure to obtain the best available part in Xilinx X4000 series. The criteria for selection is resource utilization.

- A shell script called *design* is used to set up the link for the viewdraw.ini file. The arguments to the script indicate which libraries to use in the viewdraw.ini file. Similarly, a link can be set for the techdef.vhd file in the behv subdirectory or else, it can be copied from some common area.
- Then next step involves compiling all the VHDL files in the behv subdirectory. These VHDL files are supplied by the designer. The wir file is generated for all the modules. This is done with the shell script *sample*. A UNIX command is used to pass the name of the VHDL files as arguments to the above shell script.
- A symbol is generated for the top-level schematic. A wir file (frame.1) is generated for the above symbol which doesn't have any nets connected to its pins. This wir file is used to create a wir file (chip.1) of the above symbol with i/o pads connected to it. A C program is used to create chip.1 .
- The wir file, chip.1, is changed to Xilinx Netlist Format using the *wir2xnf* utility. Then placement and routing is performed using the *xmake* utility. The largest part available in the X4000 series is selected to get an idea of resource utilization for the given design. This leads to the selection of the

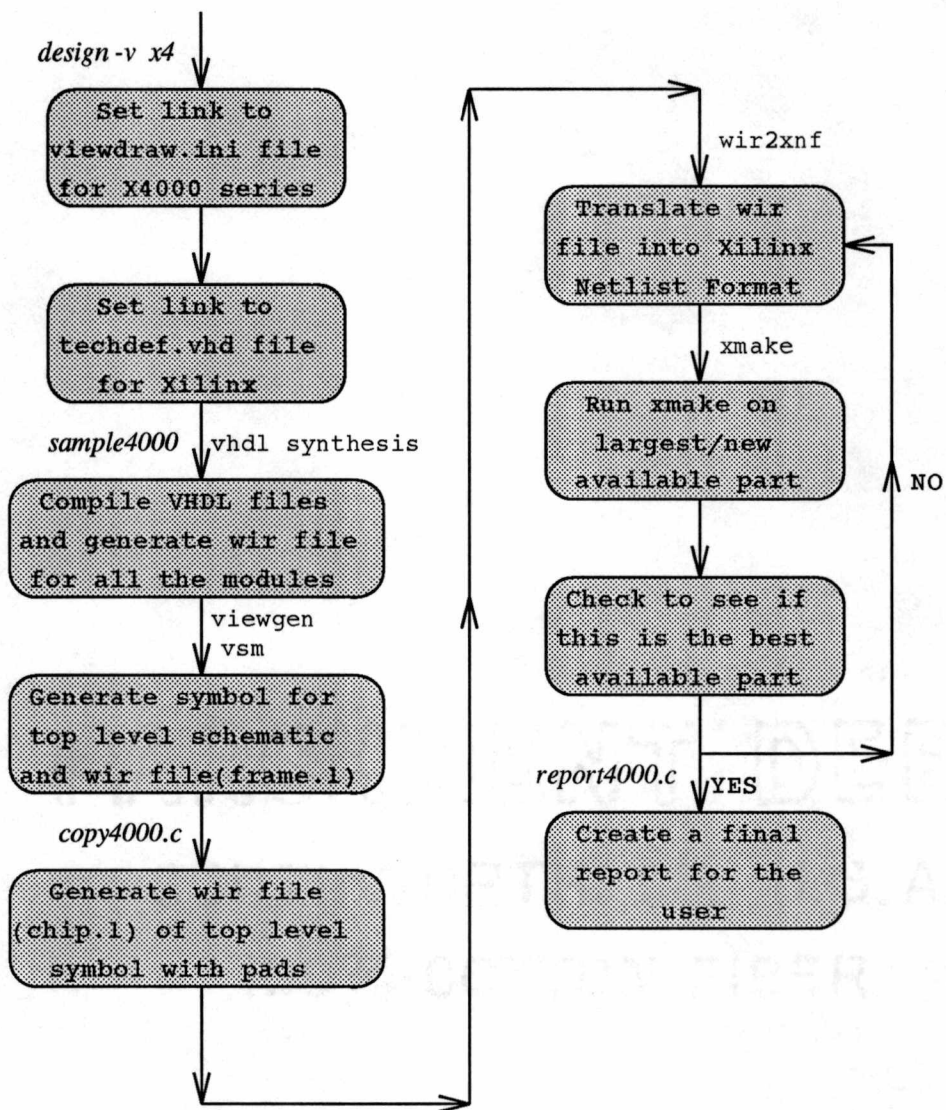


Figure 3.2: Design Flow for the Xilinx X4000 Series.

most suitable part in the X4000 series. The *wir2xnf* and *xmake* utilities are run again using this part.

- The final report is presented to the designer.

The logic used for selecting the best available part in the X3000 series is the same as the X4000 series except that the various programs and shell-scripts involved are different. Unlike X4000 series which uses the Partitioned Placement and Routing (ppr) program, X3000 series uses the Automatic Placement and Routing (apr) program. Figure 3.3 shows the procedure to obtain the best available part in Xilinx X3000 series.

3.5 Actel Series

The design flow to obtain the best available part in the ACT1 series of Actel for a given design is shown in Figure 3.4.

- As for Xilinx, a shell script called *design* is used to set up the link for the viewdraw.ini file. The arguments to the script indicate which libraries to use in the viewdraw.ini file. Similarly, a link can be set for the techdef.vhd file in the behv subdirectory or else, it can be copied from some common area.
- All the VHDL files in the behv subdirectory are compiled with the technology set to Actel. These VHDL files are supplied by the designer. The wir file is generated for all the modules. This is done with the shell script *sample*.

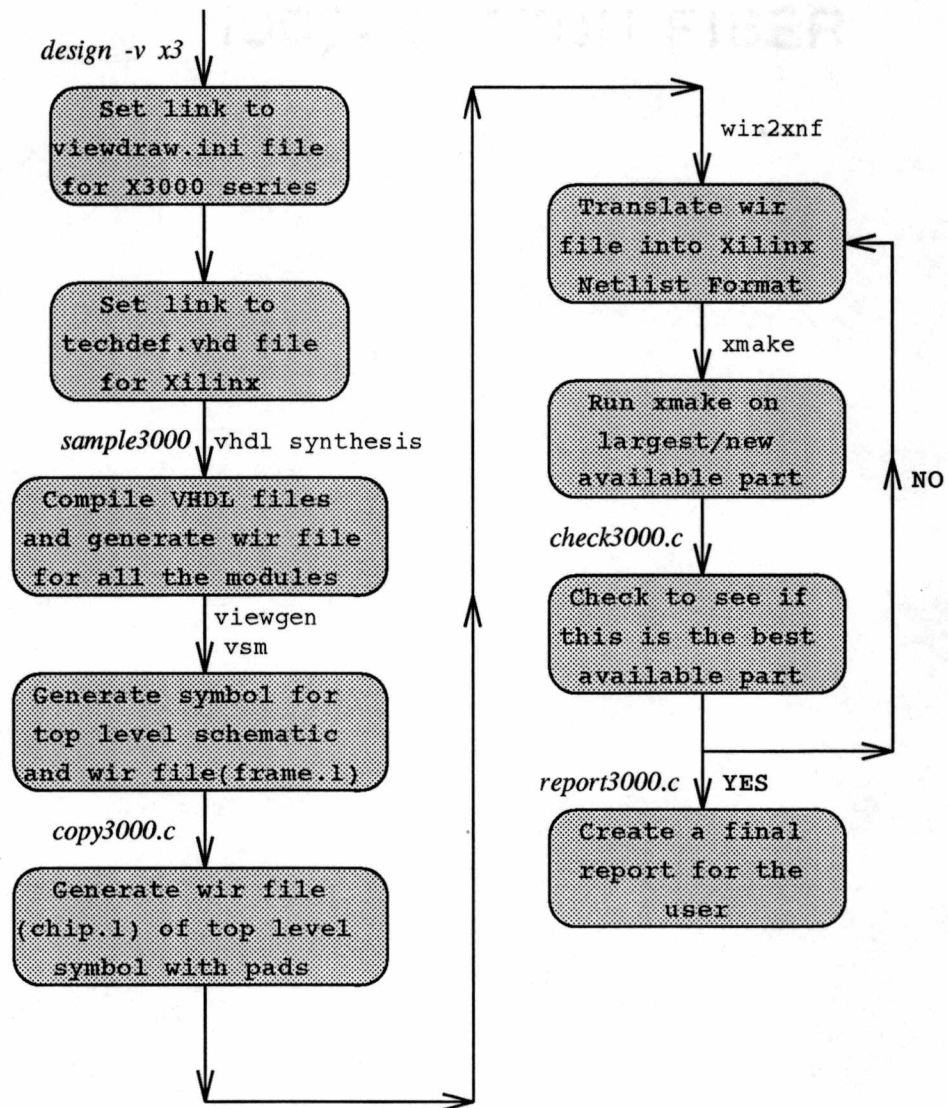


Figure 3.3: Design Flow for the Xilinx X3000 Series.

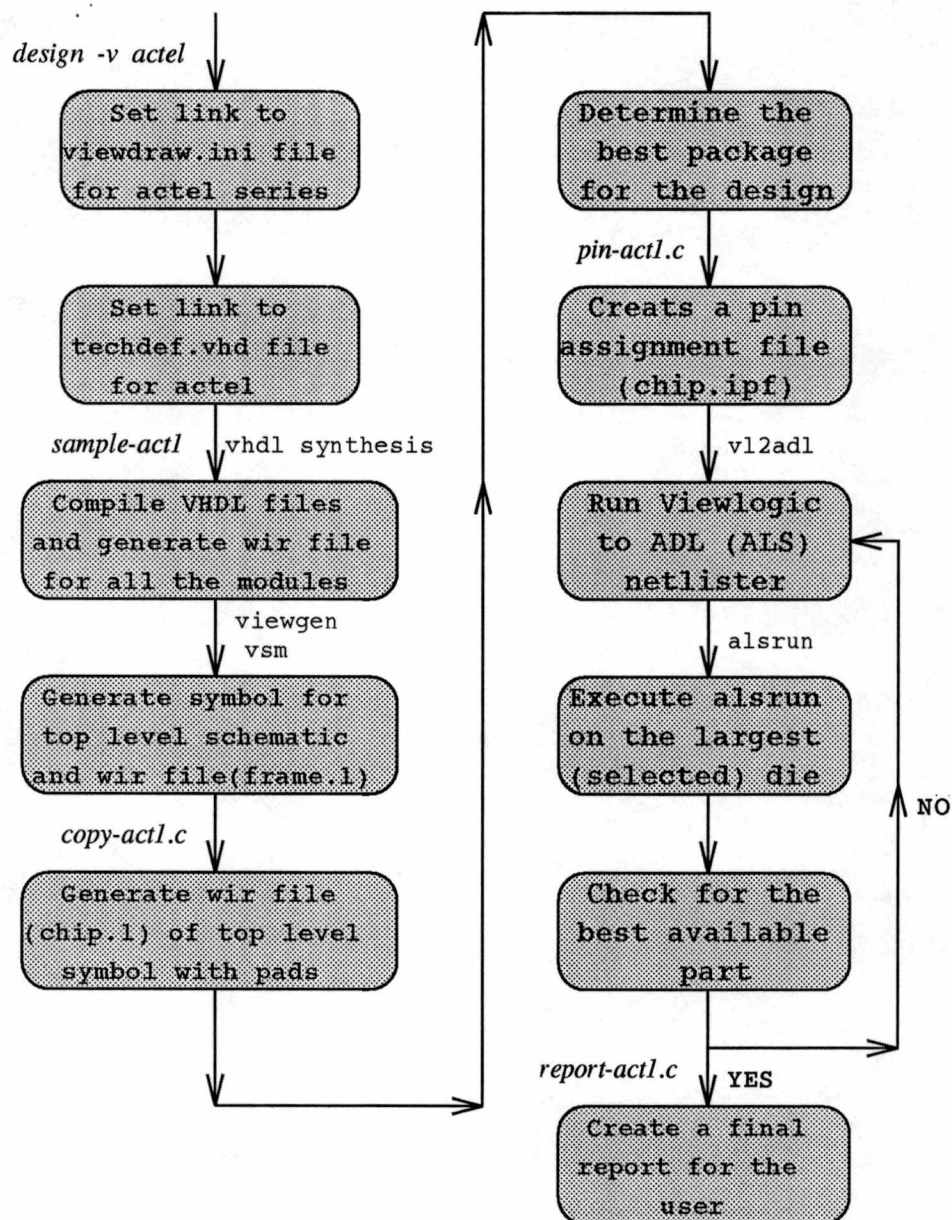


Figure 3.4: Design Flow for the Actel Series.

A UNIX command is used to pass the name of the VHDL files as arguments to the above shell script.

- A symbol is generated for the top-level schematic. A wir file (frame.1) is generated for the above symbol which doesn't have any nets connected to its pins. This wir file is used to create a wir file (chip.1) of the above symbol with i/o pads connected to it. A C program is used to create chip.1 .
- After the Viewlogic netlist is written, a target package is selected so that the maximum number of pins can be used in the selected package. Then, a pin assignment file called chip.ipf is created for that package with the help of a C program.
- The netlist file is then translated from Viewlogic format to Actel format using the *vl2adl* command.
- The largest available device is selected for the design. This gives an approximation of the resources the given design utilizes. A script called *alsrun* is used to perform automatic placement and routing of a design into the target device.
- The best part is selected using the data obtained from executing *alsrun* on the largest part. Necessary steps are repeated and a final report is created for the user.

CHAPTER 4

USING THE SOFTWARE AND RESULTS

A typical implementation of a design would require the designer to type various commands interactively. The designer also would have to connect manually a pad to each input and output of the top-level schematic by invoking CAD tools. While implementing a design in an Actel FPGA, the designer would have to assign pin numbers to all the inputs and outputs, by creating an .ipf file. The software we have developed can help the designer save his time and effort. Once the designer invokes the software by typing in a single command, it does the above tasks automatically. The software can be run in batch mode also. It enhances the productivity of the designer.

4.1 Tutorial

The various steps involved in performing automated evaluation and FPGA implementation of VHDL-based designs are:

- Make a directory (e.g. *temp*) in your home area. Make subdirectories *behv*, *wir*, *sch* in the *temp* directory.
- Copy all the VHDL code files into the *behv* subdirectory. Copy the top-level schematic with various modules connected into the *sch* subdirectory.

- Copy the C programs which generate the top-level wir file (chip.1) into the *wir* subdirectory. The rest of the C programs and shell scripts are copied into the *temp* directory.
- Make sure that the path specified to create links to the viewdraw.ini file and techdef.vhd file is correct.
- Finally, at the UNIX prompt in the *temp* directory, type *compare*. This action will execute a shell script which runs other shell scripts and C programs to evaluate and implement the given VHDL design.

4.2 Results

Many designs concerning the application of the software developed were successfully tested. Three of those designs will be discussed here. Some of the designs used in these test cases were developed by the author as part of his class project requirements.

4.2.1 Test Case # 1

The design of the card-dealer in a Black-Jack game is discussed here. Figure 4.1 shows the top-level schematic of this design with various modules connected. It consists of several functional blocks, each of which performs a particular task. A principal task is that of dealing cards randomly from the deck. To accomplish this, the PRBS module acts as a pseudo-random binary sequence generator which produces a sequence of ONE's and ZERO's.

Black-Jack Table Card-Dealer

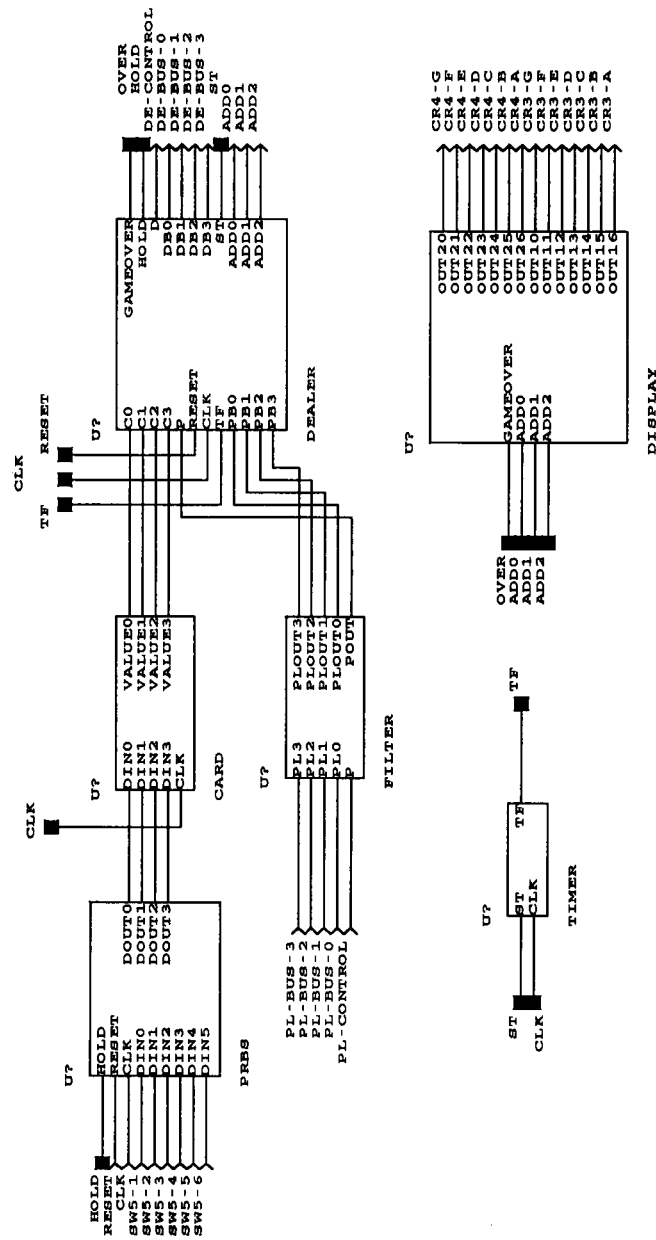


Figure 4.1: Top-Level Schematic of Card-Dealer.

The module CARD acts as a filter. It discards all values that are either 0 or greater than 13. Jack (11), Queen (12) and King (13) are passed as Ten (10). The DEALER module is the finite state machine (FSM). It assigns the cards to various players. It also sends the value of the dealer-player upcard to different players. If the player asks for another card, it assigns a new card to that player.

The DISPLAY module displays the address of the player with which the FSM is playing. The TIMER module keeps track of the time. If the player doesn't respond within 9 clock cycles, it moves to the next player. The FILTER module checks to see that the signal it is getting from the player is a valid signal.

The report generated by the software looks like this:

```

XILINX - 4000
-----
The test for Xilinx 4000 series is complete
The best available part in X4000 series for your design is 4003APC84
For more information check 4003APC84.rpt and 4003APC84.out

ACTEL - ACT1
-----
The test for ACT-1 series is complete
The best available part in Act1 series for your design is 1020alcc68
For more information check 1020alcc68.log

XILINX - 3000
-----
The test for Xilinx 3000 series is complete
The best available part in X3000 series for your design is 3042PC84
For more information check 3042PC84.rpt and 3042PC84.out
```

Figure 4.2: *Report generated by the software for test case # 1.*

Table 4.1: *The X4000 Series Comparision for test case #1*

	<i>Design Uses</i>	<i>4003</i>	<i>4002</i>	<i>4004</i>
Occupied CLBs	95	100	64	144
Packed CLBs	88	100	64	144
Flip Flops	22	360	256	480
Package Pins	35	84 (Min.)	84 (Min.)	84 (Min.)

Xilinx 4000

Table 4.1 shows the partitioned, place and route summary of the given design. It also shows some of the parameters of the X4000 series part which the software evaluates as the best available part and its two closest neighbors.

We observe that the number of Configurable Logic Blocks used by the design is 95, out of which, 85 are packed. The part in the X4000 series in which this design can have the highest percentage of occupancy is the XC4003. Other design statistics also show that the design can be fitted in this part. The package type used for this design is PC84. The other packages which support the XC4003 have more pins.

Xilinx 3000

Table 4.2 gives us the design statistics along with some of the parameters of the X3000 series part. The XC3042 is the closest available part to accomodate 117 CLBs used by the design. Also, the PC84 seems to be the best available package

Table 4.2: *The X3000 Series Comparision for test case #1*

	<i>Design Uses</i>	<i>3042</i>	<i>3030</i>	<i>3064</i>
Occupied CLBs	<i>117</i>	<i>144</i>	<i>100</i>	<i>224</i>
Flip Flops	<i>22</i>	<i>480</i>	<i>360</i>	<i>688</i>
Package Pins	<i>35</i>	<i>84 (Min.)</i>	<i>44 (Min.)</i>	<i>84 (Min.)</i>

Table 4.3: *The Actel ACT1 Series Comparision for test case #1*

	<i>Design Uses</i>	<i>A1020A</i>	<i>A1010A</i>
Logic Modules	<i>304</i>	<i>547</i>	<i>295</i>
User I/O	<i>35</i>	<i>69 (Max.)</i>	<i>69 (Max.)</i>

as other packages supporting XC3042 have higher numbers of pins.

Actel ACT1

Table 4.3 discusses some of the family members of Actel ACT1 series along with the resources utilized by the design. It is observed that the design uses 304 Logic Modules. This can be accomodated in the A1020A since it has 547 Logic Modules. The A1010A has only 295 Logic Modules. Also, the best available package looking at the number of I/O pins seems to be the lcc68. We cannot use the lcc44 sine it has 34 user I/O pins.

4.2.2 Test Case # 2

The design used here is a hardware compression engine based on the Huffman encoding algorithm [9]. The user enters 8-bit ASCII characters via a 4×3 keypad. The characters are then compressed based on the Huffman algorithm, which is a variable length, string-dependent code. The algorithm represents the characters with the highest statistical probabilities of occurrence with the shortest code. By following this scheme, a compression ratio as high as 40% may be achieved. Figure 4.4 shows the various functional blocks of the ASIC.

The following report was generated by the software.

```

XILINX - 4000
-----
The test for Xilinx 4000 series is complete
The best available part in X4000 series for your design is 4003APC84
For more information check 4003APC84.rpt and 4003APC84.out

ACTEL - ACT1
-----
The test for ACT-1 series is complete
The best available part in Act1 series for your design is 1020alcc68
For more information check 1020alcc68.log

XILINX - 3000
-----
The test for Xilinx 3000 series is complete
The best available part in X3000 series for your design is 3042PC84
For more information check 3042PC84.rpt and 3042PC84.out
```

Figure 4.3: *Report generated by the software for test case # 2.*

Huffman Encoding Algorithm

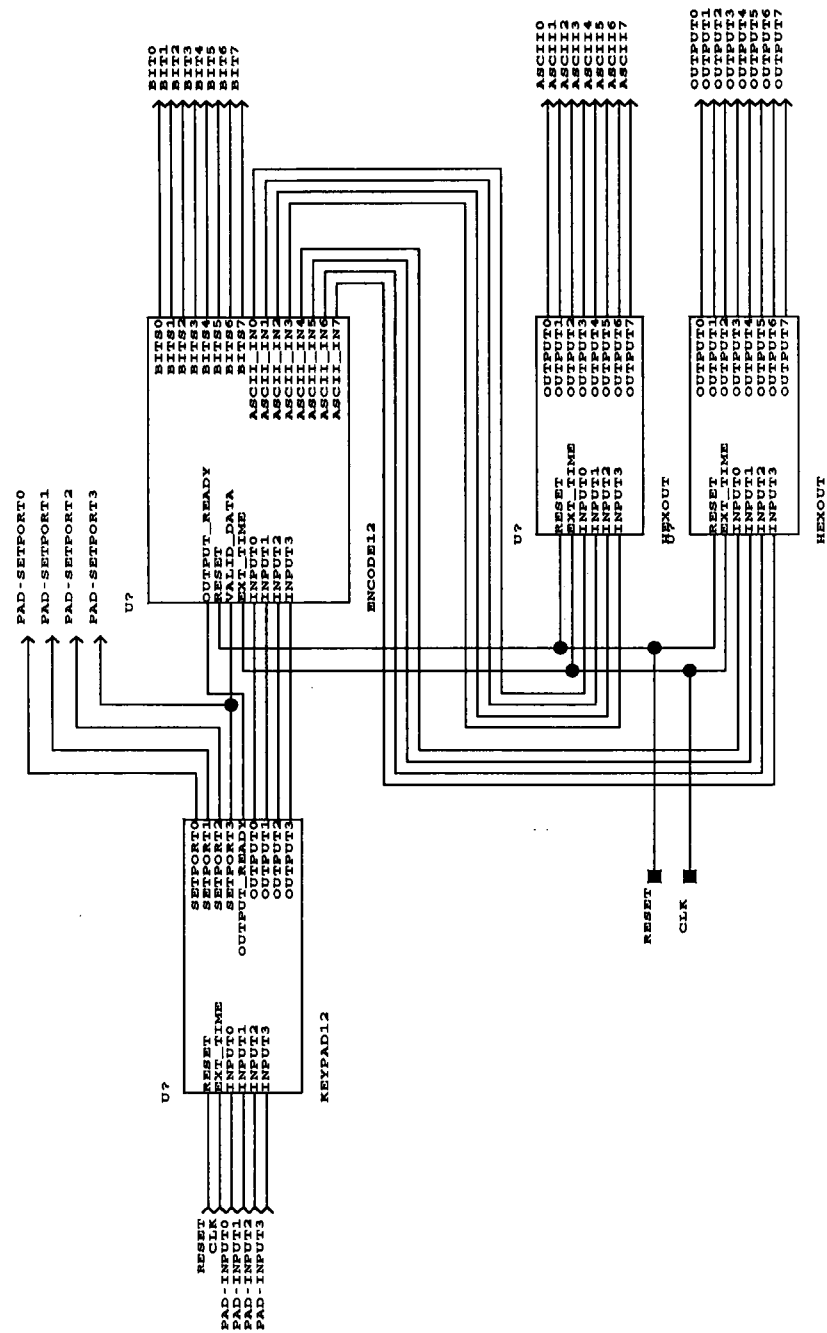


Figure 4.4: Top-Level Schematic of Huffman Encoding Algorithm.

Table 4.4: *The X4000 Series Comparision for test case #2*

	<i>Design Uses</i>	<i>4003</i>	<i>4002</i>	<i>4004</i>
Occupied CLBs	89	100	64	144
Packed CLBs	82	100	64	144
Flip Flops	41	360	256	480
Package Pins	34	84 (Min.)	84 (Min.)	84 (Min.)

Xilinx 4000

Table 4.4 shows the partitioned, place and route summary of the given design. It also shows some of the parameters of the X4000 series part which the software evaluates as the best available part and its two closest neighbors.

We observe that the number of Configurable Logic Blocks used by the design is 89, out of which, 82 are packed. The part in the X4000 series in which this design can have the highest percentage of occupancy is the XC4003. Other design statistics also show that the design can be fitted in this part. The package type used for this design is the PC84. Other packages which support the XC4003 have more pins.

Xilinx 3000

Table 4.5 gives us the design statistics along with some of the parameters of the X3000 series part. The XC3042 is the closest available part to accomodate 115 CLBS used by the design. Also, the PC84 seems to be the best available package

Table 4.5: *The X3000 Series Comparision for test case #2*

	<i>Design Uses</i>	<i>3042</i>	<i>3030</i>	<i>3064</i>
Occupied CLBs	115	144	100	224
Flip Flops	41	480	360	688
Package Pins	34	84 (Min.)	44 (Min.)	84 (Min.)

Table 4.6: *The Actel ACT1 Series Comparision for test case #2*

	<i>Design Uses</i>	<i>A1020A</i>	<i>A1010A</i>
Logic Modules	350	547	295
User I/O	34	69 (Max.)	69 (Max.)

as other packages supporting the XC3042 have higher numbers of pins.

Actel ACT1

Table 4.6 discusses some of the family members of the Actel ACT1 series along with the resources utilized by the design. It is observed that the design uses 350 Logic Modules. This can be accomodated in the A1020A since it has 547 Logic Modules. The A1010A has only 295 Logic Modules. Also, the best available package looking at the number of I/O pins seems to be the lcc68. We cannot use the lcc44 since it has 34 user I/O pins.

4.2.3 Test Case # 3

The design being tested here is a binary image segmenter for binary images [10]. It takes a binary image as an input. The image is required to be transmitted serially starting with the upper lefthand pixel. Scanning of the image proceeds left-to-right, top-to-bottom. The *sync* signals the binary image segmenter that a new image is being transmitted.

As it receives an image, the binary image segmenter outputs the current position and the label for that position. If the current video pixel is inactive, then it is assigned a label 0. If it is active, then its immediate surrounding neighbors are examined. Since the image is being scanned left-to-right, top-to-bottom, a pixel can only have at most four neighbors, with border pixels having even fewer. If one of the surrounding pixels is labelled, or if more than one neighbor are labelled the same, then the current pixel is simply given the same label. However, it is possible for the neighbors to contain two, but no more than two, different labels. U shaped regions are one example of where this situation exists. This occurrence is referred to as merge. However, a problem arises when two regions merge which cause some previously stored labels to be incorrect. To correct this problem, a translator is provided. Every time the stored image is read, it should be done through this translator. The binary image segmenter also outputs a count of the number of pixels in a user-specified region.

The design consists of several functional blocks, each of which performs a particular task. Figure 4.5 shows the various functional blocks of the ASIC.

42

Following report was generated by the software.

XILINX - 4000

The test for Xilinx 4000 series is complete
The best available part in X4000 series for your design is 4005APC84
For more information check 4005APC84.rpt and 4005APC84.out

ACTEL - ACT1

The test for ACT-1 series is complete
Your design doesn't fit in Act-1 series

XILINX - 3000

The test for Xilinx 3000 series is complete
The best available part in X3000 series for your design is 3064PC84
For more information check 3064PC84.rpt and 3064PC84.out

Figure 4.6: *Report generated by the software for test case # 3.*

Xilinx 4000

Table 4.7 shows the partitioned, place and route summary of the given design. It also shows some of the parameters of the X4000 series part which the software evaluates as the best available part and its two closest neighbors.

We observe that the number of Configurable Logic Blocks used by the design is 186, out of which 171 are packed. The part in the X4000 series in which this design can have the highest percentage of occupancy is the XC4005. Other design statistics also show that the design can be fitted in this part. The package type used for this design is the PC84. Other packages which support XC4005 have

Table 4.7: *The X4000 Series Comparision for test case #3*

	<i>Design Uses</i>	<i>4005</i>	<i>4006</i>	<i>4004</i>
Occupied CLBs	<i>186</i>	<i>196</i>	<i>256</i>	<i>144</i>
Packed CLBs	<i>171</i>	<i>196</i>	<i>256</i>	<i>144</i>
Flip Flops	95	616	768	480
Package Pins	48	84 (Min.)	156 (Min.)	84 (Min.)

Table 4.8: *The X3000 Series Comparision for test case #3*

	<i>Design Uses</i>	<i>3064</i>	<i>3042</i>	<i>3090</i>
Occupied CLBs	<i>223</i>	<i>224</i>	<i>144</i>	<i>320</i>
Flip Flops	95	688	480	928
Package Pins	48	84 (Min.)	84 (Min.)	84 (Min.)

more pins.

Xilinx 3000

Table 4.8 gives us the design statistics along with some of the parameters of X3000 series part. The XC3064 is the closest available part to accomodate the 223 CLBs used by the design. Also, the PC84 seems to be the best available package as other packages supporting XC3042 have higher numbers of pins.

Table 4.9: *The Actel ACT1 Series Comparision for test case #3*

	<i>Design Uses</i>	<i>A1020A</i>	<i>A1010A</i>
Logic Modules	708	547	295
User I/O	48	69 (Max.)	69 (Max.)

Actel ACT1

Table 4.9 discusses some of the family members of Actel ACT1 series along with the resources utilized by the design. It is observed that the design uses 708 Logic Modules. This cannot be accomodated in the A1020A since it has only 547 Logic Modules. Also, the A1020A is the largest available part in the ACT1 series. Hence the design will not fit in the Actel ACT1 series.

CHAPTER 5

SUMMARY, CONCLUSION AND FUTURE WORK

5.1 Summary and Conclusion

In the past five years, the FPGA market has experienced tremendous growth and currently is the fastest growing segment of the \$5.7 billion ASIC market [5]. A top priority for every engineer today is how to shorten the design cycle. FPGAs have become an important factor in new product development. Offering the design flexibility and capacity of gate arrays, plus the convenience and speed of desktop programming, FPGAs deliver a quicker time-to-market solution than any other method of logic integration. At the same time, engineers are still facing the traditional demands to make new products cheaper and smaller. FPGAs provide an important solution to this problem, too. By integrating thousands of gates of random logic onto one chip, the designer can reduce board space, component count, and power requirements at the same time.

This has flooded the market with a number of FPGA companies. In order to remain competitive in the marketplace, companies are being driven to reduce product manufacturing costs while adding more features and improving reliability. This has lead to the evolution of various series of FPGAs. At times, designers are often baffled with the choice of FPGAs available to implement their design. They

feel the need for a software which can give them a comparison of commercially available FPGAs in terms of costs involved, resource utilization, etc. This is where the software developed in this thesis comes into the picture.

As previously noted, the purpose of the software developed here is to help a designer in determining which FPGA is best suited for a design in terms of the highest percent occupancy. As the name suggests, it automatically evaluates and implements VHDL-based designs on the FPGAs available from different vendors. The designer supplies VHDL code and a top-level schematic with various modules connected. The software does the rest of the work and the results of the evaluation are summarized in a file. This file contains the necessary information about the part and design implementation. The software can help the designer save his time by eliminating interactive use of cad tools. The software doesn't require technical expertise in using the cad tools on the part of the designer. The software also enhances the productivity of the designer.

5.2 Future Work

There are several ways in which this software can be made more sophisticated and more user friendly. Some of them are described below.

- Currently, the criteria for selecting the best available part is resource utilization. Additional features like critical delay, speed, power consumption, etc. could be added so that the designer has several choices.

- A Graphical User Interface could be added to the developed software so as to make it more user friendly. A user could select various options by clicking on them with the help of a pointing device.
- The software developed supports the X4000 series and the X3000 series of Xilinx and the ACT1 series of Actel. More vendors could be supported (provided the license to run their technology mapping and place and route software is available).
- More information about the design implementation could be given to the designer in the final report.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] Actel Corporation Actel FPGA Data Book and Design Guide.
Sunnyvale, CA : Actel Corporation, 1993.
- [2] Bouldin, Donald W. Designing Field-Programmable Gate Arrays.
ECE 551/552 Class Notes, The University of Tennessee, 1993.
- [3] Bouldin, Donald W. VLSI Designer's Interface.
IEEE Circuits & Devices pp 5-6, September 1993.
- [4] Brown Stephen, Francis Robert, Rose Jonathan and Vranesic Zvonko
Field-Programmable Gate Arrays. University of Toronto : Kluwer
Academic Publishers, 1992.
- [5] Electronic Engineering Times, March 22, 1993.
- [6] Giovanni De Micheli Synthesis and Optimization of Digital Circuits.
McGraw-Hill Publishers, 1994.
- [7] Huber Successful ASIC Design The First Time Through
Van Nostrand Rheinhold, 1992.
- [8] Jenkins Jesse Designing with FPGAs and CPLDs
Prentice-Hall Publications, 1994.
- [9] Kendall Matthew and King Eric Huffman Algorithm. ECE 494 Class
Project Report, The University of Tennessee, Dec. 1993.
- [10] Smith Doug and Chattopadhyay Subhomoy Image Segment Identification
System. ECE 551 Class Project Report, The University of Tennessee, Dec.
1993.
- [11] Trimberger, Stephen M. Field-Programmable Gate Array Technology.
San Jose, CA : Kluwer Academic Publishers, 1994.
- [12] Weste N. and Eshraghian K. Principles of CMOS VLSI Design.
Addison Wesley Publishing Company, 1993.
- [13] Xilinx, Inc. XILINX The Programmable Logic Data Book.
San Jose, CA : Xilinx, Inc., 1993.

APPENDIXES

APPENDIX A

VARIOUS C-SHELL SCRIPTS

```
:::::::::::::
hello4000
:::::::::::::
```

```
#!/bin/csh -f
# hello4000 is a script that invokes various other scripts and
# C programs to select the best available part for given design
# in Xilinx 4000 series.
# Any line starting with three dots (...) shows the continuation
# of previous command.
#####
```

```
rm viewdraw.ini
pushd wir
rm *.1 *.2 *.3 *.4 *.5
popd
```

```
pushd sch
mv core.1 core.temp
mv frame.1 frame.temp
rm *.1 *.2 *.3 *.4 *.5
mv core.temp core.1
mv frame.temp frame.1
popd
```

```
design -v x4
```

```
pushd behv
rm techdef.vhd
cp ~/techdef.vhd.xilinx techdef.vhd
rm *.lis
set modules = `ls *.vhd | cut -d. -f1,5`
popd
```

```
sample4000 $modules
```

```
vsm core
```

```

viewgen core -makesym -noschem
vsm frame

set part = 4010PQ208

pushd wir
rm chip.1
set IN = 'grep PINTYPE=IN frame.1 | wc -l'
set OUT = 'grep PINTYPE=OUT frame.1 | wc -l'
cc -o create4000 copy4000.c
create4000 $IN $OUT
popd

rm chip.xnf chip.xnx
wir2xnf -b -x -p $part chip chip.xnf
xmake -p $part chip

set CHECK = 'awk '/You selected what/{ print $8}' chip.out'
set RECO = 'awk '/Note: by initial estimates,{ print $13}'
... chip.out'

if ($CHECK == "best") then
    cp chip.rpt $part.rpt
    cp chip.out $part.out
    cp chip.bit $part.bit
    rm chip.*
    rm *.xnf *.xnx
    echo "The best part for your design is $part"
    cc -o rep report4000.c
    rep $part
    goto last
else
    rm chip.xnf chip.xnx
    wir2xnf -b -x -p $RECO chip chip.xnf
    xmake -p $RECO chip
    cp chip.rpt $RECO.rpt
    cp chip.out $RECO.out
    cp chip.bit $RECO.bit
    rm chip.*
    rm *.xnf *.xnx
    echo "The best part for your design is $RECO"
    cc -o rep report4000.c
    rep $RECO
    goto last

```

```
endif
```

```
#####
```

```
last:
```

```
echo "The test for Xilinx 4000 series is complete."
```

```
exit -1
```

```
:::::::::::::
hello-act1
:::::::::::::
```

```
#!/bin/csh -f
# helloactel is a script that invokes various other scripts and
# C programs to select the best available part for given design
# in Actel ACT-1 series.
# Any line starting with three dots (...) shows the continuation
# of previous command.
```

```
#####
```

```
rm viewdraw.ini
rm *.def *.fus *.log
pushd wir
rm *.1 *.2 *.3 *.4 *.5
popd
```

```
pushd sch
mv core.1 core.temp
mv frame.1 frame.temp
rm *.1 *.2 *.3 *.4 *.5
mv core.temp core.1
mv frame.temp frame.1
popd
```

```
design -v actel
```

```
pushd behv
rm techdef.vhd
cp ~/techdef.vhd.actel techdef.vhd
rm *.lis
set modules = 'ls *.vhd | cut -d. -f1,5'
popd
```

```
sample-act1 $modules
```

```
vsm -w core
vsm -w frame
```

```
pushd wir
set IN = 'grep PINTYPE=IN frame.1 | wc -l'
set OUT = 'grep PINTYPE=OUT frame.1 | wc -l'
set number = 'awk '/AS core PINORDER =/
... {for (i = 1; i <= NF; i++)
```

```

... if($i == "CLK") print i;}' frame.1'

cc -o create-act1 copy-act1.c
create-act1 $IN $OUT $number
popd

rm chip.ipf chip.adl chip.aal alsrun.log alsrun.temp

if (($IN + $OUT) <= 30) then
    set pack = lcc44
else if (((($IN + $OUT) > 30) && (($IN + $OUT) <= 53)) then
    set pack = lcc68
else if (((($IN + $OUT) > 53) && (($IN + $OUT) <= 65)) then
    set pack = lcc84
else
    cc -o rep report-act1.c
    rep not-actel
    goto last
endif

cc -o create-pin pin-act1.c
create-pin $IN $OUT

vl2adl fam:act1 chip
als -set die 1020a chip
als -set package $pack chip
alsrun chip
cp alsrun.log alsrun.temp

cc -o trim out-act1.c
trim
set LM = 'awk '/logic modules/{ print $1}' final.out'
if ($LM <= 295) then
    rm alsrun.log chip.adl chip.aal
    vl2adl fam:act1 chip
    als -set die 1010a chip
    als -set package $pack chip
    alsrun chip
    mv alsrun.log 1010a$pack.log
    mv chip.fus 1010a$pack.fus
    mv chip.def 1010a$pack.def
    rm chip.*

echo "The best available part for your design in ACT-1 series
... is 1010a$pack"

```



```

cc -o rep report-act1.c
rep 1010a$pack
goto last

else if (($LM > 295) && ($LM <= 547)) then
    mv alsrn.log 1020a$pack.log
    mv chip.fus 1020a$pack.fus
    mv chip.def 1020a$pack.def
    rm chip.*
    echo "The best available part for your design in ACT-1 series
    ... is 1020a$pack"
    cc -o rep report-act1.c
    rep 1020a$pack
    goto last

else
    echo "Your design doesn't fit in Actel ACT-1 series."
    cc -o rep report-act1.c
    rep not-actel
    goto last

endif

#####
last:
echo "The test for Actel ACT-1 series is complete."
exit -1

```

```
:::::::::::::
hello3000
:::::::::::::
```

```
#!/bin/csh -f
# hello3000 is a script that invokes various other scripts and
# C programs to select the best available part for given design
# in Xilinx 3000 series.
# Any line starting with three dots (...) shows the continuation
# of previous command.
#####
```

```
rm viewdraw.ini
pushd wir
rm *.1 *.2 *.3 *.4 *.5
popd
```

```
pushd sch
mv core.1 core.temp
mv frame.1 frame.temp
rm *.1 *.2 *.3 *.4 *.5
mv core.temp core.1
mv frame.temp frame.1
popd
```

```
design -v x3
```

```
pushd behv
rm techdef.vhd
cp ~/techdef.vhd.xilinx techdef.vhd
rm *.lis
set modules = 'ls *.vhd | cut -d. -f1,5'
popd
```

```
sample3000 $modules
```

```
vsm core
vsm frame
```

```
set part = 3090PP175
```

```
pushd wir
set IN = 'grep PINTYPE=IN frame.1 | wc -l'
set OUT = 'grep PINTYPE=OUT frame.1 | wc -l'
cc -o create3000 copy3000.c
```

```

create3000 $IN $OUT
popd

rm chip.xnf chip.xnx

wir2xnf -b -x -p $part chip chip.xnf
xmake -p $part chip

cc -o trim out3000.c
trim
set CLB = 'awk '/CLBs used/{ print $1}' final.out'
set IOPINS = 'awk '/0 pins used/{ print $1}' final.out'
set FF = 'awk '/flipflops used/{ print $1}' final.out'
set TSS = 'awk '/three-state signals/{ print $1}' final.out'

cc -o better check3000.c
set best = 'better $CLB $IOPINS $FF $TSS'

if ($best == "3090PP175") then
    mv chip.rpt $part.rpt
    mv chip.out $part.out
    mv chip.bit $part.bit
    rm chip.*
    rm *.xnf *.xnx
    echo "The best part for your design is $part"
    cc -o rep report3000.c
    rep $best
    goto last

else if ($best == "not-3000") then
    mv chip.rpt $part.rpt
    mv chip.out $part.out
    rm chip.*
    rm *.xnx *.xnf
    echo "Your design doesn't fit in Xilinx 3000 series."
    cc -o rep report3000.c
    rep not-3000
    goto last

else
    rm chip.*
    wir2xnf -b -x -p $best chip chip.xnf
    xmake -p $best chip
    mv chip.out $best.out
    mv chip.rpt $best.rpt

```

```

mv chip.bit $best.bit
rm *.xnx *.xnf
rm chip.*
echo "The best available part seems to be $best."
cc -o rep report3000.c
rep $best
goto last
endif
#####
last:
echo "The test for Xilinx 3000 series is complete."
exit -1

```

```

::::::::::::
design
::::::::::::

#! /bin/csh -f

# design is a script that creates links to lager files and
# viewdraw.ini files for use in local directories
# Any line starting with three dots (...) shows the continuation
# of previous command.
#####

if ($#argv <= 0) goto usage

while ($#argv > 0)
    switch ($argv[1])
        case -v :
            set view_only = $argv[1]
            breaksw

        case -l :
            set lager_only = $argv[1]
            breaksw

        case -* :
            goto usage
            breaksw

        default :
            set target = $argv[1]
            breaksw

    endsw
    if ($#argv > 0) shift argv
end
#####

switch ($target)
    case x2 :
        set tech = xilinx_2000
        breaksw

    case x3 :
        set tech = xilinx_3000

```

```

        breaksw

case x4 :
    set tech = xilinx_4000
    breaksw

case altera :
    set tech = altera
    breaksw

case actel :
    set tech = actel
    breaksw

case dio :
    set tech = dio
    breaksw

case stdcell :
    set tech = stdcell
    breaksw

case hyper :
    set tech = hyper
    breaksw

case * :
    goto usage
    breaksw

endsw
#####

if (! $?view_only && ! $?lager_only) then
    set view_only = 1
    set lager_only = 1
endif

if ($?view_only) then
    rm -f viewdraw.ini
    if ( -e 'echo $WDIR | sed 's%:.*$%%' '/viewdraw.ini_
    ...${tech}) then
        ln -s 'echo $WDIR | sed 's%:.*$%%' '/viewdraw.ini_
        ...${tech} viewdraw.ini
    else

```

```

        ln -s /u4/bagri/viewdraw.ini_${tech} viewdraw.ini
    endif
endif

if ($?lager_only) then
    rm -f lager
    if ( -e 'echo $WDIR | sed 's%:.*$%%' /lager_${tech}) then
        ln -s 'echo $WDIR | sed 's%:.*$%%' /lager_${tech} lager
    else
        ln -s /common/lager/common/LagerIV/lib/lager_${tech}
        ... lager
    endif
endif

mkdir behv sch wir sym
mkdir sch/sav sch/bac
mkdir sym/sav sym/bac

#####
exit 0
#####

usage:
echo "Usage:"
echo "  design [options] [target]"
echo ""
echo "  options: -v generate viewdraw.ini link only"
echo "           -l generate lager link only"
echo "  target: altera -> altera programmable gate array"
echo "           x2 -> xilinx 2000 series programmable gate array"
echo "           x3 -> xilinx 3000 series programmable gate array"
echo "           x4 -> xilinx 4000 series programmable gate array"
echo "           actel -> actel programmable gate array"
echo "           dio -> Data IO library, use with ViewPLD tool"
echo "           stdcell -> Standard Cell Design with Lager"
echo "           hyper -> Design using the Hyper/Lager libraries"
echo ""
echo ""

exit -1

```

```

::::::::::::
sample-act1
::::::::::::

#!/bin/csh -f
# sample-act1 is a script that compiles the various vhd1 modules
# and creates their wir file and their symbol.
# Any line starting with three dots (...) shows the continuation
# of previous command.
#####

vhdldes techdef.vhd -tec=act1
if ($#argv <= 0) goto usage
while ($#argv > 0)
    if (($argv[1] == "techdef") && ($#argv > 0)) then
        shift argv
    else
        vhdldes $argv[1].vhd -tec=act1
        viewgen $argv[1] -makesym
        vsm $argv[1]
        if ($#argv > 0) shift argv
    endif
end
#####

usage:
exit -1

```



```

:::::::::::::
sample3000
:::::::::::::

#! /bin/csh -f
# sample3000 is a script that compiles the various vhd1 modules
# and creates their wir file and their symbol.
# Any line starting with three dots (...) shows the continuation
# of previous command.
#####

vhdldes techdef.vhd -tec=x3000
if ($#argv <= 0) goto usage
while ($#argv > 0)
    if (($argv[1] == "techdef") && ($#argv > 0)) then
        shift argv
    else
        vhdldes $argv[1].vhd -tec=x3000
        viewgen $argv[1]
        vsm $argv[1]
        if ($#argv > 0) shift argv
    endif
end
#####

usage:
exit -1

```

```

:::::::::::::
sample4000
:::::::::::::

#! /bin/csh -f
# sample4000 is a script that compiles the various vhd1 modules
# and creates their wir file and their symbol.
# Any line starting with three dots (...) shows the continuation
# of previous command.
#####

vhdlldes techdef.vhd -tec=x4000
if ($#argv <= 0) goto usage
while ($#argv > 0)
    if (($argv[1] == "techdef") && ($#argv > 0)) then
        shift argv
    else
        vhdlldes $argv[1].vhd -tec=x4000
        viewgen $argv[1]
        vsm $argv[1]
        if ($#argv > 0) shift argv
    endif
end
#####

usage:
exit -1

```

APPENDIX B

VARIOUS C PROGRAMS

```
:::::::::::::
copy4000.c
:::::::::::::
```

```
\* copy4000.c is a C program in the wir subdirectory which
creates top level wir file with the x4000 pads included. \*
```

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
```

```
void main ( argc, argv )
int argc;
int *argv[ ];
```

```
{
    FILE *from,*to;
    char str[500];
    char string[500];
    char str1[20], str2[20];
    char str3[20], str4[20], str5[7];
    int i, j;
    int temp=0, num1, num2;
    int INPIN, OUTPIN;
```

```
/*see for correct # of command line arguments */
```

```
if (argc < 3) {
    printf("Usage: create <IN-PINS> <OUT-PINS>\n");
    exit(1);
}
```

```
INPIN = atoi(argv[1]);
OUTPIN = atoi(argv[2]);
```

```
num1 = 3;
num2 = INPIN+3;
```

```

/* printf("%s\n", argv[1]);*/
/* Open Source File */

if((from = fopen("frame.1","r"))==NULL) {
    printf("Cannot open source file\n");
    exit(1);
}

/* Open Destination File */

if((to = fopen("chip.1","w"))==NULL){
    printf("Cannot open file\n");
    exit(1);
}

/* Copy the File */

while(!feof(from)) {
    fgets(str,499,from);
    sscanf(str, "%s %s %s %s", str1, str2, str3, str4);
    if (ferror(from)) {
        printf("error on input\n");
        break;
    }
    if((!(strcmp(str1, "AP"))) && (!(strcmp(str4,
"PINTYPE=IN")))) {
        sprintf(str5,"$1N%d ", num1);
        num1++;
        /* printf("%s", str5); */
        strcat(string, str5);
    }

    if((!(strcmp(str1, "AP"))) && (!(strcmp(str4,
"PINTYPE=OUT")))) {
        sprintf(str5,"$1N%d ", num2);
        num2++;
        strcat(string, str5);
    }

    if (!feof(from)) {
        if((!(strcmp(str1, "DW"))) && (!(strcmp(str2,
"frame")))) {
            fprintf( to, "%s\n","DW chip");
            fprintf( to, "AS x4000:opad EXT=BPAD \n");
            fprintf( to, "AS x4000:opad LEVEL=XILINX \n");
            fprintf( to, "AS x4000:opad PINORDER = PAD \n");
        }
    }
}

```

```

fprintf( to, "AS x4000:obuf @GTSDLY1=0 \n");
fprintf( to, "AS x4000:obuf @GTSDLY0=0 \n");
fprintf( to, "AS x4000:obuf DEVICE=OBUF \n");
fprintf( to, "AS x4000:obuf LEVEL=XILINX \n");
fprintf( to, "AS x4000:obuf @ODLY1=1ns \n");
fprintf( to, "AS x4000:obuf @ODLY0=1ns \n");
fprintf( to, "AS x4000:obuf @IDLY1=0 \n");
fprintf( to, "AS x4000:obuf @IDLY0=0 \n");
fprintf( to, "AS x4000:obuf PINORDER = GTS I 0 \n");
fprintf( to, "AP x4000:obuf 1 PINTYPE=IN \n");
fprintf( to, "AP x4000:obuf 2 PINTYPE=IN \n");
fprintf( to, "AP x4000:obuf 3 PINTYPE=CHIPTRI \n");
fprintf( to, "AS x4000:ipad EXT=BPAD \n");
fprintf( to, "AS x4000:ipad LEVEL=XILINX \n");
fprintf( to, "AS x4000:ipad PINORDER = PAD \n");
fprintf( to, "AS x4000:ibuf @IDLY0=0 \n");
fprintf( to, "AS x4000:ibuf @IDLY1=0 \n");
fprintf( to, "AS x4000:ibuf @ODLY0=1ns \n");
fprintf( to, "AS x4000:ibuf @ODLY1=1ns \n");
fprintf( to, "AS x4000:ibuf LEVEL=XILINX \n");
fprintf( to, "AS x4000:ibuf DEVICE=IBUF \n");
fprintf( to, "AS x4000:ibuf PINORDER = I 0 \n");
fprintf( to, "AP x4000:ibuf 1 PINTYPE=CHIPIN \n");
fprintf( to, "AP x4000:ibuf 2 PINTYPE=OUT \n");
}

```

```

else if ((!(strcmp(str1, "I"))) && (!strcmp(str3,
"core")))) {
    fprintf(to, "%s %s %s %s", str1, str2, str3, string);
    fprintf(to, "REFDES=U? ' \n");

```

```

    for ( i=1; i<=INPIN; i++ ) {
        fprintf(to, "W x4000:ibuf $1I%d \n", 100+i);
        fprintf(to, "I $1I%d x4000:ibuf INPAD-%d $1N%d
        \n", 100+i, i, i+2);
    }
    for ( i=1; i<=OUTPIN; i++ ) {
        fprintf(to, "W x4000:obuf $1I%d \n", 100+INPIN+i);
        fprintf(to, "I $1I%d x4000:obuf ? $1N%d OUTPAD-%d
        \n", 100+INPIN+i, i+2+INPIN, i);
    }
    j = 0;
    for ( i=1; i<=OUTPIN; i=i++ ) {
        fprintf(to, "W x4000:opad $1I%d\n", 200+i+j);
        fprintf(to, "I $1I%d x4000:opad OUTPAD-%d\n",

```

```

        200+i+j, i);
        j++;
    }
    j = 0;
    for ( i=1; i<=INPIN; i=i++ ) {
        fprintf(to, "W x4000:ipad $1I%d \n", 300+i+j);
        fprintf(to, "I $1I%d x4000:ipad INPAD-%d \n",
            300+i+j, i);
        j++;
    }
}

else
    fputs(str,to);
}

if (ferror(to)) {
    printf("Error on output\n");
    break;
}
}
fclose(from);
fclose(to);
}

```

```
.....:
copy3000.c
.....:
```

```
\* copy3000.c is a C program in the wir subdirectory which
creates top level wir file with the x3000 pads included. \*
```

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
```

```
void main ( argc, argv )
int argc;
int *argv[ ];
```

```
{
    FILE *from,*to;
    char str[500];
    char string[500];
    char str1[20], str2[20];
    char str3[20], str4[20], str5[7];
    int i, j;
    int temp=0, num1, num2;
    int INPIN, OUTPIN;

    /*see for correct # of command line arguments */
    if (argc < 3) {
        printf("Usage: create <IN-PINS> <OUT-PINS>\n");
        exit(1);
    }

    INPIN = atoi(argv[1]);
    OUTPIN = atoi(argv[2]);

    num1 = 3;
    num2 = INPIN+3;
    /* printf("%s\n", argv[1]);*/
    /* Open Source File */
    if((from = fopen("frame.1","r"))==NULL) {
        printf("Cannot open source file\n");
        exit(1);
    }

    /* Open Destination File */
    if((to = fopen("chip.1","w"))==NULL){
```

```

    printf("Cannot open file\n");
    exit(1);
}

/* Copy the File */
while(!feof(from)) {
    fgets(str,499,from);
    sscanf(str, "%s %s %s %s", str1, str2, str3, str4);
    if (ferror(from)) {
        printf("error on input\n");
        break;
    }
    if((!(strcmp(str1, "AP"))) && (!strcmp(str4,
"PINTYPE=IN")))) {
        sprintf(str5,"$1N%d ", num1);
        num1++;
        /* printf("%s", str5); */
        strcat(string, str5);
    }

    if((!(strcmp(str1, "AP"))) && (!strcmp(str4,
"PINTYPE=OUT")))) {
        sprintf(str5,"$1N%d ", num2);
        num2++;
        printf("%s", str5);
        strcat(string, str5);
    }

    if (!feof(from)) {
        if((!(strcmp(str1, "DW"))) && (!strcmp(str2,
"frame")))) {
            fprintf( to, "%s\n","DW chip");
            fprintf( to, "AS x3000:opad EXT=OPAD \n");
            fprintf( to, "AS x3000:opad LEVEL=XILINX \n");
            fprintf( to, "AS x3000:opad PINORDER = OPAD \n");
            fprintf( to, "AS x3000:ipad EXT=IPAD \n");
            fprintf( to, "AS x3000:ipad LEVEL=XILINX \n");
            fprintf( to, "AS x3000:ipad PINORDER = IPAD \n");
            fprintf( to, "AS x3000:obuf @IDLY0=0 \n");
            fprintf( to, "AS x3000:obuf @IDLY1=0 \n");
            fprintf( to, "AS x3000:obuf @ODLY0=1ns \n");
            fprintf( to, "AS x3000:obuf @ODLY1=1ns \n");
            fprintf( to, "AS x3000:obuf LEVEL=XILINX \n");
            fprintf( to, "AS x3000:obuf DEVICE=OBUF \n");
            fprintf( to, "AS x3000:obuf PINORDER = I 0 \n");
        }
    }
}

```



```

    fprintf( to, "AP x3000:obuf 1 PINTYPE=IN \n");
    fprintf( to, "AP x3000:obuf 2 PINTYPE=CHIPOUT \n");
    fprintf( to, "AS x3000:ibuf @IDLY0=0 \n");
    fprintf( to, "AS x3000:ibuf @IDLY1=0 \n");
    fprintf( to, "AS x3000:ibuf @ODLY0=1ns \n");
    fprintf( to, "AS x3000:ibuf @ODLY1=1ns \n");
    fprintf( to, "AS x3000:ibuf LEVEL=XILINX \n");
    fprintf( to, "AS x3000:ibuf DEVICE=IBUF \n");
    fprintf( to, "AS x3000:ibuf PINORDER = I 0 \n");
    fprintf( to, "AP x3000:ibuf 1 PINTYPE=CHIPIN \n");
    fprintf( to, "AP x3000:ibuf 2 PINTYPE=OUT \n");

}
else if (((!strcmp(str1, "I"))) && (!strcmp(str3,
"core")))) {
    fprintf(to, "%s %s %s %s",str1, str2, str3, string);
    fprintf(to,"REFDES=U?' \n");

    for ( i=1; i<=INPIN; i++ ) {
        fprintf(to, "W x3000:ibuf $1I%d \n", 100+i);
        fprintf(to, "I $1I%d x3000:ibuf INPAD-%d $1N%d
\n", 100+i, i, i+2);
    }
    for ( i=1; i<=OUTPIN; i++ ) {
        fprintf(to, "W x3000:obuf $1I%d \n", 100+INPIN+i);
        fprintf(to, "I $1I%d x3000:obuf $1N%d OUTPAD-%d
\n", 100+INPIN+i, i+2+INPIN, i);
    }
    j = 0;
    for ( i=1; i<=OUTPIN; i=i++ ) {
        fprintf(to, "W x3000:opad $1I%d\n", 200+i+j);
        fprintf(to, "I $1I%d x3000:opad  OUTPAD-%d\n",
200+i+j, i);
        j++;
    }
    j = 0;
    for ( i=1; i<=INPIN; i=i++ ) {
        fprintf(to, "W x3000:ipad $1I%d \n", 300+i+j);
        fprintf(to, "I $1I%d x3000:ipad  INPAD-%d \n",
300+i+j, i);
        j++;
    }
}

else

```

```
        fputs(str,to);
    }

    if (ferror(to)) {
        printf("Error on output\n");
        break;
    }
}
fclose(from);
fclose(to);
}
```

```

::::::::::::
copy-act1.c
::::::::::::

```

```

/* copy-act1.c is a C program in the wir subdirectory which
creates top level wir file with the Actel pads included. */

```

```

#include "stdio.h"
#include "stdlib.h"
#include "string.h"

```

```

void main ( argc, argv )
int argc;
int *argv[ ];
{

```

```

    FILE *from,*to;
    char str[500];
    char string[500];
    char str1[20], str2[20];
    char str3[20], str4[20], str5[7];
    int i, j, tem;
    int temp=0, num1, num2, num3, num4;
    int INPIN, OUTPIN, CLKNUM;

```

```

/*see for correct # of command line arguments */
    if (argc < 3) {
        printf("Usage: create <IN-PINS> <OUT-PINS>\n");
        exit(1);
    }

```

```

    INPIN = atoi(argv[1]);
    OUTPIN = atoi(argv[2]);
    CLKNUM = atoi(argv[3]) - 4;

```

```

    num3 = 0;
    num1 = 3;
    num2 = INPIN+3;
    tem = 1;

```

```

/* printf("%s\n", argv[1]);*/
/* Open Source File */
    if((from = fopen("frame.1","r"))==NULL) {
        printf("Cannot open source file\n");
        exit(1);
    }

```

```

/* Open Destination File */
if((to = fopen("chip.1","w"))==NULL){
    printf("Cannot open file\n");
    exit(1);
}

/* Copy the File */
while(!feof(from)) {
    fgets(str,499,from);
    sscanf(str, "%s %s %s %s", str1, str2, str3, str4);
    if (ferror(from)) {
        printf("error on input\n");
        break;
    }
    if((!(strcmp(str1, "AP"))) && (!(strcmp(str4,
"PINTYPE=IN")))) {
        num3++;
        if (num3 == CLKNUM) num4 = num1 - 2;
        sprintf(str5,"$1N%d ", num1);

        num1++;
        strcat(string, str5);
    }

    if((!(strcmp(str1, "AP"))) && (!(strcmp(str4,
"PINTYPE=OUT")))) {
        num3++;
        sprintf(str5,"$1N%d ", num2);
        num2++;
        strcat(string, str5);
    }

    if (!feof(from)) {
        if((!(strcmp(str1, "DW"))) && (!(strcmp(str2,
"frame")))) {
            fprintf( to, "%s\n","DW chip");
            fprintf( to, "AS ACTELCELLS:outbuf LEVEL=UNIT \n");
            fprintf( to, "AS ACTELCELLS:outbuf PINORDER = D PAD \n");
            fprintf( to, "AP ACTELCELLS:outbuf 1 PINTYPE=IN \n");
            fprintf( to, "AP ACTELCELLS:outbuf 2 PINTYPE=OUT \n");
            fprintf( to, "AS ACTELCELLS:inbuf LEVEL=UNIT \n");
            fprintf( to, "AS ACTELCELLS:inbuf PINORDER = PAD Y \n");
            fprintf( to, "AP ACTELCELLS:inbuf 1 PINTYPE=IN \n");
            fprintf( to, "AP ACTELCELLS:inbuf 2 PINTYPE=OUT \n");
        }
    }
}

```

```

    fprintf( to, "AS ACTELCELLS:clkbuf LEVEL=HARD \n");
    fprintf( to, "AS ACTELCELLS:clkbuf PINORDER = PAD Y \n");
    fprintf( to, "AP ACTELCELLS:clkbuf 1 PINTYPE=IN \n");
    fprintf( to, "AP ACTELCELLS:clkbuf 2 PINTYPE=OUT \n");

}
else if (((strcmp(str1, "I"))) && (!strcmp(str3,
"core")))) {
    fprintf(to, "%s %s %s %s",str1, str2, str3, string);
    fprintf(to,"REFDES=U?' \n");
    for ( i=1; i<=INPIN; i++ ) {
        if(i == num4) {
            fprintf(to, "W ACTELCELLS:clkbuf $1I%d \n", 100+i);
            fprintf(to, "I $1I%d ACTELCELLS:clkbuf PAD-C $1N%d
\n", 100+i, i+2);
        }
        else {
            fprintf(to, "W ACTELCELLS:inbuf $1I%d \n", 100+i);
            fprintf(to, "I $1I%d ACTELCELLS:inbuf PAD-%d $1N%d
\n", 100+i, tem, i+2);
            tem++;
        }
    }
    for ( i=1; i<=OUTPIN; i++ ) {
        fprintf(to, "W ACTELCELLS:outbuf $1I%d \n",
100+INPIN+i);
        fprintf(to, "I $1I%d ACTELCELLS:outbuf $1N%d PAD-%d
\n", 100+INPIN+i, i+2+INPIN, tem);
        tem++;
    }
}

else
    fputs(str,to);
}

if (ferror(to)) {
    printf("Error on output\n");
    break;
}
}
fclose(from);
fclose(to);
}

```

```
.....  
check3000.c  
.....
```

```
\* A C program to select the best available part in x3000  
series. */
```

```
#include "stdio.h"  
#include "stdlib.h"  
#include "string.h"
```

```
void main ( argc, argv )  
int argc;  
int *argv[ ];
```

```
{  
    int CLB, IOPINS, FF, TBUF, TSS;  
    char str[20];
```

```
/*see for correct # of command line arguments */  
if (argc < 5) {  
    printf("Usage: create <IN-PINS> <OUT-PINS>\n");  
    exit(1);  
}
```

```
CLB = atoi(argv[1]);  
IOPINS = atoi(argv[2]);  
FF = atoi(argv[3]);  
/* TBUF = atoi(argv[4]); */  
TSS = atoi(argv[4]);
```

```
if ((CLB < 320) && (FF < 928) && (TSS < 40)) {  
    if (IOPINS < 144)    sprintf(str, "3090PP175");  
    if (IOPINS < 138)    sprintf(str, "3090PQ160");  
    if (IOPINS < 70)     sprintf(str, "3090PC84");  
}
```

```
if ((CLB < 224) && (FF < 688) && (TSS < 32)) {  
    if (IOPINS < 120)    sprintf(str, "3064PQ160");  
    if (IOPINS < 110)    sprintf(str, "3064PG132");  
    if (IOPINS < 70)     sprintf(str, "3064PC84");  
}
```

```
if ((CLB < 144) && (FF < 480) && (TSS < 24)) {  
    if (IOPINS < 96)     sprintf(str, "3042PG132");
```

```

        if (IOPINS < 82)    sprintf(str, "3042PQ100");
        if (IOPINS < 74)    sprintf(str, "3042PC84");
    }

    if ((CLB < 100) && (FF < 360) && (TSS < 20)) {
        if (IOPINS < 80)    sprintf(str, "3030PQ100");
        if (IOPINS < 74)    sprintf(str, "3030PC84");
        if (IOPINS < 58)    sprintf(str, "3030PC68");
        if (IOPINS < 34)    sprintf(str, "3030PC44");
    }

    if ((CLB < 64) && (FF < 256) && (TSS < 16)) {
        if (IOPINS < 64)    sprintf(str, "3020PC84");
        if (IOPINS < 58)    sprintf(str, "3020PC68");
    }

    if ((CLB >= 320) || (FF >= 928) || (TSS >= 40) ||
        (IOPINS >= 144))
        printf("not-3000\n");

    else {
        printf("%s\n", str);
    }
}

```

```
.....  
out-act1.c  
.....
```

```
/* A C program to output particular strings in a file */
```

```
#include "stdio.h"  
#include "stdlib.h"  
#include "string.h"
```

```
void main ()  
{
```

```
    FILE *from,*to;  
    char str[1000];  
    char str1[100], str2[100];  
    int i;
```

```
/* Open Source File */
```

```
    if((from = fopen("alsrun.log","r"))==NULL) {  
        printf("Cannot open source file\n");  
        exit(1);  
    }
```

```
/* Open Destination File */
```

```
    if((to = fopen("final.out","w"))==NULL){  
        printf("Cannot open file\n");  
        exit(1);  
    }
```

```
    while(!feof(from)) {  
        fgets(str,999,from);  
        fscanf(from, "%s %s", str1, str2);  
        if (ferror(from)) {  
            printf("error on input\n");  
            break;  
        }  
        if((!(strcmp(str1, "VLD5300:")))) {  
            for (i=1; i < 10; i++) {  
                fgets(str,999,from);  
                fputs(str, to);  
            }  
        }
```

```
    }  
}
```



```
.....  
out3000.c  
.....
```

```
/* A C program to output a particular strings in a file */
```

```
#include "stdio.h"  
#include "stdlib.h"  
#include "string.h"
```

```
void main ()
```

```
{  
    FILE *from,*to;  
    char str[1000], str1[100], str2[100];  
    int i;
```

```
/* Open Source File */
```

```
if((from = fopen("chip.out","r"))==NULL) {  
    printf("Cannot open source file\n");  
    exit(1);  
}
```

```
/* Open Destination File */
```

```
if((to = fopen("final.out","w"))==NULL){  
    printf("Cannot open file\n");  
    exit(1);  
}
```

```
while(!feof(from)) {  
    fgets(str,999,from);  
    fscanf(from, "%s %s", str1, str2);  
    if (ferror(from)) {  
        printf("error on input\n");  
        break;  
    }  
    if((!(strcmp(str1, "DESIGN"))) && (!(strcmp(str2,  
    "SUMMARY:")))) {  
        for (i=1; i < 10; i++) {  
            fgets(str,999,from);  
            fputs(str, to);  
        }  
    }  
}
```

```
    }  
    fclose(from);  
    fclose(to);  
}
```

```
.....  
pin-act1.c  
.....
```

```
/* A C program which creates the top level ipf file with  
all the i/o pin defined in ACTEL series. */
```

```
#include "stdio.h"  
#include "stdlib.h"  
#include "string.h"
```

```
void main ( argc, argv )  
int argc;  
int *argv[ ];
```

```
{  
    FILE *to;  
    int i, j, temp, total;
```

```
/*see for correct # of command line arguments */  
    if (argc < 3) {  
        printf("Usage: create <IN-PINS> <OUT-PINS>\n");  
        exit(1);  
    }
```

```
    temp = (atoi(argv[1]) + atoi(argv[2]) - 1);  
    total = atoi(argv[1]) + atoi(argv[2]);
```

```
/* Open Destination File */  
    if((to = fopen("chip.ipf","w"))==NULL){  
        printf("Cannot open file\n");  
        exit(1);  
    }
```

```
    fprintf( to, "DEF chip.\n");
```

```
    if (total <= 30) {  
        fprintf( to, "NET PAD-C; ; PIN:33.\n");
```

```
        for(i = 1; i <= temp; i++) {  
            if (i < 3)  
                fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i);  
            if ((i >= 3) && (i <= 8))  
                fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+1);  
            if ((i >= 9) && (i <= 11))  
                fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+2);
```

```

    if (i == 12)
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+3);
    if ((i >= 13) && (i <= 16))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+4);
    if ((i >= 17) && (i <= 19))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+5);
    if ((i >= 20) && (i <= 25))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+6);
    if ((i >= 26) && (i <= 28))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+14);
    if (i == 29)
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+15);
}
}

if ((total > 30) && (total <= 53)) {
    fprintf( to, "NET PAD-C; ; PIN:52.\n");

    for(i = 1; i <= temp; i++) {
        if (i < 4)
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i);
        if ((i >= 4) && (i <= 12))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+1);
        if ((i >= 13) && (i <= 17))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+3);
        if ((i >= 18) && (i <= 20))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+4);
        if ((i >= 21) && (i <= 26))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+5);
        if ((i >= 27) && (i <= 31))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+6);
        if ((i >= 32) && (i <= 41))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+7);
        if ((i >= 42) && (i <= 43))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+8);
        if (i == 44)
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+9);
        if ((i >= 45) && (i <= 50))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+15);
        if ((i >= 51) && (i <= 52))
            fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+16);
    }
}

if ((total > 53) && (total <= 65)) {

```

```

fprintf( to, "NET PAD-C; ; PIN:64.\n");

for(i = 1; i <= temp; i++) {
    if (i < 4)
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i);
    if ((i >= 4) && (i <= 10))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+1);
    if ((i >= 11) && (i <= 15))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+2);
    if ((i >= 16) && (i <= 20))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+4);
    if ((i >= 21) && (i <= 26))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+6);
    if ((i >= 27) && (i <= 32))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+7);
    if ((i >= 33) && (i <= 37))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+8);
    if ((i >= 38) && (i <= 50))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+9);
    if ((i >= 51) && (i <= 52))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+11);
    if (i == 53)
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+12);
    if ((i >= 54) && (i <= 56))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+15);
    if ((i >= 57) && (i <= 62))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+19);
    if ((i >= 63) && (i <= 64))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+20);
}
}

/* fprintf( to, "NET PAD-C; ; PIN:90.\n");

```

```

for(i = 1; i <= temp; i++) {
    if (i < 8)
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+5);
    if ((i >= 8) && (i <= 12))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+6);
    if ((i >= 13) && (i <= 19))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+7);
    if ((i >= 20) && (i <= 24))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+11);
    if ((i >= 25) && (i <= 29))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+13);
}

```

```

    if ((i >= 30) && (i <= 35))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+15);
    if ((i >= 36) && (i <= 41))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+21);
    if ((i >= 42) && (i <= 46))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+22);
    if ((i >= 47) && (i <= 53))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+23);
    if ((i >= 54) && (i <= 59))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+26);
    if ((i >= 60) && (i <= 61))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+28);
    if (i == 62)
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+29);
    if ((i >= 63) && (i <= 68))
        fprintf( to, "NET PAD-%d; ; PIN:%d.\n", i, i+32);
} */

fclose(to);
}

```

```

:::::::::::::
report-act1.c
:::::::::::::

/* Writes in report.txt file for the ACT-1 part. */

#include "stdio.h"
#include "stdlib.h"
#include "string.h"

void main ( argc, argv )
int argc;
int *argv[ ];

{
    FILE *from,*to;
    char str[1000];
    char str1[100], str2[100];
    int i;

    /*see for correct # of command line arguments */
    if (argc < 1) {
        printf("Usage: create <IN-PINS> <OUT-PINS>\n");
        exit(1);
    }

    /* Open Source File
    if((from = fopen("chip.out","r"))==NULL) {
        printf("Cannot open source file\n");
        exit(1);
    } */

    /* Open Destination File */
    if((to = fopen("report.txt","a+"))==NULL){
        printf("Cannot open file\n");
        exit(1);
    }

    if((!(strcmp(argv[1], "not-actel")))) {
        fprintf(to, "\n\n");
        fprintf(to, "\n                ACTEL - ACT1  ");
        fprintf(to, "\n                -----  \n");
        fprintf(to, "The test for ACT-1 series is complete\n");
        fprintf(to, "Your design doesn't fit in Act-1 series\n");
    }
}

```

```

}

else {
    fprintf(to, "\n\n");
    fprintf(to, "\n                ACTEL - ACT1  ");
    fprintf(to, "\n                ----- \n");
    fprintf(to, "The test for ACT-1 series is complete\n");
    fprintf(to, "The best available part in Act1 series for\n");
    fprintf(to, "your design is %s\n", argv[1]);
    fprintf(to, "For more information check %s.log\n", argv[1]);
}

fclose(to);
}

```

```

:::::::::::::
report3000.c
:::::::::::::

/* Writes in report.txt file for the X3000 part. */

#include "stdio.h"
#include "stdlib.h"
#include "string.h"

void main ( argc, argv )
int argc;
int *argv[ ];

{

    FILE *from,*to;
    char str[1000];
    char str1[100], str2[100];
    int i;

    /*see for correct # of command line arguments */
    if (argc < 1) {
        printf("Usage: create <IN-PINS> <OUT-PINS>\n");
        exit(1);
    }

    /* Open Source File
    if((from = fopen("chip.out","r"))==NULL) {
        printf("Cannot open source file\n");
        exit(1);
    } */

    /* Open Destination File */
    if((to = fopen("report.txt","a+"))==NULL){
        printf("Cannot open file\n");
        exit(1);
    }

    if((!(strcmp(argv[1], "not-3000")))) {
        fprintf(to, "\n\n");
        fprintf(to, "\n                XILINX - 3000  ");
        fprintf(to, "\n                -----  \n");
        fprintf(to, "The test for Xilinx 3000 series is

```



```

        complete\n");
        fprintf(to, "Your design doesn't fit in Xilinx 3000
        series\n");
    }

    else {
        fprintf(to, "\n\n");
        fprintf(to, "\n                XILINX - 3000  ");
        fprintf(to, "\n                -----  \n");
        fprintf(to, "The test for Xilinx 3000 series is
        complete\n");
        fprintf(to, "The best available part in X3000 series
        for your design is %s\n", argv[1]);
        fprintf(to, "For more information check %s.rpt and
        %s.out\n", argv[1], argv[1]);
    }

    fclose(to);
}

```

```

:::::::::::::
report4000.c
:::::::::::::

/* Writes in report.txt file for the x4000 part. */

#include "stdio.h"
#include "stdlib.h"
#include "string.h"

void main ( argc, argv )
int argc;
int *argv[ ];

{

    FILE *from,*to;
    char str[1000];
    char str1[100], str2[100];
    int i;

    /*see for correct # of command line arguments */
    if (argc < 1) {
        printf("Usage: create <IN-PINS> <OUT-PINS>\n");
        exit(1);
    }

    /* Open Source File
    if((from = fopen("chip.out","r"))==NULL) {
        printf("Cannot open source file\n");
        exit(1);
    } */

    /* Open Destination File */
    if((to = fopen("report.txt","w"))==NULL){
        printf("Cannot open file\n");
        exit(1);
    }

    fprintf(to, "\n                XILINX - 4000 ");
    fprintf(to, "\n                ----- \n");
    fprintf(to, "The test for Xilinx 4000 series is complete\n");
    fprintf(to, "The best available part in X4000 series for
    your design is %s\n", argv[1]);

```

```
fprintf(to, "For more information check %s.rpt and %s.out\n",  
argv[1], argv[1]);  
fclose(to);  
}
```

APPENDIX C

VHDL code for test case # 1

```
.....:
card.vhd
.....:

-- The card module discards all values that are either 0 or
-- greater than 13.
library synth,work;
use synth.stdsynth.all, work.techdef.all;

entity card is port(
    signal clk      : in vlbit;
    signal din      : in vlbit_1d(3 downto 0);
    signal value    : out vlbit_1d(3 downto 0)
);
end card;

architecture behavior of card is
    signal n_state, p_state : vlbit_1d(3 downto 0);
    signal clock: vlbit;

begin
    clock <= FALLING xor clk;
    dff_v(n_state, clock, p_state);
    value <= p_state(3 downto 0);

    process (din, p_state)
    begin
        if ((din = B"0000") or (din > B"1101")) then
            n_state <= p_state;
        elsif ((din = B"1011") or (din = B"1100") or (din = B"1101")) then
            n_state <= B"1010";
        else
            n_state <= din;
        end if;
    end process;
end behavior;
```

```

:::::::::::::
dealer.vhd
:::::::::::::

```

--The Dealer module is the finite state machine.

```

library synth,work;
use synth.stdsynth.all, work.techdef.all;

```

```

entity dealer is port(

```

```

-- inputs

```

```

    signal pb:          in vlbit_1d(3 downto 0);
    signal tf:          in vlbit;
    signal clk:         in vlbit;
    signal reset:       in vlbit;
    signal p:           in vlbit;
    signal c:           in vlbit_1d(3 downto 0);

```

```

-- outputs

```

```

    signal add:         out vlbit_1d(2 downto 0);
    signal st:          out vlbit;
    signal db:          out vlbit_1d(3 downto 0);
    signal d:           out vlbit;
    signal hold:        out vlbit;
    signal gameover:    out vlbit

```

```

);

```

```

end dealer;

```

```

architecture behavior of dealer is

```

```

    signal add_bus:      vlbit_1d(2 downto 0);
    signal p_state:      vlbit_1d(4 downto 0);
    signal n_state:      vlbit_1d(4 downto 0);
    signal temp:         vlbit_1d(3 downto 0);
    signal clk_l:        vlbit;

```

```

    constant ATTENTION:  vlbit_1d(3 downto 0) := B"1111";
    constant UPCARD:     vlbit_1d(3 downto 0) := B"1110";

```

```

    constant ACK         :  vlbit_1d(3 downto 0) := B"1111";
    constant HIT         :  vlbit_1d(3 downto 0) := B"0001";
    constant STAND       :  vlbit_1d(3 downto 0) := B"0010";
    constant BUST        :  vlbit_1d(3 downto 0) := B"0100";
    constant BLACKJACK   :  vlbit_1d(3 downto 0) := B"1000";

```

```

constant s0:          vlbit_1d(4 downto 0) := B"00000";
constant s1:          vlbit_1d(4 downto 0) := B"00001";
constant s2:          vlbit_1d(4 downto 0) := B"00010";
constant s3:          vlbit_1d(4 downto 0) := B"00011";
constant s4:          vlbit_1d(4 downto 0) := B"00100";
constant s5:          vlbit_1d(4 downto 0) := B"00101";
constant s6:          vlbit_1d(4 downto 0) := B"00110";
constant s7:          vlbit_1d(4 downto 0) := B"00111";
constant s8:          vlbit_1d(4 downto 0) := B"01000";
constant s9:          vlbit_1d(4 downto 0) := B"01001";
constant s10:         vlbit_1d(4 downto 0) := B"01010";
constant s11:         vlbit_1d(4 downto 0) := B"01011";
constant s12:         vlbit_1d(4 downto 0) := B"01100";
constant s13:         vlbit_1d(4 downto 0) := B"01101";
constant s14:         vlbit_1d(4 downto 0) := B"01110";
constant s15:         vlbit_1d(4 downto 0) := B"01111";
constant s16:         vlbit_1d(4 downto 0) := B"10000";
constant s17:         vlbit_1d(4 downto 0) := B"10001";
constant s18:         vlbit_1d(4 downto 0) := B"10010";
constant s19:         vlbit_1d(4 downto 0) := B"10011";
constant s20:         vlbit_1d(4 downto 0) := B"10100";

```

```
begin
```

```
clk_l <= FALLING xor clk;
```

```
-- this is for the state registers
```

```
state          :dff_v(n_state,clk_l,p_state);
```

```
address        :dff_v(add,clk_l,add_bus);
```

```
process( p_state, tf, p, pb, db, c,reset,add_bus,temp)
```

```
begin
```

```
-- default assignments -- VERY IMPORTANT
```

```
    n_state <= p_state;
```

```
    st <= '0';
```

```
    add <= B"111";
```

```
    db <= B"0000";
```

```
    d <= '0';
```

```
    hold <= '0';
```

```
    gameover <= '0';
```

```
    add <= add_bus;
```

```
-----
--This resets all the flip-flops to the following values
```

```

if (reset = '1') then
    n_state <= s0;
    st<='0';
    add <= B"000";
    db <= B"0000";
    temp <= B"0000";
    d<='0';
    hold <= '0';
    gameover <= '0';

else

    case p_state(4 downto 0) is -- send attention signal to
                                -- the players
        when s0 =>
            st<='1';
            d<='1'; -- signal player
            db <= ATTENTION;
            n_state <= s1;

        -----
        -- begin hand shake state
        when s1 =>
            if (p='0') then
                if (tf='0') then
                    n_state <= s1; -- wait
                    db <= ATTENTION;
                    d<='1';
                else
                    n_state <= s4;
                end if;
            else
                if (pb=ACK) then
                    n_state <= s3;
                    d<='0'; -- lower dealer signal
                    st<='1'; -- begin delay counter
                    hold <= '1';
                else
                    n_state <= s4;
                end if;
            end if;

        -----
        -- give player first card state
        when s3 =>

```

```

    if (tf = '0') then
        db <= c;
        d <= '1';           -- signal player at current
                             -- address that card is
                             -- available for reading.

        hold <= '1';
        n_state <= s3;       -- wait
    else
        n_state <= s4;       -- time is up go to the
                             -- next player

        d <= '0';
        db <= B"0000";
        hold <= '0';

    end if;

-----
-- increment address values here
    when s4 =>
        if (add_bus = B"000") then
            add <= B"001";
            n_state <= s0;
        end if;

        if (add_bus = B"001") then
            add <= B"010";
            n_state <= s0;
        end if;

        if (add_bus = B"010") then
            add <= B"011";
            n_state <= s0;
        end if;

        if (add_bus = B"011") then
            add <= B"100";
            n_state <= s0;
        end if;

        if (add_bus = B"100") then
            add <= B"101";
            n_state <= s0;
        end if;

    -- last player(dealer-player)

```



```

        if (add_bus = B"101") then
            add <= B"000";
            n_state <= s5;
        end if;

-----
-----
-- starting part for second round of cards

    when s5 =>
        n_state <= s6;
        st<='1';
        hold <= '1';

-----
-----
-- concluding part for second round of cards

    when s6 =>
        if (tf = '0') then
            db <= c;
            d <= '1';
            -- signal player at current
            -- address that card is
            -- available for reading.
            hold <= '1';
            n_state <= s6;      -- wait
        else
            n_state <= s7;      -- time is up go to the
            -- next player
            d <= '0';
            db <= B"0000";
            hold <= '0';
        end if;

-----
-- increment address values here for round two

    when s7 =>
        if (add_bus = B"000") then
            add <= B"001";
            n_state <= s5;
        end if;

```

```

if (add_bus = B"001") then
    add <= B"010";
    n_state <= s5;
end if;

```

```

if (add_bus = B"010") then
    add <= B"011";
    n_state <= s5;
    n_state <= s5;
end if;

```

```

if (add_bus = B"011") then
    add <= B"100";
    n_state <= s5;
end if;

```

```

if (add_bus = B"100") then
    add <= B"101";
    n_state <= s8;
    st <= '1';
    hold <= '1';
end if;

```

```

-----
-----

```

```

-- give dealer player second card (or the upcard )

```

```

when s8 =>
    if (tf = '0') then
        d <= '1';
        db <= c;
        temp <= c;
        hold <= '1';
        n_state <= s8;  -- wait
    else
        n_state <= s9;  -- time is up go to the
-- next player
        d <= '0';
        add <= B"000";
        hold <= '0';
    end if;

```

```

-----
-----

```

```

-- starting part of giving players, upcard

    when s9 =>
        n_state <= s2;
        st<='1';

-----
-- give players upcard prompt

    when s2 =>
        if (tf = '0') then
            db <= UPCARD;
            d <= '1';
            n_state <= s2;      --wait
        else
            n_state <= s16;
            d <= '0';
            db <= UPCARD;
        end if;

-----

    when s16 =>
        n_state <= s10;
        db <= UPCARD;
        st<='1';

-----
-- concluding part of giving players, upcard

    when s10 =>
        if (tf = '0') then
            db <= temp;
            d <= '1';
            n_state <= s10;    -- wait
        else
            n_state <= s11;    -- time is up go to the
-- next player
            d <= '0';
            db <= B"0000";
        end if;

-----
-- increment address values here for giving upcard

    when s11 =>

```

```

if (add_bus = B"000") then
    add <= B"001";
    n_state <= s9;
end if;

if (add_bus = B"001") then
    add <= B"010";
    n_state <= s9;
end if;

if (add_bus = B"010") then
    add <= B"011";
    n_state <= s9;
end if;

if (add_bus = B"011") then
    add <= B"100";
    n_state <= s9;
end if;

if (add_bus = B"100") then
    add <= B"101";
    n_state <= s9;
end if;

if (add_bus = B"101") then
    add <= B"000";
    n_state <= s12;
end if;

```

```

when s12 =>
    n_state <= s13;
    st <= '1';
    d <= '0';
    db <= B"0000";
    hold <= '1';

```

```

when s13 =>
    if (tf='1') then
        n_state <= s15;
        hold <= '0';
    end if;

```

```

else
    if (p='1') then
        if (pb=HIT) then
            db <= c;
            d <= '1';
            hold <= '1';
            n_state <= s14;
        else
            hold <= '0';
            d <= '1';
            n_state <= s17;
        end if;
    else
        n_state <= s13;
    end if;
end if;
-----

```

```

when s14 =>
    if (tf='0') then
        db <= c;
        d <= '1';
        hold <= '1';
        n_state <= s14;
    else
        hold <= '0';
        n_state <= s18;
    end if;
-----

```

```

when s18 =>
    d <= '0';
    db <= B"0000";
    n_state <= s19;
-----

```

```

when s19 =>
    d <= '0';
    db <= B"0000";
    n_state <= s20;
-----

```

```

when s20 =>
    d <= '0';
    db <= B"0000";
    n_state <= s12;
-----

```

```

when s17 =>
    d <= '1';
    n_state <= s15;
-----

when s15 =>
    if (add_bus = B"000") then
        add <= B"001";
        n_state <= s12;
    end if;

    if (add_bus = B"001") then
        add <= B"010";
        n_state <= s12;
    end if;

    if (add_bus = B"010") then
        add <= B"011";
        n_state <= s12;
    end if;

    if (add_bus = B"011") then
        add <= B"100";
        n_state <= s12;
    end if;

    if (add_bus = B"100") then
        add <= B"101";
        n_state <= s12;
    end if;

    if (add_bus = B"101") then
        gameover <= '1';
    end if;
-----

end case;
end if;

end process;
end behavior;

```

```

:::::::::::::
display.vhd
:::::::::::::

-- The display module displays the address of the player with
-- which FSM is playing.

-- vhdl model for
--*****
-- IO Interface Declaration
--*****

entity display is
port
(
-- inputs
  signal add: in vlbit_1d(2 downto 0);
  signal gameover: in vlbit;

-- outputs
  signal out1: out vlbit_1d(6 downto 0);
  signal out2: out vlbit_1d(6 downto 0)
);
end display;

--*****
-- Architecture body
--*****

architecture behavior of display is
begin
  process (add, gameover)
  begin

    if (gameover = '1') then
      out1 <= B"0100000";
      out2 <= B"0000001";          --G0
    else
      if (add = B"000") then
        out1 <= B"0001000";
        out2 <= B"0000001"; --A0
      end if;
      if (add = B"001") then
        out1 <= B"0001000";
        out2 <= B"1001111"; --A1
      end if;
    end if;
  end process;
end behavior;

```

```

        end if;
        if (add = B"010") then
            out1 <= B"0001000";
            out2 <= B"0010010"; --A2
        end if;
        if (add = B"011") then
            out1 <= B"0001000";
            out2 <= B"0000110"; --A3
        end if;
        if (add = B"100") then
            out1 <= B"0001000";
            out2 <= B"1001100"; --A4
        end if;
        if (add = B"101") then
            out1 <= B"0001000";
            out2 <= B"0100100"; --A4
        end if;

        end if;
    end process;

end behavior;

```



```

::::::::::::
filter.vhd
::::::::::::

-- The filter module checks for valid signal form the player.

library synth;
use synth.stdsynth.all, work.techdef.all;

entity filter is
port
(
--inputs
    signal p:          in vlbit;
    signal pl0:         in vlbit;
    signal pl1:         in vlbit;
    signal pl2:         in vlbit;
    signal pl3:         in vlbit;

--outputs
    signal pout:        out vlbit;
    signal plout0:      out vlbit;
    signal plout1:      out vlbit;
    signal plout2:      out vlbit;
    signal plout3:      out vlbit
);
end filter;

architecture behavior of filter is
begin
    pout <= '1' when p='1' else
           '0' when p='0' else
           '0' when p='X' else '0';
    plout0 <= '1' when pl0='1' else
             '0' when pl0='0' else
             '0' when pl0='X' else '0';
    plout1 <= '1' when pl1='1' else
             '0' when pl1='0' else
             '0' when pl1='X' else '0';
    plout2 <= '1' when pl2='1' else
             '0' when pl2='0' else
             '0' when pl2='X' else '0';
    plout3 <= '1' when pl3='1' else
             '0' when pl3='0' else
             '0' when pl3='X' else '0';

```

end behavior;

```

::::::::::::
prbs.vhd
::::::::::::

-- The PRBS module is a pseudo-random binary sequence generator.

library synth,work;
use synth.stdsynth.all, work.techdef.all;

entity prbs is port(
signal din      : in vlbit_1d(5 downto 0);
signal clk      : in vlbit;
signal reset    : in vlbit;
signal hold     : in vlbit;
signal dout     : out vlbit_1d(3 downto 0)
);
end prbs;

architecture behavior of prbs is

signal p_state, n_state: vlbit_1d(5 downto 0);
signal clock : vlbit;

begin
clock <= FALLING xor clk;
dff_v(n_state, clock, p_state);
dout <= p_state(3 downto 0);

process (din, p_state, reset, hold)
begin
    if (reset = '1') then
        n_state <= din;
    else
        if (hold = '1') then
            n_state <= p_state;
        else
            n_state(5) <= p_state(0) xor p_state(1);
            for i in 5 downto 1 loop
                n_state(i-1) <= p_state(i);
            end loop;
        end if;
    end if;
end if;

end process;
end behavior;

```

```

:::::::::::::
timer.vhd
:::::::::::::

-- The timer module keeps track of the elapsed time.

library synth;
use synth.stdsynth.all, work.techdef.all;

entity timer is
port
(
--inputs
    signal clk:          in vlbit;
    signal st:           in vlbit;

--outputs
    signal tf:           out vlbit
);
end timer;

--*****
-- Architecture body
--*****

architecture behavior of timer is
    signal clk_l:        vlbit;
    signal n_state:      vlbit_1d(3 downto 0);
    signal p_state:      vlbit_1d(3 downto 0);

    constant s0:         vlbit_1d(3 downto 0) := B"0000";
    constant s1:         vlbit_1d(3 downto 0) := B"0001";
    constant s2:         vlbit_1d(3 downto 0) := B"0010";
    constant s3:         vlbit_1d(3 downto 0) := B"0011";
    constant s4:         vlbit_1d(3 downto 0) := B"0100";
    constant s5:         vlbit_1d(3 downto 0) := B"0101";
    constant s6:         vlbit_1d(3 downto 0) := B"0110";
    constant s7:         vlbit_1d(3 downto 0) := B"0111";
    constant s8:         vlbit_1d(3 downto 0) := B"1000";
    constant s9:         vlbit_1d(3 downto 0) := B"1001";
    constant s10:        vlbit_1d(3 downto 0) := B"1010";

begin

```

```

clk_l <= FALLING xor clk;

dff_v(n_state,clk_l,p_state);
--dff(tf_in,clk_l,tf_out);

--tf <= tf_out;

counter:process(st, p_state)

begin

    n_state <= p_state;
--    tf_in <= tf_out;
    tf <= '0';

    if st = '1' then
        n_state <= s0;
    else
        if p_state = s0 then n_state <= s1; end if;
        if p_state = s1 then n_state <= s2; end if;
        if p_state = s2 then n_state <= s3; end if;
        if p_state = s3 then n_state <= s4; end if;
        if p_state = s4 then n_state <= s5; end if;
        if p_state = s5 then n_state <= s6; end if;
        if p_state = s6 then n_state <= s7; end if;
        if p_state = s7 then n_state <= s8; end if;
        if p_state = s8 then n_state <= s9; end if;
        if p_state = s9 then tf <= '1'; n_state <= s10;
    end if;
    end if;

end process counter;

end behavior;

```

VITA

Akhil Bagri was born in Raipur, India on April 7, 1969. After graduating from the Holy Cross H. S. School, Raipur in 1985 with honors, he entered the Government Engineering College, Raipur for his undergraduate studies. He was awarded a Bachelor of Engineering degree in Electrical Engineering in August of 1990. In August of 1991 he joined the University of Tennessee for the graduate program in Electrical Engineering. He was awarded a Master of Science degree in Electrical Engineering in August of 1994.