

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

**MULTI-TARGET AHPL NETLIST FORMAT
TRANSLATOR**

by
Teng-I Wang

A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE
In the Graduate College
THE UNIVERSITY OF ARIZONA

1 9 9 5

UMI Number: 1375991

UMI Microform 1375991

Copyright 1995, by UMI Company. All rights reserved.

**This microform edition is protected against unauthorized
copying under Title 17, United States Code.**

UMI

**300 North Zeeb Road
Ann Arbor, MI 48103**

STATEMENT BY AUTHOR

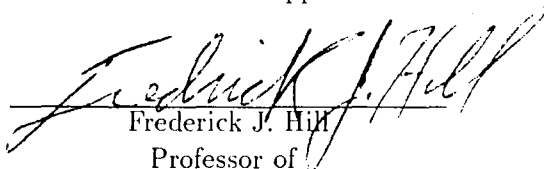
This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: 

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:


Frederick J. Hill
Professor of

Electrical and Computer Engineering


Date

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Frederick J. Hill for his patience, encouragement and guidance throughout this research. I consider it an honor to be able to work with him.

I would also like to thank my friends Mr. Chung-Yen Chang and Mr. Gong-San Yu for advising me both academically and spiritually.

Lastly, I would like to express my gratitude to my parents and my brother. Studying abroad has been a dream of mine since I was young. It is their continued strong encouragement and support that make this dream come true.

TABLE OF CONTENTS

LIST OF FIGURES	7
LIST OF TABLES	9
ABSTRACT	10
1. Introduction	11
1.1. Hardware Description Language and AHPL	11
1.2. Introduction to the AHPL Synthesis System	12
1.3. Goals of the Project	13
2. Review of The AENT91	15
2.1. Functions of the AENT91	15
2.2. Limitations of the AENT91	16
3. Expanding the AENT91	20
3.1. General Considerations of the Tool System	21
3.2. Element Modeling	22
3.3. The Libraries	26
3.3.1. Tool Dependent Style and Information	26
3.3.2. Gate Information for PLEs, FLEs and LETs	27
3.3.3. The LETs for DFFs and I/O Pads	28
3.3.4. I/O pin attributes	30
3.4. The Bus sub-type and the Buses	31
3.5. Fanin limit	32
3.6. Fanout limit and Drive Capability	33
3.7. The BLACK BOX Elements and Special Gate Mapping	34
3.8. Inverted and Invertible Inputs and Outputs	35
3.9. The Output files and Their Formats	36
3.10. The ANFT Organization	37
4. Implementation	40
4.1. The Element Types	40
4.2. The Data Structures	41
4.3. Library Format	47
4.4. Default Object Transformation	55
4.4.1. Redundant Signal and Object Trimming	56

4.4.2.	Bus Construction	57
4.4.3.	I/O pad Construction	62
4.4.4.	Gate Fitting Using PLE Models	66
4.4.5.	NAND-NAND Implementation	68
4.4.6.	BLACK BOX Fitting Using FLEs	70
4.4.7.	DFE Fitting using LETs	73
4.5.	Special Gate Mapping	77
4.6.	Invertability Utilization	78
4.7.	Fanout Limit	79
4.8.	Output File Generation	81
5.	Using The ANFT	83
5.1.	Execution Mode and Option	83
5.2.	Applications of ANFT Output	87
5.2.1.	Porting ANFT Output to L-Edit TM and GateSim TM	88
5.2.2.	Porting ANFT Output to OrCAD/VST TM	89
5.2.3.	Porting the ANFT Output to Xilinx TM 's Tool System	90
5.2.4.	Porting the ANFT Output to COMPASS Tools TM	92
6.	Examples and Evaluation	93
6.1.	The AHPL Demo Description: DEMO	93
6.1.1.	Porting DEMO to GateSim TM Using AENT91 and ANFT	95
6.1.2.	Porting DEMO to OrCAD/VST TM Using AENT_TTL and ANFT	97
6.1.3.	Incorporating BLACK BOX in DEMO.HPC	99
6.2.	The Time-of-Day Clock: TODCLK	101
6.2.1.	Test Using GateSim TM	104
6.2.2.	Test Using OrCAD/VST TM	111
6.2.3.	Test Using Xilinx TM 's Tool System	115
6.2.4.	Test Using COMPASS Tools TM System	121
6.3.	Evaluation	125
6.3.1.	Comparison of ANFT and AENT91 Results	126
6.3.2.	Comparison of ANFT and AENT_TTL Results	132
7.	Conclusion	135
7.1.	Final Thought	136
7.2.	Future Work	137
Appendix A. ANFT Libraries		139

Appendix B. ANFT Translated Output and Simulation Command Files of DEMOX	154
Appendix C. TODCLK Simulation Files for GateSim™ Ver 1.11 . . .	162
Appendix D. TODCLK Simulation Files for OrCAD/VST™ Ver 1.10	166
Appendix E. TODCLK Simulation Files for Xilinx™'s Tool System .	169
Appendix F. TODCLK Simulation Files for COMPASS Tools™ System V8R4.4	172
Appendix G. SWA416 Detail Critical Path Timing Information	175
REFERENCES	179

LIST OF FIGURES

1.1. The AHPL Design Flow.	13
2.1. The AENT91 Processes Flow.	16
3.1. The ANFT Processes Flow.	22
3.2. The Sub-type Parameter and Its Effect.	32
3.3. Tree Buffering for High Fanout Gate.	34
3.4. Example of Utilizing the Inverted Output Pin.	36
3.5. The Organization of ANFT.	38
4.1. Typical Realization of An AHPL Netlist.	43
4.2. Data Structure of ANFT Internal Element.	44
4.3. The Dynamic Element Array.	46
4.4. The Library Format.	48
4.5. Default Object Transformation Flow.	55
4.6. Source of Redundant Gates.	56
4.7. Bus Construction Flow.	58
4.8. Example of Constructing a BUS (4014) object.	58
4.9. Example of Erroneous BUS Element Connections.	59
4.10. Example of Constructing an EXBUS (4020) object.	60
4.11. Example of Constructing EX- and OUTPUT (4019/4013) objects.	61
4.12. Example of Using the IPAD.	63
4.13. Example of Using the OPAD.	64
4.14. Example of Using the BPAD.	65
4.15. Gate Fitting Process Flow.	66
4.16. Example of Typical Gate Fitting Results.	69
4.17. Examples of AND to NAND Conversion.	70
4.18. Examples of Declaring and Using BLACK BOXes.	72
4.19. Redirecting the Preset Signal.	74
4.20. Redirecting the Clock Enable Signal of a DFF object.	76
4.21. Clock Signal Connections.	76
4.22. Example of Mapping the AOI21 Gate.	77
4.23. Example of Utilizing The Invertible Input Pins.	78
4.24. Example of Using Buffer Tree.	80
4.25. Example of Pin-to-Pin Connection Netlist Style.	81
4.26. Example of I/O Connection Netlist Style.	82

5.1. Three Types of XOR Equivalent Logic.	87
5.2. Application Using L-Edit™ and GateSim™.	88
5.3. Application Using OrCAD/VST™.	90
5.4. Application Using Xilinx™'s Tool System.	91
5.5. Application Using COMPASS Tools™ System.	92
6.1. The DEMO AHPL Description.	94
6.2. Simulation Results of DEMO Using GateSim™.	96
6.3. Simulation Results of DEMO Using OrCAD/VST™.	98
6.4. AHPL Benchmark DEMOX.HPC.	99
6.5. The Schematic Diagram of DEMOX.	100
6.6. Simulation Result of DEMOX Using COMPASS Tools™.	100
6.7. Simulation Results of TODCLK Using GateSim™.	105
6.7. Continued.	106
6.7. Continued.	107
6.7. Continued.	108
6.7. Continued.	109
6.7. Continued.	110
6.8. Simulation Results of TODCLK Using OrCAD/VST™.	112
6.8. Continued.	113
6.8. Continued.	114
6.9. Simulation Results of TODCLK Using Xilinx™'s Tool System.	116
6.9. Continued.	117
6.9. Continued.	118
6.9. Continued.	119
6.10. Simulation Results of TODCLK Using COMPASS Tools™.	122
6.10. Continued.	123
6.10. Continued.	124

LIST OF TABLES

3.1. HPCOM object Codes and their logic function.	23
3.2. Type of Required Information.	28
4.1. The Element Type Definition of AENT91.	41
4.2. The Element Type Definition of ANFT.	42
4.3. I/O Pin Attribute Symbols.	49
4.4. The Tokens of The Logic Tree Expression.	54
6.1. Translation Settings for DEMO Using AENT91 and ANFT.	95
6.2. Translation Settings for DEMO Using AENT_TTL and ANFT.	97
6.3. Settings for Running ANFT and GateSim TM	104
6.4. Settings for Running ANFT and OrCAD/VST TM	111
6.5. Settings for Running ANFT and Xilinx TM 's Tools.	115
6.6. Settings for Running ANFT and COMPASS Tools TM	121
6.7. Characteristics of AHPL Benchmarks.	125
6.8. ANFT and AENT91 AND-OR-INV Implementation Results.	127
6.9. ANFT and AENT91 NAND-NAND Implementation Results.	128
6.10. Comparing the ANFT and AENT91 Results.	129
6.11. ANFT and AENT_TTL AND-OR-INV Implementation Results and Comparison.	132
6.12. ANFT and AENT_TTL NAND-NAND Implementation Results and Comparison.	133

ABSTRACT

AENT91 and AENT_TTL are AHPL to EDIF netlist translator for the AHPL hardware programming language. Although they are able to provide connections to L-EditTM, GateSimTM and OrCAD/VSTTM, some embedded limitations inhibit them from providing more useful connections to other advanced commercial CAD tools. This thesis describes a new AHPL Netlist Format Translator (ANFT) which serves as a multi-target translator with some local optimization processes for AHPL. The limitations of AENT91 and AENT_TTL as well as the design considerations of ANFT are discussed in the first of the three parts of this thesis. Implementation details are provided in the second part. In the last part, tests of ANFT output files and comparisons of results from AENT91, AENT_TTL and ANFT translated AHPL benchmarks are presented. The results show that ANFT not only provides better results but also successfully connects AHPL to more commercial CAD tool systems.

CHAPTER 1

Introduction

1.1 Hardware Description Language and AHPL

Integrated VLSI synthesis systems have been playing important roles in VLSI design automation. These synthesis systems usually incorporate some Hardware Description Languages, HDLs, as an indispensable link between themselves and designers. HDLs provide designers a compact and most importantly a formal procedure for their design tasks. The designers can easily translate ideas into descriptions in HDLs, step by step and module by module. These descriptions can be compiled, optimized and verified in an integrated fashion. The benefits of using such a design approach are productivity and economics.

HDLs can be classified into four categories or levels of abstraction [1] including: (1) System level, (2) Register transfer level, (3) Gate level and (4) transistor level. The system level descriptions have the highest abstraction and lowest detail while the transistor level ones have the lowest abstraction and highest detail. Designs using the system level descriptions can be easy. However, designers can have little control over the final realization. Yet both the gate and transistor level descriptions require the specifications such as wiring and structural details from designer. Only the register transfer level descriptions can save designers from handling unnecessary detail

without losing too much control over the actual realization. Timing constraints can also be specified in this level. Therefore, the designers have large degree of freedom to generate desired designs in a fairly convenient way.

A Hardware Programming Language, AHPL [2], is a register transfer level HDL for synchronized clock-mode designs. It has been successfully applied to many VLSI design tasks [3] [4] [5] [6] [7]. Several powerful CAD tools have been developed to help the designers accomplishing their design tasks in various computing environments.

1.2 Introduction to the AHPL Synthesis System

The AHPL synthesis system uses a design process that is based on a describe-and-synthesis methodology. As shown in Figure 1.1, designers begin by translating their ideas into AHPL descriptions. The designs can then be simulated and verified using HPSIM, a function level AHPL simulator [8].

Once the function of the designs are verified, they can be compiled into a netlist using HPCOM. HPCOM is a three-stage hardware compiler for AHPL. Stage 1 of HPCOM performs the semantic and syntactical error checking and translates the AHPL descriptions into tabular representations. Stage 2 reconstructs the outputs of stage 1 into internal element interconnections. It also performs the control logic minimizations and some data logic optimizations [9]. Stage 3 produces abstract gate-level AHPL netlist output files. They are abstract netlists in the sense that the internal elements are independent from application details and technology information.

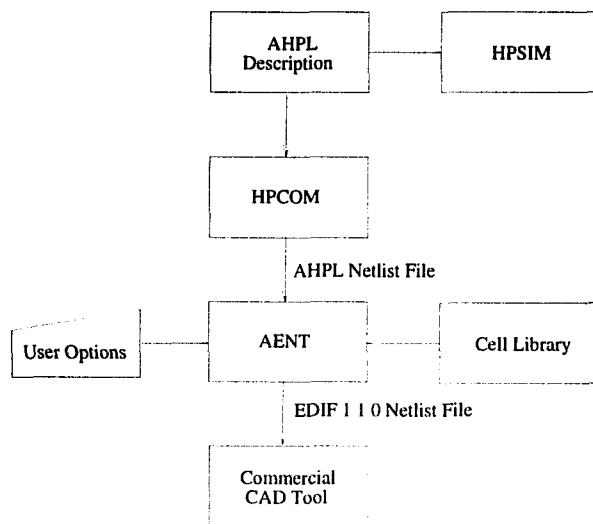


Figure 1.1: The AHPL Design Flow.

The Stage 3 outputs can be further utilized if they are translated into netlist files in various standard formats (e.g., EDIF 1 1 0) using available gate or standard cell libraries. These transformed standard netlist files can be used as sources of many useful CAD tools.

1.3 Goals of the Project

The outputs of HPCOM stage 3 are intended for porting the designs to an almost universal list of CAD tools. However, the gaps between HPCOM and other CAD tools are not well bridged. Currently, all the available stage 3 of HPCOM are designed for translating AHPL to EDIF netlist; hence the name AHPL to EDIF Netlist Translator or AENT. There have been several versions of AENT developed namely AENT89 [10], AENT_TTL [11] and AENT91 [12]. AENT_TTL and AENT91 are most widely used

stage 3 programs today. AENT_TTL uses the OrCAD™ Systems Corporation's TTL library and generates EDIF 1 1 0 format input files for OrCAD/VST™. AENT91 is designed for generating EDIF 1 1 0 netlist files using Tenner Research incorporation's scalable CMOS standard cell library. Its outputs can be used by Nettran™ to generate input files for L-Edit™ and GateSim™.

However, these translation programs are too primitive. Even AENT91, the most recent and powerful one, can not serve as a proper link between AHPL designs and powerful commercial CAD tools. Both of them are designed for generating only one specific format netlist file. Due to this restriction, they can only make the HPCOM compatible with those CAD tools which accept EDIF 1 1 0 netlists. A new multi-target AHPL netlist format translator with enhanced functions based on AENT91 is needed. Therefore, the goals of the project point mainly in three directions:

1. Utilizing the new capabilities of the latest version of HPCOM.
2. Enhancing the functions and capabilities of the netlist translator, including post HPCOM logic optimization.
3. Porting the AHPL netlists to commercial CAD tools by generating output files in different standard netlist formats.

With these new capabilities, the usefulness of AHPL can be further expanded.

CHAPTER 2

Review of The AENT91

2.1 Functions of the AENT91

AENT91 successfully provides interfaces to L-EditTM and GateSimTM. It allows users to assign different vendor-specific libraries. Thus, its output files can be associated with different gate library specifications. It is also the first version of AENT that incorporates both optimization and implementation options. The AENT91 process flow is shown in Figure 2.1.

In the module EDIF, users can specify which library to use and set options such as NAND-NAND implementation, data D Flip-Flop (DFF) type, I/O pad generation and AND-OR-Inverter (AOI) mapping. By specifying the desired library, the designers are able to generate EDIF 1 1 0 files associated with desired logic gate or standard cell library. The NAND-NAND implementation, I/O pad generation and DFF type options allow the designs to match some specific implementation preferences. The AOI gate mapping provides the designer a way to bind the abstract AHPL netlist elements into some special designed gates.

The module INIT reads the HPCOM stage 2 output files and reconstructs the AHPL netlists in its internal form. The assigned library files will be handled by module ASSIGN. The module BUILD converts the abstract elements into real gates

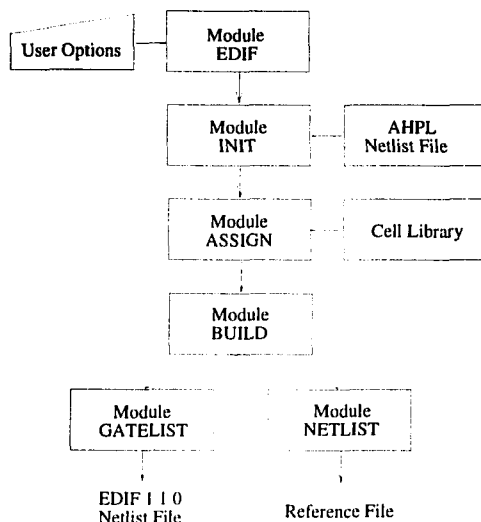


Figure 2.1: The AENT91 Processes Flow.

according to the user options. Fanin limit and some local optimizations such as AOI gates mapping, inverted DFF output utilization and redundant gate trimming are also performed in this stage. The GATELIST and NETLIST modules are two output file generators which generate readable reference file and EDIF 1 1 0 netlist file respectively.

2.2 Limitations of the AENT91

Despite the presence of many useful features mentioned, several limitations and weaknesses have been identified in the AENT91:

First, AENT91 does not include functions to handle BLACK BOX elements and bus sub-type information. There are two powerful capabilities in the latest version of

HPCOM namely BLACK BOX and bus sub-type compilation [9]. BLACK BOX compilation expands the hierarchical design capability of AHPL designs. The BLACK BOX elements can be viewed as function cells, units or even modules. It is highly possible that well defined function cells or units are included in the libraries provided. Using these special function cells can save designers a lot of works to design and test their own ones. Bus sub-type is a parameter for preserving application-dependent information assigned by designers. In AHPL compilation, variables declared as BUSES and EXBUSES will be treated as bus instances¹. They can be implemented either as AND-OR buses or tri-state buses. Designers can individually specify the type used for each bus construction. Unfortunately, AENT91 can not make use of these information.

Second, the library format is not flexible enough. It can only provide single-device package information to AENT91. Multi-device package information for describing such as TTL and CMOS chip families are not allowed. Moreover, it fails to provide the information like the triggering edge of special input pins, availability of invertible pins and drive capabilities of gates which are vital for designs that use specific gate or cell libraries.

Third, the gates provided in libraries are not fully utilized. Although users can select any library for the AENT91 (as long as the format matches,) only certain available gates can be really used. Take the AND type gates for example. Only the 2-input,

¹Although OUTPUT and EXOUTPUT will also be treated as buses, the bus sub-type can not be applied to them

3-input and 4-input AND gates can be selected and used in the implementations. Any other AND gates with input pin number different from two, three and four are skipped by the process ASSIGN. The similar situation applies to many other type of gates.

Fourth, AENT91 does not include a fanout limit process. Fanout limit is an important process for reducing circuit delay, ensuring logic correctness or protecting circuits; depending on the type and technology of the libraries used. Since AHPL netlists are abstract netlists, gates can have more than 100 fanouts in some large designs. It is necessary to apply a fanout limit process to the designs before they are converted to the final outputs.

Fifth, the mapping process used is too restricted. It can only map two specific AOI gates: AOI22 and AOI21. In addition, the processing routines are fixed. Any other special designed gates can not be mapped and bound into the AHPL netlists.

Sixth, AENT91 can only generate EDIF 1 1 0 output files. The compatibility problems between AHPL designs and other commercial CAD tools result mainly from this restriction. Although the EDIF 1 1 0 format is a commonly used standard, a newer version of it, EDIF 2 0 0 [13], has already been wildly used among CAD tools. Some of them even have their own netlist formats. It should be the responsibility of a proper HPCOM stage 3 to generate output files in different standard formats for further use.

These embedded limitations make the AENT91 unable to port the AHPL netlists to advanced commercial CAD tools. As are the goals of the project, the capability of the AENT91 must be expanded. The discussions of how these limitations and weaknesses are expanded and improved will be given in the next chapter.

CHAPTER 3

Expanding the AENT91

In this chapter, the discussions of the general ideas and considerations for designing the new AHPL Netlist Format Translator, ANFT, will be presented. The structure of the ANFT tool system will be given in the first section; followed by discussions supporting the following definitions and processes:

- Object modeling.
- Element modeling.
- Library format.
- Bus sub-type handling and bus constructions.
- Fanin and fanout limit.
- BLACK BOX handling and special gate mapping.
- Invertible and inverted I/O pin utilization.
- Output files and their formats.

Finally, the organization of the ANFT will be discussed. All the implementation details will be given in the next chapter.

3.1 General Considerations of the Tool System

ANFT is in charge of converting the abstract AHPL netlists into real implementations under different specifications. What should be the criteria for the synthesis processes? It would be too complicated for ANFT to handle all the possible requirements for different design styles. Therefore, ANFT uses two simple criteria in the synthesis processes: minimum gate count and minimum number of levels in the longest path.

According to the goals, ANFT should be capable of taking advantages of features added to the latest version of HPCOM, accepting different user options and generating output in different formats. A modular multi-stage synthesis process is needed to achieve these objectives. Different stages should be responsible for different tasks. The synthesis flow of ANFT can be roughly divided into three stages as shown in Figure 3.1. All the stages can be sub-divided into as many processing modules as required. The first stage must be able to handle the user options and set up required databases for synthesis. The databases must provide enough design and format dependent information for the following stages. After being processed by the first stage, data will be transformed into desired form in the second stage. All the processes for the transformation will follow the criteria mentioned previously. In the third stage, the desired output file will be extracted from the internal netlist.

The advantages for multiple stage system is that each stage or module can be a stand alone model as is the case for the entire AHPL synthesis system. Once a new

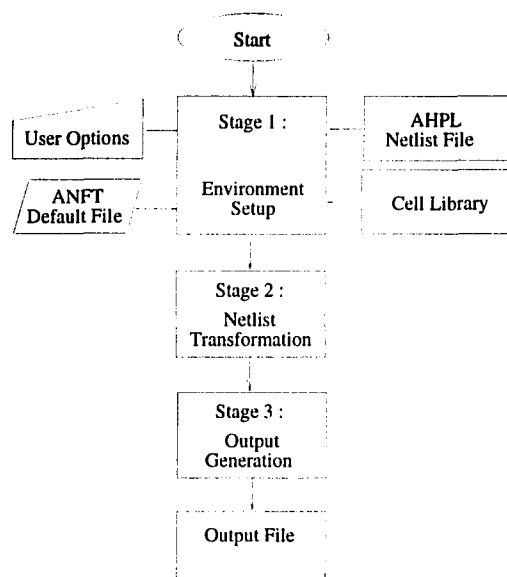


Figure 3.1: The ANFT Processes Flow.

synthesis methodology is derived for a stage or module, it can be updated without affecting the remaining modules. In addition, the system can be easily expanded as required. If any additional process has to be added to the synthesis flow then it is easy to find a place for it.

3.2 Element Modeling

The objects or elements used in the AHPL netlists are abstract logic objects¹. As shown in Table 3.1 there are twenty one different primitive object types defined and used by the HPCOM. Each of them is assigned a unique object code. The codes from

¹Unless other specified, we will use the term **object** for referring to the abstract elements in AHPL netlists and the term **gate** for the real elements that realize the objects. The term **element** can be applied to both of them.

HPCOM code	Logic function	HPCOM code	Logic function
4001	AND	4019	EXOUTPUT
4002	NAND	4020	EXBUS
4003	OR	4024	BLACK BOX
4004	XOR	4029	CND(Control unit AND)
4005	NOR	4030	OR(for No-Delay step)
4006	DFF(for control)	4031	AND(for CLU)
4009	DFF(for data)	4032	NAND(for CLU)
4012	INPUT	4033	OR(for CLU)
4013	OUTPUT	4034	XOR(for CLU)
4014	BUS	4035	NOR(for CLU)
4018	EXINPUT		

Table 3.1: HPCOM object Codes and their logic function.

4001 to 4005 and from 4029 to 4035 are for pure objects, i.e., the corresponding logic functions are single boolean operations. The 4006, 4009 and 4024 are for functional objects since they incorporate more complex logic operations. 4012 to 4020 are types for input, output and bus objects. In order to apply the intended processes to the AHPL netlists, we define the Pure Logic Element (PLE,) Function Logic Element (FLE) and Logic Element Template (LET) for modeling the objects.

Definition 3.2.1 *A pure logic element is a m -input and n -output, $0 < m, n$ and $m, n \in I$, logic instance which performs only a single boolean logic operation. The inputs and outputs serve purely for data input and output without any specific control meaning.*

Definition 3.2.2 *A functional logic element is a m -input and n -output, $0 \leq m, n$ and $m, n \in I$, functional instance which performs complex or more than one logic*

operations. The inputs serve either for data or control input and the outputs are for data output.

Definition 3.2.3 *A logic element template is a m -input and n -output, where m and n are fixed and $m, n \in I$, object template which specifies the order and availability of the element I/O pins.*

Regardless of the configurations, each of the objects used in an AHPL netlist will be modeled as either a PLE or FLE or LET individually, e.g., two 2-input AND objects will be treated as two individual instances (PLEs.)

The PLE modeling is necessary for handling the object which performs a single boolean logic operation and has no designated I/O pin usage. For the PLEs, ANFT will only be concerned with the variations of their I/O capacities since gates with suitable I/O pin numbers must be found from the assigned library for implementations. Elements in pure object types (AND, OR, NAND, NOR ...etc.) will be modeled as PLEs. These object types are similar in that the real gates used to implement them differ only in the number of available I/O pins. There is no typical value for m due to the high variations in the available input numbers. The typical value for n is 1. However, in some cases, the corresponding gates of PLEs provided may have two outputs; one provides the complement output value of the other. Therefore, a PLE model with m inputs and 2 outputs will be used for each of the pure objects. For example, an AND gate may have 2, 3 or even 16 inputs. Regardless the input numbers, an AND gate performs the boolean logic-AND to all the inputs for the

output. They will be modeled as a 2-input-2-output, 3-input-2-output and 16-input-2-output PLEs. The input, output and bus objects all have similar properties as the pure objects because they can be viewed as “gates” with multiple input and single output pins. They will also be modeled as PLEs.

The LET modeling is necessary for handling the object which has designated I/O pin usages. Some special object categories such as DFFs and I/O pads are handled in special ways by ANFT. Objects in these categories have common input and output pin patterns and are designed for special requirements. They will be handled using different LETs. For example, the edge triggered DFFs all have a data input, a clock input and a registered output, Q . While in some designs, inputs such as preset, reset, clock enable and inverted output, \bar{Q} , are also available. Moreover, the clock input may be triggered by positive or negative edge. All these inputs and outputs have specific usages. By using a LET, ANFT can identify which pin is presented and which is not. Therefore, objects with properties like edge triggered DFFs must be transformed using LETs so that ANFT can handle the I/O pin variations.

As mentioned, BLACK BOX elements can be function cells, units or modules. Although a BLACK BOX type may have several similar elements, the variations of them are too high to let them be described using a template. They can neither be modeled as PLEs since not all of their I/O pins are for data I/O. In addition, regardless the usages of its I/O pins, they are optional for the designs. They will only be invoked by designers and their I/O connections will be handled by HPCOM.

Therefore, The processes of ANFT do not have to have the capabilities to handle the variations between them. Due to these facts, BLACK BOXes will be modeled as FLEs. Unlike the PLEs, there is no typical value for n . A m -input and n -output FLE will be used for each of them.

3.3 The Libraries

AHPL is a hardware description language. It is not dedicated to implement designs using any specific type of technology. The connection between the design descriptions and the implementations are provided by ANFT. Therefore, the ANFT libraries are important databases regarding the implementations. They must provide required technology information to the ANFT. In most cases, vendor-specific libraries are also designed and used by some specific CAD tools. These CAD tools, whether they use a standard or a self-defined netlist format, have their own conventions in interpreting netlists. In order to generate suitable netlists for individual tool, some tool dependent information must also be included in the ANFT libraries. Due to the same reasons, different ANFT libraries must be created for different technology libraries.

3.3.1 Tool Dependent Style and Information

All the standard and vendor-specific netlist formats use keywords for special meanings. A commonly required instance for all the formats may be referred by using different keywords in different CAD tool environments. For example, the keywords

for “high” and “low” power sources may be VDD/GND, VCC/GND or VDD/VSS. ANFT can not handle this kind of variations by using some default assumptions. They must be reflected in the libraries. The followings are commonly required for all the ANFT libraries:

1. Necessity for connecting power and ground of each element.
2. Default drive capability of a normal gate.
3. Key words for power and ground instances.

In some cases, tool dependent information such as the chip part number, netlist format version and special user comment are also needed for generating usable netlists. They must be submit to ANFT via a default information file. Usually, the information will be included to the outputs in the final stage of the translation processes. Therefore, the format of the default information file must be recognizable by the corresponding output file generator. There will be no standard format for them.

3.3.2 Gate Information for PLEs, FLEs and LETs

Table 3.2 shows the element information that must be presented for each available element of an ANFT library. In general, these data are required for all the PLEs, FLEs and LETs (the actual formats for specifying them will be different.) The gate name, library name and I/O pin ID must be identifiable by the CAD tool which the library is associated with. The input and output pin numbers indicate the I/O

Type of Information	Data type	Example
Gate Name	Character String	X_74LS08
Library Name	Character String	TTL_LIB
Input Pin #	Integer	2
Output Pin #	Integer	1
Gate #	Integer	4
I/O Pin ID(s)	Character Strings	I1, I2, O

Table 3.2: Type of Required Information.

capacities of the gate. ANFT uses a single-package-multiple-gate model to assign serial unit numbers of gates in a netlist. Thus, the gate duplicate number tells the number of duplicate gates in the gate package. The AND gate package shown in the Table 3.2 has four AND gates. In a real design, every four AND gates will be assigned one unit number. If a package has only one gate in it then every gate will receive a unit number.

3.3.3 The LETs for DFFs and I/O Pads

Fitting the DFF and I/O pad objects into real elements presents special problems for ANFT. As are the reasons for treating them as LETs, elements in one category are different yet have similar I/O pin patterns. ANFT will be able to combine the DFF and I/O pad objects with suitable elements from the assigned library only if the presence of each possibly used I/O pin is clearly specified.

For DFFs, ANFT will treat all the memory variables (4009) declared in the AHPL descriptions and the control DFFs (4006) in the same way. By default, the HPCOM

uses a DFF with preset (P), clear (C), clock enable (E) and one output pin (Q) as the memory register model (DFFPCEQ.) Since there are usually four types of DFFs available namely DFF, DFFP, DFFC and DFFPC². The DFFPCEQ model is general enough for handling all the possible variations when combining the DFF objects and the DFF gates. However, in most cases, both the Q and \bar{Q} outputs will be available. Utilizing the \bar{Q} output can reduce some unnecessary inverters. Therefore, ANFT will take the advantage of DFFPCEQ's properties and add an additional \bar{Q} for defining the DFF templates. The LETs for the four DFF types are:

DFF LET	Default Data	I/O Pin Template						
DFFE	As in Table 3.2	Data	Clock		Clock_Enable	Q	Q	
DFFPE	As in Table 3.2	Data	Clock	Preset		Clock_Enable	Q	Q
DFFCE	As in Table 3.2	Data	Clock		Clear	Clock_Enable	Q	Q
DFFPCE	As in Table 3.2	Data	Clock	Preset	Clear	Clock_Enable	Q	Q

All the DFF elements, if provided in the library, must indicate the availability of each I/O pin accordingly.

The I/O pads provide a circuit the connections to the external world. Although they have similar attributes as the DFFs, they are not generated by HPCOM. Because they are not necessarily available in a library or needed for an implementation, their presence are optional. Usually, there are four types of I/O pad available in a library namely blank pad (BLANKPAD) input pad (IPAD), output pad (OPAD) and bi-directional pad (BPAD.) The LETs for them are:

²As a result from the AHPL design paradigm, ANFT will only consider these types of DFFs.

PAD LET	Default Data	I/O Pin Template			
		Input Pin		Output Pin	
BLANKPAD	As in Table 3.2				
IPAD	As in Table 3.2	The_PAD	D_Out	D_Out	
OPAD	As in Table 3.2	D_Out			The_PAD
BPAD	As in Table 3.2	Out_En	D_Out	D_In	D_In The_PAD

Notice that the BLANKPAD has no I/O pin and the field “The_Pad” is for the pad itself. It can be connected to an external I/O port.

3.3.4 I/O pin attributes

The inputs and outputs of an object are corresponding to the I/O pins of a library gate. They are either for data I/O or for controlling the behaviors of the gate. In order for ANFT to manipulate the netlist at gate level, every I/O pin must have a unique identifiable name within the gate and must present enough information of their own attribute(s). ANFT recognizes two important I/O pin attributes: polarity and invertability.

The polarity can only be applied to control input pins. For an enabling control pin, it decides whether the action of control will be activated when the signal is “high” or “low”. For a triggering control pin, it decides whether the triggering signal should be “low-to-high” or “high-to-low”. Its value can either be positive or negative which corresponds to active-high/leading-edge and active-low/trailing-edge respectively.

The invertability decides whether a pin is invertible or has an inverted pin of itself. In the real world, the invertability of a pin can either type but not both and it can be applied to both the data or control pin. A pin is invertible means the signal passing

through it can be complemented directly by setting some control flag. The inversion is accomplished within the associated gate. Therefore, an invertible pin can make the particular I/O signal either complemented or uncomplemented but not at the same time. There will be only one physically available pin. On the other hand, if a pin, say P , has an inverted pin of itself, there will be another physically accessible pin, \bar{P} , for complementing the signal. In this case, both the original and inverted signals can be applicable to or originated from P and \bar{P} at the same time.

The I/O pin attributes assigned to every gate in an ANFT library should faithfully reflect what they resemble in the vendor-specific library. Otherwise, the output netlists will be unacceptable for any application.

3.4 The Bus sub-type and the Buses

In the AHPL implementation convention [14], both the AND-OR and tri-state bus are allowed. The AND-OR bus structure is fine for buses with small number of I/O. In some particular cases, a “huge” OR may needed to adopt numbers of I/O connections. In this case, the tri-state bus structure should be considered. Of course, this is only a general guide. Determining which bus type to use should be based on the type of technology for implementation.

HPCOM will construct all the bus type elements, i.e., BUS, EXBUS, OUTPUT and EXOUTPUT, as AND-OR buses. However, that does not mean that there is no way to incorporate tri-state buses into a design. Within HPCOM [9], a special

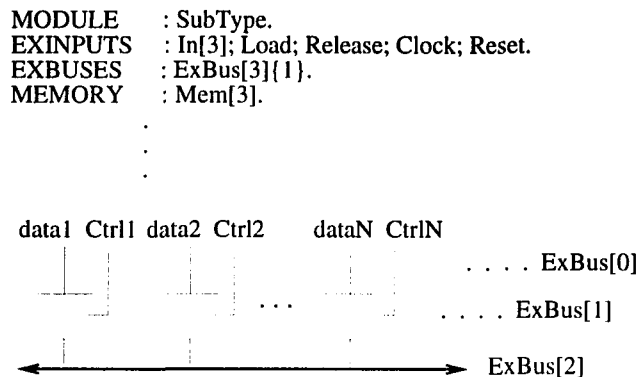


Figure 3.2: The Sub-type Parameter and Its Effect.

bus sub-type parameter is allowed for AHPL descriptions. It determines the type the particular bus element(s) will be constructed. If a bus sub-type, {1}, is attached as shown in Figure 3.2 when declaring a bus variable then it can be implemented as a tri-state bus. Otherwise, by default, the bus will be implemented as an AND-OR bus. The bus sub-type, as its name suggested, can only be applied to the BUS and EXBUS variables. It still can not control the type for constructing OUTPUT and EXOUTPUT elements. If for some particular reasons the latter must be constructed as tri-state buses then ANFT must be able to meet the requirement by providing an user option for it.

3.5 Fanin limit

Fanin limit or gate breaking is an important process for fitting the PLEs into available gates. The fitting process for a PLE is based on the available input pin numbers of its type provided in the library. Once a desired library is assigned, the

process must correspondingly take all the available input pin numbers of a type into considerations.

In AHPL netlists, there is no limit on the input number of pure objects. It is almost impossible for all the objects to find a compatible gate (i.e., gate with same input number) in the assigned library. If a PLE is compatible to a gate then it can be realized using the gate. On the other hand, the incompatible objects must be broken into a set of equivalent gates for the implementation. For example, if the incompatible object type is AND or OR then the gate breaking will be based on the available input pin numbers of themselves. If the type is NAND or NOR, they will be transformed into AND-INV and OR-INV equivalent logics, and then the AND or OR gate(s) will be further fitted. The transformation processes must not depend only on the specific sets of input pin numbers of one library.

3.6 Fanout limit and Drive Capability

For overloaded outputs, ANFT must apply fanout limit process for the reasons discussed in Section 2.2. ANFT will use the default drive capability specified in the libraries as the drive capabilities of all gates. In some large designs, the default drive capability will be inadequate for the fanouts of some gates. Leaving them unchanged will result in unusable implementations.

ANFT has two ways to resolve the fanout problems. The first one is using the replicate of a gate with higher drive capability. If a gate has one or more replications

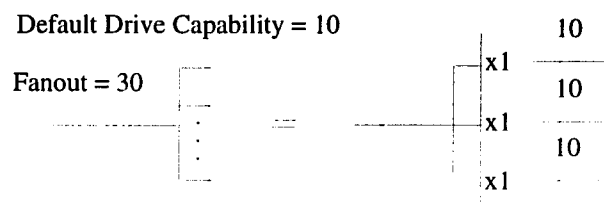


Figure 3.3: Tree Buffering for High Fanout Gate.

with different drive capabilities then a suitable one with sufficient fanout will be used as a substitute. The ANFT libraries must contain the replication and drive capability information of all applicable gates. The second method is using tree buffering. If the highest drive capability available is still less than the fanout, buffers must be inserted for sharing the load. As shown in the Figure 3.3, the load will be divide and deployed to the newly inserted buffers. Although using the tree buffering will increase the gate count, it is necessary for generating a usable circuit. ANFT will try to find a suitable replicate with enough drive capability first. If it can not be found then the tree buffering will be used.

3.7 The BLACK BOX Elements and Special Gate Mapping

Special designed functional elements can be used either as BLACK BOXes or special gates. Using them can reduce both the gate count and delay of a design. In order to take advantages of some special designed elements in the libraries, two special mechanisms namely BLACK BOX compilation and special gate mapping will be used. If a functional element is declared but without definition in the original AHPL description then it will be treated as a BLACK BOX (4024). A BLACK BOX

can be as simple as an AOI21 gate or as complex as a 16-to-1 multiplexer. Its naming and I/O connections must be done by the designer while the wiring of its I/O pins will be handled by HPCOM. As long as the desired functional elements are available, the designer can invoke them as BLACK BOXES in his design directly.

HPCOM only uses the primitives shown in Table 3.2 for compilation. Therefore, most of the objects in an AHPL netlist are pure objects. No matter how many BLACK BOXES are used in a design, there are still chances for combining sets of gates into some special gates. This can be done by applying the special gate mapping in the ANFT. Unlike the BLACK BOXes, the special gates are only used for replacing 2-level logic. The restriction is due to the fact that most of the special elements provided are 2-level logic and complex elements can be implemented using BLACK BOXes. Thus, ANFT will only accept 2-level logic tree assignments in the libraries and search for possible matchings.

3.8 Inverted and Invertible Inputs and Outputs

In most of the vendor-specific libraries, the I/O pins of certain types of gates are prone to incorporate invertability. HPCOM assumes no invertability for both the input and output pins of any object. Therefore, if there are invertible or inverted I/O pins available in the assigned library, it is possible to remove some unnecessary inverters by utilizing them. As the example shown in Figure 3.4, an inverted (or invertable) I/O pin is useless unless it connects to an inverter. If an output pin is

invertible and connected to an inverter then the inversion of it can remove the inverter. In addition, if all the fanout from an inverter are connected to input pins which are all with invertability then the inverter can be removed, too.

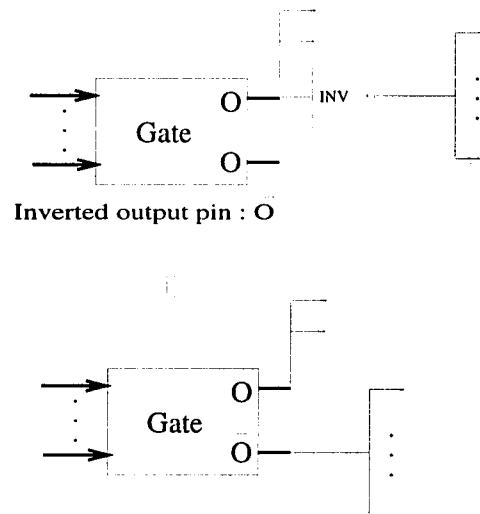


Figure 3.4: Example of Utilizing the Inverted Output Pin.

3.9 The Output files and Their Formats

Generating netlists in various netlist formats is an important goal of designing the ANFT. In general, the standard or vendor-specific netlist formats are very different from each other. Even the same standard format with different versions may seem totally different. Due to the high variations, the ANFT uses different output file generators to generate different output netlists. Currently, there are four file generators available. The first one, reference file generator, generates human readable connection

lists of designs. The other three are for generating EDIF 1 1 0, EDIF 2 0 0 and Xilinx's XNF netlist respectively.

The functions of ANFT has covered what both of the AENT91 and AENT_TTL can achieve. An output in one of these formats based on a proper library can be used by any CAD tool who accepts netlists in that format. Although three generators and formats may seem not quite enough, the output functions can be expanded for new netlist formats. New output file generators can be added into the last stage of the synthesis processes for new tasks. Once the expansion has been done, additional outputs can be available directly form the AHPL sources.

3.10 The ANFT Organization

This section summarizes the design considerations and presents the organization of ANFT in Figure 3.5. The users can assign files and options through an integrated man-machine interface. The options available are:

1. Output file format.
2. Data DFF type.
3. Implementation style: AND-OR or NAND-NAND implementation.
4. Special gate mapping.
5. VDD/GND block splitting³.

³The option is for NettranTM version 2.25 only.

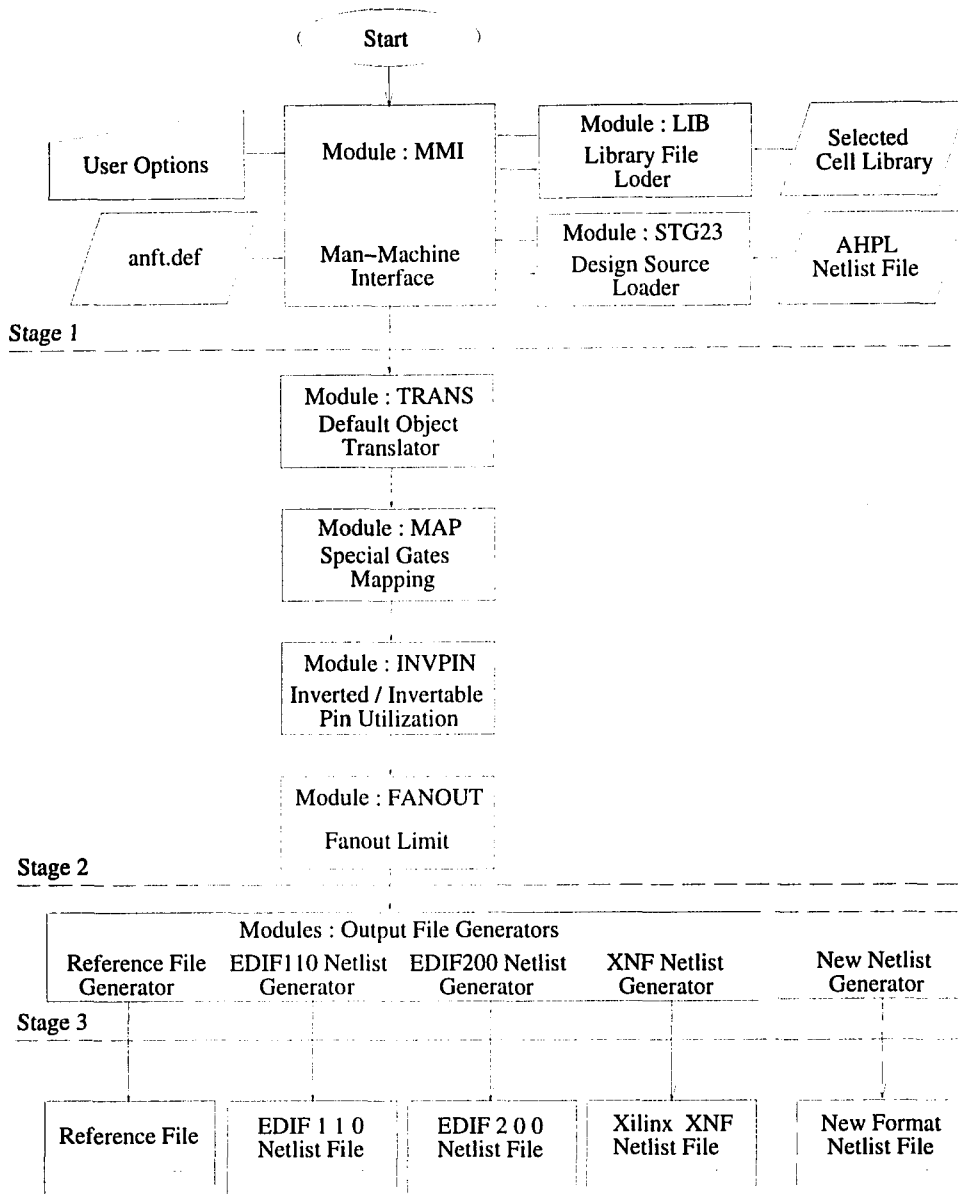


Figure 3.5: The Organization of ANFT.

6. Bus type.
7. XOR flattening option.
8. I/O pad generation.

The libraries and AHPL netlists are handled separately by different process modules since several netlists may use the same library for implementation. The AHPL netlist will be read by module STG23 where an abstract netlist for the design will be restored. Fanin limit and gate allocation for PLE and FLE are then performed in module TRANS. The module MAP maps the available special gates into the netlist. Then, if any gate has invertible or inverted pins then the module INVPIN will try to utilize them. After applying the fanout limit to the netlist in module FANOUT, the design will be ready for the output processes. The readable reference file will be generated automatically. The output netlist format is decided by the answer to the output file format option. As mentioned, additional output file generator can be added into the synthesis processes. As long as the output generators are available, the ANFT should be able to generate netlist files in the desired format using different libraries in full scale.

CHAPTER 4

Implementation

In this chapter, the implementation of all the design considerations will be presented. The discussions will follow the flow of modules listed in Figure 3.5. ANFT uses the AHPL netlist generated by HPCOM as the input file. We will not review its format here but only cite some portions when necessary. For detailed information see also [9] [12].

4.1 The Element Types

All the object types that are used by the stage23 of HPCOM are shown in Table 3.1. In AENT91, the integer object codes are converted into characters for saving memory. They are shown in Table 4.1. Notice that the type BLACKBOX (4024) is not included; hence, AENT91 can not handle designs that contain BLACK BOXes. ANFT will make a little adjustment of the original types and add additional types for handling new objects as shown in Table 4.2. For the D type flip-flop objects, the original types CDFE (4006) and DDFE (4009) will represent the DFFPC and DFF. Two new types DFFPE (DFF with preset and clock enable) and DFFCE (DFF with clear and clock enable) are added for handling new elements when substitutions are needed. For the

Object and Function	Symbol	Object and Function	Symbol
4001 (AND)	G	4018 (EXINPUT)	D
4002 (NAND)	H	4019 (EXOUTPUT)	E
4003 (OR)	I	4020 (EXBUS)	F
4004 (XOR)	J	4029 (CND for Control unit)	N
4005 (NOR)	K	4030 (OR for No-Delay step)	O
4006 (DFF for control)	X	4031 (AND for CLU)	P
4009 (DFF for data)	M	4032 (NAND for CLU)	Q
4012 (INPUT)	A	4033 (OR(for CLU)	R
4013 (OUTPUT)	B	4034 (XOR(for CLU)	S
4014 (BUS)	C	4035 (NOR(for CLU)	T
N/A IPAD(input pad)	U	N/A BPAD(bi-directional pad)	S
N/A OPAD(output pad)	O		

Table 4.1: The Element Type Definition of AENT91.

bus type elements, if the tri-state implementation is selected the BUS, EXBUS, OUTPUT and EXOUTPUT will be converted to TRIBUS, EXTRIBUS, TRIOUT and EXTRIOUT respectively. Extra types such as INV (inverter), BUF (non-inverting buffer), TRIBUF (tri-state buffer) and DON'T CARE ("*") are also included. The DON'T CARE, which will appear in the gate mapping process, will be discussed in Section 4.3 and 4.5.

4.2 The Data Structures

ANFT must restore the AHPL netlist in a structure that reflect the design details and is ready for manipulations. An AHPL netlist can be represented by using a directed graph. Figure 4.1 shows the structure of a typical AHPL netlist. Each node is representing an object of the netlist and the I/O connections among them are represented by the directed edges. As shown in Figure 4.1, the element data

Object Code	Function	Symbol
4001 4031	AND	A
4002 4032	NAND	B
4003 4030 4033	OR	C
4004 4034	XOR	D
4005 4035	NOR	E
4006	Control DFF / DFFPC	F
4009	Data DFF / DFF	G
N/A	DFFP	H
N/A	DFFC	X
4012	INPUT	I
4013	OUTPUT	J
4014	BUS	K
4018	EXINPUT	L
4019	EXOUTPUT	M
4020	EXBUS	N
N/A	INPUT PAD	O
N/A	OUTPUT PAD	P
N/A	BI-DIRECTIONAL PAD	Q
N/A	BLANK PAD	a
4029	CND (Control AND)	R
4024	BLACK BOX	S
N/A	TRIBUS	T
N/A	EXTRIBUS	b
N/A	TRIOUT	Y
N/A	EXTRIOUT	Z
N/A	TRI-STATE BUFFER	U
N/A	INV	V
N/A	NON-INVERTING BUFFER	W
N/A	DON'T CARE	*

Table 4.2: The Element Type Definition of ANFT.

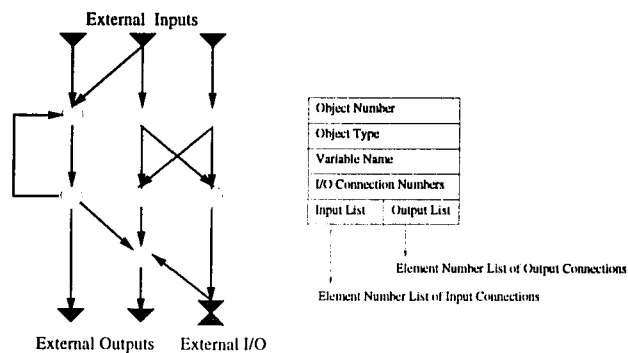


Figure 4.1: Typical Realization of An AHPL Netlist.

structure used by AHPL netlist contains only basic information. The ID of every object is a unique element number, EN, assigned by HPCOM. All the I/O connection information will be stored in the I/O link lists. For ANFT, the data structure used by AHPL netlist is too primitive for the intended manipulations. Therefore, based on the element model discussed in Section 3.2, a new element data structure as shown in Figure 4.2 will be used instead. The **EN** is the element number inherited directly from the AHPL netlist. **EType** and **Entry** are for element type. **EType** is used to identify which type (e.g., PLE, FLE or LET) the element is. The **Entry** as well as the **ENa** and **lib** are used to locate the gate information for implementing the element. **NI** holds the number of input elements or pins. There will be equivalent number of entries in the input link list **I**. Similarly, **NOP** is the number of output pins. Each output pin has its own output link list, **O[i]**, that identifies the fanout connections. All the I/O link lists are one way lists. Every entry in a list is a data structure that holds connection and pin usage information of the associated I/O pin. Both the **tag** and **modu** are for generating new serial numbers as well as locating the I/O pin ID

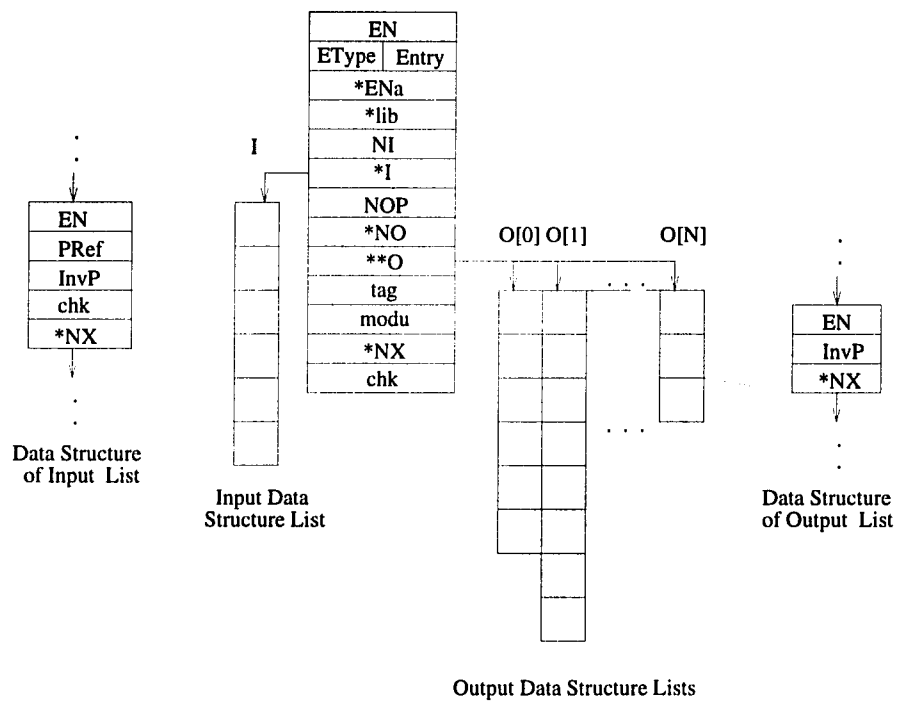


Figure 4.2: Data Structure of ANFT Internal Element.

strings. The **chk** is an internal flag used by some processes and **NX** is a pointer for connecting similar type elements. An element may have more than one output link list but only one input link list. The multiple output lists are necessary because we may have to locate all the fanout elements in an efficient way. Therefore all the entries in an output list are associated with only one output pin. On the other hand, each entry of an input list is associated to one input pin. I/O information for each pin is stored in the corresponding data structure of the I/O link list. The common fields, **EN**, **InvP** and **NX** are for holding the element number where the I/O pin is connected to, the pin usage and the location of the next entry. The **PRef** of an input element structure is for identifying which output pin of the element EN (the number kept in the same structure) the current input pin is connected to.

The graph will be reconstructed using an one dimensional element structure array as shown in Figure 4.3. The contents of the array are pointers to the new element data structures which represent the nodes. Edges are implemented by the new I/O structure lists. The size of the array is decided by the `Total_Object_Count`, integer parameter next to the label `GATELIST` in an AHPL netlist:

```
GATE LIST:   Total_Object_Count
```

The number is usually more than required. Nevertheless, we will increase it by the increment number provided in the ANFT default file since elements may be created during the conversion. In order to locate an element data structure efficiently, the element numbers will be used as the array indices for themselves. They usually are

Element pointer array (Total_Object_Count = 95)

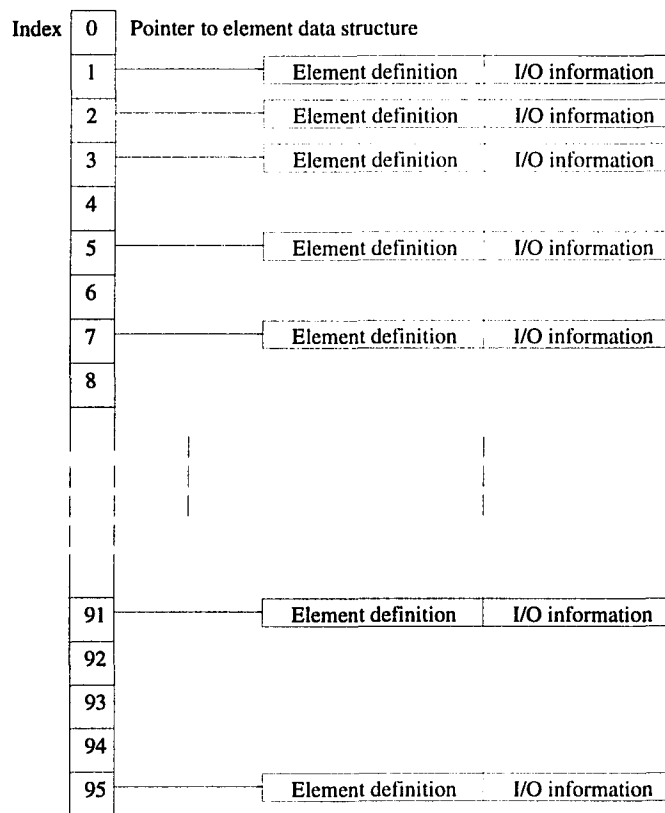


Figure 4.3: The Dynamic Element Array.

not consecutive. Therefore, not all the entries in the array will be used. Unused entries such as 4 and 6 can be assigned to newly created elements later.

4.3 Library Format

In this section, we will first discuss how to present the tool dependent specifications in an ANFT library. The discussions of assigning vendor-specific gate information and user defined element information will be given later.

An ANFT library can be viewed as a specification database keeping all the detail information associated with the gates to be used. Figure 4.4 shows the format of an ANFT library. The general and tool dependent information discussed in section 3.3.1 will be presented in the beginning of an ANFT library:

P1 P2 P3 L1 L2

P1 : Flag for individual power connection.

P2 : Default drive strength.

P3 : Number of libraries referred.

L1 : Power “high” label.

L2 : Power “low” label.

These parameters will be used in the ways as suggested above. Notice that the P3 is used as a count of different library IDs that appear in the ANFT library.

The following five sections: DEFELEM, DFFTYPE, IOPAD, MAPPING and DRIVECAP are for defining gates used by different process module. The first parameter of each of them as well as the nine subsections of DEFELEM indicates the number of entries presented in that section or subsection. The number in the first

P1	P2	P3	L1	L2
DEFELEM :	9			
AND :	N1		Input Number List	
OR :	N2		Input Number List	
NAND :	N3		Input Number List	
NOR :	N4		Input Number List	
INV :	N5		Input Number List	
BUF :	N6		Input Number List	
XOR :	N7		Input Number List	
TRIBUF :	N8		Input Number List	
BLKBOX :	N9			Element definitions
DFFTYPE :	4			
DFFPCE				
DFFE				
DFFPE				
DFFCE				
IOPAD :	4			
BLNKPAD				
IPAD				
OPAD				
BPAD				
MAPPING :	N10			
DRIVECAP :	N11			

Figure 4.4: The Library Format.

Symbol	Meaning	Attribute Type
+	1. Active-high 2. Positive Edge Trigger	Polarity
-	1. Active-low 2. Negative Edge Trigger	Polarity
^	Inverted Pin Available	Invertability
~	Invertible	Invertability
	ID strings separator	Invertability

Table 4.3: I/O Pin Attribute Symbols.

three sections are fixed since ANFT only accepts nine logic primitives and four types of DFFs and I/O pads. Elements defined in different sections may be specified in different ways. However, commonly required information provided must conform to the following format:

Name ID	Library ID	I/O Pin #	Gate Number	I/O pin IDs	Power pin IDs
---------	------------	-----------	-------------	-------------	---------------

The gate name, associated library name, I/O pin numbers , gate number and I/O pin IDs must be clearly defined according to the vendor-specific library. I/O pin attributes can be assigned alone with the IDs. Table 4.3 is a list of symbols for specifying the polarity and invertability. When both the polarity and invertability are necessary for a pin, the polarity should be assigned first. ANFT will use but not check the assigned information. Any erroneous assignment will result in unusable implementation.

The polarity of all control input pins by default is positive. Therefore, the ID “Clk” shown below is equivalent to the “+Clk”:

```
ETDFF 3 1 1 D Clk -CE Q
```

while the “-CE” means the clock enable pin is “active-low”. The “+” and “-” if assigned to a data input ID will be ignored without any warning.

Assigning the invertability is more complicated. Ideally, every I/O pin can be invertible or has an inverted pin of itself. If the invertability is invertible than an i -input gate will contain exactly i input pins. The symbol “~” must be added to the IDs which are invertible as shown below:

```
OR3    3 1 1  ~1 2 ~3  ~0
```

If the input pin attributes indicate that some or all the input pins have inverted pins we say that the gate has i set of input pins. Only one out of each set can be in use since we are utilizing but not re-deriving the logic using the inverted pins. As discussed before, the typical I/O values of a PLE are m and 2 where $m > 0$. When fitting m using the $2i$ available pins, only the number of set (i.e., i) should be counted. Thus, the input pin number of the gate should be the number of set of pins if inverted pin(s) are available. As shown in the example below the AND3 has 3 sets of input pins and totally 5 input pins:

```
AND3    3 2 1  ^1|4 2 ^3|6  7  8  PW_HIGH PW_LOW
```

The “^” indicates that the pin 4 is an inversion of pin 1 and they are separated by the vertical bar “|”. The example also shows that the AND3 has an inverted output pin 8 in addition to 7, the normal one. Unlike the inverted input pins, they are specified as two separate pins because it is possible to use them simultaneously. If for some reasons the inverted I/O pins of the AND3 are not desired we can simply define it as:

```
AND3      3 1 1      1 2 3      7      PW_HIGH PW_LOW
```

ANFT will only use the normal I/O pins. The power pin IDs (PW_HIGH and PW_LOW) if presented are used for connecting the “high” and “low” power sources to the gate. They are optional since not all the design styles require power connections to all the elements individually. The I/O pin attributes can not be applied to them. If the connections are necessary each gate must have two extra input pins as shown above and the P1 must be set to “1”.

The first section, DEFELEM, contains gates for implementing the PLEs and FLEs. It contains nine subsections for defining AND, OR NAND, NOR, INV, BUF, XOR, TRIBUF and BLKBOX type elements. All the pure gates and BLACK BOX units that are to be used must be defined here. ANFT insists that the first six of them must contain at least one gate and each of the first four of them must contain a 2-input gate. The first set of parameters of each of them must be the entry number and the input number list. The input number list tells which input pin number is available for the associated type of gate. The numbers in a list must be given from high to low.

Subsection DFFTYPE and IOPAD will provide the elements for implementing LET objects. As mentioned, ANFT will only make use of four types of DFFs. Therefore, one can provide the required templates similar to the example shown below:

```
DFFTYPE:      4
DFFE          ETDFD      lib_1  3 2 4  D C !      Q !
DFFPCE        ETDFPCE   lib_1  5 2 1  D C S R CE  Q QN
DFFPE         ETDFP     lib_1  4 2 2  D C R !      Q !
DFFCE         !
```

For the DFFE, its name is ETDFF and it is contained in a vendor-specific library called lib_1. It has three input pins: D, C and ! and two output pins: Q and !. The following number, 4, means an ETDFF package consists of 4 of them. There are no power source input pins. The two “!” means the pins they represent, clock enable and inverted output \bar{Q} , are not available. Otherwise, they should be the actual ID strings like those provided in DFFPCE. ANFT assumes all the DFFs must incorporate the uncomplemented output pin Q. There will be no safety check for using it. If any of the four types is not available (e.g., the DFFPE) there should be a “!” for indicating the absence.

Similarly, section IOPAD contains four types of I/O pads: BLANKPAD, IPAD, OPAD and BPAD. The example given below shows how to specify the available I/O pad templates:

```

IOPAD:      4
  BLNKPAD   BLANKPAD  lib_2  0 0 1
  IPAD      InPAD     lib_2  1 2 1   PAD      DI !
  OPAD      OutPAD    lib_2  1 1 1  ^DO|DO_BAR PAD
  BPAD      BiPAD     lib_2  2 3 1   OE ~DO   DI DI_BAR PAD

```

The BLANKPAD which has no I/O pins can be used in a pad frame. For the IPAD, the second output pin is not available. For the OPAD, in addition to the normal I/O pins, is has an inverted input pin DO_BAR. The BPAD has two input (OE and \sim DO) and three output (DI, DI_BAR and PAD) pins where DO is invertable. The DI (data in) is for conveying data that go into the circuit while the DO (data out) is for connecting signals to the external world. The OE (output enable) is used to control

the data flow direction. Since it is an output enable pin, ANFT assumes the outgoing data has the priority when OE is activated. Otherwise, the incoming data will gain the access to the path. The PAD represents the pad itself. Generally, it will be used as the physical I/O connection port to the circuit.

The information defined in section MAPPING are used by the special gate mapping process. The entry format will be a logic tree expression followed by either the corresponding special gate definition or an index to a previously defined gate as shown in the following example:

```

MAPPING:    10
            :
            2+(2&,2&) AOI22  4 1 1  A1 A2 B1 B2  0
            2&(2&,* ) = AND  2 3 1
            :

```

A logic tree can only represent a 2-level logic. The tree expression would be an i-input gate, the root, followed by i gates as the inputs of root. The expression “2+(2&,2&)” means a 2-input OR with two 2-input AND connect to it where “+” and “&” are tokens for OR and AND gate respectively. The recognizable tokens of the mapping process are shown in Table 4.4. Notice that the DON’T CARE means the type of the corresponding gate is not important. Since the “space” is not a recognizable token. Thus, an expression like “2 + (2 & , 2 &)” is not acceptable. In the entry “2&(2&,*)”, the logic tree is equivalent to a previously defined 3-input AND gate. It is not necessary to redefine the same gate again. Thus, the “AND”, “2”, “3” and “1” represent the type, entry number of the gate, input pin number and output pin

Gate	Token
AND	&
OR	+
NAND	~
NOR	#
INV	I
BUF	B
XOR	@
DON'T CARE	*
left-parenthesis	(
right-parenthesis)
comma	,

Table 4.4: The Tokens of The Logic Tree Expression.

number of the 3-input AND previously defined in the DEFELEM section. For more examples of assigning logic trees see also Appendix A.

The last section, DRIVECAP, is for specifying gates with higher drive capability. Since same gates with different drive strength differ only in their names, it is sufficient to provide only the gate names and strength parameters without defining them. A drive strength parameter is the times of the default drive capability. Therefore, an AND with strength 2 can drive twice as many fanout as an AND with strength 1. The following example shows how to specify the drive strength information of the gate AND2x1:

```

DRIVECAP: 15
          :
          AND2x1 3 4 2 1 AND2x4 AND2x2 AND2x1
          :

```

The statement tells the gate AND2x1 has three different drive strength: 4, 2 and 1. Since the first number is three, there must be equal number of strength parameters and name strings. The strength parameters must be given from high to low; hence, the corresponding names will also be given in the same order. Notice that the default drive strength and gate name must also be included in the list.

4.4 Default Object Transformation

After an AHPL netlist has been loaded and reconstructed, the first process will be applied to it is the default object transformation. Its objective is to convert and replace the original object specifications with vendor-specific element information. The process flow is shown in Figure 4.5. All the elements will be fitted into suitable gates after the process, and the netlist will be ready for the special gate mapping.

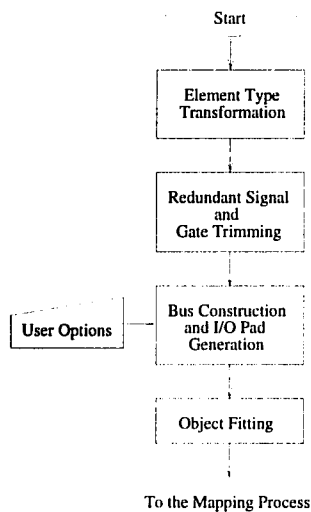


Figure 4.5: Default Object Transformation Flow.

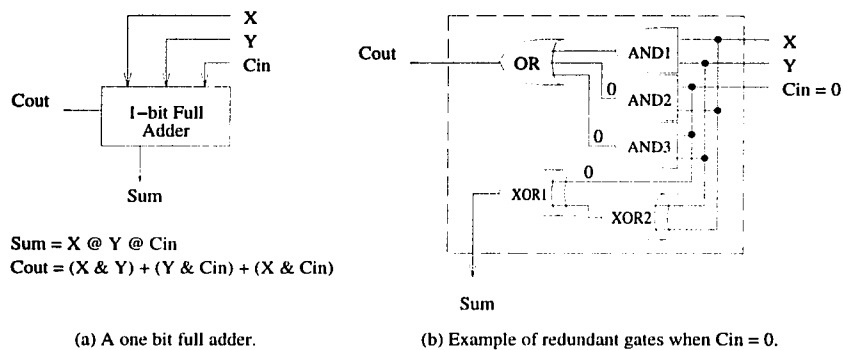


Figure 4.6: Source of Redundant Gates.

4.4.1 Redundant Signal and Object Trimming

HPCOM sometimes generates isolated objects (i.e., objects have no connections to and from others) when there exist declared but unused variables in the source AHPL description. They are redundant to the design and will be removed. Another situation that will generate redundant gates is when connecting constant logic value, 1 or 0, to variable or CLU inputs. For example, we may use an 1-bit full adder with “0” carry as shown in Figure 4.6 (a.) HPCOM will generate a flattened circuit as shown in Figure 4.6 (b.) Since 0 is the control value of an AND operation, the output of AND2 and AND3 will always be 0 which make the OR unnecessary. Similarly, the XOR1 is not necessary since its output is equivalent to (X@Y), the output value of XOR2. Although they may seem redundant at first glance, ANFT will not remove them simply because they are parts of the full adder and the 0 input is assigned by the designer.

HPCOM sometimes will also connect redundant ground signals to OUTPUT and EXOUTPUT elements. They will be removed by ANFT. Finally, for some improper object usages, ANFT will analyze and make necessary changes as shown below:

- An 1-input NAND will be converted to an inverter.
- An 1-input NOR will be converted to an inverter.
- An 1-input AND will be removed.
- An 1-input OR will be removed.

These conversions must be done prior to the **gate fitting process**.

4.4.2 Bus Construction

A BUS (4014), EXBUS (4020), OUTPUT (4013) and EXOUTPUT (4014) can be implemented either as an AND-OR bus or a tri-state bus. Figure 4.7 shows how ANFT decides which type of structure is to be used for them. Since designers can assign the type of bus variables, for tri-state buses, ANFT must make sure that the tri-state buffer is available. Otherwise, regardless the user specifications, only AND-OR buses will be built.

For a BUS, both of its I/O connections must at least be one. If either the all-tri-state-bus implementation is desired or the BUS has a subtype “{1}” then it will become a TRIBUS. An example of constructing a 3-input-2-output BUS as both an AND-OR and a tri-state bus is given in Figure 4.8. The data connections to the

For Every Bus Type Object :

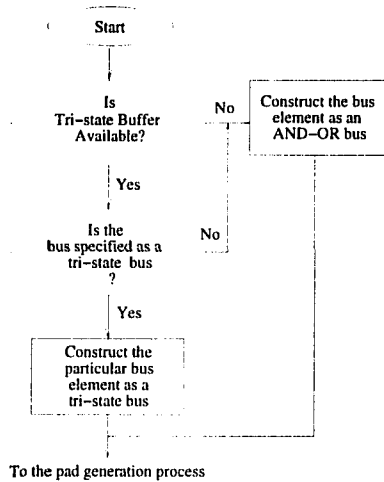


Figure 4.7: Bus Construction Flow.

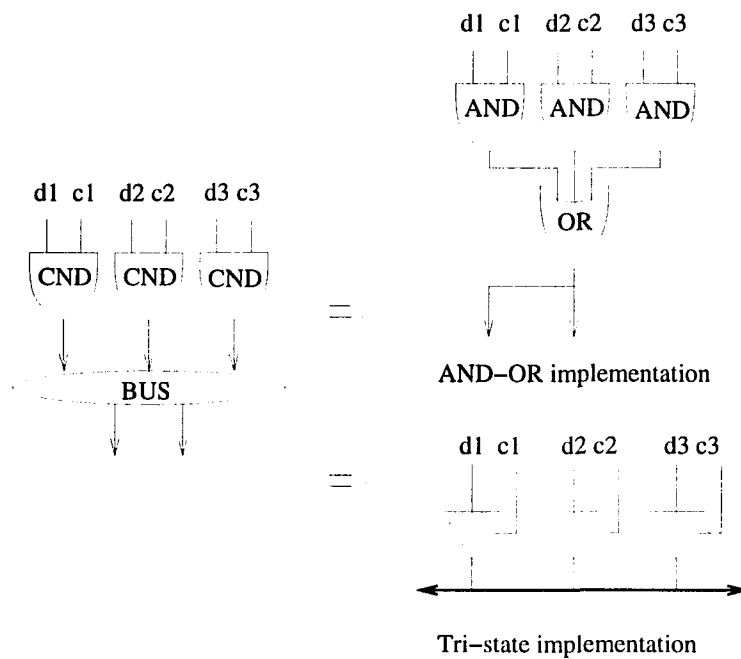


Figure 4.8: Example of Constructing a BUS (4014) object.

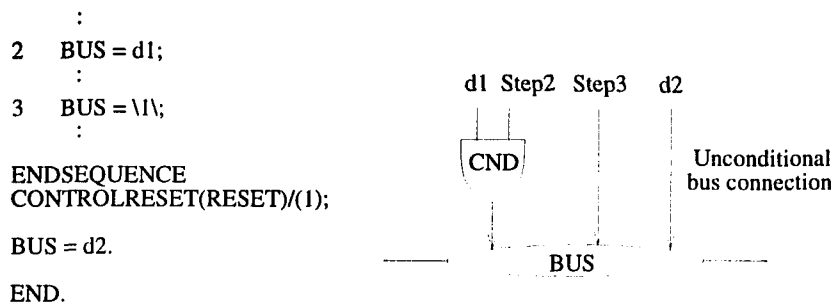


Figure 4.9: Exmample of Erroneous BUS Element Connections.

BUS object is conditional, i.e., the corresponding connection statement is stated within the SEQUENCE section of the source file. The CNDs (4029) are used to control the transfers accordingly. If the data connection is specified after the SEQUENCE section it is an unconditional connection. Data will be transferred to the bus all the time.

Using a bus type element does have restrictions. Figure 4.9 shows an example of erroneous BUS element (declared as BUSES) usage. Because of possible data conflict, once an input signal is conditional, there can not be any unconditional connections to it at the same time. Therefore, if any input element of a BUS is a CND all the others must be CNDs or control signals. It is the designer's responsibility to prevent any data conflict. ANFT will stop the transformation process and give error message if such kind of error is encountered. Notice that the control signals (e.g., c1, c2 and c3 shown in Figure 4.8) of bus connections are always connected to the second input pin of the CNDs. For tri-state implementation, these control signal must be connected to the OE pins of the tri-state buffers. Consequently, when specifying the input pin IDs of a tri-state buffer, the OE pin ID must be the second one. If the TRIBUS connections

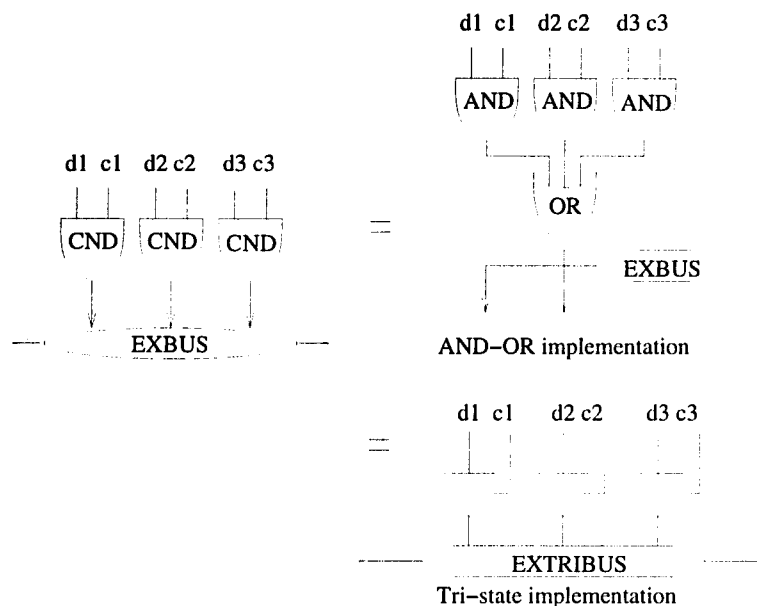


Figure 4.10: Example of Constructing an EXBUS (4020) object.

are unconditional ANFT will connect suitable control signals (i.e., “high” or “low”) to the OE pins; depending on the polarities of them.

Unlike the BUS object, the number of input or of output connection to an EXBUS object can be zero but not both. However, an EXBUS is supposed to be used bi-directionally. Whenever a 0-input or 0-output EXBUS object is found, ANFT will give warning messages and let the designers to decide whether it is necessary to modify the type of it. The EXBUSes will be handled in the same way as BUSes except that there will be additional tags appended as shown in the example of Figure 4.10.

For AND-OR implementation, an EXBUS object will be treated as an OR object. The extra EXBUS tag is needed for holding the original EXBUS label. For tri-state implementation, the EXBUS will be converted into an EXTRIBUS. No extra element

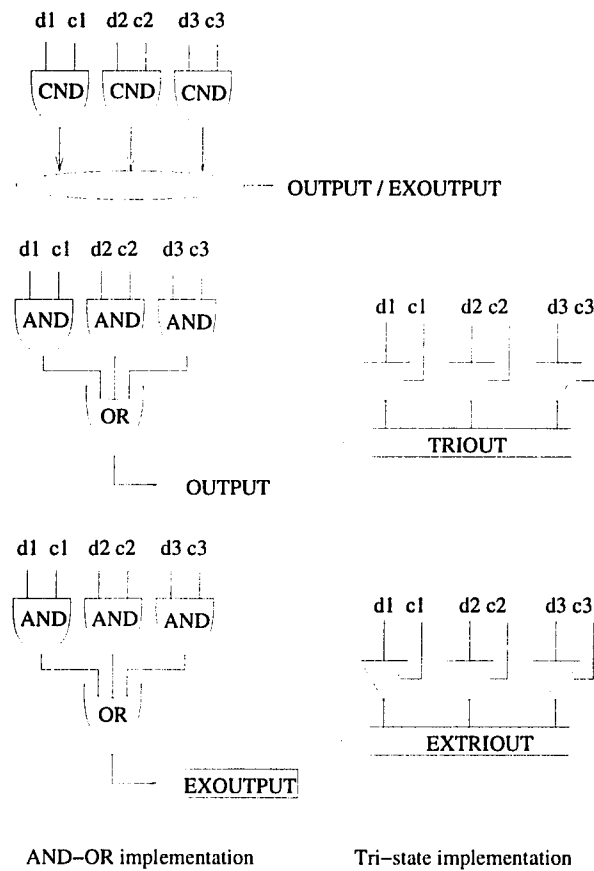


Figure 4.11: Example of Constructing EX- and OUTPUT (4019/4013) objects.

will be created since the EXTRIBUS represents both the bus line and the tag at the same time. Restrictions for using EXBUSes and EXTRIBUSes are the same as using BUSes. In fact, all the bus type elements must be carefully used to avoid any data conflict.

OUTPUT and BUS are equivalent except that OUTPUT has no output connection to any other element in the same module. Figure 4.11 shows how an OUTPUT element will be constructed in different ways. An OUTPUT is invisible to the

external world. In a single module design, it is equivalent and will be converted to an EXOUTPUT. Otherwise, there will be no trace of it externally. An EXOUTPUT is similar to an OUTPUT. The only difference between them is that the former is used as an external output port while the later is for sending signal to other modules within the system. Therefore, like an EXBUS, an extra tag is necessary when implemented as an AND-OR bus.

4.4.3 I/O pad Construction

In some CAD systems, pad elements can be included in the netlist file which would be used in place and route process later. Since I/O pads are not generated by HPCOM, ANFT will be responsible for adding pads to target netlists. If I/O pads are needed the designer can specify either the minimum or the desired number of pads to be included. Every input, output and bi-directional port will be connected to an IPAD, OPAD and BPAD respectively.

An IPAD will be connected to an EXINPUT as shown in Figure 4.12. The DI is the output pin and the Pad itself is the input port. If the \overline{DI} is also available the output number of the IPAD definition should be 2:

```
IPAD InPAD 1 2 1 Pad DI  $\overline{DI}$ 
```

otherwise, it should be specified as:

```
IPAD InPAD 1 1 1 Pad DI
```

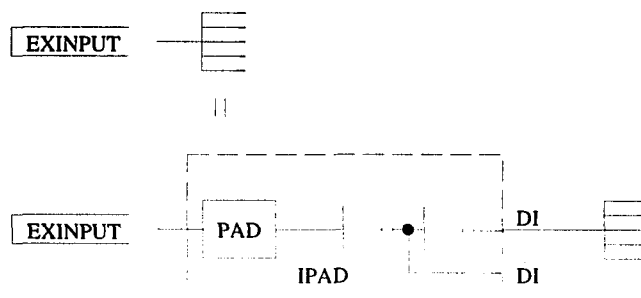


Figure 4.12: Example of Using the IPAD.

Connecting an OPAD to an EXOUTPUT is similar. If the EXOUTPUT is to be implemented as an AND-OR bus then the output of the bus-OR will be connected to the DO of an OPAD. For EXTRIOUT, the bus line will be connected directly to the DO as shown in Figure 4.13.

In AENT91, BPAD can be invoked only when there is only one input connection to the EXBUS. For multiple input connections, it is suggested that the EXBUS should be changed to equivalent EXINPUT and EXOUTPUT. The reason being that one output enable pin is insufficient for connecting multiple control signals. However, if study carefully, the restriction is indeed not necessary. The argument is that we can OR all the control signals together and connect it to the OE as shown in Figure 4.14. When the connections are conditional, control signals will be the second input signal to all the CNDs. If the connections are unconditional we can simply put a suitable power source on the OE. If this is the case then the EXBUS either has no output connection or has no input connection. For the first case, a “high” will be used if the OE is active-low and a “low” will be used if the OE is active-high. For the second

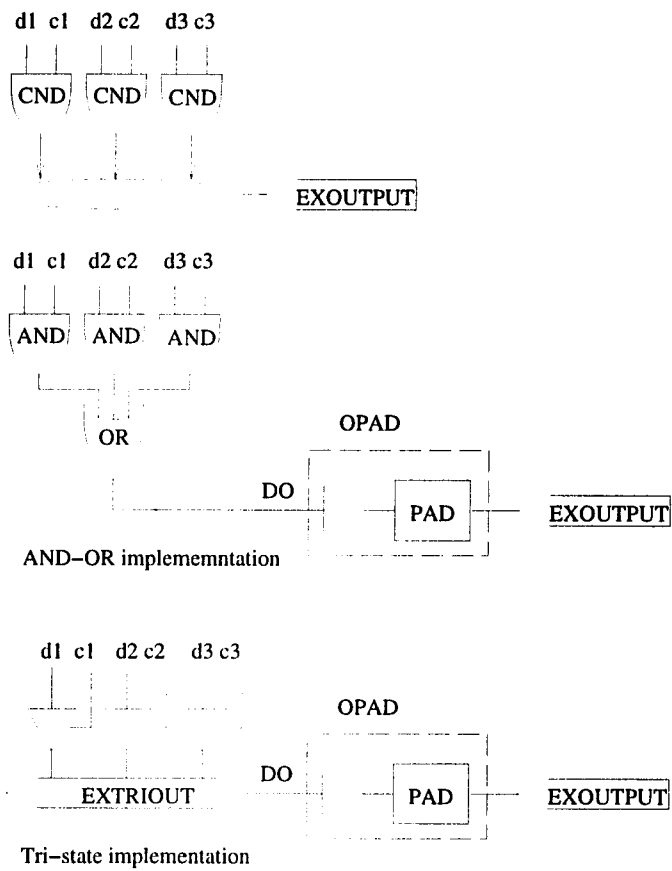


Figure 4.13: Example of Using the OPAD.

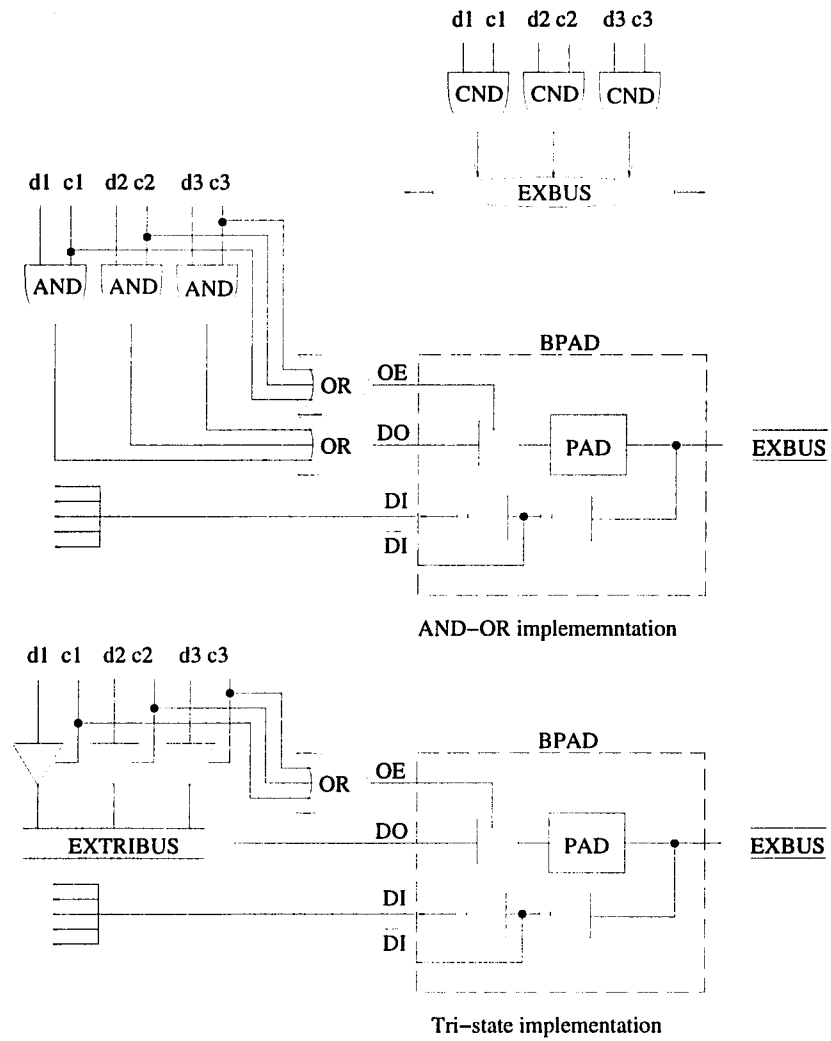


Figure 4.14: Example of Using the BPAD.

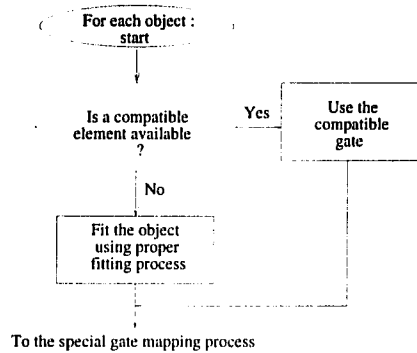


Figure 4.15: Gate Fitting Process Flow.

case, the situation will be reversed. The only concern when allowing multiple data connections is again the possible data conflict. Designers must not invoke more than one data connection onto a bus type element at the same time.

4.4.4 Gate Fitting Using PLE Models

Every object used in a design must be realized using available gates that are specified in the assigned library. For pure objects, the most important parameter is the input pin number. Their values depend totally on the original AHPL description. HPCOM will take whatever numbers are necessary for a design and generate objects with as many input pins. As a result, objects with incompatible input numbers is inevitable. If an object is compatible to the library it will be implemented directly using the matched gate. Otherwise, it must be broken into equivalent logic consisting of compatible objects. Figure 4.15 shows the process flow for gate fitting. Fortunately, the major problems lie only in four types of objects: AND, OR, NAND and NOR. ANFT assumes that sometimes the XOR might not be provided because

AND, OR, NAND, NOR and INV are sufficient for a functionally complete subset of connectives. Should this be the case, the XOR will also be flattened into an equivalent AND-OR-INV logic.

The processes of breaking and fitting incompatible objects is based on the following algorithm:

```

FitObject(Input Number: n, Object Type: T)
{ Find the highest available input number i where i <= n;
  if(i = n)
    { Use the i-input type T gate to implement the object;
      return;
    }
  else
    { Let q = n / i;
      Let r = n % i;
      Create q type T objects according to equation 4.1 to 4.5;
      Use q i-input type T gates to implement the q new objects;
      Let next_level_input_No = q + r;
      FitObject(next_level_input_No, T);
    }
}

```

and the Boolean equations:

$$\begin{aligned} (A \cdot B \cdot C) &= ((A \cdot B) \cdot C) \\ &= (A \cdot (B \cdot C)) \end{aligned} \quad (4.1)$$

$$\begin{aligned} (A + B + C) &= ((A + B) + C) \\ &= (A + (B + C)) \end{aligned} \quad (4.2)$$

$$(A \uparrow B \uparrow C) = \overline{(A \cdot B \cdot C)} \quad (4.3)$$

$$(A \downarrow B \downarrow C) = \overline{(A + B + C)} \quad (4.4)$$

$$\begin{aligned} (A \oplus B) &= ((\bar{A} \cdot B) + (A \cdot \bar{B})) \\ &= (((A \uparrow B) \uparrow A) \uparrow ((A \uparrow B) \uparrow B)) \end{aligned} \quad (4.5)$$

The algorithm is based on the criteria and equation 4.1 through 4.5. Recall that each subsection in the DEFELEM starts with a line telling how many and what input

numbers are available for the particular gate type. These numbers are the reference numbers for deciding the value of i . Based on equation 4.1 and 4.2, an incompatible AND or OR gate will be handled directly by the process FitObject without any type conversion. For NAND and NOR, since they are not associative operations, type conversion and inverter insertion as shown in equation 4.3 and 4.4 are needed. Figure 4.16 shows some of the typical gate fitting results.

4.4.5 NAND-NAND Implementation

NAND-NAND implementation is preferred in certain types of technology. If the NAND-NAND implementation option is selected by the designer ANFT will perform the NAND conversion based on the following Boolean equations:

$$(I_1 \cdot I_2 \cdot \dots \cdot I_n) = \overline{(I_1 \uparrow I_2 \uparrow \dots \uparrow I_n)} \quad (4.6)$$

$$(I_1 + I_2 + \dots + I_n) = \overline{(\bar{I}_1 \uparrow \bar{I}_2 \uparrow \dots \uparrow \bar{I}_n)} \quad (4.7)$$

$$(I_1 \downarrow I_2 \downarrow \dots \downarrow I_n) = \overline{(\bar{I}_1 \uparrow \bar{I}_2 \uparrow \dots \uparrow \bar{I}_n)} \quad (4.8)$$

$$\begin{aligned} (I_1 \oplus I_2) &= ((\bar{I}_1 \cdot I_2) + (I_1 \cdot \bar{I}_2)) \\ &= ((\bar{I}_1 \uparrow I_2) \uparrow (I_1 \uparrow \bar{I}_2)) \\ &= (((I_1 \uparrow I_2) \uparrow I_1) \uparrow ((I_1 \uparrow I_2) \uparrow I_2)) \end{aligned} \quad (4.9)$$

Only the AND, OR, NOR and XOR to NAND conversions will be performed. The DFFs will remain unchanged. For an AND gate, if its input number, say eight, matches one of the available input number of NANDs then the AND will be converted into an 8-input NAND with an inverter attached as shown in Figure 4.17 (a.) In case of a mismatch, the AND will be broken into ANDs according to the available **NAND** gate input numbers. Then the newly created ANDs will be converted directly into

Available input number of AND : 4 3 2
 OR : 3 2
 NAND : 8 4 3 2
 NOR : 8 4 3 2

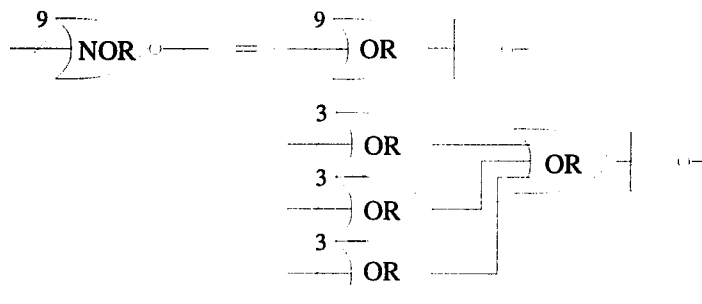
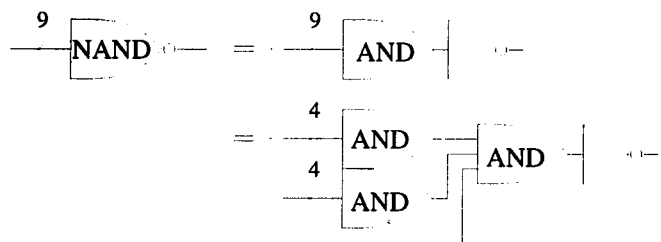
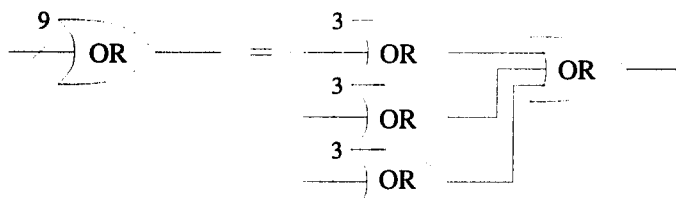
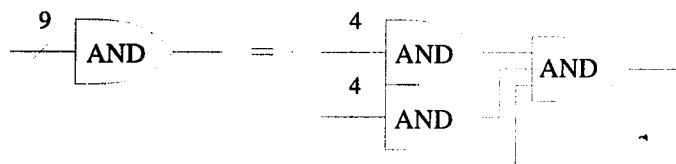
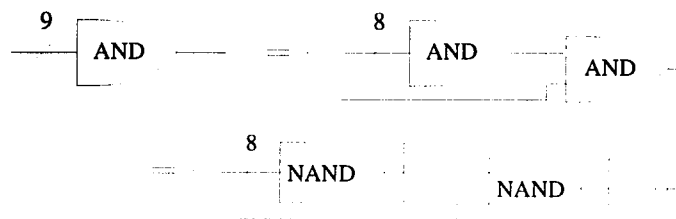


Figure 4.16: Example of Typical Gate Fitting Results.

Available input number of NAND : 8 4 3 2



(a) Converting An 8-input AND to Equivalent NAND Logic.



(b) Converting A 9-input AND to Equivalent NAND Logic.

Figure 4.17: Examples of AND to NAND Conversion.

NANDs with inverters attached as shown in Figure 4.17 (b.) Converting OR to NAND is the same as converting AND to NAND except that the inverters will be inserted to each input pin of the new NANDs rather than attached to their output pins. For NOR to NAND, the NOR will be changed to an equivalent OR-INV logic first. Then the OR part will be converted into NAND logic. For XOR to NAND, the XOR will be flattened into the equivalent OR-AND-INV logic and the ANDs and OR will be converted into NAND logics accordingly.

4.4.6 BLACK BOX Fitting Using FLEs

BLACK BOXes are modeled as FLEs. Fitting a FLE element is easy because all the necessary usage checks are done by HPCOM and its interior world is irrelevant to

the whole transformation process. A FLE will be treated as an integrated functional unit; thus, the object breaking process can not and will not be applied to it. All the ANFT has to do is to make sure that the definition of the FLE element is provided in the assigned library and the I/O pin numbers are matched. If any of the conditions are not satisfied ANFT will stop transforming and give error messages.

BLACK BOX elements can be included in a design in various forms. If Figure 4.18 shows three different ways of declaring and using 2-bit ripple carry adders (RCADD2) as BLACK BOXes. First of all, there must be a RCADD2 element defined in the BLKBOX section of the assigned library. It must contain five input and three output pins:

```

BLKBOX:    12
           ⋮
           RCADD2  5 3 1  A1 A0 B1 B0 Cin  Cout S1 S0
           ⋮

```

In the first example, BBOX3, a BLACK BOX RCADD2 is declared as:

```

CLUNITS: RCADD2[3].

```

and is used as:

```

2  M <= RCADD2(A;B;Cin);

```

The “[3]” indicates that RCADD2 is declared as a 3-bit CLU hence has three output pins. The I/O connections must be aligned with the I/O pin assignments. There is no way for ANFT to do any safety check of the kind.

```

Example 1:  MODULE    : BBOX3
              EXINPUTS  : A[2]; B[2]; Cin; Clcok; Reset.
              EXOUTPUTS : OUT[3].
              MEMORY    : M[3].
              CLUNITS   : RCADD2[3].

              BODY SEQUENCE : Clock.
              :
              2  M <= RCADD2(A; B; Cin);
              :

Example 2:  MODULE    : BBOX5
              EXINPUTS  : A[2]; B[2]; C[2]; D[2]; Cin1; Cin2; Clcok; Reset.
              EXOUTPUTS : OUT[3].
              MEMORY    : M1[3]; M2[3].
              CLUNITS   : RCADD2[6].

              BODY SEQUENCE : Clock.
              :
              2  M1 <= RCADD2[0:2](A; B; Cin1);
                 M2 <= RCADD2[3:5](C; D; Cin2);
              :

Example 3:  MODULE    : BBOX9
              EXINPUTS  : A[2]; B[2]; C[2]; D[2]; Cin1; Cin2; Clcok; Reset.
              EXOUTPUTS : OUT[3].
              MEMORY    : M1[3]; M2[3].
              CLUNITS   : RCADD2<2>[3].

              BODY SEQUENCE : Clock.
              :
              2  M1 <= RCADD2<0>(A; B; Cin1);
                 M2 <= RCADD2<1>(C; D; Cin2);
              :

```

Figure 4.18: Examples of Declaring and Using BLACK BOXes.

If more than one RCADD2 are needed in a design. There are two ways to declare them. As shown in the second example, BBOX5, two RCADD2s are declared as:

```
CLUNITS : RCADD2[6].
```

and is used as:

```
2 M1 <= RCADD2[0:2](A;B;Cin1);
  M2 <= RCADD2[3:5](C;D;Cin2);
```

Since one RCADD2 has only three output pins, RCADD2[6] represents 2 adders: RCADD2[0:2] and RCADD2[3:5]. Such kind of one dimensional declaration is not recommended when a BLACK BOX element is to be used many times. In stead, one should declare them as two dimensional elements as shown in module BBOX9:

```
CLUNITS : RCADD2<2>[3].
```

and is used as:

```
2 M1 <= RCADD2<0>(A;B;Cin1);
  M2 <= RCADD2<1>(C;D;Cin2);
```

Different declaration styles will cause different results of usage specifications in the HPCOM stage 2 output file. ANFT will accept all the different results and fit them into assigned functional units.

4.4.7 DFF Fitting using LETs

D type flip-flops are designated memory registers in AHPL designs. They are categorized by HPCOM as either CDFFs (4006) or DDFFs (4009.) HPCOM uses

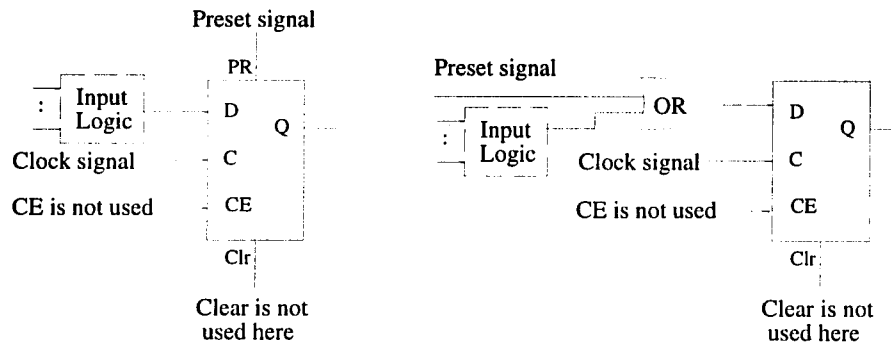


Figure 4.19: Redirecting the Preset Signal.

the DFFPCEQ model to manipulate their connections. Therefore, if there is no completely matched element for the implementation fitting them using DFF LETs become necessary.

The CDFFs if used are in charge of state transitions. In certain designs, they will be set to some initial state when initializing the circuit. Due to the necessities of presetting or resetting them, DFFPCEs will be used for the implementations. If a CDFF must initially be set as “high” (“low”) then the reset signal must be connected to the preset (clear) pin. Problem comes when the DFFPCE element is not available in the assigned library. Both of the DFFE and DFFPE elements are not suitable substitutes. ANFT can only use DFFCE elements as replacements. If a DFFCE element must be preset in a reset process, the reset signal will be redirected to the data input for providing a “high” signal. Figure 4.19 shows the adjustment that will be applied. Because the CE is never been controlled and will be always activated for a CDFF, the clock signal will be transmitted directly into a CDFF element. Thus, the adjustment will guarantee that the DFFCE been preset to “high” when resetting.

Once the DFF type for implementation is decided, fitting a CDFE depends totally on the LET of the chosen DFF type. Unnecessary input connection(s) may be removed from the CDFE object's input list.

For DDFEs, any of the four types of DFFs would be a feasible choice because none of their preset or reset pin will be in use. ANFT can let designers to select desired type for implementing the DDFEs. If the selected type is not available then ANFT will turn to the DFEE, DFFPE, DFFCE and DFFPCE elements as substitutes. Although there is no difficulty for searching substitutes, CE connection adjustments are sometimes required. If a clock condition is used in a transfer statement then the clock signal will be controlled by the condition specified and the control signal will be connected to CE. However, designer can decide whether the clock condition signals will be connected to the CE inputs directly or redirected to the D inputs by combining them with the data signals during the compiling process. If they are moved to D inputs then no adjustment is needed and the CE by default will be connected to a "high" signal. A problem comes if the CE pin is not available in the chosen DFF element when clock control signals are assigned. ANFT will NAND the original clock signal and control signal together and use it as the new clock signal as shown in the example of Figure 4.20.

For all the DFF control input pins, their polarities will also be considered in the fitting process. For example, the AHPL designs incorporate trailing clock edges as triggering signals. Therefore, the clock source will be connected to a DFF via

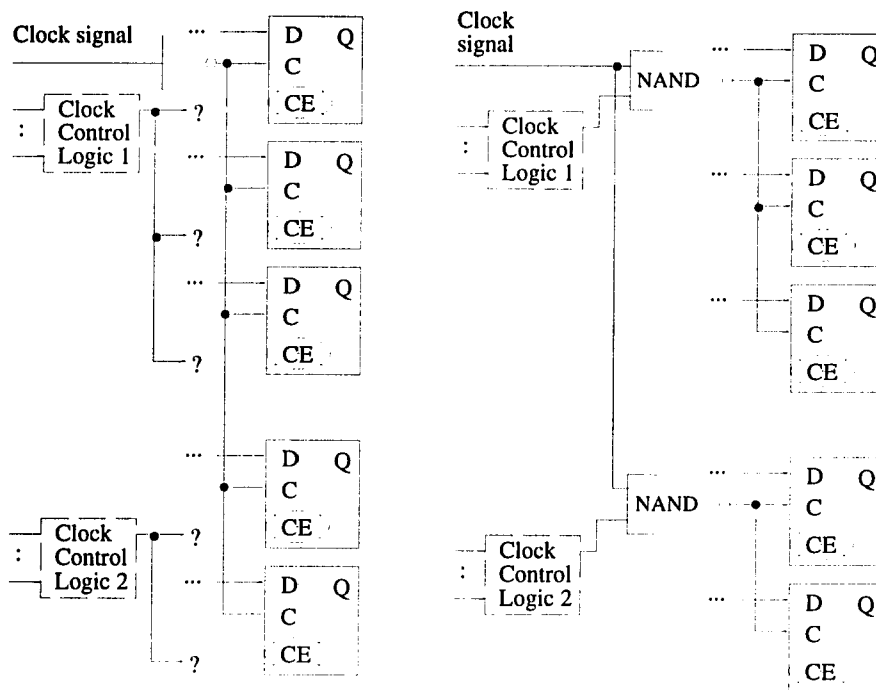


Figure 4.20: Redirecting the Clock Enable Signal of a DFF object.

an inverter when the clock pin's polarity is positive. If the polarity is negative the inverter will be replaced by an inverter pair as shown in Figure 4.21.

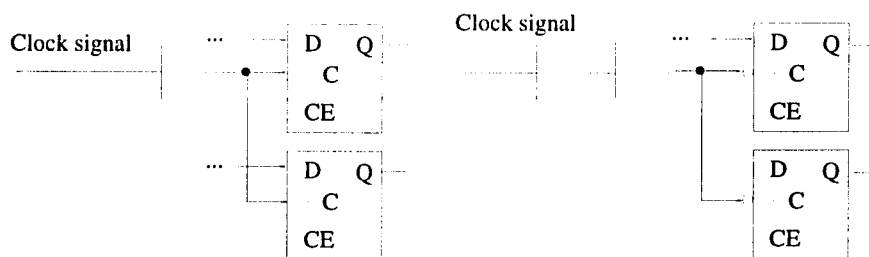


Figure 4.21: Clock Signal Connections.

4.5 Special Gate Mapping

The mapping process is designed in a dynamic fashion. Designers can control the gates to be mapped by specifying mapping logic trees. Every logic tree expression will be used for searching possible matches. If no logic tree is provided in the MAPPING section of the assigned library then the process will be skipped. The logic tree expressions can be assigned in any order. However, mapping one logic expression may reduce the chance of mapping another logic tree. Thus, designers can arrange their order in a way that is best for combining more gates.

Figure 4.22 shows a “2+(2&,*)” tree expression and the corresponding specification. The expression “2+(2&,*)” consists of a 2-input OR gate with a 2-input AND and a don't care as inputs which is exactly an AOI21 gate. When don't cares are used, ANFT will try all the possible combinations for a match. Therefore, there are two possible matchings as shown in the example. The input number of a tree expression must be the same as the input pin number of the corresponding special gate. Otherwise, the matching of the particular tree expression will be skipped.

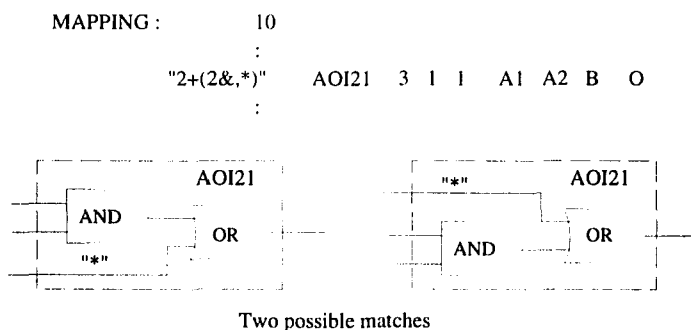


Figure 4.22: Example of Mapping the AOI21 Gate.

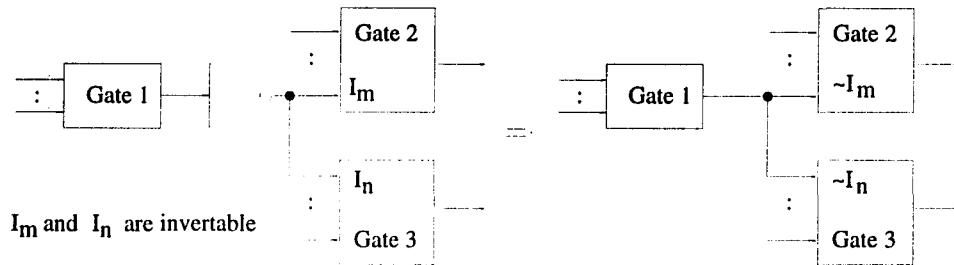


Figure 4.23: Example of Utilizing The Invertible Input Pins.

4.6 Invertability Utilization

In some of the vendor-specific libraries, invertible or inverted I/O pins are provided. As mentioned, utilizing them can reduce the total gates needed. However, they can be utilized only when they have connections to or from inverters. ANFT will scan the entire circuit and remove unnecessary inverters. The scan process will not examine the I/O pins of all gates for seeking possible reductions. Only the elements that connect to inverters will be checked.

If a gate has two output pins, say O and \bar{O} , then the second one will be the inversion of the first one and vice versa. If either or both of them have connections to inverters, say $INV1$ and $INV2$, then the output connection(s) of the $INV1$ can be reconnected to \bar{O} . Similarly, the output connection(s) of the $INV2$ can be moved to the O . Both the $INV1$ and $INV2$ can then be discarded. If a gate has only one output pin, O , and has a output connection to an INV it is still possible to remove the INV as shown in Figure 4.23. Suppose both I_m and I_n that the inverter connects to are invertible.

By reconnecting the input signal of the inverter directly to the inversions of I_m and I_n , the original function will be preserved and the inverter can be removed.

4.7 Fanout Limit

The default fanout number for all gates must be specified as the parameter P2 in every ANFT library. When fitting objects, the drive capability of all the elements are assumed to be the default fanout number. The maximum fanout of a gate is the product of its drive strength and the default number.

Suppose the default drive capability is 10 and the drive strength of a gate is 3 then the maximum fanout of the gate is $3 \times 10 = 30$. If a n-input AND gate is designed to have drive strength of 1, 2 and 3 and a non-inverting buffer has drive strength of 1 and 4 then the possible fanout of the AND gate can be from 10 up to 1200 (i.e., $(3 \times 10) \times (4 \times 10)$.)

When selecting the suitable combination, ANFT will try to avoid using excessive drive strength on small fanout requirements. Therefore, suitable components with smaller drive capabilities will be considered first. The default fanout number usually ranges from 7 to 16. Thus, using minimum drive strength components in a 1-level buffer tree is enough for handling 49 to 256 fanout as shown in Figure 4.24 (a.) The maximum value, 256, is high enough for most of the ordinary designs. In some special cases, 2-level buffer trees may be needed for extra high fanout as shown in Figure 4.24

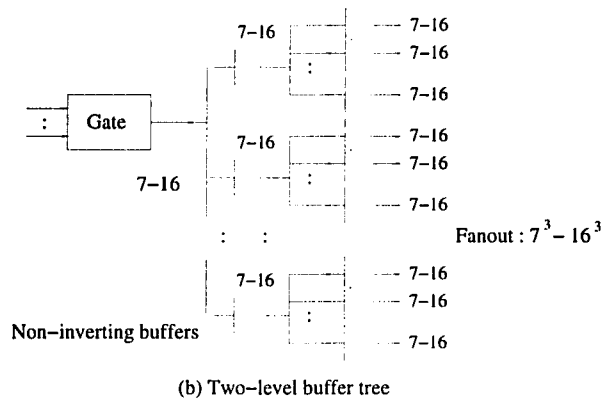
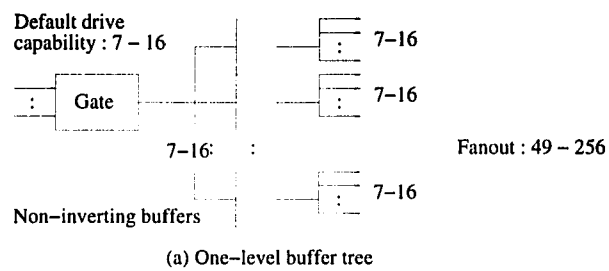


Figure 4.24: Example of Using Buffer Tree.

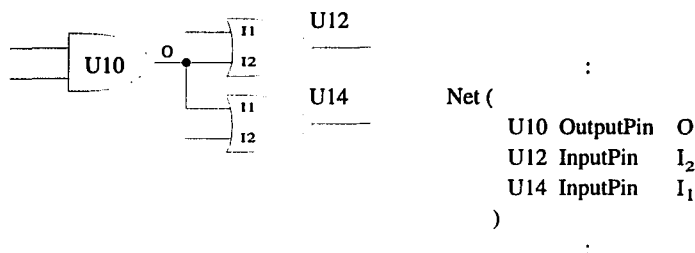


Figure 4.25: Example of Pin-to-Pin Connection Netlist Style.

(b.) Should it fail to provide enough drive capability, ANFT will not insert any buffer tree for the specific gate but only prompt a warning message.

4.8 Output File Generation

Once an AHPL netlist has been successfully transformed, generating the output file is easy. ANFT by default will generate a reference file automatically for every design. Then the user selected output file generator will produce the target netlist. There are two different styles for describing the connection behavior among the three available target formats. The first one, pin-to-pin connection style, is based on describing the interconnections of sets of I/O pins. As shown in Figure 4.25, the net statement describes that the output pin of unit 10 (O,) input pin 2 of unit 12 (I2) and input pin 1 of unit 14 (I1) are interconnected. When generating netlist file using this style, the output generator will examine the outgoing connections of the output pins of all elements instead of the input connections of them.

The second style, I/O connection style, is based on describing the I/O connections of each gate. Figure 4.26 is an example of describing I/O connections to an 2-input

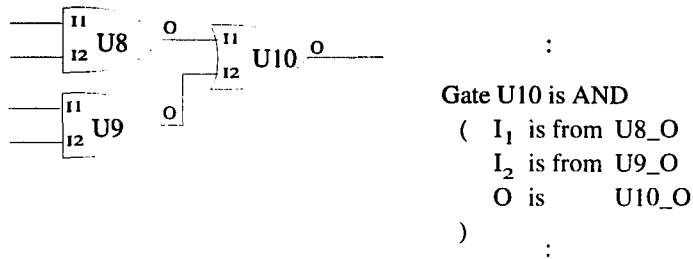


Figure 4.26: Example of I/O Connection Netlist Style.

AND gate. In this style, the inputs are associated with the signal sources respectively while the outputs are used as signal sources. The file generator will produce the connection descriptions by examining the I/O pins of each element.

Generating the reference and target file section by section will force the output generators to repeatedly search the entire element array for different elements. Recall that the maximum element number is usually more than needed and the elements are not contained in consecutive entries of the element pointer array. The searching processes used will waste a lot of time checking the array and element type. Therefore, while generating the reference file, ANFT will build some internal dynamic linked lists. The purpose is to speedup the target file generation process by avoiding unnecessary checking and searching of some specific I/O connections or elements.

CHAPTER 5

Using The ANFT

Currently, the ANFT can be run on both the MS-DOSTM and the UNIXTM platform. The instructions of running the MS-DOSTM version ANFT will be briefly discussed. Running the UNIXTM version ANFT is similar; thus, most of the discussion can be applied to it. The applications for ANFT output will also be presented.

5.1 Execution Mode and Option

ANFT has two execution modes: batch and interactive mode. The batch mode is useful when one wishes to run ANFT using batch files. If thirteen parameters are provided at the command line in the following format ANFT will run in the batch mode.

```
anft F1 F2 F3 F4 O1 O2 O3 O4 O5 O6 O7 O8 O9
```

```
F1: [[drive:\][path\]] *.glf: input source file name.
F2: [[drive:\][path\]] *.* : reference output file name.
F3: [[drive:\][path\]] *.* : target output file name.
F4: [[drive:\][path\]] *.lb : gate/cell library file name.
O1: (1-4) output file format.
O2: (1-4) DFF type for implementing the DDF (4009).
O3: (Y|N) map the special gate(s).
O4: (Y|N) NAND-NAND implementation.
O5: (Y|N) split the VDD/GND block.
O6: (Y|N) all-tri-state-bus implementation.
O7: (1-2) replace XOR with 1-OR-2-AND-2INV logic or 4-NAND.
O8: Unused.
O9: (M|#|N) the number of I/O pads or no pad to be used.
```

These parameters can only be assigned at the command line. ANFT will not ask for any options from users while translating. If any problem is encountered when opening files F1 to F4 or any erroneous assignment among O1 to O9 is found ANFT will abort the translation and give error messages.

If the number of command line arguments is not fourteen (one for "anft" itself and thirteen for the parameters) then ANFT will enter the interactive mode. The following menu will be displayed on the screen when ANFT runs in the interactive mode.

<p style="text-align: center;">A N F T : AHPL Netlist Format Translator << May 12, 1995 >> Dept. of Electrical & Computer Engineering, UofA, Tucson</p>	
Input file name	:-
Info. output file name .	:
Netlist output file name :	:
Library file name	:
Output format :	1.<EDIF110> 2.<EDIF200> 3.<XNF2000> 4.<XNF3000> (1-4) ?
Data DFF type :	1.<DFF> 2.<DFFPC> 3.<DFFP> 4.<DFFC>... (1-4) ?
Map the special gates provided in the library file : (Y/N) ?
Use NAND-NAND implementation : (Y/N) ?
Split VDD/GND block for NETTRAN : (Y/N) ?
Build all internal and external buses as Tri-State BUSES : (Y/N) ?
Replace XOR with 1.<1-OR-2-AND-2INV> or 2.<4-NAND> logic : (1/2) ?
[Unused]	?
Use PAD frame [MIN number / number specified / NO] : (M/number/N) ?
Use [Tab Up Dn] to move, [Space] to execute, [Esc] to Quit.	

The menu contains two windows. The upper one shows the copyright and version of ANFT. The lower one is further divided into three parts. The top and middle parts

are for assigning I/O files and user options. The bottom part is for displaying the translation messages. The available options from top to bottom are:

Output File Format Currently the available output file formats are EDIF 1 1 0, EDIF 2 0 0, XNF2000 and XNF3000. The XNF2000 and XNF3000 are XNF Netlist (specification version 4.00b [15]) using LCA 2000 and 3000 family chips.

Data DFF Type Designers can select the type for implementing the memory variables declared in the AHPL description. The available types are DFF, DFFPC, DFFP and DFFC. Whenever the user type is not available, ANFT will use default type instead as discussed in Chapter 4.

Mapping the Special Gates This option is for enabling or disabling the special gate mapping process. If the answer is “Y” ANFT will map the assigned logic trees in the library. The NAND-NAND implementation will be disabled if the mapping process is activated.

NAND-NAND Implementation This option decides whether the circuit will be implemented using NAND-NAND logic only. Using this option will disable the special gate mapping.

Split the VDD/GND block This option is useful when one wants to generate EDIF 1 1 0 file for the NetTran™ version 2.25. NetTran™ version 2.25 can not handle more than 400 joined I/O pins. Designs using more than 400 elements

will result in unacceptable joined VDD and GND sections. Therefore, answering “Y” to this option will break the overflowed VDD/GND joined section into acceptable subsections.

All-Tri-State-Bus Implementation Designers may implement all the bus type elements as tri-state buses by answering “Y” to this option. It will override the effect of any sub-type parameter. If the answer is “N” then the bus elements will be built according to their original type assignment.

XOR Replacement This option is effective only if the XOR is not available in the assigned library or the NAND-NAND implementation is selected. The available options are 1-OR-2-AND-2-INV and 4-NAND. When using the AND-OR-INV implementation, a XOR will be changed to the selected logic if XOR is not available. When using the NAND-NAND implementation, the 1-OR-2-AND-2-INV equivalent logic will result in a 3-NAND-2-INV logic while the 4-NAND one will not be changed as shown in Figure 5.1.

Pad Generation Designers can include pads into the design by answering “M” or desired number to this option. Minimum and specified number of pads will be generated respectively. Unlike the AENT91 which will generate six pads less than the number assigned, ANFT will generate the exact number of I/O pads specified. If the number is not enough ANFT will make the necessary increment automatically. Answering “N” will disable the pad generation.

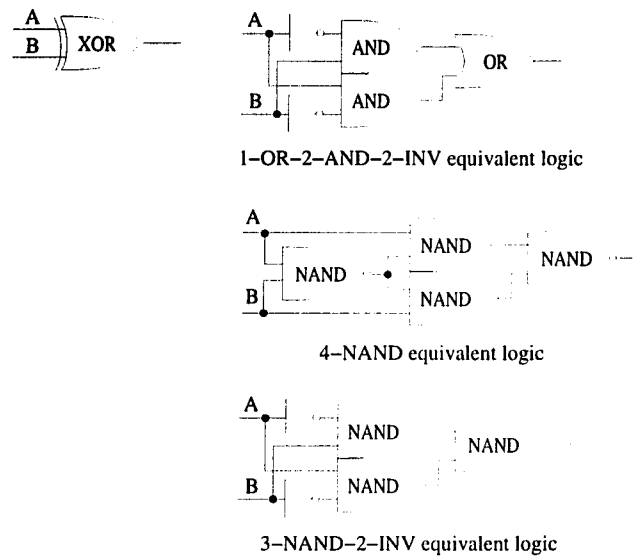


Figure 5.1: Three Types of XOR Equivalent Logic.

Notice that there is an unused option that can be used in the future. The interactive mode provides designers a convenient way of translation. If one wishes to generate multiple outputs then only partial adjustment of the settings are required. The settings can be stored in a default file called **anft.def**. It can be created or updated for the current settings by pressing “F1” key. ANFT will try to load and use the default settings when running in the interactive mode. If the default file does not exist one has to setup all the options before the translation can start.

5.2 Applications of ANFT Output

In this section, brief instructions of how to port the ANFT outputs to the compatible CAD tools will be presented. Only the procedures will be discussed. One should refer to proper manuals for the detail information of each of the CAD tool.

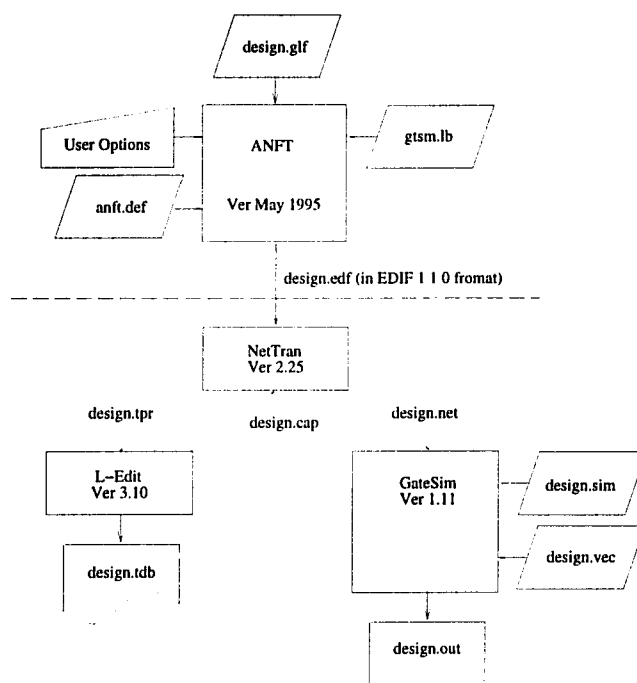


Figure 5.2: Application Using L-Edit™ and GateSim™.

5.2.1 Porting ANFT Output to L-Edit™ and GateSim™

L-Edit™ and GateSim™ are PC based layout and simulation tools. As shown in Figure 5.2, the ANFT output, generated in EDIF 1 1 0 format using the **gtsm.lb** library, can be ported to them for chip layout and post-layout timing simulation. The ANFT output file, **design.edf**, must be translated into acceptable format using Nettran™.

The L-Edit™ input file may be generated by the following commands

```
nettran -m sch2tpr.mac -P user.mac design.edf design.tpr
nettran -m user.mac design.edf design.tpr
```

or

```
nettran -m sch2tpr.mac design.edf design.tpr
```

The first set of commands should be used only when memory space is not enough for loading the required NetTran™ macro definitions. The **design.tpr** can be used for chip layout in the L-Edit™ which can generate the layout file, **design.tdb**, and the post-layout capacitance file, **design.cap**.

The GateSim™ input file, **design.net**, can be generated using the following commands

```
nettran -m sch2sim.mac -p user.mac design.edf design.net
nettran -m user.mac -c design.cap design.edf design.net
or
nettran -m sch2sim.mac design.edf design.net
```

If the capacitance information is desired for more accurate delay estimation then the **design.cap** file should be included as shown below

```
nettran -m sch2sim.mac -p user.mac -c design.cap design.edf design.net
nettran -m user.mac -c design.cap design.edf design.net
or
nettran -m sch2sim.mac -c design.cap design.edf design.net
```

Additional GateSim™ parameter files such as **design.sim**, the simulation command file, and **design.vec**, the input signal vector file, may also be needed for the simulation.

5.2.2 Porting ANFT Output to OrCAD/VST™

AHPL designs can be implemented using OrCAD™ Systems Corporation's library, TTL.LIB, and verified using OrCAD/VST™ version 1.10. The ANFT output files should be generated in EDIF 1 1 0 format using the **ttl.lib** and can be used by OrCAD/VST™ directly. Figure 5.3 shows the procedure of simulating designs

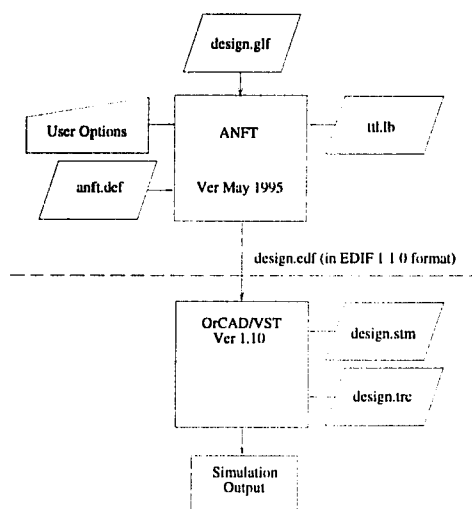


Figure 5.3: Application Using OrCAD/VST™.

using OrCAD/VST™. Notice that the stimulus file, **design.stm**, and trace file, **design.trc**, must be created for running the simulation.

5.2.3 Porting the ANFT Output to Xilinx™'s Tool System

ANFT output files generated in XNF format using **lca2000.lb** and **lca3000.lb** can be accepted by the Xilinx's tool system as shown in Figure 5.4. The default file, **lcanet.def**, can provide the version and part ID to be used in the LCA NET and PART records of the output file respectively. A possible assignment is shown as follows

2 2064pc68-100

If the default file is presented, the output generator will use these settings directly; otherwise, default values, 4 and 2064pc68-100, will be used.

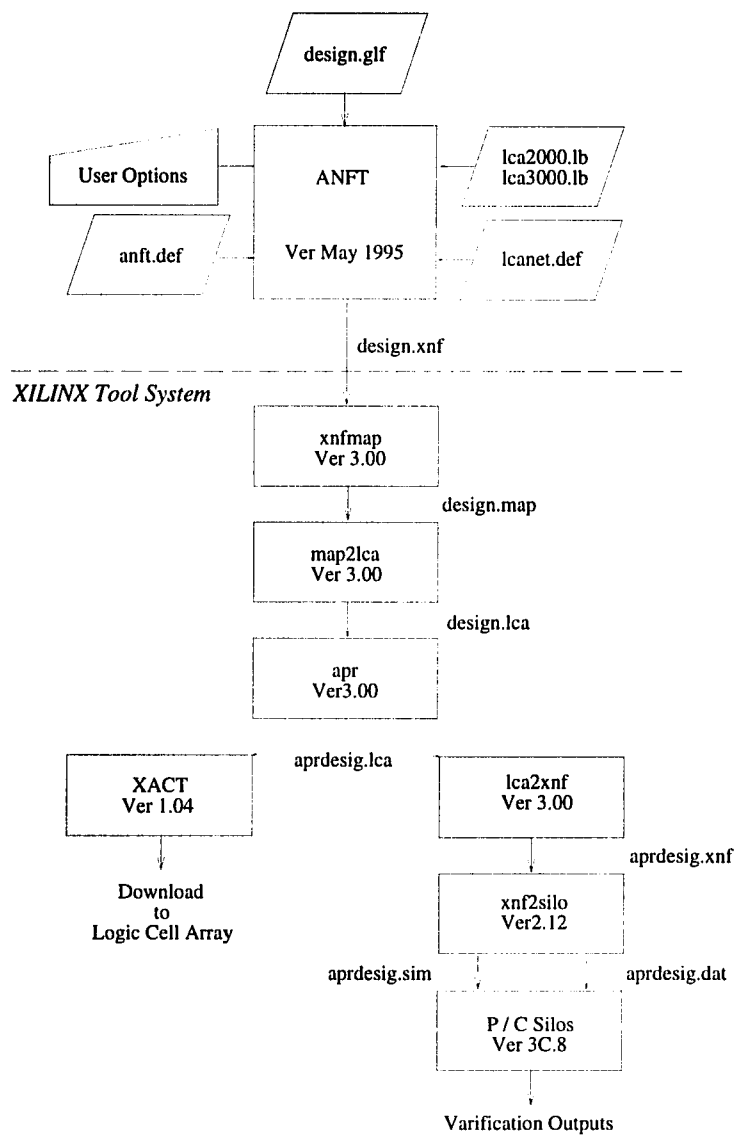


Figure 5.4: Application Using Xilinx™'s Tool System.

5.2.4 Porting the ANFT Output to COMPASS Tools™

The COMPASS Tools™ system can accept the EDIF 2 0 0 files generated by the ANFT, using the **cpss.lb** library, as shown in Figure 5.5. The default file, **edif200.def**, is for assigning information such as translator name, translator version and comment. It is optional to the translation process. If ANFT can not locate it then default values will be used. The ANFT output file, **design.edf** must be loaded by the utility **netlist** and converted to COMPASS's own netlist format for further applications.

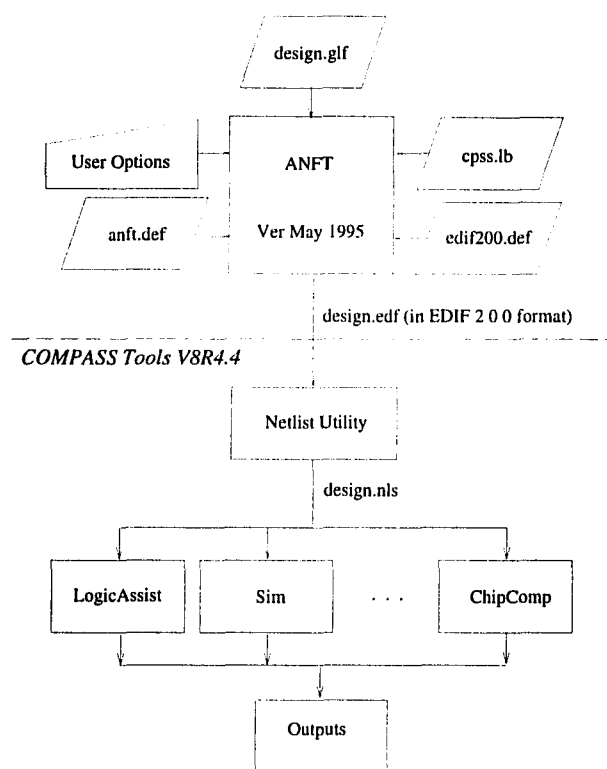


Figure 5.5: Application Using COMPASS Tools™ System.

CHAPTER 6

Examples and Evaluation

In this Chapter, two AHPL benchmarks will be used to demonstrate the applications and verify the correctness of ANFT. The first one, DEMO.HPC, will be used to show that the function of ANFT has covered what AENT91 and AENT_TTL can achieve. A modified version of DEMO.HPC, DEMOX.HPC, will also be generated to illustrate how to incorporate BLACK BOXes into the design. The second one, TODCLK.HPC, will be run and tested exhaustively using all the tools mentioned in Chapter 5. In the last section, fifteen AHPL benchmarks will be used to compare the results generated by ANFT and AENT91 and AENT_TTL.

6.1 The AHPL Demo Description: DEMO

DEMO is a very simple AHPL benchmark that shows how to specify and what will be the effects of the descriptions. Its contents are shown in Figure 6.1. There are only two steps or states in the design. The RESET signal will bring the circuit back to state 1 where LOOK will be set to logic high. The state will switch to state 2 when the next trailing edge of clock comes. The input X will also be stored in memory element A. The BUS will stay low in state 1 but will equal to the contents of memory A or \bar{R} in state 2 depending on the value of X. If X is high then BUS

```

MODULE      : DEMO.                                " TEST MODULE : DEMO "
EXINPUTS   : X; B; CLOCK; RESET.
EXOUTPUTS  : Z; LOOK.
MEMORY     : R; A.
BUSES      : BUS.
BODY SEQUENCE: CLOCK.

1   A <= X;
    LOOK = \1\.

2   BUS = (A ! ^R) * (X, ^X);
    R * B <= BUS;
    => (A, ^A) / (2,1).

ENDSEQUENCE
CONTROLRESET(RESET)/(1);

Z = ^R.

END.

```

Figure 6.1: The DEMO AHPL Description.

will be connected to output of A; otherwise, it will be connected to \bar{R} . Whether the contents of BUS will be transferred to memory R depends on the input value of B. B is a clock condition that controls the triggering signal of R. If B is low then R will not be triggered by the clocks. The EXOUTPUT Z is unconditionally connected to R. Therefore, Z is equivalent to the R. The circuit will go back to state 1 if A is low when the next trailing edge of clock comes. Once A is changed to high, the circuit will be forced to stay in state 2 until a RESET signal is issued.

User Options	AENT91	ANFT
Output Format	Default	EDIF 1 1 0
Special cell mapping	Yes	Yes
NAND-NAND implementation	No	No
Memory DFF type	DFF	DFF
\bar{Q} utilization	Yes	Default
Pad generation	No	No
Split the VDD/GND blocks	N/A	No
All-tri-state-bus	N/A	No
XOR Equivalent Logic	N/A	1
Gate/Cell Library	gtsm.lb	New gtsm.lb

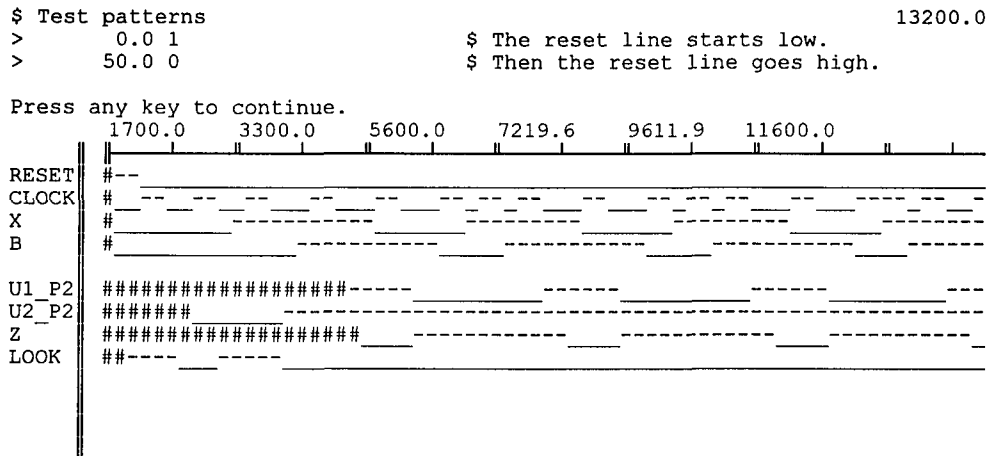
Table 6.1: Translation Settings for DEMO Using AENT91 and ANFT.

In the following subsections, the ANFT, AENT91 and AENT_TTL translated DEMO will be ported to and simulated on GateSimTM, OrCAD/VSTTM and COMPASS ToolsTM. The test file of DEMO is compiled by HPCOM (stage01 and stage23x) using the controlled clock input and one hot encoding (i.e., answer “Y” and “Y” to the two inquiries of stage23x.)

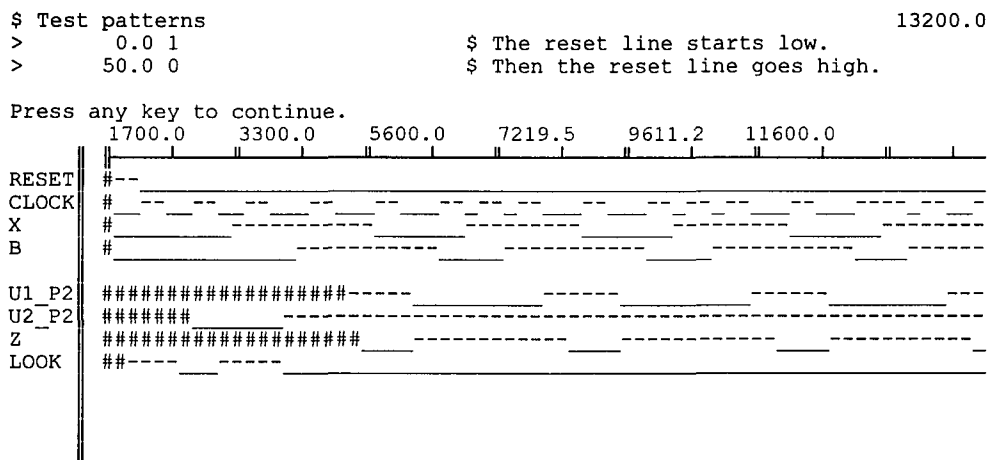
6.1.1 Porting DEMO to GateSimTM Using AENT91 and ANFT

In this subsection, the DEMO is translated using both AENT91 and ANFT and simulated using GateSimTM. The libraries used by both of the translators are different versions of **gtsm.lb** libraries. The settings for the translations are shown in Table 6.1.

Figure 6.2 shows the simulation results. One can examine that both the simulation results are conform to the behavior described at the beginning of the section. The pound character, “#”, means the value is undetermined. Therefore, the value of R remains unknown until B is high and the circuit is in state 2. This is true since its



(a) AENT91 Translated DEMO



(b) ANFT Translated DEMO

Figure 6.2: Simulation Results of DEMO Using GateSim™.

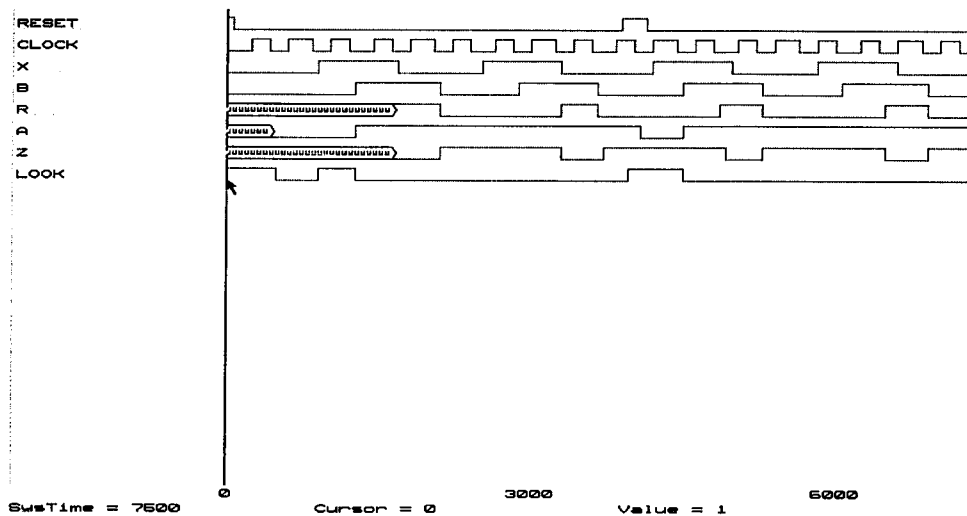
User Options	AENT_TTL	ANFT
Output Format	Default	EDIF 1 1 0
Special cell mapping	N/A	No
NAND-NAND implementation	Yes	Yes
Memory DFF type	DFF	DFF
\bar{Q} utilization	N/A	Default
Pad generation	N/A	No
Split the VDD/GND blocks	N/A	No
All-tri-state-bus	N/A	No
XOR Equivalent Logic	N/A	1
Gate/Cell Library	Internal Settings	ttl.lib

Table 6.2: Translation Settings for DEMO Using AENT_TTL and ANFT.

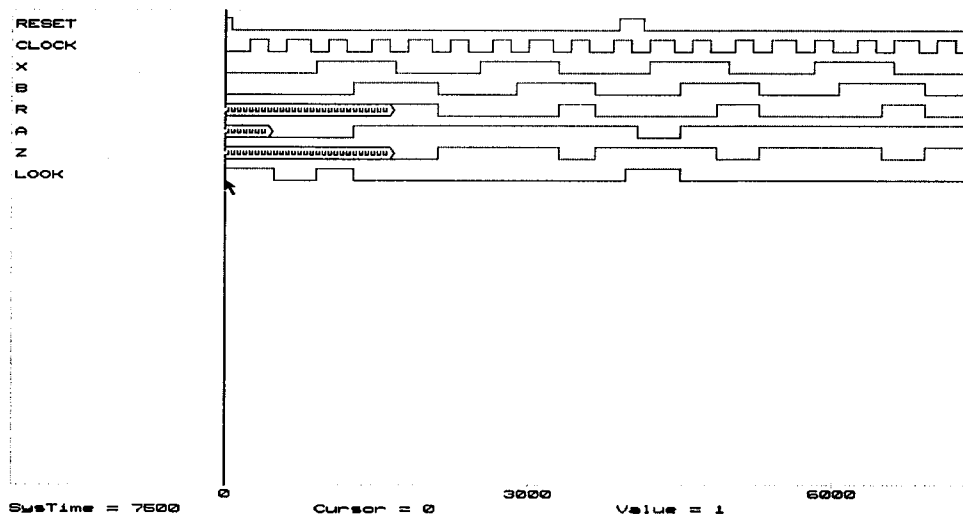
value is not set in step 1 and it can only be loaded if B is high in the state 2 when triggering signal arrives. Notice that the output waveform shown in Figure 6.2 is displayed in text mode; so the pulse widths shown are not drawn to the same scale.

6.1.2 Porting DEMO to OrCAD/VST™ Using AENT_TTL and ANFT

The output of AENT_TTL is only compatible with the OrCAD/VST™. Thus, the DEMO is translated by ANFT using the `ttl.lib` and by AENT_TTL. As shown in table 6.2, there is no library for AENT_TTL. The setting of NAND-NAND implementation is “Yes” so that the outputs will be in NAND-NAND logic. The simulation results are shown in Figure 6.3. Note that the **RESET** is activated at the 10th clock cycle again which cause the circuit goes back to state 1. The **Z** is not undetermined since R is “high” at that time.



(a) AENT_TTL Translated DEMO.



(b) ANFT Translated DEMO.

Figure 6.3: Simulation Results of DEMO Using OrCAD/VST™.

```

MODULE      : DEMOX.                                " TEST MODULE : DEMOX "
EXINPUTS   : X; B; CLOCK; RESET.
EXOUTPUTS  : Z; LOOK.
MEMORY     : R; A.
BUSES      : BUS.
CLUNITS    : mx21d1.    "BLACK BOX: 2-to-1 MUX, drive strength = x1"
BODY SEQUENCE:  CLOCK.
1   A <= X;
    LOOK = \1\.
2   BUS = mx21d1(^R; A; X);                          " modified statement "
    R * B <= BUS;
    =>(A, ^A) / (2,1).
ENDSEQUENCE
CONTROLRESET(RESET)/(1);
Z = ^R.
END.

```

Figure 6.4: AHPL Benchmark DEMOX.HPC.

6.1.3 Incorporating BLACK BOX in DEMO.HPC

In the original DEMO.HPC, the first statement in state 2 specifies the conditional connection of BUS as “BUS = (A ! \hat{R}) * (X , \hat{X});”. This statement will be implemented in a logic that is equivalent to a 2-to-1 multiplexer. We can utilize the available 2-to-1 multiplexer cell, **mx21d1**, provided in the **cpss.lb**. Therefore, in this subsection, the ANFT output of a modified version of DEMO.HPC, **DEMOX.HPC**, that uses the **mx21d1** as a BLACK BOX element will be ported to the COMPASS Tools™. The AHPL description of DEMOX is shown in Figure 6.4 where **mx21d1** must be an undefined CLUNIT and its I/O signals must be correctly connected. The schematic diagram of the DEMOX is shown in Figure 6.5. The simulation result is similar to the previous cases and is shown in Figure 6.6. Notice that the cell **U5** is the

Compass Design Automation plot [a]demoxy_nmac by s3 on 12-May-95 at 11:24 P.M.

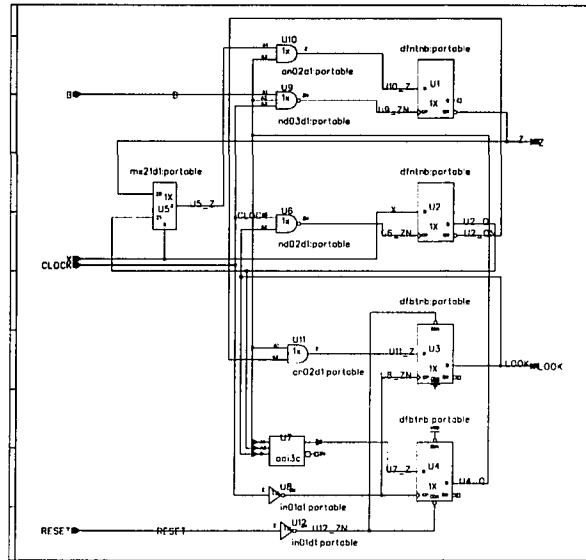


Figure 6.5: The Schematic Diagram of DEMOX.

Compass Design Automation plot [a]demoxy10M by s3 on 13-May-95 at 2:24 A.M.

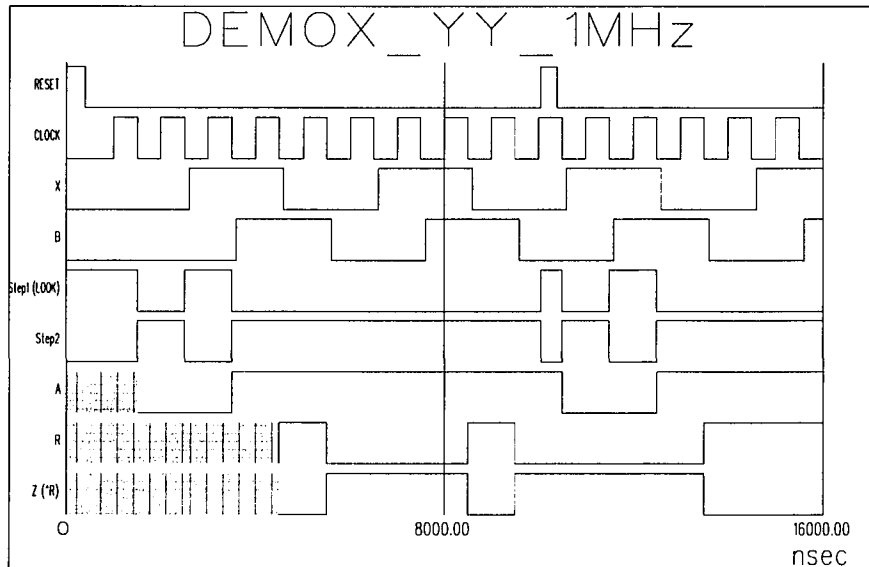


Figure 6.6: Simulation Result of DEMOX Using COMPASS Tools™.

2-to-1 multiplexer that implements the BLACK BOX **mx21d1** declared in the source file. The **U7** or **aoi3c** is a tiny module that is functionally equivalent to an AOI21. It is created separately and can be used either as a BLACK BOX element or a special gate.

6.2 The Time-of-Day Clock: TODCLK

The TODCLK, time-of-day clock [16], is a four BCD digit time of day clock. Its AHPL description is shown below:

```

MODULE: TODCLK.

EXINPUTS :RESET;SET;HRSET;LEDTEST;TWOHZ;CLOCK.
MEMORY   :PRESCR[7];MINUR[4];MINTENR[3];HRUR[4];HRTR;DISPR[2].
OUTPUTS  :MINENBL;HRENBL;DIG[5].
EXOUTPUTS:OUT[4].
CLUNITS  :INCA[7] <: INC{7}.
CLUNITS  :INCB[4] <: INC{4}.
CLUNITS  :INCC[3] <: INC{3}.
CLUNITS  :INCD[4] <: INC{4}.
CLUNITS  :INCE[2] <: INC{2}.

BODY SEQUENCE:CLOCK.

1  PRESCR <= 7$0;  MINUR <= 4$0;  MINTENR <= 3$0;

   HRUR <= 4$0;   HRTR <= 1$0;   DISPR <= 2$0;

   => (^HRSET+SET)/(1).

2  PRESCR * TWOHZ <= (INCA(PRESCR)!7$0) * (^SET & ^MINENBL, SET+MINENBL);

   MINENBL = &/(PRESCR[0:2] & PRESCR[4:6]);

   MINUR * (TWOHZ &(MINENBL+SET)) <= (INCB(MINUR) ! 4$0) *
                                     (^MINUR[0]&MINUR[3]), (MINUR[0]&MINUR[3]));

   MINTENR * (TWOHZ&(MINENBL+SET) & (MINUR[0]&MINUR[3])) <=
   (INCC(MINTENR)!3$0) * (^MINTENR[0]&MINTENR[2]),(MINTENR[0]&MINTENR[2]));

   HRENBL = (MINTENR[0]&MINTENR[2]) & (MINUR[0]&MINUR[3]);

```

```

HRUR * (TWOHZ & ((MINENBL&HRENBL)+HRSET)) <=
      (INCD(HRUR) & (^HRUR[0] & ^HRTR + ^HRUR[3]));

HRTR * (TWOHZ & ((MINENBL&HRENBL)+HRSET)) <=
      (HRTR & ^HRUR[3]) + (HRUR[0] & HRUR[3]);

DISPR <= INCE(DISPR);

DIG = ^DISPR[0] & ^DISPR[1] & ^LEDTEST, ^DISPR[0] & DISPR[1] & ^LEDTEST,
      DISPR[0] & ^DISPR[1] & ^LEDTEST, DISPR[0] & DISPR[1] & ^LEDTEST,
      LEDTEST;

OUT = (MINUR ! 1$0, MINTENR ! HRUR ! 3$0, HRTR ! 4$8) * DIG;

=> (2).

ENDSEQUENCE
CONTROLRESET(RESET)/(1).
END.

CLU      : INC(X){N}.
INPUTS  : X[N].
OUTPUT  : INC[N].
CTERMS  : CUM[N].
BODY
  INC[N-1] = ^X[N-1];
  CUM[N-1] = X[N-1];
  FOR I = N-2 TO 0 CONSTRUCT
    INC[I] = X[I]@CUM[I+1];
    IF I <> 0 THEN CUM[I] = X[I]&CUM[I+1] FI
  ROF.
END.

```

The 7-bit memory array **PRESCR[0-6]** serves as the base counter for the whole circuit. The external input signal **TWOHZ** will trigger and increase the **PRESCR** from 0 to 119 in one minute. The **MINUR[0-3]** and **MINTENR[0-2]** are counters for the unit and tenth digit of the minute. The BCD of **MINUR** will count from 0 to 9 while the BCD of **MINTENR** will only count from 0 to 5. Similarly, the **HRUR[0-3]** and **HRTR** are counters for the hour and will count from 0 to 9 and 0 to 1 respectively. The hour will be displayed in a 12-hour format; thus, the hour

counters, **HRUR** and **HRTR**, will only count from 0 to 11. The **DISPR[0-1]** will in turn connect the unit digit of minute, tenth digit of minute, unit digit of hour and tenth digit of hour to the **OUT[0-3]** which serves as the connection to the external display.

In the following subsections, the ANFT translated TODCLK will be ported to and simulated and/or implemented on GateSim™, OrCAD/VST™, Xilinx™'s tool system and COMPASS Tools™ system. The test file of TODCLK is compiled by HPCOM (stage01 and stage23x) using controlled clock input and one hot encoding (i.e., answer “Y” and “Y” to the two inquiries of stage23x). Since all the main counters will be traced exhaustively, the simulation input signal of **TWOHZ** will be half the rate of the **CLOCK** signal rather than 2Hz to shorten the simulation time. This is also the fastest rate that can be applied to the **TWOHZ** since we would like to use the **TWOHZ** to control the triggering signals of the counters. The simulations demonstrate only the correctness of the circuits being implemented; therefore, the clock rates used for the **CLOCK** are not necessarily the maximum values. As shown in Figure 5.2 to 5.5, each of the four simulators may need some simulation command and input signal vector files. They will be shown in Appendix C to F respectively. In addition, in order to trace the contents of the counters directly, some external output ports may be needed to connect to their output pin Q. It will be mentioned in the corresponding subsections if the modifications are necessary.

Settings for ANFT		Settings for GateSim™	
Output Format	EDIF 1 1 0	CLOCK Signal	1.6MHz clock
Special cell mapping	Yes	TWOHZ Signal	0.8MHz clock
NAND-NAND implementation	No	Simulation Type	Post-Layout
Memory DFF type	DFF		
Inversion Utilization	Default		
Pad generation	No		
Split the VDD/GND blocks	No		
All-tri-state-bus	No		
XOR Equivalent Logic	1		
Gate/Cell Library	gtsm.lb		

Table 6.3: Settings for Running ANFT and GateSim™.

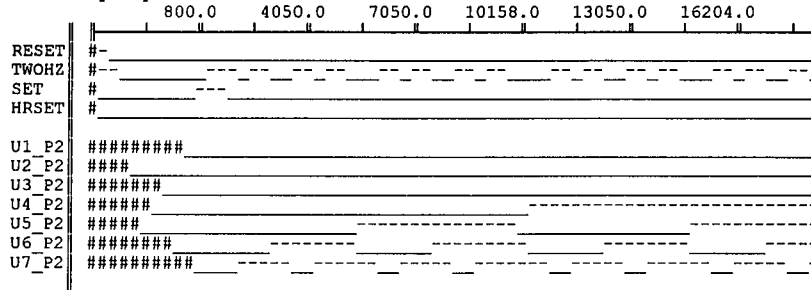
6.2.1 Test Using GateSim™

The processes of porting the TODCLK to GateSim™ follows the translation flow shown in Figure 5.2. The layout of the design is done by the L-Edit™ (no optimization time is allowed for the place and route since it does not guarantee a better result.) Then the post-layout capacitance file will be included for generating the input file of GateSim™. The settings for the translation and the simulation are shown in Table 6.3. The **TWOHZ** is set to 0.8MHz which is twice the frequency of the **CLOCK** signal. The simulation results are shown in Figure 6.7. The U1 to U7, U8 to U11, U12 to U14, U15 to U18 and U19 in Figure 6.7 represent **PRESCR[0-6]**, **MINUR[0-3]**, **MINTENR[0-2]**, **HRUR[0-3]** and **HRTR** respectively. The label P2 stands for the output pin number of Q of a DFF. One can identify that the counters counts exactly as predicted. Figure 6.7 (p) and (q) also shows that the **HRUR[0-3]** and **HRTR** are triggered only when the **TWOHZ** is “high”.

```
$ Test patterns
> 0.0 1
> 250.0 0
```

18604.0

Press any key to continue.

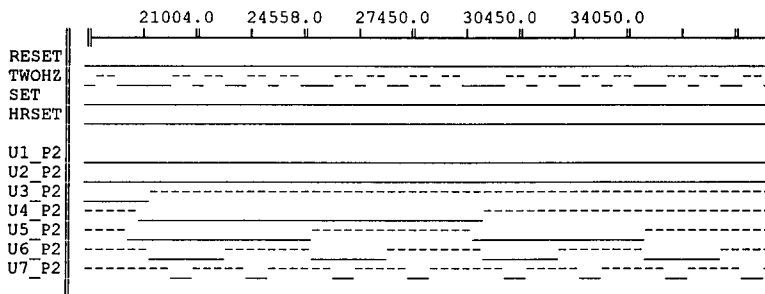


(a) PRESCR[0-6] (Part 1/8).

Continue simulation

37804.0

Press any key to continue.

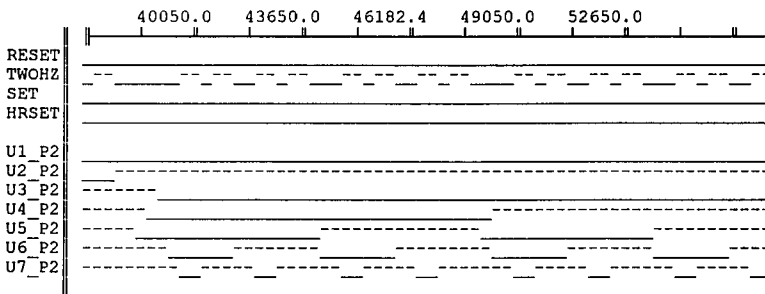


(b) PRESCR[0-6] (Part 2/8).

Continue simulation

56850.0

Press any key to continue.

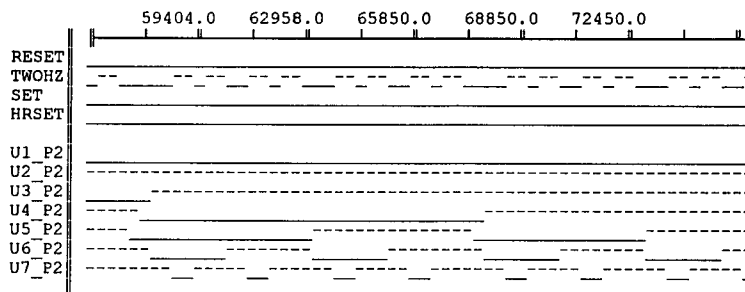


(c) PRESCR[0-6] (Part 3/8).

Figure 6.7: Simulation Results of TODCLK Using GateSim™.

Continue simulation 75450.0

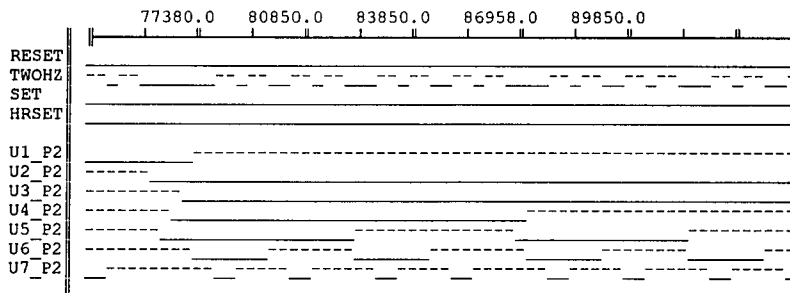
Press any key to continue.



(d) PRESCR[0-6] (Part 4/8).

Continue simulation 94650.0

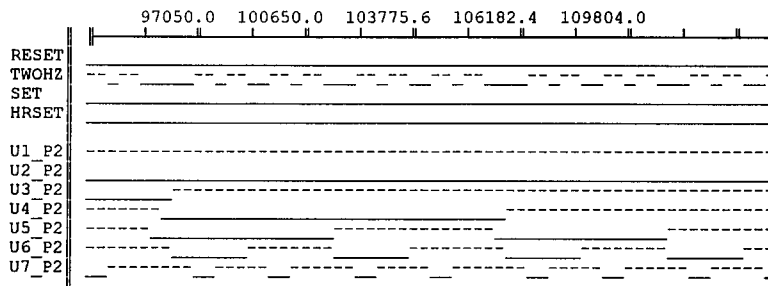
Press any key to continue.



(e) PRESCR[0-6] (Part 5/8).

Continue simulation 113850.0

Press any key to continue.

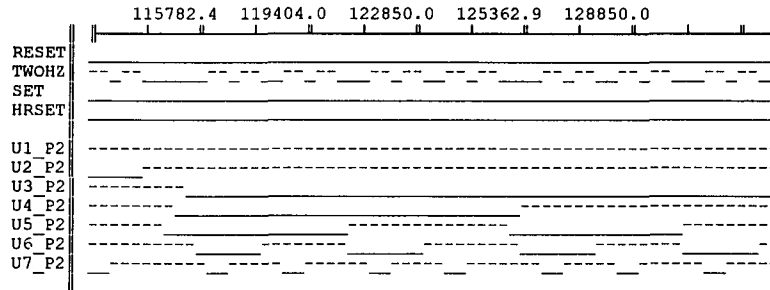


(f) PRESCR[0-6] (Part 6/8).

Figure 6.7: Continued.

Continue simulation 132582.4

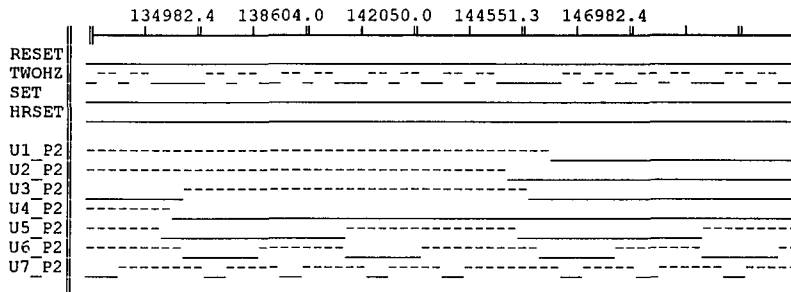
Press any key to continue.



(g) PRESCR[0-6] (Part 7/8).

Continue simulation 151782.4

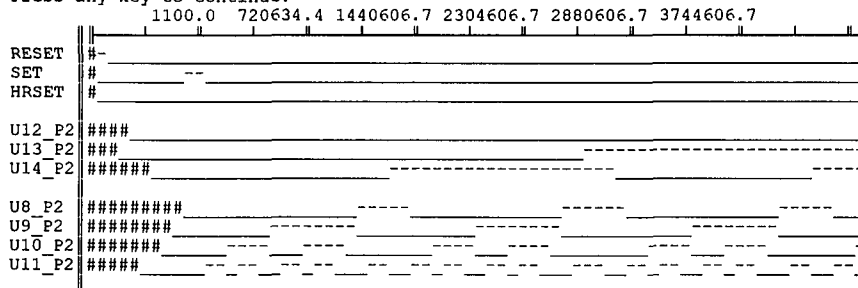
Press any key to continue.



(h) PRESCR[0-6] (Part 8/8).

\$ Test patterns 4464634.4
 > 0.0 1
 > 250.0 0

Press any key to continue.



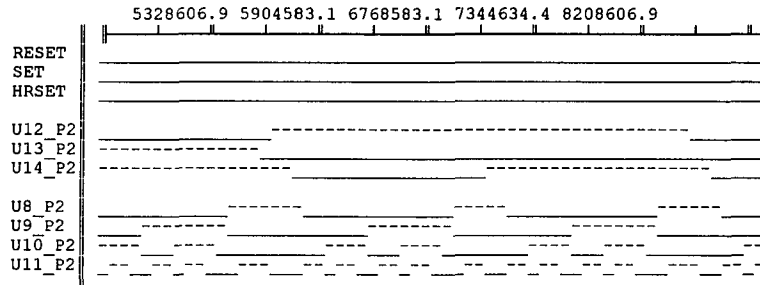
(i) MINUR[0-3] and MINTENR[0-2] (Part 1/2).

Figure 6.7: Continued.

Continue simulation

8928606.7

Press any key to continue.



(j) MINUR[0-3] and MINTENR[0-2] (Part 2/2).

\$ Test patterns

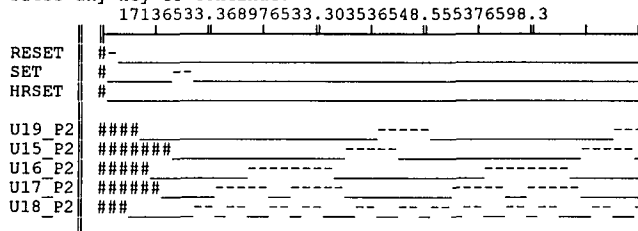
198576548.2

> 0.0 1

> 250.0 0

199900709.6

Press any key to continue.



(k) HRUR[0-3] and HRTR.

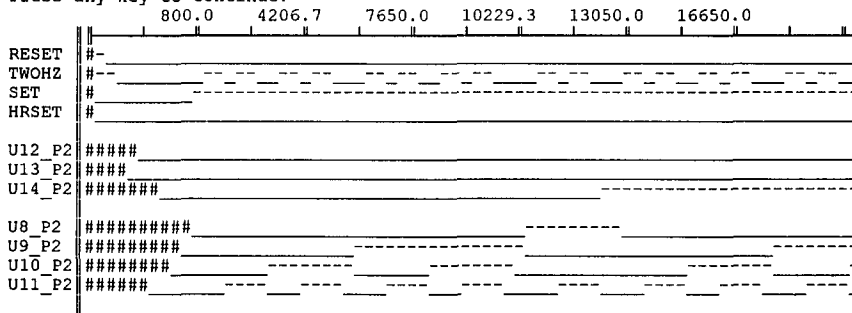
\$ Test patterns

19834.4

> 0.0 1

> 250.0 0

Press any key to continue.

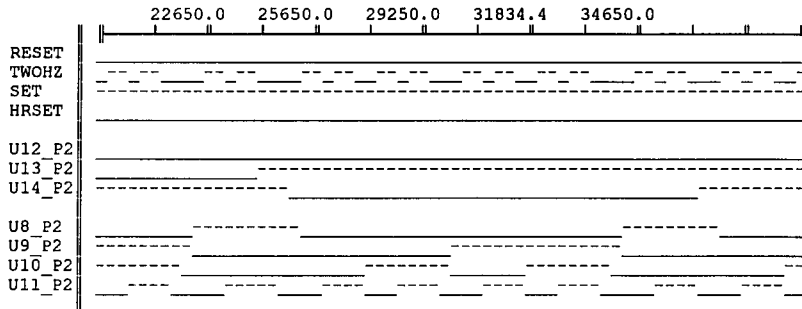


(l) SET (Part 1/4).

Figure 6.7: Continued.

Continue simulation 39450.0

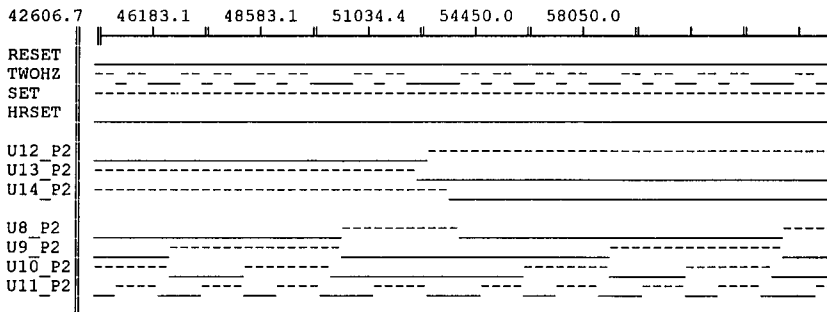
Press any key to continue.



(m) SET (Part 2/4).

Continue simulation 59850.0

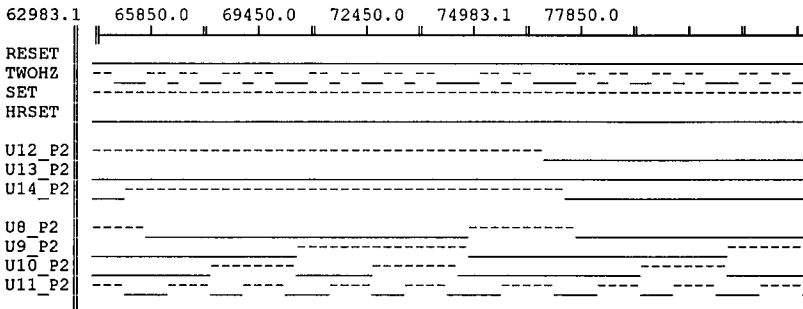
Press any key to continue.



(n) SET (Part 3/4).

Continue simulation 79834.4

Press any key to continue.



(o) SET (Part 4/4).

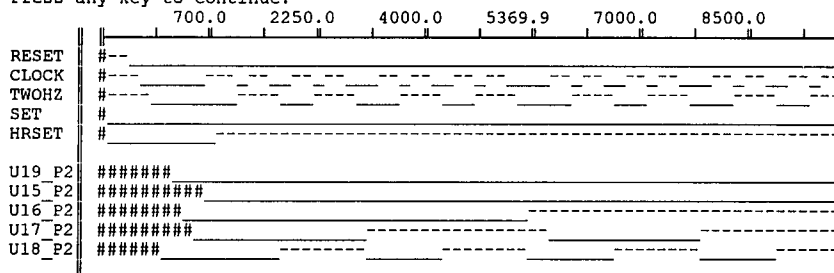
Figure 6.7: Continued.

```

$ Test patterns                                     10000.0
> 0.0 1
> 250.0 0

```

Press any key to continue.



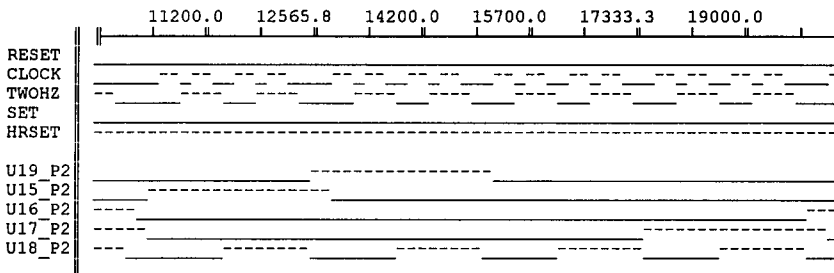
(p) HRSET (Part 1/2).

```

Continue simulation                                 19900.0

```

Press any key to continue.



(q) HRSET (Part 2/2).

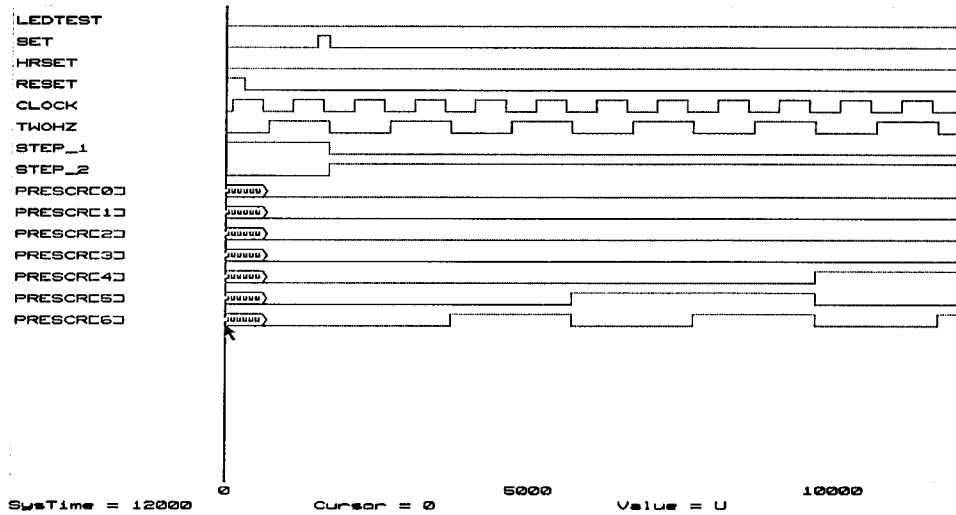
Figure 6.7: Continued.

6.2.2 Test Using OrCAD/VST™

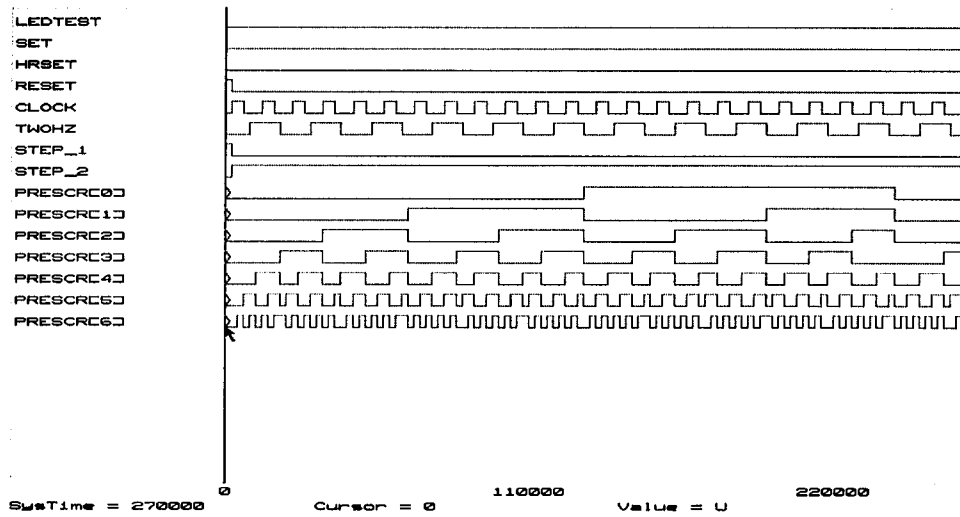
The OrCAD/VST™ Ver 1.10 can simulate designs implemented using OrCAD's TTL library. It uses a transport delay model for the gates and does not take the inter-gate wiring delays into consideration. Therefore, the results are only approximations to the real conditions. Table 6.4 shows the settings for the translation and simulation. Since the NAND-NAND implementation is selected, the translation output will be in NAND-NAND logic. The signal assignment of the **CLOCK** and **TWOHZ** is similar to the previous case and the rates are 1MHz and 0.5MHz respectively. The simulation results are shown in Figure 6.8. Due to the limitation of display resolution, the **CLOCK**, **TWOHZ** and **MINENBL** in Figure 6.8 (b) to (e) are not properly displayed by the simulator. However, the results do match the original design.

Settings for ANFT		Settings for OrCAD/VST™	
Output Format	EDIF 1 1 0	CLOCK Signal	1MHz clock
Special cell mapping	No	TWOHZ Signal	0.5MHz clock
NAND-NAND implementation	Yes	Simulation Type	Timing Simulation
Memory DFF type	DFF		
Inversion Utilization	Default		
Pad generation	No		
Split the VDD/GND blocks	No		
All-tri-state-bus	No		
XOR Equivalent Logic	1		
Gate/Cell Library	t11.lib		

Table 6.4: Settings for Running ANFT and OrCAD/VST™.

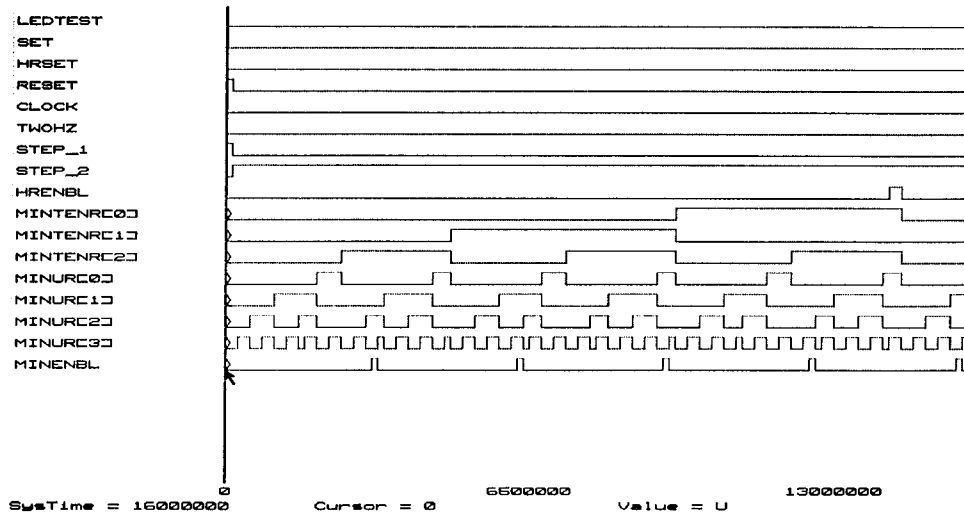


(a) CLOCK and TWOHZ.

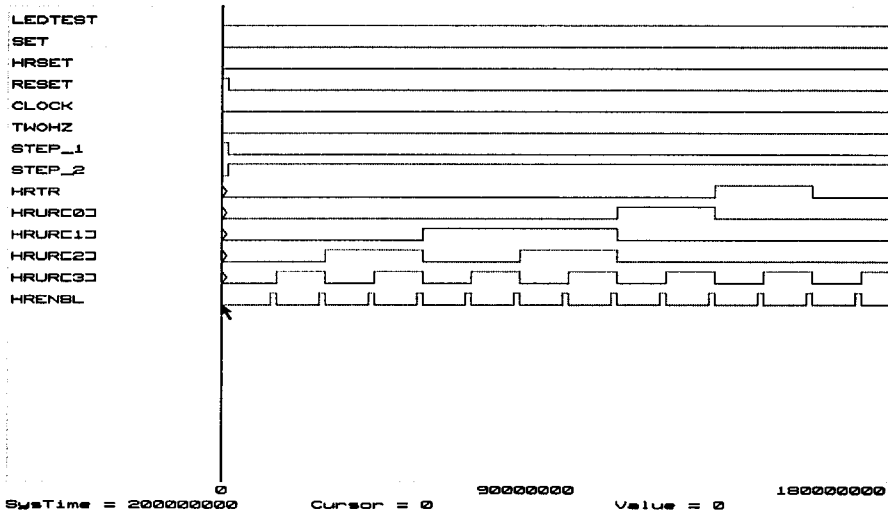


(b) PRESCR[0-6].

Figure 6.8: Simulation Results of TODCLK Using OrCAD/VST™.

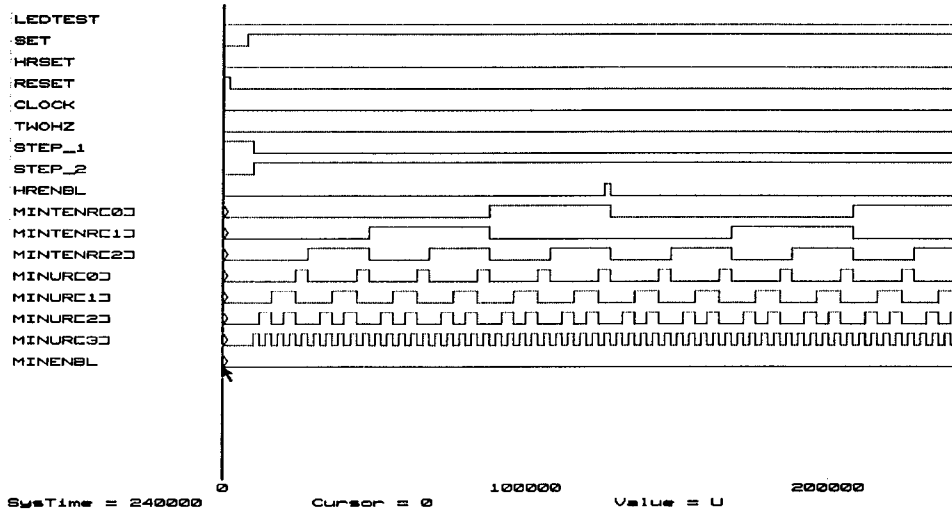


(c) MINUR[0-3] and MINTENR[0-2].

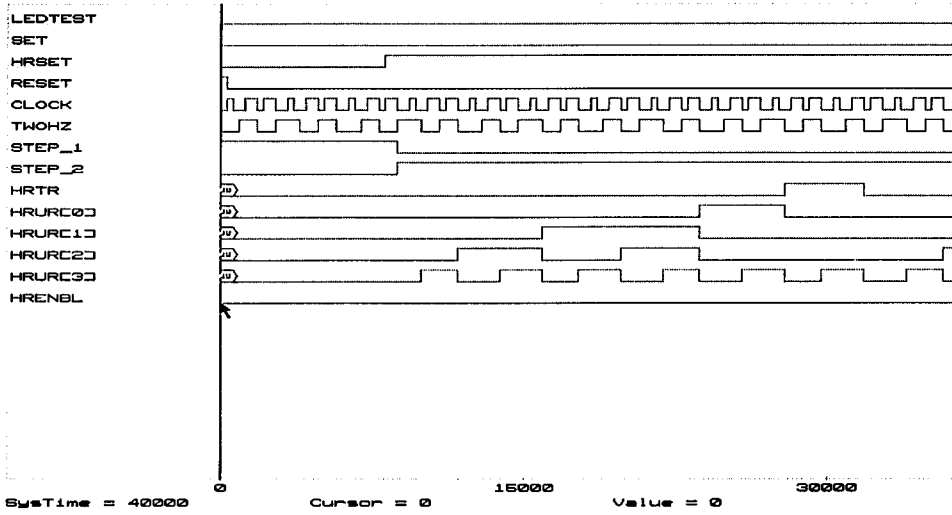


(d) HRUR[0-3] and HRTR.

Figure 6.8: Continued.



(e) SET.



(f) HRSET.

Figure 6.8: Continued.

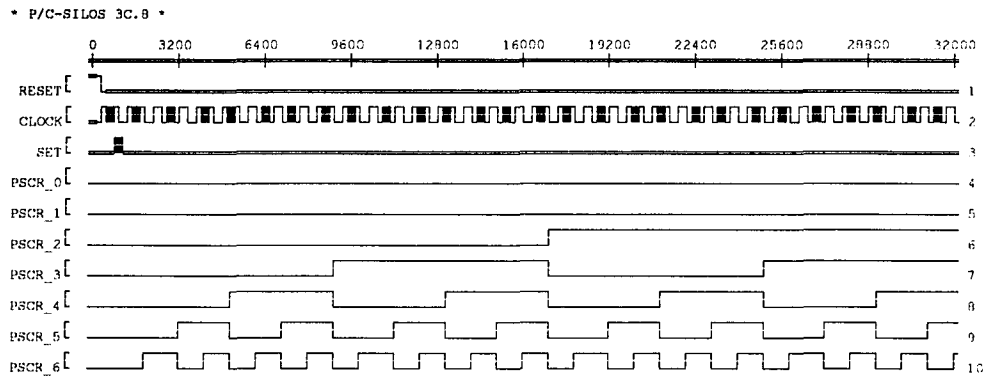
Settings for ANFT		Settings for Xilinx™'s Tools	
Output Format	XNF2000	XNF Specification	Version 2
Special cell mapping	Yes	LCA Part Type	2064pc68-100
NAND-NAND implementation	No	CLOCK Signal	2MHz
Memory DFF type	DFFPC	TWOHZ Signal	1MHz
Inversion Utilization	Default	Simulation Type	Post-layout
Pad generation	Min No.		
Split the VDD/GND blocks	No		
All-tri-state-bus	No		
XOR Equivalent Logic	1		
Gate/Cell Library	lca2000.lb		

Table 6.5: Settings for Running ANFT and Xilinx™'s Tools.

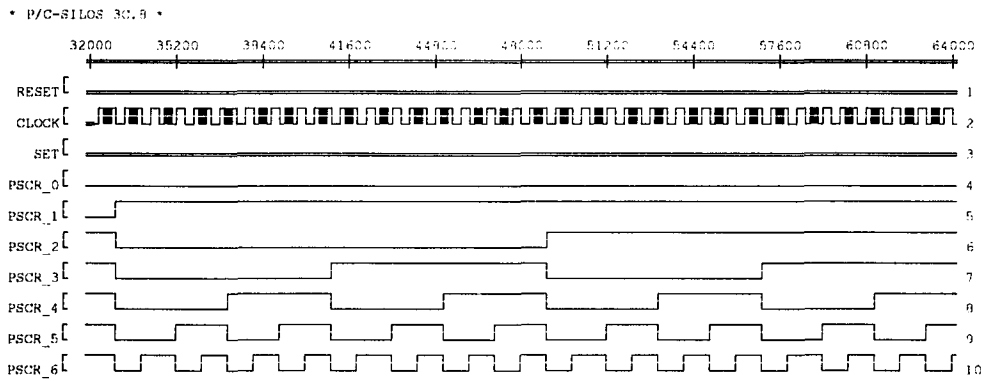
6.2.3 Test Using Xilinx™'s Tool System

The TODCLK can be implemented using Xilinx™'s LCA chip families. As shown in Figure 5.4, once the place and route is done, the post-layout simulation can be done by using the P / C Silos™ as shown in Figure 5.4. Table 6.5 summarizes the settings used by ANFT and Xilinx's tools. Notice that the pads are required for the implementation; thus, the answer to the pad generation option should be "M". The version of the XNF specification should be 2 or lower; otherwise, the `xnfmap` will not accept the file. In the LCA 2000 family, DFFPC is the only available DFF type. Therefore, ANFT will ignore any user option that is different from the DFFPC and use it as the data DFF type.

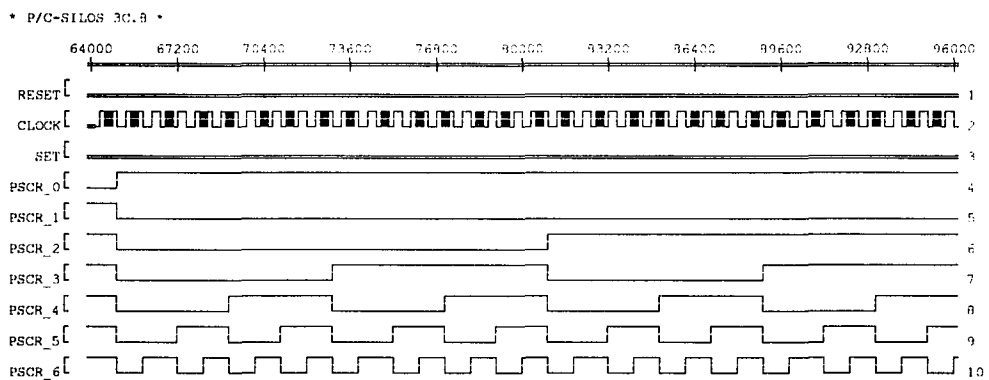
The simulation results from the P / C Silos™ are shown in Figure 6.9. In order to trace the contents of the counters, all of their output pin Q must be connected



(a) PRESCR[0-6] (Part 1/4).

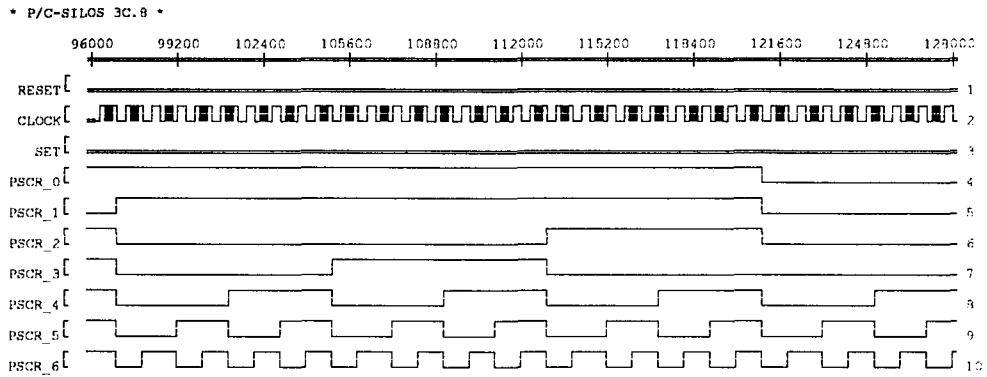


(b) PRESCR[0-6] (Part 2/4).

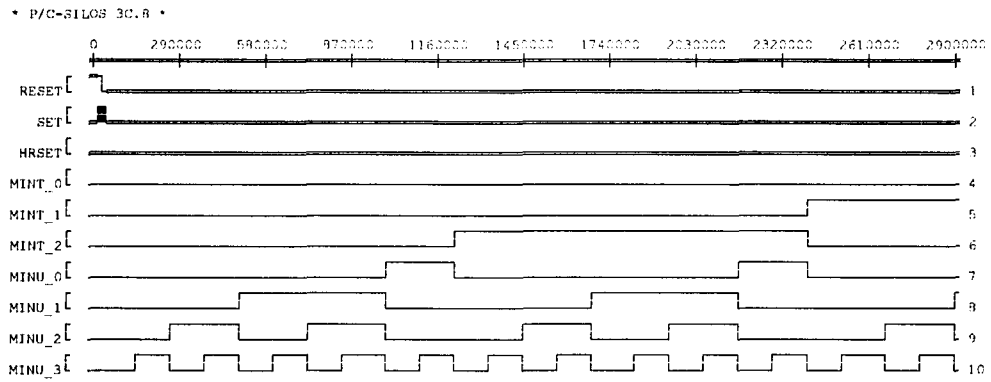


(c) PRESCR[0-6] (Part 3/4).

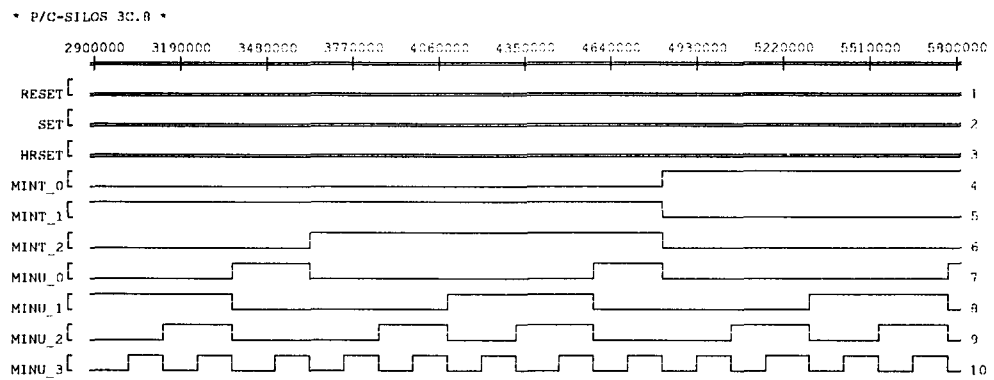
Figure 6.9: Simulation Results of TODCLK Using XilinxTM's Tool System.



(d) PRESCR[0-6] (Part 4/4).

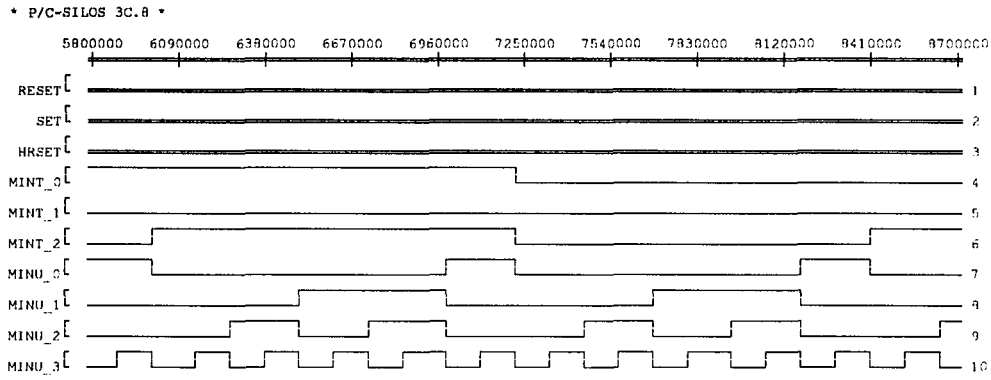


(e) MINUR[0-3] and MINTENR[0-2] (Part 1/3).

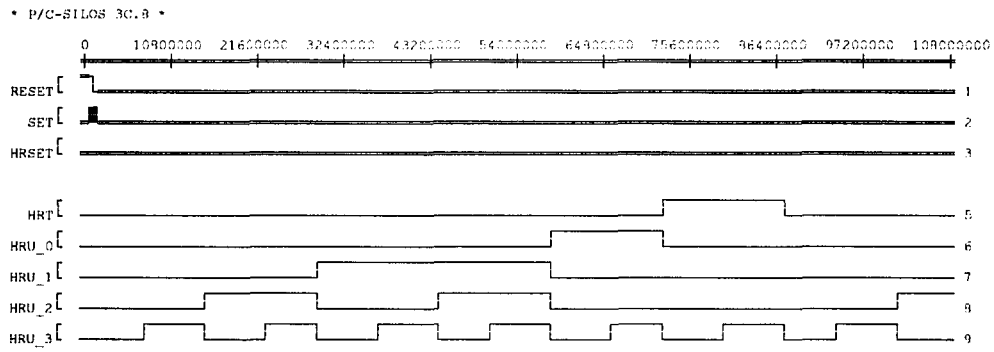


(f) MINUR[0-3] and MINTENR[0-2] (Part 2/3).

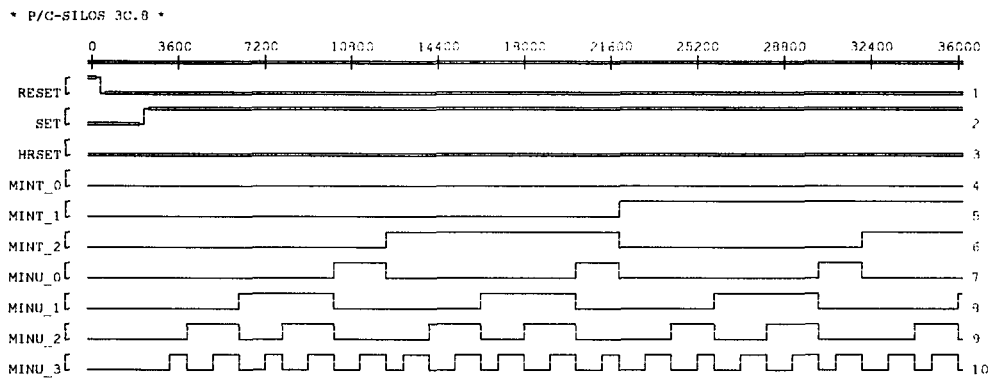
Figure 6.9: Continued.



(g) MINUR[0-3] and MINTENR[0-2] (Part 3/3).

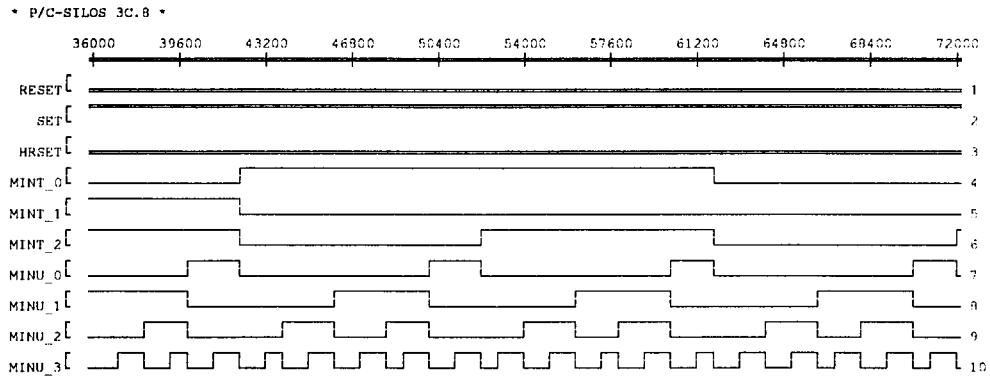


(h) HRUR[0-3] and HRTR.

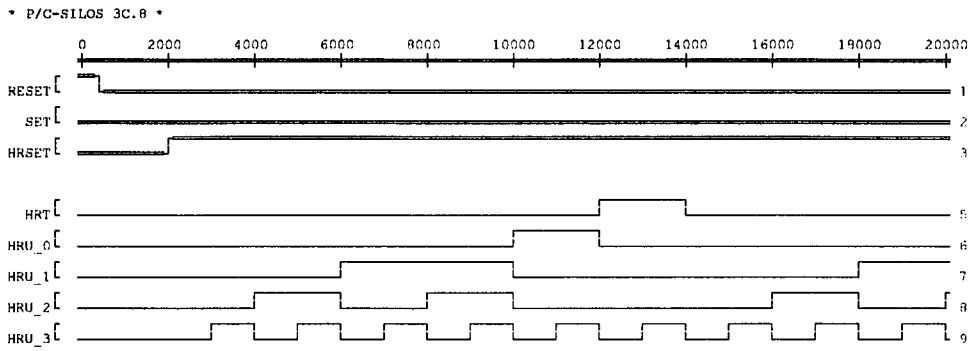


(i) SET (Part 1/2).

Figure 6.9: Continued.



(j) SET (Part 2/2).



(k) HRSET.

Figure 6.9: Continued.

to an external output port. Thus, two sets of statements are added to the original description as shown below:

```

MODULE: TODCLK.

      :

      EXOUTPUTS:OUT[4];PSCR[7];MINU[4];MINT[3];HRU[4];HRT.

      :

BODY SEQUENCE:CLOCK.

      :

CONTROLRESET(RESET)/(1);

PSCR = PRESCR;
MINU = MINUR;
MINT = MINTENR;
HRU  = HRUR;
HRT  = HRTR.

END.

CLU      : INC(X){N}.

      :

BODY

      :

END.

```

The **PSCR_0** to **_6**, **MINU_0** to **_3**, **MINT_0** to **_2**, **HRU_0** to **_3** and **HRT** will represent the **PRESCR[0-6]**, **MINUR[0-3]**, **MINTENR[0-2]**, **HRUR[0-3]** and **HRTR** respectively.

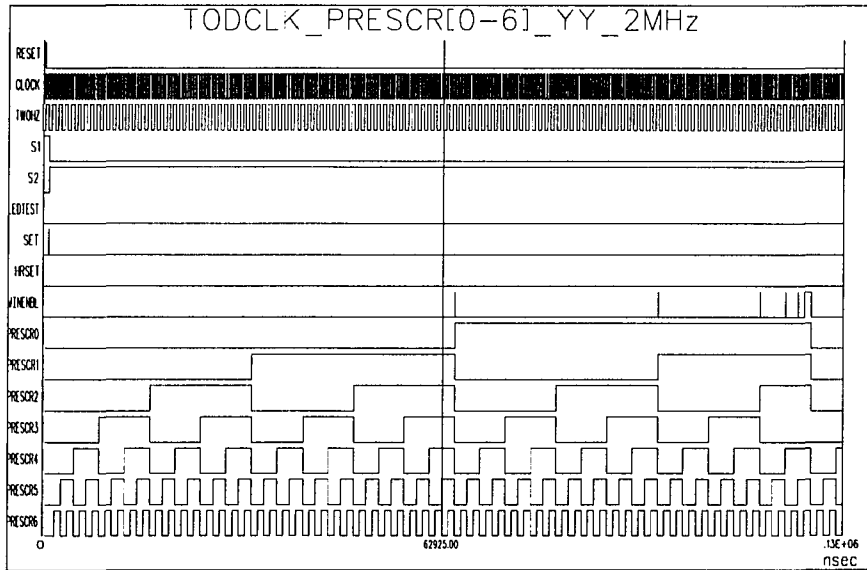
6.2.4 Test Using COMPASS Tools™ System

Simulation of TODCLK using COMPASS Tools™ system can be done by following the process flow shown in Figure 5.5. The **sim** or mixed-mode simulator of the system will be used. Table 6.6 summarizes the settings used by ANFT and COMPASS Tools™. The simulation results are shown in Figure 6.10 which proves the correctness of ANFT. In order to trace the contents of the counters, each of their Q output pin is manually connected to an external output port after the ANFT output file has been converted to the COMPASS's own netlist file. Therefore, the **PRESCR0** to **6**, **MINUR0** to **3**, **MINTENR0** to **2**, **HRUR0** to **3** and **HRTR** will represent the **PRESCR[0-6]**, **MINUR[0-3]**, **MINTENR[0-2]**, **HRUR[0-3]** and **HRTR** respectively. The **S1** and **S2** which stands for state 1 and 2 are also added manually.

Settings for Running ANFT		Settings for COMPASS Tools™	
Output Format	EDIF 2 0 0	CLOCK Signal	2MHz
Special cell mapping	Yes	TWOHZ Signal	1MHz
NAND-NAND implementation	No	Simulation Type	Timing Simulation
Memory DFF type	DFF		
Inversion Utilization	Default		
Pad generation	No		
Split the VDD/GND blocks	No		
All-tri-state-bus	No		
XOR Equivalent Logic	1		
Gate/Cell Library	cpss.lb		

Table 6.6: Settings for Running ANFT and COMPASS Tools™.

Compass Design Automation plot [gdf]tickpre by s3 on 13-May-95 at 1:47 A.M.



Compass Design Automation plot [gdf]tickmin by s3 on 13-May-95 at 1:46 A.M.

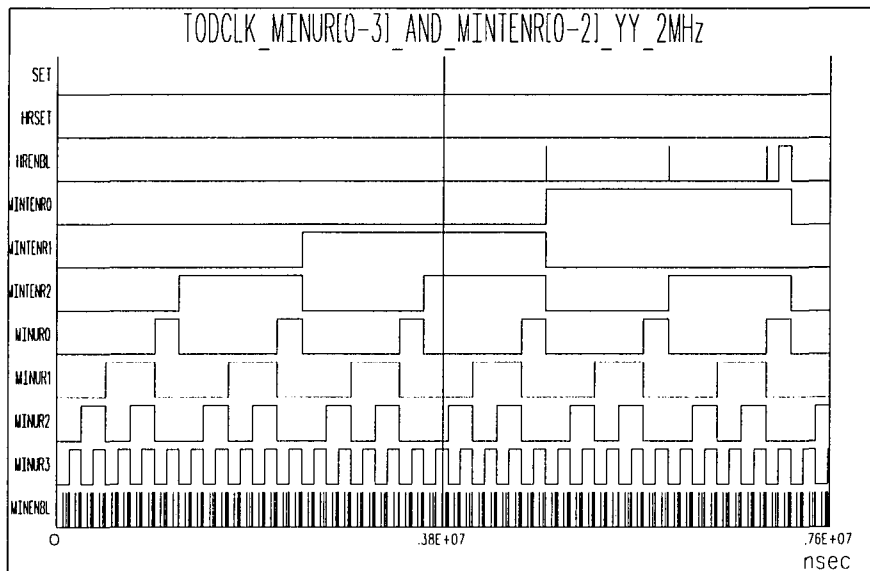
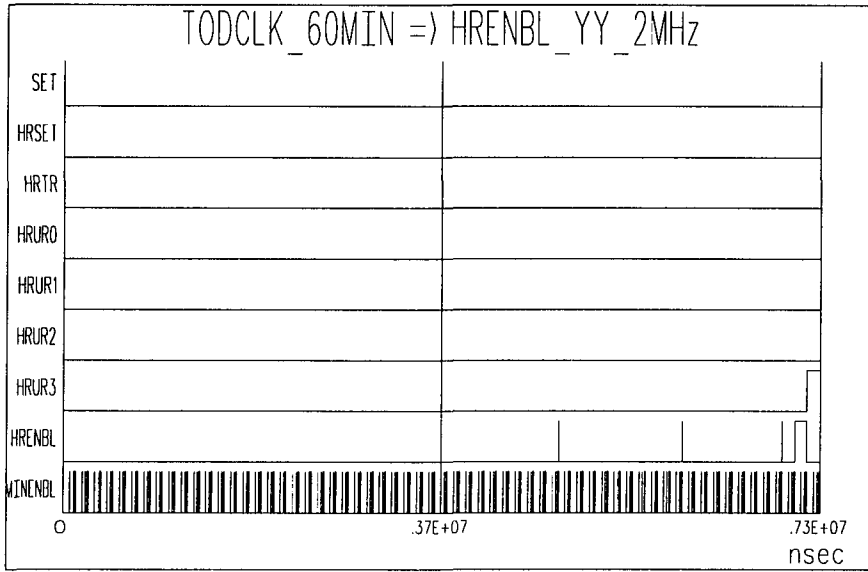


Figure 6.10: Simulation Results of TODCLK Using COMPASS Tools™.

Compass Design Automation plot [gdf]cklr by a3 on 13-May-95 at 1:42 A.M.



Compass Design Automation plot [gdf]cklr by a3 on 13-May-95 at 1:53 A.M.

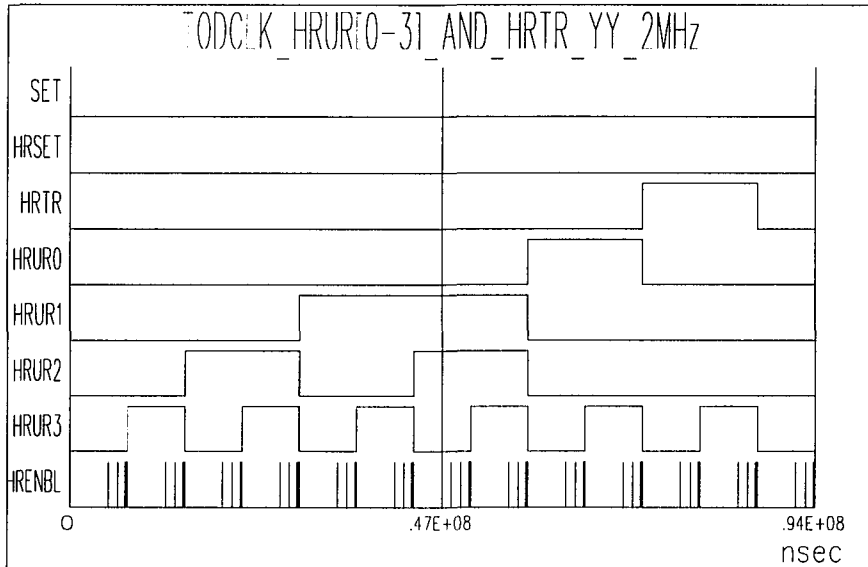
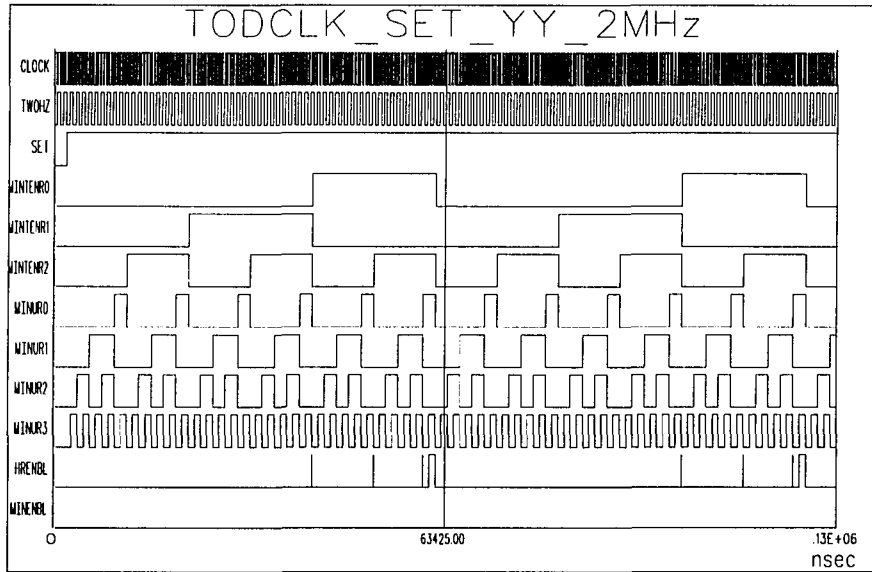


Figure 6.10: Continued.

Compass Design Automation plot [gdf]tckset by a3 on 13-May-95 at 1:49 A.M.



Compass Design Automation plot [gdf]tckset by a3 on 13-May-95 at 1:44 A.M.

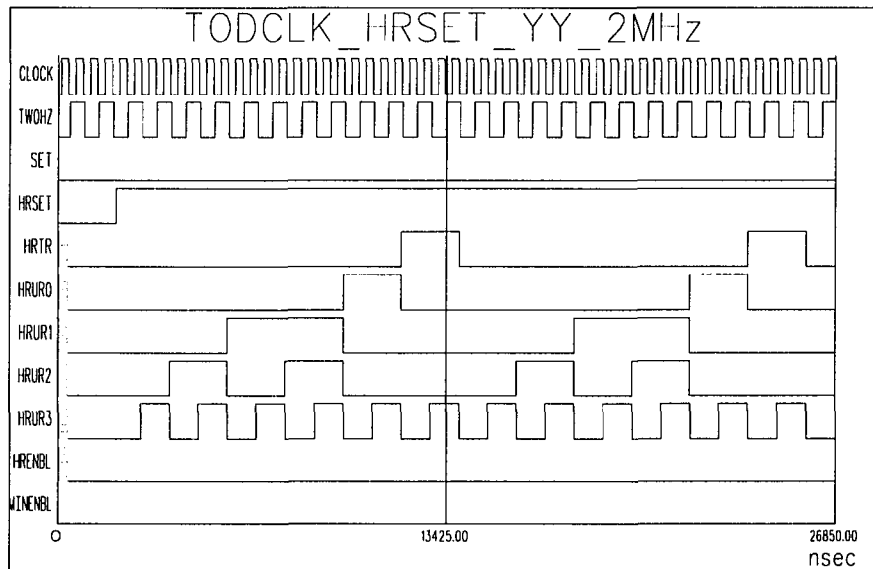


Figure 6.10: Continued.

6.3 Evaluation

In this section, fifteen AHPL benchmarks will serve as the test modules for evaluating the results generated by ANFT, AENT91 and AENT_TTL. They will be compiled by the HPCOM (stage01 and stage23x) with controlled clock and one-hot encoding (i.e. answer “Y” and “Y” to the two inquiries of stage23x.) Table 6.7 summarizes the characteristics of the benchmarks.

Benchmark	Function
ADD3AR	Three step adder with four 8-bit arguments.
ALU	16-bit ALU using carry look ahead adder.
CSR	Communication module (part 2).
DEMO	Sample AHPL description.
HWTEST	4-bit circular shift register module.
RDATAR	Communication module (part 1).
SEQSEL	Registers sequentially selecting strings and delivering most recent one.
SERCOM	Serial comparison module checking for duplication bits.
SHIFTR	4-bit shift register with parallel load.
STACK4	4-bit stack with a 4×4 RAM.
STEST1	4-state state transition test module.
SWA42	Four processor array (2 bits).
SWA410	Four processor array (10 bits).
SWA416	Four processor array (16 bits).
TODCLK	Time of day clock with LED test.

Table 6.7: Characteristics of AHPL Benchmarks.

For the comparison of ANFT and AENT91, the evaluation is based on the number of gates and number of gate inputs collected directly from the output files implemented in both the AND-OR-INV and NAND-NAND logic. The output files will then be compiled by the **ChipComp** tool of COMPASS Tools™ using 1.2μ **pvsc370d.lib**. After

the compilation processes, the transistor number, the layout area and the maximum delay time of the layouts will also be evaluated. These three data categories will be collected by allowing no optimization. For comparing the ANFT and AENT_TTL, only the first two numbers will be compared. Because both the AENT91 and AENT_TTL do not have fanout limit process, the non-inverting buffer if used will be excluded when counting the gate numbers and input numbers of ANFT outputs. In addition, since different XOR equivalent logics will result in different gate count, when comparing the NAND-NAND implementation, the ANFT results used will be the ones with less gate count or layout area.

6.3.1 Comparison of ANFT and AENT91 Results

Because the output of AENT91 can only be in EDIF 1 1 0 format, a utility program called **EDIF1122**, EDIF 1 1 0 to 2 0 0 format translator [17], will be used to convert the output files of AENT91 into equivalent EDIF 2 0 0 files. The option settings of ANFT and AENT91 are the same as those shown in Table 6.6 and Table 6.1 except that answers for the special gate mapping and NAND-NAND implementation will be exchanged for NAND-NAND evaluation. The results of AND-OR-INV and NAND-NAND implementation are listed in Table 6.8 and Table 6.9 respectively. The ratio of ANFT to AENT91 results for both types of implementations are shown in Table 6.10.

Table 6.10 shows that most of the ANFT outputs are better than the AENT91's in terms of all the items compared. For the AND-OR-INV implementation, most of the

AND-OR-INV Implementation						
Benchmark	ANFT Results					
	Gate No. (wo/ BUF)	Gate Input No. (wo/ BUF)	BUF No.	Tran- sistor No.	Area (μ^2)	MAX Delay (<i>ns</i>)
ADD3AR	993	2,161	6	7,047	2,031,916	46.94
ALU	366	816	3	2,768	715,725	25.94
CSR	164	437	5	1,884	434,991	15.97
DEMO	15	34	0	176	41,983	7.27
HWTEST	39	85	0	418	91,853	9.69
RDATAR	50	115	0	486	113,131	8.11
SEQSEL	61	133	0	817	187,328	11.03
SERCOM	96	210	0	1,350	299,519	10.14
SHIFTR	17	35	0	186	39,112	5.92
STACK4	137	312	0	1,806	422,204	10.75
STEST1	61	158	0	554	136,194	9.49
SWA42	236	498	4	1,952	517,514	20.00
SWA410	1,068	2,226	16	8,432	2,551,298	56.74
SWA416	1,692	3,522	24	13,288	4,184,420	82.40
TODCLK	128	291	1	1,437	341,613	17.39
AENT91 Results						
ADD3AR	995	2,163	0	7,023	1,979,783	46.14
ALU	369	819	0	2,760	721,963	20.95
CSR	186	459	0	1,902	494,204	19.01
DEMO	18	37	0	188	44,048	8.02
HWTEST	39	85	0	418	98,328	9.56
RDATAR	E	E	E	E	E	E
SEQSEL	62	134	0	821	193,359	11.14
SERCOM	119	233	0	1,436	305,791	12.15
SHIFTR	18	36	0	186	40,154	5.91
STACK4	148	323	0	1,836	441,454	11.66
STEST1	69	166	0	570	146,124	9.67
SWA42	237	500	0	1,940	458,745	18.71
SWA410	1,069	2,228	0	8,372	2,314,591	61.41
SWA416	1,693	3,567	0	13,196	3,922,599	97.40
TODCLK	149	312	0	1,487	334,528	15.58
E : Output file incorrect.						

Table 6.8: ANFT and AENT91 AND-OR-INV Implementation Results.

NAND-NAND Implementation							
Benchmark	ANFT Results						
	XOR Equ.	Gate No.	Gate Input No.	BUF	Transistor	Area	MAX Delay
	Logic	(wo/ BUF)	(wo/ BUF)	No.	No.	(μ^2)	(<i>ns</i>)
ADD3AR	1	1,009	2,179	6	5,878	2,024,304	38.39
ALU	2	579	1,173	3	2,604	955,511	24.45
CSR	1	209	482	5	1,734	478,623	14.86
DEMO	1	22	41	0	178	45,091	6.66
HWTEST	1	41	87	0	366	94,771	7.27
RDATAR	1	64	129	0	450	114,173	6.84
SEQSEL	2	79	160	0	800	176,062	11.54
SERCOM	1	145	291	0	1,326	336,135	10.82
SHIFTR	1	20	38	0	164	37,882	4.76
STACK4	1	170	349	0	1,726	429,726	10.84
STEST1	1	85	182	0	510	151,273	9.09
SWA42	2	260	540	1	1,660	469,257	14.92
SWA410	2	1,220	2,492	13	7,212	2,909,611	50.42
SWA416	2	1,940	3,956	24	11,384	5,782,967	76.24
TODCLK	1	246	439	1	1,420	406,153	17.11
AENT91 Results							
ADD3AR		1,010	2,181	0	5,858	2,006,840	40.60
ALU		579	1,179	0	2,616	920,588	18.08
CSR		212	499	0	1,778	518,753	18.81
DEMO		22	41	0	178	47,176	6.95
HWTEST		41	87	0	366	91,178	7.26
RDATAR		E	E	0	E	E	E
SEQSEL		84	165	0	810	189,413	11.43
SERCOM		161	323	0	1,390	352,758	11.89
SHIFTR		20	38	0	164	40,612	4.75
STACK4		178	359	0	1,744	445,284	11.30
STEST1		107	204	0	554	165,725	9.67
SWA42		261	542	0	1,660	486,497	15.31
SWA410		1,221	2,494	0	7,164	2,615,413	55.66
SWA416		1,941	4,001	0	11,292	4,365,459	79.67
TODCLK		252	465	0	1,480	410,465	15.58
E : Output file incorrect.							

Table 6.9: ANFT and AENT91 NAND-NAND Implementation Results.

Comparing the ANFT and AENT91 Results					
Benchmark	AND-OR-INV Implementation Ratio				
	Gate No. (wo/ BUF)	Gate Input No. (wo/ BUF)	Tran- sistor No.	Area (μ^2)	MAX Delay (<i>ns</i>)
ADD3AR	99.80%	99.91%	100.34%	102.63%	101.73 %
ALU	99.19%	99.63%	100.29%	99.14%	123.82 %
CSR	88.17%	95.21%	99.05%	88.02%	84.01 %
DEMO	83.33%	91.89%	93.62%	95.31%	90.65 %
HWTEST	100.00%	100.00%	100.00%	93.41%	101.36 %
RDATAR	N/A	N/A	N/A	N/A	N/A
SEQSEL	98.39%	99.25%	99.51%	96.88%	99.01 %
SERCOM	80.67%	90.13%	94.01%	97.95%	83.46 %
SHIFTR	94.44%	97.22%	100.00%	97.40%	100.17 %
STACK4	92.57%	96.59%	98.37%	95.64%	92.20 %
STEST1	88.41%	95.18%	97.19%	93.20%	98.14 %
SWA42	99.58%	99.60%	100.62%	112.81%	106.89 %
SWA410	99.91%	99.91%	100.72%	110.23%	92.40 %
SWA416	99.94%	98.74%	100.70%	106.67%	84.60 %
TODCLK	85.91%	93.27%	96.64%	102.12%	111.62 %
NAND-NAND Implementation Ratio					
ADD3AR	99.90%	99.91%	100.34%	100.87%	94.56%
ALU	100.00%	99.49%	99.54%	103.79%	135.23%
CSR	98.58%	96.59%	97.53%	92.26%	79.00%
DEMO	100.00%	100.00%	100.00%	95.58%	95.83%
HWTEST	100.00%	100.00%	100.00%	103.94%	100.14%
RDATAR	N/A	N/A	N/A	N/A	N/A
SEQSEL	94.05%	96.97%	98.77%	92.95%	100.96%
SERCOM	90.06%	90.09%	95.40%	95.29%	91.00%
SHIFTR	100.00%	100.00%	100.00%	93.28%	100.21%
STACK4	95.51%	97.21%	98.97%	96.51%	95.93%
STEST1	79.44%	89.22%	92.06%	91.28%	94.00%
SWA42	99.62%	99.63%	100.00%	96.46%	97.45%
SWA410	99.92%	99.92%	100.67%	111.25%	90.59%
SWA416	99.95%	98.88%	100.81%	132.47%	95.69%
TODCLK	97.62%	94.41%	95.95%	98.95%	109.82%

Table 6.10: Comparing the ANFT and AENT91 Results.

ratios are between 95% to 100% which mean the improvements are not much. This is reasonable because the ANFT has to accept the stage 2 outputs and manipulates the netlist according to the available gates or cells provided in the library. The degree of freedom for ANFT is limited to optimizing the implementation of given objects locally. There is no way for ANFT to redo the logic synthesis and undo broaden ranging redundancy. Consequently, most of the improvements result from the flexible gate fitting, special gate mapping processes and redundant input and/or object deletion. The best result is the case of SERCOM where the number of gates and gate inputs are 80.67% and 90.13% of those generated by AENT91. The reasons are that ANFT is able to utilize AND, OR, NAND and NOR gates with more than four input pins provided by the **cpss.lib** (from **pvc370d.lib**.) It also maps 17 “I(2&)*”, 4 “2&(4&)*”, 1 “2^(2&)*” and 1 “2^(3&)*” logics into **nd02d1**, **an05d1**, **nd03d1** and **nd04d1** gates respectively. As mentioned, when comparing the gate numbers, the number of non-inverting buffers are excluded from the ANFT results. However, when comparing the the layout areas, the ANFT results contain the non-inverting buffers. Therefore, the chip layout results from COMPASS Tools™ are not directly reflecting the results of reduced number of gates. For example, in the TODCLK case, the gate number ratio is 85.91% but the area and maximum delay time ratios are 102.12% and 111.62%. The same situation also happens to the ADD3AR, ALU and SWA42 where extra non-inverting BUFFERS are used and the different orders of gates provided

to the COMPASS Tools™ cause different placement decisions. It is possible that extending optimization time for layout would eliminate the inconsistency.

For the NAND-NAND implementation, most of the improvements comes from the deletion of unnecessary inverter pairs as well as unnecessary inverter trees in the ANFT. ANFT has two ways to flatten the XOR; therefore, the better results can be selected. Other reasons are again the flexible gate fitting process and redundancy adjustment. The best case is the STTEST1 where the ratio of the five categories compared are 79.44%, 89.22%, 92.06%, 91.28% and 94.00% respectively. However, in the cases of ALU, the ratio of layout area and maximum delay time are 103.79% and 135.23% while the ratio of gate numbers and gate input numbers are 100.00% and 99.49%. This phenomenon, again, can only result from the different placement decisions made by the COMPASS Tools™.

As for the maximum delay time in both types of implementation, most of the ANFT results are better or equal to the AENT91 results. One may notice that in some cases, the area ratios are higher than 100.00% but the maximum delay time ratios are less than 100.00%. For example, in the SWA416 case, even though the area ratios are 106.67% and 132.47% for both types of implementation, the maximum delay time ratios are 84.60% and 95.69% respectively. Because the delay time ratios are similar to the above values when stray delay is not included in the delay time estimation. We conclude the improvement of the delay time result directly from the fanout limit process used by ANFT rather than the different placement decision made

ANFT and AENT_TTL AND-OR-INV Implementation Results							
Benchmark	ANFT			AENT_TTL		ANFT/AENT_TTL Ratio	
	Gate No. (wo/ BUF)	Gate Input No. (wo/ BUF)	BUF No.	Gate No.	Gate Input No.	Gate No.	Gate Input No.
	ADD3AR	1,183	2,445	10	1,296	2,558	91.28%
ALU	426	876	11	426	876	100.00%	100.00%
CSR	215	504	9	221	510	97.29%	98.82%
DEMO	16	39	0	20	43	80.00%	90.70%
HWTEST	42	96	0	43	97	97.67%	98.97%
RDATAR	60	129	0	61	130	98.36%	99.23%
SEQSEL	70	174	1	71	175	98.59%	99.43%
SERCOM	102	272	1	119	289	85.71%	94.12%
SHIFTR	18	44	0	18	44	100.00%	100.00%
STACK4	159	398	3	168	407	94.64%	97.79%
STEST1	72	169	0	79	176	91.14%	96.02%
SWA42	277	577	5	279	580	98.28%	99.48%
SWA410	1,237	2,561	28	1,239	2,564	99.84%	99.88%
SWA416	1,957	4,049	49	1,959	4,052	99.90%	99.93%
TODCLK	140	345	2	171	383	81.87%	90.08%

Table 6.11: ANFT and AENT_TTL AND-OR-INV Implementation Results and Comparison.

by COMPASS ToolsTM. One can refer to the Appendix G and [18] for detail critical path and delay time estimation information.

6.3.2 Comparison of ANFT and AENT_TTL Results

Table 6.11 and Table 6.12 summarizes the results of comparing the ANFT and AENT_TTL outputs in both AND-OR-INV and NAND-NAND logic. The translation settings of ANFT and AENT_TTL are the same as shown in Table 6.2.

ANFT and AENT_TTL NAND-NAND Implementation Results								
Benchmark	ANFT				AENT_TTL		ANFT/AENT_TTL Ratio	
	XOR Equ. Logic	Gate No. (wo/ BUF)	Gate Input No. (wo/ BUF)	BUF No.	Gate No.	Gate Input No.	Gate No.	Gate Input No.
ADD3AR	1	1,010	2,273	10	1,127	2,392	89.53%	95.03%
ALU	2	583	1,177	11	579	1,179	100.69%	99.83%
CSR	1	219	508	7	250	553	87.60%	91.86%
DEMO	1	22	45	0	25	48	88.00%	93.75%
HWTEST	1	41	95	0	46	100	89.13%	95.00%
RDATAR	1	64	133	0	89	158	71.91%	84.18%
SEQSEL	2	79	192	1	86	199	91.86%	96.48%
SERCOM	1	145	347	1	162	380	89.51%	91.32%
SHIFTR	1	20	46	0	20	46	100.00%	100.00%
STACK4	1	170	413	2	202	447	84.16%	92.39%
STEST1	1	85	182	0	107	204	79.44%	89.22%
SWA42	2	260	578	2	288	607	90.28%	95.22%
SWA410	2	1,220	2,658	25	1,248	2,687	97.76%	98.92%
SWA416	2	1,940	4,218	45	1,968	4,247	98.58%	99.32%
TODCLK	1	250	485	2	U	U	N/A	N/A
U : Unable to obtain the output file.								

Table 6.12: ANFT and AENT_TTL NAND-NAND Implementation Results and Comparison.

For the AND-OR-INV implementation, almost all the ANFT results are better or equal to the AENT_TTL's results. The reasons are that AENT_TTL does not utilize the \bar{Q} output pins of DFFs and it can not combine cascaded gates into one equivalent gate. For example, in the DEMO case, ANFT is able to remove 2 unnecessary INVs by utilizing the \bar{Q} and map 2 "2&(2&,*)" logics into 2 3-input AND gates. The results of the SHIFTR case from both of the translators are the same because there is no

inverter used and no gates can be further combined or mapped when using the `ttl.lib` library.

For the NAND-NAND implementation, the improvements are slightly more than for the AND-OR-INV implementation. The main reason is that `AENT_TTL` never removes any inverter pairs or trees resulting from the NAND-NAND conversion. In the `ADD3AR` case, the output form `ANFT` contains 125 inverters less than the `AENT_TTL` result. The different XOR flattening options also provide `ANFT` ways to generate outputs with less gates. For example, if selecting the 1-OR-2-AND-2-INV equivalent logic, the gate count of the `ANFT` translated `SWA416` output will be 1999 (worse than the result, 1968, from `AENT_TTL`) while selecting the 4-NAND equivalent logic, the result is 1940. The reason is that the 4-NAND logic contains one gate less than the 1-OR-2-AND-2-INV logic. However, using the 1-OR-2-AND-2-INV or 3-NAND-2-INV (converted from 1-OR-2-AND-2-INV when NAND-NAND implementation is used) equivalent logic can also result in better outputs if the two inverters can be removed by the invertability utilization process. This is exactly what happened in the `TODCLK` case. Overall, we can conclude that `ANFT` produces better results than `AENT_TTL`.

CHAPTER 7

Conclusion

The objectives of this thesis was to expand the function and capability of the stage 3 of HPCOM. In Chapter 2, the weakness and limitation of AENT91 was discussed. We also demonstrated what were considered when defining the new stage 3 and how they were implemented. The evaluation presented in Section 6.3 showed that ANFT generates better results than AENT91 and AENT_TTL. The test results shown in Chapter 6 also indicated that the following goals were achieved:

- New features of stage 2 of HPCOM are fully utilized.
- Components in a library can be fully utilized.
- Additional optimization and fine tuning options for implementation are supported.
- Output files can be generated in various formats and used by different CAD tool systems.

With the completion of ANFT, designs can be described, implemented and ported to compatible systems for further applications.

7.1 Final Thought

ANFT has overcome most of the inconveniences and limits encountered when using the AENT91 and AENT_TTL. Currently the most critical problem for using the AHPL synthesis system is the memory space limit since most of the the tools were developed for running on MS-DOSTM environment. Although ANFT can work under UNIXTM systems, its source files are still generated from the PC version tools. Thus, the size of designs are still limited under UNIXTM environment.

One way to resolve the difficulty is to utilize the enhanced hierarchical design capabilities, i.e., using BLACK BOX elements. A BLACK BOX unit can cover at least two ordinary elements; hence, reduce the memory consumption. Currently, BLACK BOX elements must be declared as combinational logic units and they will be considered as undefined CLUs. In the course of this project, it was discovered that a BLACK BOX can be a special unit that contains memory elements and serves as data or control sources to the parent module. For example, it could be a cyclic clock edge counter. Should there be any timing problem using single clock phase, double phase clocking scheme is achievable. Declaring BLACK BOXes containing memory element(s) as CLUNITS may confuse the designer. This is also a side effect caused by the multiple stages approach of designing a tool system. Unless the module of different stages are designed by the same group of designers, no one can perfectly predefine what will really be required in the future. Therefore, the differences of

specifications between consecutive stages may cause tool designers to spend much effort in rebuilding and recombining them.

The AHPL synthesis system is still suited for the educational application. Each stage of the system is transparent to the designer. Evaluations of the results from every stage can provide accurate grasp of the synthesis processes and the effects.

7.2 Future Work

As mentioned, the limit of memory space faced by the AHPL synthesis system can be avoided using BLACK BOXes. On the second thought, porting the whole system to environments such as OS/2TM, Linux or UNIXTM would be necessary if large designs are to be accommodated. In these systems, memory usage over 640KB can be easily achieved without the burden of dealing with extra memory management programs.

As for ANFT, several areas can still be modified to strengthen the functions and capabilities:

- The cost criteria used is very primitive. The cost of area and performance is based on counting the number of total gates used and the level of gates passed. In real vendor-specific libraries, gates will have different areas and delay times. Treating their cost contributions equally will not generate optimal results. If optimizations are needed, detail area and delay information of all

gates must be provided individually. New cost functions are also needed for accurate calculations.

- The mapping process of module MAP can only convert 2-level combinational logics into special gates. Although multilevel logics can be invoked as BLACK BOXes, a new mapping scheme is still necessary in order to maximally utilize the components in the library.
- Currently, the output file can only be in EDIF 1 1 0, EDIF 2 0 0 and Xilinx™ XNF netlist format. Additional output file generators for new formats such as EDIF 3 0 0 can be derived for future needs.
- In order to declare and use CLUNITs and BLACK BOXes properly, a new keyword “BLACKBOX” for declaring BLACK BOX elements is needed.

These subjects will be the main targets for revised ANFT to achieve. The ultimate goal would be to generate designs optimal in both area and performance.

Appendix A

ANFT Libraries

(1) Contents of the **gtsm.lib**:

1 16 2 VDD GND

```

DEFELEM:          9
AND      3      4 3 2
          AND4  SCHEMLB1_LIB 4 1 1    1 2 3 4      5      7 6
          AND3  SCHEMLB1_LIB 3 1 1    1 2 3        4      6 5
          AND2  SCHEMLB1_LIB 2 1 1    1 2          3      5 4
OR       3      4 3 2
          OR4   SCHEMLB1_LIB 4 1 1    1 2 3 4      5      7 6
          OR3   SCHEMLB1_LIB 3 1 1    1 2 3        4      6 5
          OR2   SCHEMLB1_LIB 2 1 1    1 2          3      5 4
NAND    4      8 4 3 2
          NAND8 SCHEMLB1_LIB 8 1 1    1 2 3 4 5 6 7 8  9      11 10
          NAND4 SCHEMLB1_LIB 4 1 1    1 2 3 4      5      7 6
          NAND3 SCHEMLB1_LIB 3 1 1    1 2 3        4      6 5
          NAND2 SCHEMLB1_LIB 2 1 1    1 2          3      5 4
NOR     2      3 2
          NOR3  SCHEMLB1_LIB 3 1 1    1 2 3        4      6 5
          NOR2  SCHEMLB1_LIB 2 1 1    1 2          3      5 4
INV     1      1
          INV   SCHEMLB2_LIB 1 1 1    1          2      4 3
BUF     1      1
          BUF   SCHEMLB1_LIB 1 1 1    1          2      4 3
XOR     1      2
          XOR2  SCHEMLB1_LIB 2 1 1    1 2          3      5 4
TRIBUF  1      2
          BUFZ  SCHEMLB1_LIB 2 1 1    1 -3        2      5 4
BLKBOX  1
          MUX2  SCHEMLB1_LIB 3 1 1    1 2 3        4      6 5

DFFTYPE:          4
DFF     DFF     SCHEMLB2_LIB 3 2 1    1 4          !      2 3 6 5
DFFPC  DFFPC  SCHEMLB2_LIB 5 2 1    1 6 -3 -2 !    4 5 8 7
DFFP   DFFP   SCHEMLB2_LIB 4 2 1    1 2 -3      !    4 5 7 6
DFFC   DFFC   SCHEMLB2_LIB 4 2 1    1 3      -2 !    4 5 7 6

IOPAD:          4
BLNKPAD !
IPAD   IPADC   SCHEMLB2_LIB 1 2 1    1          2 3 5 4

```

OPAD	OPAD	SCHEMLB2_LIB	1	1	1	2		1		4	3	
BPAD	BPADZH	SCHEMLB2_LIB	2	3	1	4	3	2	!	1	6	5

MAPPING: 47

I(2~)	= AND	3	2	1									
I(3~)	= AND	2	3	1									
I(4~)	= AND	1	4	1									
I(2#)	= OR	3	2	1									
I(3#)	= OR	2	3	1									
I(4#)	= OR	1	4	1									
I(2&)	= NAND	4	2	1									
I(3&)	= NAND	3	3	1									
I(4&)	= NAND	2	4	1									
I(8&)	= NAND	1	8	1									
I(2+)	= NOR	2	2	1									
I(3+)	= NOR	1	3	1									
2&(2&,2&)	= AND	1	4	1									
2&(2&,*)	= AND	2	3	1									
2&(3&,*)	= AND	1	4	1									
3&(2&,* ,*)	= AND	1	4	1									
2+(2+,2+)	= OR	1	4	1									
2+(2+,*)	= OR	2	3	1									
2+(3+,*)	= OR	1	4	1									
3+(2+,* ,*)	= OR	1	4	1									
4~(2&,2&,2&,2&)	= NAND	1	8	1									
3~(2&,2&,4&)	= NAND	1	8	1									
3~(2&,3&,3&)	= NAND	1	8	1									
4~(2&,2&,3&,*)	= NAND	1	8	1									
5~(2&,2&,2&,* ,*)	= NAND	1	8	1									
2~(2&,2&)	= NAND	2	4	1									
2~(2&,6&)	= NAND	1	8	1									
2~(3&,5&)	= NAND	1	8	1									
2~(4&,4&)	= NAND	1	8	1									
3~(2&,5&,*)	= NAND	1	8	1									
3~(3&,4&,*)	= NAND	1	8	1									
4~(2&,4&,* ,*)	= NAND	1	8	1									
4~(3&,3&,* ,*)	= NAND	1	8	1									
5~(2&,3&,* ,* ,*)	= NAND	1	8	1									
6~(2&,2&,* ,* ,* ,*)	= NAND	1	8	1									
2~(2&,*)	= NAND	3	3	1									
2~(3&,*)	= NAND	2	4	1									
3~(2&,* ,*)	= NAND	2	4	1									
2~(7&,*)	= NAND	1	8	1									
3~(6&,* ,*)	= NAND	1	8	1									
4~(5&,* ,* ,*)	= NAND	1	8	1									
5~(4&,* ,* ,* ,*)	= NAND	1	8	1									
6~(3&,* ,* ,* ,* ,*)	= NAND	1	8	1									
7~(2&,* ,* ,* ,* ,* ,*)	= NAND	1	8	1									
2#(2+,*)	= NOR	1	3	1									
2+(2&,2&)	AOI4C	SCHEMLB2_LIB	4	2	1	1	2	3	4	6	5	8	7
2+(2&,*)	AOI3C	SCHEMLB2_LIB	3	2	1	1	2	3		5	4	7	6


```

I(8&)           = NAND  1 8 1
I(2+)          = NOR   2 2 1
I(3+)          = NOR   1 3 1
2&(2&,2&)      = AND   1 4 1
2&(2&,* )      = AND   2 3 1
2&(3&,* )      = AND   1 4 1
3&(2&,* ,* )   = AND   1 4 1
4~(2&,2&,2&,2&) = NAND  1 8 1
3~(2&,2&,4&)   = NAND  1 8 1
3~(2&,3&,3&)   = NAND  1 8 1
4~(2&,2&,3&,* ) = NAND  1 8 1
5~(2&,2&,2&,* ,* ) = NAND  1 8 1
2~(2&,2&)      = NAND  2 4 1
2~(2&,6&)      = NAND  1 8 1
2~(3&,5&)      = NAND  1 8 1
2~(4&,4&)      = NAND  1 8 1
3~(2&,5&,* )   = NAND  1 8 1
3~(3&,4&,* )   = NAND  1 8 1
4~(2&,4&,* ,* ) = NAND  1 8 1
4~(3&,3&,* ,* ) = NAND  1 8 1
5~(2&,3&,* ,* ,* ) = NAND  1 8 1
6~(2&,2&,* ,* ,* ,* ) = NAND  1 8 1
2~(2&,* )      = NAND  3 3 1
2~(3&,* )      = NAND  2 4 1
3~(2&,* ,* )   = NAND  2 4 1
2~(7&,* )      = NAND  1 8 1
3~(6&,* ,* )   = NAND  1 8 1
4~(5&,* ,* ,* ) = NAND  1 8 1
5~(4&,* ,* ,* ,* ) = NAND  1 8 1
6~(3&,* ,* ,* ,* ,* ) = NAND  1 8 1
7~(2&,* ,* ,* ,* ,* ,* ) = NAND  1 8 1
2#(2+,* )      = NOR   1 3 1
4#(2&,3&,3&,2&) X_74LS54 TTL_LIB 10 1 1 1 2 3 4 5 9 10 11 12 13 6 14 7
4#(2&,2&,2&,2&) X_7454 TTL_LIB 8 1 1 1 13 2 3 4 5 9 10 8 14 7
2#(4&,4&)      X_74LS55 TTL_LIB 8 1 1 1 2 3 4 10 11 12 13 8 14 7
2#(2&,2&)      X_7451 TTL_LIB 4 1 2 2 3 4 5 1 13 10 9 6 8 14 7

```

```

DRIVECAP:      1
X_74AS34      1 1 X_74AS34

```

(3) Contents of the pvsc370d.lb:

```
0 16 2 VDD VSS
```

```

DEFELEM:      9
AND           7 8 7 6 5 4 3 2
an08d1       pvsc370d 8 1 1 A1 A2 A3 A4 A5 A6 A7 A8 Z
an07d1       pvsc370d 7 1 1 A1 A2 A3 A4 A5 A6 A7 Z
an06d1       pvsc370d 6 1 1 A1 A2 A3 A4 A5 A6 Z

```

	an05d1	pvsc370d	5 1 1	A1 A2 A3 A4 A5	Z
	an04d1	pvsc370d	4 1 1	A1 A2 A3 A4	Z
	an03d1	pvsc370d	3 1 1	A1 A2 A3	Z
	an02d1	pvsc370d	2 1 1	A1 A2	Z
OR	7		8 7 6 5 4 3 2		
	or08d1	pvsc370d	8 1 1	A1 A2 A3 A4 A5 A6 A7 A8	Z
	or07d1	pvsc370d	7 1 1	A1 A2 A3 A4 A5 A6 A7	Z
	or06d1	pvsc370d	6 1 1	A1 A2 A3 A4 A5 A6	Z
	or05d1	pvsc370d	5 1 1	A1 A2 A3 A4 A5	Z
	or04d1	pvsc370d	4 1 1	A1 A2 A3 A4	Z
	or03d1	pvsc370d	3 1 1	A1 A2 A3	Z
	or02d1	pvsc370d	2 1 1	A1 A2	Z
NAND	7		8 7 6 5 4 3 2		
	nd08d1	pvsc370d	8 1 1	A1 A2 A3 A4 A5 A6 A7 A8	ZN
	nd07d1	pvsc370d	7 1 1	A1 A2 A3 A4 A5 A6 A7	ZN
	nd06d1	pvsc370d	6 1 1	A1 A2 A3 A4 A5 A6	ZN
	nd05d1	pvsc370d	5 1 1	A1 A2 A3 A4 A5	ZN
	nd04d1	pvsc370d	4 1 1	A1 A2 A3 A4	ZN
	nd03d1	pvsc370d	3 1 1	A1 A2 A3	ZN
	nd02d1	pvsc370d	2 1 1	A1 A2	ZN
NOR	7		8 7 6 5 4 3 2		
	nr08d1	pvsc370d	8 1 1	A1 A2 A3 A4 A5 A6 A7 A8	ZN
	nr07d1	pvsc370d	7 1 1	A1 A2 A3 A4 A5 A6 A7	ZN
	nr06d1	pvsc370d	6 1 1	A1 A2 A3 A4 A5 A6	ZN
	nr05d1	pvsc370d	5 1 1	A1 A2 A3 A4 A5	ZN
	nr04d1	pvsc370d	4 1 1	A1 A2 A3 A4	ZN
	nr03d1	pvsc370d	3 1 1	A1 A2 A3	ZN
	nr02d1	pvsc370d	2 1 1	A1 A2	ZN
INV	1		1		
	in01d1	pvsc370d	1 1 1	I	ZN
BUF	1		1		
	ni01d1	pvsc370d	1 1 1	I	Z
XOR	1		2		
	xo02d1	pvsc370d	2 1 1	A1 A2	Z
TRIBUF	1		2		
	nt01d1	pvsc370d	2 1 1	I OE	Z
BLKBOX	13				
	ad01d1	pvsc370d	3 2 1	A B CI	S CO
	ad02d1	pvsc370d	5 3 1	AO A1 B0 B1 CI	S0 S1 CO
	as01d1	pvsc370d	4 2 1	ADD A B CI	S CO
	as02d1	pvsc370d	7 3 1	ADDO ADD1 AO A1 B0 B1 CI	S0 S1 CO
	dc24d1	pvsc370d	2 4 1	AO A1	ZON Z1N Z2N Z3N
	dc38d1	pvsc370d	3 8 1	AO A1 A2 ZON Z1N Z2N Z3N	Z4N Z5N Z6N Z7N
	de24d1	pvsc370d	3 4 1	AO A1 EN	ZON Z1N Z2N Z3N
	fn02d1	pvsc370d	3 1 1	A B C	ZN
	mx21d1	pvsc370d	3 1 1	IO I1 S	Z
	mx41d1	pvsc370d	6 1 1	IO I1 I2 I3 S0 S1	Z
	mx81d1	pvsc370d	11 1 1	IO I1 I2 I3 I4 I5 I6 I7 S0 S1 S2	Z
	scbtnb	pvsc370d	4 3 1	CI CP SDN CDN	CO Q QN
	scctnb	pvsc370d	3 3 1	CI CP CDN	CO Q QN

```

DFFTYPE:      4
DFF   dfntnb  pvsc370d 3 2 1  D CP           !      Q QN
DFFPC dfbntb  pvsc370d 5 2 1  D CP -SDN -CDN !      Q QN
DFFP  dfptnb  pvsc370d 4 2 1  D CP -SDN     !      Q QN
DFFC  dfctnb  pvsc370d 4 2 1  D CP         -CDN !      Q QN

```

```
IOPAD:      0
```

```

MAPPING:      245
I(20)  xn02d1  pvsc370d 2 1 1  A1 A2      ZN
I(2~)  = AND   7 2 1
I(3~)  = AND   6 3 1
I(4~)  = AND   5 4 1
I(5~)  = AND   4 5 1
I(6~)  = AND   3 6 1
I(7~)  = AND   2 7 1
I(8~)  = AND   1 8 1
I(2#)  = OR    7 2 1
I(3#)  = OR    6 3 1
I(4#)  = OR    5 4 1
I(5#)  = OR    4 5 1
I(6#)  = OR    3 6 1
I(7#)  = OR    2 7 1
I(8#)  = OR    1 8 1
I(2&)  = NAND  7 2 1
I(3&)  = NAND  6 3 1
I(4&)  = NAND  5 4 1
I(5&)  = NAND  4 5 1
I(6&)  = NAND  3 6 1
I(7&)  = NAND  2 7 1
I(8&)  = NAND  1 8 1
I(2+)  = NOR   7 2 1
I(3+)  = NOR   6 3 1
I(4+)  = NOR   5 4 1
I(5+)  = NOR   4 5 1
I(6+)  = NOR   3 6 1
I(7+)  = NOR   2 7 1
I(8+)  = NOR   1 8 1
4&(2&,2&,2&,2&) = AND  1 8 1
3&(2&,2&,2&)   = AND  3 6 1
3&(2&,2&,3&)   = AND  2 7 1
4&(2&,2&,2&,* ) = AND  2 7 1
3&(2&,2&,4&)   = AND  1 8 1
3&(2&,3&,3&)   = AND  1 8 1
4&(2&,2&,3&,* ) = AND  1 8 1
5&(2&,2&,2&,* ,*) = AND  1 8 1
2&(2&,2&)      = AND  5 4 1
2&(2&,3&)      = AND  4 5 1
3&(2&,2&,* )   = AND  4 5 1
2&(2&,4&)      = AND  3 6 1
2&(3&,3&)      = AND  3 6 1

```

3&(2&,3&,*)	= AND	3 6 1
4&(2&,2&,* ,*)	= AND	3 6 1
2&(2&,5&)	= AND	2 7 1
2&(3&,4&)	= AND	2 7 1
3&(2&,4&,*)	= AND	2 7 1
3&(3&,3&,*)	= AND	2 7 1
4&(2&,3&,* ,*)	= AND	2 7 1
5&(2&,2&,* ,* ,*)	= AND	2 7 1
2&(2&,6&)	= AND	1 8 1
2&(3&,5&)	= AND	1 8 1
2&(4&,4&)	= AND	1 8 1
3&(2&,5&,*)	= AND	1 8 1
3&(3&,4&,*)	= AND	1 8 1
4&(2&,4&,* ,*)	= AND	1 8 1
4&(3&,3&,* ,*)	= AND	1 8 1
5&(2&,3&,* ,* ,*)	= AND	1 8 1
6&(2&,2&,* ,* ,* ,*)	= AND	1 8 1
2&(2&,*)	= AND	6 3 1
2&(3&,*)	= AND	5 4 1
3&(2&,* ,*)	= AND	5 4 1
2&(4&,*)	= AND	4 5 1
3&(3&,* ,*)	= AND	4 5 1
4&(2&,* ,* ,*)	= AND	4 5 1
2&(5&,*)	= AND	3 6 1
3&(4&,* ,*)	= AND	3 6 1
4&(3&,* ,* ,*)	= AND	3 6 1
5&(2&,* ,* ,* ,*)	= AND	3 6 1
2&(6&,*)	= AND	2 7 1
3&(5&,* ,*)	= AND	2 7 1
4&(4&,* ,* ,*)	= AND	2 7 1
5&(3&,* ,* ,* ,*)	= AND	2 7 1
6&(2&,* ,* ,* ,* ,*)	= AND	2 7 1
2&(7&,*)	= AND	1 8 1
3&(6&,* ,*)	= AND	1 8 1
4&(5&,* ,* ,*)	= AND	1 8 1
5&(4&,* ,* ,* ,*)	= AND	1 8 1
6&(3&,* ,* ,* ,* ,*)	= AND	1 8 1
7&(2&,* ,* ,* ,* ,* ,*)	= AND	1 8 1
4+(2+,2+,2+,2+)	= OR	1 8 1
3+(2+,2+,2+)	= OR	3 6 1
3+(2+,2+,3+)	= OR	2 7 1
4+(2+,2+,2+,*)	= OR	2 7 1
3+(2+,2+,4+)	= OR	1 8 1
3+(2+,3+,3+)	= OR	1 8 1
4+(2+,2+,3+,*)	= OR	1 8 1
5+(2+,2+,2+,* ,*)	= OR	1 8 1
2+(2+,2+)	= OR	5 4 1
2+(2+,3+)	= OR	4 5 1
3+(2+,2+,*)	= OR	4 5 1
2+(2+,4+)	= OR	3 6 1
2+(3+,3+)	= OR	3 6 1

3+(2+,3+,*)	= OR	3 6 1
4+(2+,2+,* ,*)	= OR	3 6 1
2+(2+,5+)	= OR	2 7 1
2+(3+,4+)	= OR	2 7 1
3+(2+,4+,*)	= OR	2 7 1
3+(3+,3+,*)	= OR	2 7 1
4+(2+,3+,* ,*)	= OR	2 7 1
5+(2+,2+,* ,* ,*)	= OR	2 7 1
2+(2+,6+)	= OR	1 8 1
2+(3+,5+)	= OR	1 8 1
2+(4+,4+)	= OR	1 8 1
3+(2+,5+,*)	= OR	1 8 1
3+(3+,4+,*)	= OR	1 8 1
4+(2+,4+,* ,*)	= OR	1 8 1
4+(3+,3+,* ,*)	= OR	1 8 1
5+(2+,3+,* ,* ,*)	= OR	1 8 1
6+(2+,2+,* ,* ,* ,*)	= OR	1 8 1
2+(2+,*)	= OR	6 3 1
2+(3+,*)	= OR	5 4 1
3+(2+,* ,*)	= OR	5 4 1
2+(4+,*)	= OR	4 5 1
3+(3+,* ,*)	= OR	4 5 1
4+(2+,* ,* ,*)	= OR	4 5 1
2+(5+,*)	= OR	3 6 1
3+(4+,* ,*)	= OR	3 6 1
4+(3+,* ,* ,*)	= OR	3 6 1
5+(2+,* ,* ,* ,*)	= OR	3 6 1
2+(6+,*)	= OR	2 7 1
3+(5+,* ,*)	= OR	2 7 1
4+(4+,* ,* ,*)	= OR	2 7 1
5+(3+,* ,* ,* ,*)	= OR	2 7 1
6+(2+,* ,* ,* ,* ,*)	= OR	2 7 1
2+(7+,*)	= OR	1 8 1
3+(6+,* ,*)	= OR	1 8 1
4+(5+,* ,* ,*)	= OR	1 8 1
5+(4+,* ,* ,* ,*)	= OR	1 8 1
6+(3+,* ,* ,* ,* ,*)	= OR	1 8 1
7+(2+,* ,* ,* ,* ,* ,*)	= OR	1 8 1
4 ⁻ (2&,2&,2&,2&)	= NAND	1 8 1
3 ⁻ (2&,2&,2&)	= NAND	3 6 1
3 ⁻ (2&,2&,3&)	= NAND	2 7 1
4 ⁻ (2&,2&,2&,*)	= NAND	2 7 1
3 ⁻ (2&,2&,4&)	= NAND	1 8 1
3 ⁻ (2&,3&,3&)	= NAND	1 8 1
4 ⁻ (2&,2&,3&,*)	= NAND	1 8 1
5 ⁻ (2&,2&,2&,* ,*)	= NAND	1 8 1
2 ⁻ (2&,2&)	= NAND	5 4 1
2 ⁻ (2&,3&)	= NAND	4 5 1
3 ⁻ (2&,2&,*)	= NAND	4 5 1
2 ⁻ (2&,4&)	= NAND	3 6 1
2 ⁻ (3&,3&)	= NAND	3 6 1

$3^-(2\&,3\&,*)$	= NAND	3 6 1
$4^-(2\&,2\&,* ,*)$	= NAND	3 6 1
$2^-(2\&,5\&)$	= NAND	2 7 1
$2^-(3\&,4\&)$	= NAND	2 7 1
$3^-(2\&,4\&,*)$	= NAND	2 7 1
$3^-(3\&,3\&,*)$	= NAND	2 7 1
$4^-(2\&,3\&,* ,*)$	= NAND	2 7 1
$5^-(2\&,2\&,* ,* ,*)$	= NAND	2 7 1
$2^-(2\&,6\&)$	= NAND	1 8 1
$2^-(3\&,5\&)$	= NAND	1 8 1
$2^-(4\&,4\&)$	= NAND	1 8 1
$3^-(2\&,5\&,*)$	= NAND	1 8 1
$3^-(3\&,4\&,*)$	= NAND	1 8 1
$4^-(2\&,4\&,* ,*)$	= NAND	1 8 1
$4^-(3\&,3\&,* ,*)$	= NAND	1 8 1
$5^-(2\&,3\&,* ,* ,*)$	= NAND	1 8 1
$6^-(2\&,2\&,* ,* ,* ,*)$	= NAND	1 8 1
$2^-(2\&,*)$	= NAND	6 3 1
$2^-(3\&,*)$	= NAND	5 4 1
$3^-(2\&,* ,*)$	= NAND	5 4 1
$2^-(4\&,*)$	= NAND	4 5 1
$3^-(3\&,* ,*)$	= NAND	4 5 1
$4^-(2\&,* ,* ,*)$	= NAND	4 5 1
$2^-(5\&,*)$	= NAND	3 6 1
$3^-(4\&,* ,*)$	= NAND	3 6 1
$4^-(3\&,* ,* ,*)$	= NAND	3 6 1
$5^-(2\&,* ,* ,* ,*)$	= NAND	3 6 1
$2^-(6\&,*)$	= NAND	2 7 1
$3^-(5\&,* ,*)$	= NAND	2 7 1
$4^-(4\&,* ,* ,*)$	= NAND	2 7 1
$5^-(3\&,* ,* ,* ,*)$	= NAND	2 7 1
$6^-(2\&,* ,* ,* ,* ,*)$	= NAND	2 7 1
$2^-(7\&,*)$	= NAND	1 8 1
$3^-(6\&,* ,*)$	= NAND	1 8 1
$4^-(5\&,* ,* ,*)$	= NAND	1 8 1
$5^-(4\&,* ,* ,* ,*)$	= NAND	1 8 1
$6^-(3\&,* ,* ,* ,* ,*)$	= NAND	1 8 1
$7^-(2\&,* ,* ,* ,* ,* ,*)$	= NAND	1 8 1
$4\#(2+,2+,2+,2+)$	= NOR	1 8 1
$3\#(2+,2+,2+)$	= NOR	3 6 1
$3\#(2+,2+,3+)$	= NOR	2 7 1
$4\#(2+,2+,2+,*)$	= NOR	2 7 1
$3\#(2+,2+,4+)$	= NOR	1 8 1
$3\#(2+,3+,3+)$	= NOR	1 8 1
$4\#(2+,2+,3+,*)$	= NOR	1 8 1
$5\#(2+,2+,2+,* ,*)$	= NOR	1 8 1
$2\#(2+,2+)$	= NOR	5 4 1
$2\#(2+,3+)$	= NOR	4 5 1
$3\#(2+,2+,*)$	= NOR	4 5 1
$2\#(2+,4+)$	= NOR	3 6 1
$2\#(3+,3+)$	= NOR	3 6 1

3#(2+,3+,*)	= NOR	3 6 1			
4#(2+,2+,* ,*)	= NOR	3 6 1			
2#(2+,5+)	= NOR	2 7 1			
2#(3+,4+)	= NOR	2 7 1			
3#(2+,4+,*)	= NOR	2 7 1			
3#(3+,3+,*)	= NOR	2 7 1			
4#(2+,3+,* ,*)	= NOR	2 7 1			
5#(2+,2+,* ,* ,*)	= NOR	2 7 1			
2#(2+,6+)	= NOR	1 8 1			
2#(3+,5+)	= NOR	1 8 1			
2#(4+,4+)	= NOR	1 8 1			
3#(2+,5+,*)	= NOR	1 8 1			
3#(3+,4+,*)	= NOR	1 8 1			
4#(2+,4+,* ,*)	= NOR	1 8 1			
4#(3+,3+,* ,*)	= NOR	1 8 1			
5#(2+,3+,* ,* ,*)	= NOR	1 8 1			
6#(2+,2+,* ,* ,* ,*)	= NOR	1 8 1			
2#(2+,*)	= NOR	6 3 1			
2#(3+,*)	= NOR	5 4 1			
3#(2+,* ,*)	= NOR	5 4 1			
2#(4+,*)	= NOR	4 5 1			
3#(3+,* ,*)	= NOR	4 5 1			
4#(2+,* ,* ,*)	= NOR	4 5 1			
2#(5+,*)	= NOR	3 6 1			
3#(4+,* ,*)	= NOR	3 6 1			
4#(3+,* ,* ,*)	= NOR	3 6 1			
5#(2+,* ,* ,* ,*)	= NOR	3 6 1			
2#(6+,*)	= NOR	2 7 1			
3#(5+,* ,*)	= NOR	2 7 1			
4#(4+,* ,* ,*)	= NOR	2 7 1			
5#(3+,* ,* ,* ,*)	= NOR	2 7 1			
6#(2+,* ,* ,* ,* ,*)	= NOR	2 7 1			
2#(7+,*)	= NOR	1 8 1			
3#(6+,* ,*)	= NOR	1 8 1			
4#(5+,* ,* ,*)	= NOR	1 8 1			
5#(4+,* ,* ,* ,*)	= NOR	1 8 1			
6#(3+,* ,* ,* ,* ,*)	= NOR	1 8 1			
7#(2+,* ,* ,* ,* ,* ,* ,*)	= NOR	1 8 1			
2#(2&,2&)	ao01d1	pvsc370d 4 1 1	A1 A2 B1 B2		ZN
2#(3&,3&)	ao02d1	pvsc370d 6 1 1	A1 A2 A3 B1 B2 B3		ZN
3#(2&,2&,*)	ao03d1	pvsc370d 5 1 1	A1 A2 B1 B2 C		ZN
2#(2&,*)	ao04d1	pvsc370d 3 1 1	A1 A2 B		ZN
3#(2&,* ,*)	ao05d1	pvsc370d 4 1 1	A1 A2 B C		ZN
2#(2&,2#)	fn03d1	pvsc370d 4 1 1	A1 A2 B1 B2		ZN
2~(2+,2~)	fn04d1	pvsc370d 4 1 1	A1 A2 B1 B2		ZN
2~(2+,2+)	oa01d1	pvsc370d 4 1 1	A1 A2 B1 B2		ZN
2~(3+,3+)	oa02d1	pvsc370d 6 1 1	A1 A2 A3 B1 B2 B3		ZN
3~(2+,2+,*)	oa03d1	pvsc370d 5 1 1	A1 A2 B1 B2 C		ZN
2~(2+,*)	oa04d1	pvsc370d 3 1 1	A1 A2 B		ZN
3~(2+,* ,*)	oa05d1	pvsc370d 4 1 1	A1 A2 B C		ZN

DRIVECAP:	53								
ni01d1	5	5 4 3 2 1		ni01d5	ni01d4	ni01d3	ni01d2	ni01d1	
ad01d1	2	2 1		ad01d2	ad01d1				
an02d1	2	2 1		an02d2	an02d1				
an03d1	2	2 1		an03d2	an03d1				
an04d1	2	2 1		an04d2	an04d1				
an05d1	2	2 1		an05d2	an05d1				
an06d1	2	2 1		an06d2	an06d1				
an07d1	2	2 1		an07d2	an07d1				
an08d1	2	2 1		an08d2	an08d1				
ao01d1	2	2 1		ao01d2	oa01d1				
ao04d1	2	2 1		ao04d2	oa04d1				
ao05d1	2	2 1		ao05d2	oa05d1				
as01d1	2	2 1		as01d2	as01d1				
dc24d1	2	2 1		dc24d2	dc24d1				
de24d1	2	2 1		de24d2	de24d1				
dfbtnb	2	2 1		dfbtnh	dfbtnb				
dfctnb	2	2 1		dfctnh	dfctnb				
dfntnb	2	2 1		dfntnh	dfntnb				
dfptnb	2	2 1		dfptnh	dfptnb				
fn02d1	2	2 1		fn02d2	fn02d1				
fn03d1	2	2 1		fn03d2	fn03d1				
fn04d1	2	2 1		fn04d2	fn04d1				
in01d1	5	5 4 3 2 1		in01d5	in01d4	in01d3	in01d2	in01d1	
mx21d1	2	2 1		mx21d2	mx21d1				
mx41d1	2	2 1		mx41d2	mx41d1				
mx81d1	2	2 1		mx81d2	mx81d1				
nd02d1	2	2 1		nd02d2	nd02d1				
nd03d1	2	2 1		nd03d2	nd03d1				
nd04d1	2	2 1		nd04d2	nd04d1				
nd05d1	2	2 1		nd05d2	nd05d1				
nd06d1	2	2 1		nd06d2	nd06d1				
nd07d1	2	2 1		nd07d2	nd07d1				
nd08d1	2	2 1		nd08d2	nd08d1				
nr02d1	2	2 1		nr02d2	nr02d1				
nr03d1	2	2 1		nr03d2	nr03d1				
nr04d1	2	2 1		nr04d2	nr04d1				
nr05d1	2	2 1		nr05d2	nr05d1				
nr06d1	2	2 1		nr06d2	nr06d1				
nr07d1	2	2 1		nr07d2	nr07d1				
nr08d1	2	2 1		nr08d2	nr08d1				
nt01d1	5	5 4 3 2 1		nt01d5	nt01d4	nt01d3	nt01d2	nt01d1	
oa01d1	2	2 1		oa01d2	oa01d1				
oa04d1	2	2 1		oa04d2	oa04d1				
oa05d1	2	2 1		oa05d2	oa05d1				
or02d1	2	2 1		or02d2	or02d1				
or03d1	2	2 1		or03d2	or03d1				
or04d1	2	2 1		or04d2	or04d1				
or05d1	2	2 1		or05d2	or05d1				
or06d1	2	2 1		or06d2	or06d1				
or07d1	2	2 1		or07d2	or07d1				

```

or08d1  2      2 1      or08d2 or08d1
xn02d1  2      2 1      xn02d2 xn02d1
xo02d1  2      2 1      xo02d2 xo02d1

```

(4) Contents of the lca2000.lib:

0 16 1 VDD GND

```

DEFELEM:          9
AND      3      4 3 2
        AND  N/A  4 1 1  ~1 ~2 ~3 ~4  ~0
        AND  N/A  3 1 1  ~1 ~2 ~3      ~0
        AND  N/A  2 1 1  ~1 ~2          ~0
OR       3      4 3 2
        OR  N/A  4 1 1  ~1 ~2 ~3 ~4  ~0
        OR  N/A  3 1 1  ~1 ~2 ~3      ~0
        OR  N/A  2 1 1  ~1 ~2          ~0
NAND    3      4 3 2
        NAND N/A  4 1 1  ~1 ~2 ~3 ~4  ~0
        NAND N/A  3 1 1  ~1 ~2 ~3      ~0
        NAND N/A  2 1 1  ~1 ~2          ~0
NOR     3      4 3 2
        NOR  N/A  4 1 1  ~1 ~2 ~3 ~4  ~0
        NOR  N/A  3 1 1  ~1 ~2 ~3      ~0
        NOR  N/A  2 1 1  ~1 ~2          ~0
INV     1      1
        INV  N/A  1 1 1  ~I      ~0
BUF     1      1
        BUF  N/A  1 1 1  ~I      ~0
XOR     1      2
        XOR  N/A  2 1 1  ~1 ~2  ~0
TRIBUF  0
BLKBOX  0

DFFTYPE:          4
DFF      !
DFFPC   DFF      N/A  5 2 1  D ~C SD RD !  Q !
DFFP    !
DFFC    !

IOPAD:    4
BLANKPAD  !
IPAD     IBUF  N/A   1 1 1  I  0
OPAD     OBUF  N/A   1 1 1  I  0
BPAD     OBUFT N/A   2 1 1  T I 0

MAPPING:    29
I(20)     XNOR  N/A   2 1 1  ~1 ~2  ~0
I(2~)     = AND  3 2 1

```

I(3 ⁻)	= AND	2 3 1
I(4 ⁻)	= AND	1 4 1
I(2#)	= OR	3 2 1
I(3#)	= OR	2 3 1
I(4#)	= OR	1 4 1
I(2&)	= NAND	3 2 1
I(3&)	= NAND	2 3 1
I(4&)	= NAND	1 4 1
I(2+)	= NOR	3 2 1
I(3+)	= NOR	2 3 1
I(4+)	= NOR	1 4 1
2&(2&,2&)	= AND	1 4 1
2&(2&,*)	= AND	2 3 1
2&(3&,*)	= AND	1 4 1
3&(2&,* ,*)	= AND	1 4 1
2+(2+,2+)	= OR	1 4 1
2+(2+,*)	= OR	2 3 1
2+(3+,*)	= OR	1 4 1
3+(2+,* ,*)	= OR	1 4 1
2 ⁻ (2&,2&)	= NAND	1 4 1
2 ⁻ (2&,*)	= NAND	2 3 1
2 ⁻ (3&,*)	= NAND	1 4 1
3 ⁻ (2&,* ,*)	= NAND	1 4 1
2#(2+,2+)	= NOR	1 4 1
2#(2+,*)	= NOR	2 3 1
2#(3+,*)	= NOR	1 4 1
3#(2+,* ,*)	= NOR	1 4 1

DRIVECAP: 1
 BUF 1 1 BUF

(5) Contents of the lca3000.lib:

0 16 1 VDD GND

DEFELEM:	9
AND	4 5 4 3 2
AND	N/A 5 1 1 ~1 ~2 ~3 ~4 ~5 ~0
AND	N/A 4 1 1 ~1 ~2 ~3 ~4 ~0
AND	N/A 3 1 1 ~1 ~2 ~3 ~0
AND	N/A 2 1 1 ~1 ~2 ~0
OR	4 5 4 3 2
OR	N/A 5 1 1 ~1 ~2 ~3 ~4 ~5 ~0
OR	N/A 4 1 1 ~1 ~2 ~3 ~4 ~0
OR	N/A 3 1 1 ~1 ~2 ~3 ~0
OR	N/A 2 1 1 ~1 ~2 ~0
NAND	4 5 4 3 2
NAND	N/A 5 1 1 ~1 ~2 ~3 ~4 ~5 ~0
NAND	N/A 4 1 1 ~1 ~2 ~3 ~4 ~0

```

      NAND N/A 3 1 1  ~1 ~2 ~3      ~0
      NAND N/A 2 1 1  ~1 ~2      ~0
NOR    4      5 4 3 2
      NOR N/A 5 1 1  ~1 ~2 ~3 ~4 ~5 ~0
      NOR N/A 4 1 1  ~1 ~2 ~3 ~4      ~0
      NOR N/A 3 1 1  ~1 ~2 ~3      ~0
      NOR N/A 2 1 1  ~1 ~2      ~0
INV    1      1
      INV N/A 1 1 1  ~I ~O
BUF    1      1
      BUF N/A 1 1 1  ~I ~O
XOR    1      2
      XOR N/A 2 1 1  ~1 ~2 ~O
TRIBUF 0
BLKBOX 0

DFFTYPE:      4
DFF           !
DFFPC        !
DFFP         !
DFFC DFF     N/A 4 2 1  D ~C RD CE Q !

IOPAD:      4
BLANKPAD    !
IPAD       IBUF N/A      1 1 1      I      0
OPAD       OBUF N/A      1 1 1      ~I     0
BPAD       OBUFT N/A     2 1 1      ~T ~I  0

MAPPING:    53
I(20)      XNOR N/A      2 1 1  ~1 ~2 ~O
I(2~)      = AND  4 2 1
I(3~)      = AND  3 3 1
I(4~)      = AND  2 4 1
I(5~)      = AND  1 5 1
I(2#)      = OR   4 2 1
I(3#)      = OR   3 3 1
I(4#)      = OR   2 4 1
I(5#)      = OR   1 5 1
I(2&)      = NAND 4 2 1
I(3&)      = NAND 3 3 1
I(4&)      = NAND 2 4 1
I(5&)      = NAND 1 5 1
I(2+)      = NOR  4 2 1
I(3+)      = NOR  3 3 1
I(4+)      = NOR  2 4 1
I(5+)      = NOR  1 5 1
2&(2&,2&) = AND  2 4 1
2&(2&,3&) = AND  1 5 1
3&(2&,2&,* ) = AND  1 5 1
2&(2&,* ) = AND  3 3 1
2&(3&,* ) = AND  2 4 1

```

```

3&(2&,* ,*) = AND 2 4 1
2&(4& ,*) = AND 1 5 1
3&(3&,* ,*) = AND 1 5 1
4&(2&,* ,* ,*) = AND 1 5 1
2+(2+ ,2+) = OR 2 4 1
2+(2+ ,3+) = OR 1 5 1
3+(2+ ,2+ ,*) = OR 1 5 1
2+(2+ ,*) = OR 3 3 1
2+(3+ ,*) = OR 2 4 1
3+(2+ ,* ,*) = OR 2 4 1
2+(4+ ,*) = OR 1 5 1
3+(3+ ,* ,*) = OR 1 5 1
4+(2+ ,* ,* ,*) = OR 1 5 1
2^(2& ,2&) = NAND 2 4 1
2^(2& ,3&) = NAND 1 5 1
3^(2& ,2& ,*) = NAND 1 5 1
2^(2& ,*) = NAND 3 3 1
2^(3& ,*) = NAND 2 4 1
3^(2& ,* ,*) = NAND 2 4 1
2^(4& ,*) = NAND 1 5 1
3^(3& ,* ,*) = NAND 1 5 1
4^(2& ,* ,* ,*) = NAND 1 5 1
2#(2+ ,2+) = NOR 2 4 1
2#(2+ ,3+) = NOR 1 5 1
3#(2+ ,2+ ,*) = NOR 1 5 1
2#(2+ ,*) = NOR 3 3 1
2#(3+ ,*) = NOR 2 4 1
3#(2+ ,* ,*) = NOR 2 4 1
2#(4+ ,*) = NOR 1 5 1
3#(3+ ,* ,*) = NOR 1 5 1
4#(2+ ,* ,* ,*) = NOR 1 5 1

```

```

DRIVECAP:      0
BUF           1      1      BUF

```

Appendix B

ANFT Translated Output and Simulation Command Files of DEMOX

(1) The reference file of DEMOX:

University of Arizona, Tucson, Department of Electrical & Computer Engineering
<< AENT >> AHPL to Netlist Format Translator. << VER: May 1995>>

Source File Name : [demoxyy.glf]

Sun May 14 03:42:56 1995

<<< INPUT, OUTPUT and BIDIRECTIONAL IO PORTS >>>

NAME	ELEMENT#	TYPE	INPUT/OUTPUT LIST(S)
X	1	EXINPUT	O: [8] [25]
B	2	EXINPUT	O: [13]
CLOCK	3	EXINPUT	O: [12] [13] [6]
RESET	4	EXINPUT	O: [36]
Z	10	EXOUTPUT	I: [7]
LOOK	11	EXOUTPUT	I: [14]

<<<< D TYPE FLIPFLOP(s) >>>>

NAME	ELEMENT#	TYPE	PACKAGE	Data	Clock	Preset	Clear	ClkEnb
R	7	DFF	U1-1	I : [26] [13] Q :				
A	8	DFF	U2-1	I : [1] [6] Q : [25] [9] Q~: [34]				
dfbtnb	14	DFFPC	U3-1	I : [34] [12] [36] [-1] Q : [9] [11] [6] Q~:				
dfbtnb	16	DFFPC	U4-1	I : [9] [12] [-1] [36] Q : [13] [9] [34] [26] Q~:				

<<<< BALCK BOX UNIT(s) >>>>

NAME	ELEMENT#	PACKAGE	INPUT/OUTPUT LISTS
MX21D1	25	U5-1	I : [7][8][1] O1: [26]

<<<< GATE(s) >>>>

ELEM#	CHIPNAME	TYPE	PACKAGE	INPUT LIST
6	nd02d1	NAND	U6-1	[3][14]
9	aoi3c	SPECIAL	U7-1	[16][8][14]
12	in01d1	INV	U8-1	[3]
13	nd03d1	SPECIAL	U9-1	[2][16][3]
26	an02d1	AND	U10-1	[25][16]
34	an02d1	AND	U11-1	[16][8]
36	in01d1	INV	U12-1	[4]

(2) The ANFT translated DEMOX in EDIF 2 0 0:

```
(edif demoxyy_glf
  (edifVersion 2 0 0)
  (edifLevel 0)
  (keywordMap (KeywordLevel 0))
  (status
    (written
      (timeStamp 1995 5 14 3 42 56)
      (program "ANFT -- AHPL Netlist Format Translator"
        (version "May 1995"))
      (author "s3")
    )
  )
)
(library pvsc370d
  (edifLevel 0)
  (technology
    (numberDefinition))
  (cell an02d1
    (cellType GENERIC)
    (view Netlist_view
      (viewType NETLIST)
      (interface
        (port A1 (direction INPUT))
        (port A2 (direction INPUT))
        (port Z (direction OUTPUT))
        (designator "@@Label"))))
  (cell nd03d1
    (cellType GENERIC)
```

```

(view Netlist_view
 (viewType NETLIST)
 (interface
  (port A1 (direction INPUT))
  (port A2 (direction INPUT))
  (port A3 (direction INPUT))
  (port ZN (direction OUTPUT))
  (designator "@@Label"))))
(cell in01d1
 (cellType GENERIC)
 (view Netlist_view
 (viewType NETLIST)
 (interface
  (port I (direction INPUT))
  (port ZN (direction OUTPUT))
  (designator "@@Label"))))
(cell nd02d1
 (cellType GENERIC)
 (view Netlist_view
 (viewType NETLIST)
 (interface
  (port A1 (direction INPUT))
  (port A2 (direction INPUT))
  (port ZN (direction OUTPUT))
  (designator "@@Label"))))
(cell mx21d1
 (cellType GENERIC)
 (view Netlist_view
 (viewType NETLIST)
 (interface
  (port IO (direction INPUT))
  (port I1 (direction INPUT))
  (port S (direction INPUT))
  (port Z (direction OUTPUT))
  (designator "@@Label"))))
(cell dfbntb
 (cellType GENERIC)
 (view Netlist_view
 (viewType NETLIST)
 (interface
  (port D (direction INPUT))
  (port CP (direction INPUT))
  (port SDN (direction INPUT))
  (port CDN (direction INPUT))
  (port Q (direction OUTPUT))
  (port QN (direction OUTPUT))
  (designator "@@Label"))))
(cell dfntnb
 (cellType GENERIC)
 (view Netlist_view
 (viewType NETLIST)

```

```

(interface
  (port D (direction INPUT))
  (port CP (direction INPUT))
  (port Q (direction OUTPUT))
  (port QN (direction OUTPUT))
  (designator "@@Label"))))
(library aolib
  (edifLevel 0)
  (technology
    (numberDefinition))
  (cell aoi3c
    (cellType GENERIC)
    (view Netlist_view
      (viewType NETLIST)
      (interface
        (port A1 (direction INPUT))
        (port A2 (direction INPUT))
        (port B (direction INPUT))
        (port Z (direction OUTPUT))
        (port ZN (direction OUTPUT))
        (designator "@@Label")))))
(library hpc
  (edifLevel 0)
  (technology
    (numberDefinition))
  (cell demoxyy_glf
    (cellType GENERIC)
    (view Netlist_view
      (viewType NETLIST)
      (interface
        (port X          (direction INPUT))
        (port B          (direction INPUT))
        (port CLOCK     (direction INPUT))
        (port RESET     (direction INPUT))
        (port Z          (direction OUTPUT))
        (port LOOK      (direction OUTPUT))
        (designator "@@Label"))
      (contents
        (instance U1
          (viewRef Netlist_view
            (cellRef dfntnb
              (libraryRef pvsc370d))))
          (instance U2
            (viewRef Netlist_view
              (cellRef dfntnb
                (libraryRef pvsc370d))))
          (instance U3
            (viewRef Netlist_view
              (cellRef dfbntb
                (libraryRef pvsc370d))))
          (instance U4

```

```
(viewRef Netlist_view
  (cellRef dfbtnb
    (libraryRef pvsc370d))))
(instance U5
  (viewRef Netlist_view
    (cellRef MX21D1
      (libraryRef pvsc370d))))
(instance U6
  (viewRef Netlist_view
    (cellRef nd02d1
      (libraryRef pvsc370d))))
(instance U7
  (viewRef Netlist_view
    (cellRef aoi3c
      (libraryRef aoilib))))
(instance U8
  (viewRef Netlist_view
    (cellRef in01d1
      (libraryRef pvsc370d))))
(instance U9
  (viewRef Netlist_view
    (cellRef nd03d1
      (libraryRef pvsc370d))))
(instance U10
  (viewRef Netlist_view
    (cellRef an02d1
      (libraryRef pvsc370d))))
(instance U11
  (viewRef Netlist_view
    (cellRef an02d1
      (libraryRef pvsc370d))))
(instance U12
  (viewRef Netlist_view
    (cellRef in01d1
      (libraryRef pvsc370d))))
(net X
  (joined
    (portRef X)
    (portRef D (instanceRef U2))
    (portRef S (instanceRef U5))
  ))
(net B
  (joined
    (portRef B)
    (portRef A1 (instanceRef U9))
  ))
(net CLOCK
  (joined
    (portRef CLOCK)
    (portRef I (instanceRef U8))
    (portRef A3 (instanceRef U9))
```

```
    (portRef A1 (instanceRef U6))
  ))
(net RESET
  (joined
    (portRef RESET)
    (portRef I (instanceRef U12))
  ))
(net
  (rename U6_ZN "U6.ZN")
  (joined
    (portRef ZN (instanceRef U6))
    (portRef CP (instanceRef U2))
  ))
(net
  (rename U1_QN "U1.QN")
  (joined
    (portRef QN (instanceRef U1))
    (portRef IO (instanceRef U5))
    (portRef Z)
  ))
(net
  (rename U2_Q "U2.Q")
  (joined
    (portRef Q (instanceRef U2))
    (portRef I1 (instanceRef U5))
    (portRef A2 (instanceRef U7))
  ))
(net
  (rename U2_QN "U2.QN")
  (joined
    (portRef QN (instanceRef U2))
    (portRef A2 (instanceRef U11))
  ))
(net
  (rename U7_Z "U7.Z")
  (joined
    (portRef Z (instanceRef U7))
    (portRef D (instanceRef U4))
  ))
(net
  (rename U8_ZN "U8.ZN")
  (joined
    (portRef ZN (instanceRef U8))
    (portRef CP (instanceRef U3))
    (portRef CP (instanceRef U4))
  ))
(net
  (rename U9_ZN "U9.ZN")
  (joined
    (portRef ZN (instanceRef U9))
    (portRef CP (instanceRef U1))
  ))
```

```
))
(net
  (rename U3_Q "U3.Q")
  (joined
    (portRef Q (instanceRef U3))
    (portRef B (instanceRef U7))
    (portRef LOOK)
    (portRef A2 (instanceRef U6))
  ))
(net
  (rename U4_Q "U4.Q")
  (joined
    (portRef Q (instanceRef U4))
    (portRef A2 (instanceRef U9))
    (portRef A1 (instanceRef U7))
    (portRef A1 (instanceRef U11))
    (portRef A2 (instanceRef U10))
  ))
(net
  (rename U5_Z "U5.Z")
  (joined
    (portRef Z (instanceRef U5))
    (portRef A1 (instanceRef U10))
  ))
(net
  (rename U10_Z "U10.Z")
  (joined
    (portRef Z (instanceRef U10))
    (portRef D (instanceRef U1))
  ))
(net
  (rename U11_Z "U11.Z")
  (joined
    (portRef Z (instanceRef U11))
    (portRef D (instanceRef U3))
  ))
(net
  (rename U12_ZN "U12.ZN")
  (joined
    (portRef ZN (instanceRef U12))
    (portRef SDN (instanceRef U3))
    (portRef CDN (instanceRef U4))
  ))
(net VDD
  (joined
    (portRef CDN (instanceRef U3))
    (portRef SDN (instanceRef U4))
  ))
)
)
)
```

```

)
(design demoxyy_glf
 (cellRef demoxyy_glf
  (libraryRef hpc)))

```

(3) The simulation command and data file:

```

#cell12 * demo csm * 2 any 0 v8r4.4
# "3-Mar-94 GMT" "1:23:32 GMT" "3-Mar-94 GMT" "1:23:32 GMT" s3 * .
## Setup the parameters

set trace mode tabular
set options -tabularReportOnChange
set options traceAllSteps
set options tabularAbsoluteTime

set clock clock 0(1000); 1(500) 0(500)
set clock x 0(2600); 1(2000) 0(2000)
set clock b 0(3600); 1(2000) 0(2000)

set external input reset

set external output z look a r

set power high vdd
set power low vss

watch (display) reset clock x b z look r a

##### simulation begins

set inputs high reset
step 400
set inputs low reset
step 9650
set inputs high reset
step 340
set inputs low reset
step 4800

##### End of simulation

```

Appendix C

TODCLK Simulation Files for GateSim™ Ver 1.11

(1) Vector input file for testing **PRESCR[0-6]**, **MINUR[0-3]**, **MINTENR[0-2]**, **HRUR[0-3]** and **HRTR** (Figure 6.7 (a) to (k)):

```
$ TCKYY.VEC : specify the input waveforms.
$
$ Power connections
GND          .clk 0 0
VDD          .clk 0 1
CLOCK        .clk 0 0 1000 1 4000 0 7000 1 .REP 1000
TWOHZ        .clk 0 1 4500 0 10500 1 16500 0 .REP 4500
LEDTEST      .clk 0 0
SET          .clk 0 0 8000 1 11000 0
HRSET        .clk 0 0
$
$ Test patterns
$
.patt RESET
0 1
.2500 0
$
$
```

(2) Vector input file for testing **SET** (Figure 6.7 (l) to (o)):

```
$ TCKSET.VEC : specify the input waveforms.
$
$ Power connections
GND          .clk 0 0
VDD          .clk 0 1
CLOCK        .clk 0 0 1000 1 4000 0 7000 1 .REP 1000
TWOHZ        .clk 0 1 4500 0 10500 1 16500 0 .REP 4500
LEDTEST      .clk 0 0
SET          .clk 0 0 8000 1
HRSET        .clk 0 0
$
$ Test patterns
$
.patt RESET
```

```

0    1
.2500 0
$
$

```

(3) Vector input file for testing **HRSET** (Figure 6.7 (p) and (q)):

```

$ TCKHRSET.VEC : specify the input waveforms.
$
$ Power connections
GND      .clk 0 0
VDD      .clk 0 1
CLOCK    .clk 0 0 1000 1 4000 0 7000 1 .REP 1000
TWOHZ    .clk 0 1 4500 0 10500 1 16500 0 .REP 4500
LEDTEST  .clk 0 0
SET      .clk 0 0
HRSET    .clk 0 0 8000 1
$
$ Test patterns
$
.patt RESET
0    1
.2500 0
$
$

```

(4) The simulation command file for **PRESCR[0-6]**:

```

$ ===== $
$      This simulation file is for testing the AHPL benchmark TODCLK.HPC      $
$      There can be only one "space" between the 2-char command and the parameter $
$ ===== $
$ ===== $
$      Specify the source and output file      $
$ ===== $
ne tckyy
pa tckyy
re tckpre
$
$ ===== $
$      Display the simulation      $
$ ===== $
mo ON
$
$ ===== $
$ Shift the decimal point so that the time is expressed in nanoseconds $
$ because library times are in 1/10ths of nanoseconds. $
$ ===== $

```

```

dp 1 1 1
$
$ ===== $
$           Trace the following signals           $
$ ===== $
.tab RESET TWOHZ SET HRSET ; U1_P2 U2_P2 U3_P2 U4_P2 U5_P2 U6_P2 U7_P2
$
$ ===== $
$           Turn on horizontal scrolling & Turn on the waveform output. $
$ ===== $
HOrizontal ON
WAVEform ON
DLay 1000
$
$ ===== $
$           Simulate                               $
$ ===== $
si 0 190000
si +
si +
si +
si +
si +
si +
si +
$
$ ===== $
$           Quit                                   $
$ ===== $
qu

```

(5) For **MINUR[0-3]** and **MINTENR[0-2]** use the followings as substitute:

```

ne tckyy
pa tckyy
re tckmin
$
.tab RESET SET HRSET ; U12_P2 U13_P2 U14_P2 ; U8_P2 U9_P2 U10_P2 U11_P2
$
si 0 45000000
si +

```

(6) For **HRUR[0-3]** and **HRTR** use the followings as substitute:

```

ne tckyy
pa tckyy
re tckhr
$

```

```
.tab RESET SET HRSET ; U19_P2 U15_P2 U16_P2 U17_P2 U18_P2
$
si 0 2000000000
```

(7) For **SET** trace use the followings as substitute:

```
ne tckyy
pa tckset
re tckset
$
.tab RESET TWOHZ SET HRSET; U12_P2 U13_P2 U14_P2 ; U8_P2 U9_P2 U10_P2 U11_P2
$
si 0 200000
si +
si +
si +
```

(8) For **HRSET** use the followings as substitute:

```
ne tckyy
pa tckhrset
re tckhrset
$
.tab RESET CLOCK TWOHZ SET HRSET ; U19_P2 U15_P2 U16_P2 U17_P2 U18_P2
$
si 0 100000
si +
```

Appendix D

TODCLK Simulation Files for OrCAD/VST™ Ver 1.10

(1) Settings for testing **PRESCR[0-6]**:

Stimulus File		
Signal Name	Init. Value	Time/Function
VCC	1	None.
GND	0	None.
LEDTEST	0	None.
SET	0	1500/1 1700/0
HRSET	0	None.
RESET	1	300/0
CLOCK	0	100/1 600/0 1100/GOTO 100
TWOHZ	0	650/1 1650/0 2650/GOTO 650
Trace File : Nodes for PRESCR[0-6]		
PRESCR[0-6]	U1-5 U1-9 U2-5 U2-9 U3-5 U3-9 U4-5	
Misc. Settings		
Change View To	2200	
Simulation Length	270,000 (<i>ns</i>)	

(2) Settings for testing **MINUR[0-3]**, **MINTENR[0-2]**:

Stimulus File		
Signal Name	Init. Value	Time/Function
(Same as above)		
Trace File : Nodes for MINUR[0-3] and MINTENR[0-2]		
MINUR[0-3]	U4-9 U5-5 U5-9 U6-5	
MINTENR[0-2]	U6-9 U7-5 U7-9	
Misc. Settings		
Change View To	130,000	
Simulation Length	16,000,000 (<i>ns</i>)	

(3) Settings for testing **HRUR[0-3]** and **HRTR**:

Stimulus File		
Signal Name	Init. Value	Time/Function
(Same as above)		
Trace File : Nodes for HRUR[0-3] and HRTR		
HRUR[0-3]	U8-5 U8-9 U9-5 U9-9	
HRTR	U10-5	
Misc. Settings		
Change View To	1,800,000	
Simulation Length	200,000,000 (<i>ns</i>)	

(4) Settings for testing **SET**:

Stimulus File		
Signal Name	Init. Value	Time/Function
VCC	1	None.
GND	0	None.
LEDTEST	0	None.
SET	0	8000/1
HRSET	0	None.
RESET	1	300/0
CLOCK	0	100/1 600/0 1100/GOTO 100
TWOHZ	0	650/1 1650/0 2650/GOTO 650
Trace File : Trace the following counters		
MINUR[0-3]	U4-9 U5-5 U5-9 U6-5	
MINTENR[0-2]	U6-9 U7-5 U7-9	
Misc. Settings		
Change View To	2,000	
Simulation Length	240,000 (<i>ns</i>)	

(5) Settings for testing **HRSET**:

Stimulus File		
Signal Name	Init. Value	Time/Function
VCC	1	None.
GND	0	None.
LEDTEST	0	None.
SET	0	None.
HRSET	0	8000/1
RESET	1	300/0
CLOCK	0	100/1 600/0 1100/GOTO 100
TWOHZ	0	650/1 1650/0 2650/GOTO 650
Trace File : Trace the following counters		
Misc. Settings		
Change View To	300	
Simulation Length	40,000 (ns)	

(6) Miscellaneous Signals:

Signal Name	Node
State 1 Signal	U11-9
State 2 Signal	U12-5
MINENBL	U37-8
HRENBL	U43-6

Appendix E

TODCLK Simulation Files for XilinxTM's Tool System

(1) The simulation file of TODCLK_YY used by P / C Silos can be generated directly

by **xf2sil** Ver. 2.12.

(2) Simulation data and command file for **PRESCR[0-6]**:

```

$ =====$
$   Data file : tckpre.dat
$ =====$
$
$ Simulation file for design 'aprtcka.sim', type '2064pc68-100'
$ Created by XNF2SILO Ver. 2.12 at Thu Apr 13 22:04:15 1995
$
!INPUT tckaapr.sim

GLOBALRESET- .CLK 0 S0 1 S1 $ Initial pulse to reset latches
RESET        .CLK 0 S1 300 S0
CLOCK        .CLK 0 S0 100 S1 350 S0 600 S1 .REP 100
TWOHZ        .CLK 0 S1 400 S1 900 S0 1400 S1 .REP 400
LEDTEST      .CLK 0 S0
SET          .CLK 0 S0 800 S1 900 S0
HRSET        .CLK 0 S0

$ ===== $
$   Set nodes to be traced while sim
$ ===== $
.MONITOR RESET CLOCK TWOHZ ; LEDTEST SET HRSET ;
+ PSCR_0 PSCR_1 PSCR_2 PSCR_3 PSCR_4 PSCR_5 PSCR_6
$
$ ===== $
$   Set nodes to be traced in graph mode
$ ===== $
.GRAPH RESET CLOCK SET
+ PSCR_0 PSCR_1 PSCR_2 PSCR_3 PSCR_4 PSCR_5 PSCR_6
$
$ ===== $
$   Set nodes to be PLOT using "store output t1 to t2 on change"
$   Note : use "disk new_output_file_name" to change file name.
$ ===== $
$

```

```
.PLOT PSCR_6 PSCR_5 PSCR_4 PSCR_3 PSCR_2 PSCR_1 PSCR_0 ;
+ HRSET SET ; TWOHZ CLOCK RESET
```

(3) Simulation data and command file for **MINUR[0-3]** and **MINTENR[0-2]** (use the followings as substitute):

```
.MONITOR RESET LEDTEST SET HRSET ;
+ MINT_0 MINT_1 MINT_2 MINU_0 MINU_1 MINU_2 MINU_3
$
.GRAPH RESET SET HRSET
+ MINT_0 MINT_1 MINT_2 MINU_0 MINU_1 MINU_2 MINU_3
$
.PLOT MINU_3 MINU_2 MINU_1 MINU_0 ; MINT_2 MINT_1 MINT_0 ;
+ HRSET SET RESET
```

(4) Simulation data and command file for **HRUR[0-3]** and **HRTR** (use the followings as substitute):

```
.MONITOR RESET LEDTEST SET HRSET ; HRT HRU_0 HRU_1 HRU_2 HRU_3
$
.GRAPH RESET SET HRSET ; HRT HRU_0 HRU_1 HRU_2 HRU_3
$
.PLOT HRU_3 HRU_2 HRU_1 HRU_0 HRT ; HRSET SET RESET
```

(5) Simulation data and command file for **SET** (use the followings as substitute):

```
SET .CLK 0 S0 2000 S1
$
.MONITOR RESET CLOCK TWOHZ LEDTEST SET HRSET ;
+ MINT_0 MINT_1 MINT_2 MINU_0 MINU_1 MINU_2 MINU_3
$
.GRAPH RESET SET HRSET
+ MINT_0 MINT_1 MINT_2 MINU_0 MINU_1 MINU_2 MINU_3
$
.PLOT MINU_3 MINU_2 MINU_1 MINU_0 ; MINT_2 MINT_1 MINT_0 ;
+ HRSET SET TWOHZ CLOCK RESET
```

(6) Simulation data and command file for **HRSET** (use the followings as substitute):

```
SET .CLK 0 S0
HRSET .CLK 0 S0 2000 S1
$
```

```

.MONITOR RESET CLOCK TWOHZ LEDTEST SET HRSET ; HRT HRU_0 HRU_1 HRU_2 HRU_3
$
.GRAPH RESET SET HRSET ; HRT HRU_0 HRU_1 HRU_2 HRU_3
$
.PLOT HRU_3 HRU_2 HRU_1 HRU_0 HRT ; HRSET SET TWOHZ CLOCK RESET

```

(7) Simulation times and Node Names for testing the counters:

Signal Name	Node Name	Simulation Time (ns)
PRESCR[0-6]	PSCR_0 to PSCR_6	128K
MINUR[0-3] and MINTENR[0-2]	MINU_0 to MINU_3 MINTENR_0 to MINTENR_2	8800K
HRUR[0-3] and HRTR	HRU_0 to HRU_3 HRTR	108M
SET	MINU_0 to MINU_3 MINTENR_0 to MINTENR_2	72K
HRSET	HRU_0 to HRU_3 HRTR	20K

Appendix F

TODCLK Simulation Files for COMPASS Tools™ System

V8R4.4

(1) Simulation command and data file for testing **PRESCR[0-6]**:

```
#cell12 * todclk_prescr csm * 9 any 0 v8r4.4
# "12-Apr-95 GMT" "4:49:11 GMT" "12-Apr-95 GMT" "4:49:11 GMT" s3 * .
## Setup the parameters

set trace mode tabular
set options -tabularReportOnChange
set options traceAllSteps
set options tabularAbsoluteTime

set clock CLOCK 0(100); 1(250) 0(250)
set clock TWOHZ 0(400); 1(500) 0(500)

set external input RESET SET HRSET LEDTEST

set external output S1 S2 MINENBL
set external output OUT0 OUT1 OUT2 OUT3
set external output PRESCRO PRESCR1 PRESCR2 PRESCR3 PRESCR4 PRESCR5 PRESCR6
set external output MINURO MINUR1 MINUR2 MINUR3
set external output MINTENRO MINTEN1 MINTEN2
set external output HRURO HRUR1 HRUR2 HRUR3
set external output HRTR

set power high vdd
set power low vss

watch (display) RESET CLOCK TWOHZ S1 S2 LEDTEST SET HRSET MINENBL
watch (display) PRESCRO PRESCR1 PRESCR2 PRESCR3 PRESCR4 PRESCR5 PRESCR6

##### simulation begins
set inputs high RESET
set inputs low LEDTEST
set inputs low SET
set inputs low HRSET
step 300
set inputs low RESET
step 500
set inputs high SET
```

```

step 100
set inputs low SET
step 5000
##### End of simulation

```

(2) For **MINUR[0-3]** and **MINTENR[0-2]**, use the followings as substitute :

```

watch (display) SET HRSET HRENBL MINTENRO MINTENR1 MINTENR2
watch (display) MINURO MINUR1 MINUR2 MINUR3 MINENBL
##### simulation begins
set inputs high RESET
set inputs low LEDTEST
set inputs low SET
set inputs low HRSET
step 300
set inputs low RESET
step 500
set inputs high SET
step 100
set inputs low SET
step 5000
##### End of simulation

```

(3) For **HRUR[0-3]** and **HRTR**, use the followings as substitute :

```

watch (display) SET HRSET HRTR HRURO HRUR1 HRUR2 HRUR3 HRENBL MINENBL
##### simulation begins
set inputs high RESET
set inputs low LEDTEST
set inputs low SET
set inputs low HRSET
step 300
set inputs low RESET
step 500
set inputs high SET
step 100
set inputs low SET
step 5000
##### End of simulation

```

(4) For **SET**, use the followings as substitute :

```

watch (display) CLOCK TWOHZ SET MINTENRO MINTENR1 MINTENR2
watch (display) MINURO MINUR1 MINUR2 MINUR3 HRENBL MINENBL
##### simulation begins
set inputs high RESET

```

```
set inputs low LEDTEST
set inputs low SET
set inputs low HRSET
step 300
set inputs low RESET
step 1700
set inputs high SET
step 5000
##### End of simulation
```

(5) For **HRSET**, use the followings as substitute :

```
watch (display) CLOCK TWOHZ SET HRSET HRTR HRURO HRUR1 HRUR2 HRUR3 HRENBL MINENBL
##### simulation begins
set inputs high RESET
set inputs low LEDTEST
set inputs low SET
set inputs low HRSET
step 300
set inputs low RESET
step 1700
set inputs high HRSET
step 5000
##### End of simulation
```

Appendix G

SWA416 Detail Critical Path Timing Information

In the SWA416 case, the critical path of AENT91 translated output is different from the critical path of ANFT translated output. However, the pattern of them are the same. Thus, we list the detailed critical path delay information below to show the effect of the BUFFER insertion used by ANFT.

(1) In the ANFT translated output (AND-OR-INV implementation), the fanout of U134 (dfbtnb) is 78; thus, four non-inverting BUFFERs (ni01d1) are inserted to the output q of U134. The U282 is one of them. As shown, the delay time from cp to q of U134 is 6.97 ns.

```
f2
instance Name
input Pin -----> output Pin
-----
START-OF-PATH
----->U271_Z
U134
  cp --> q
U282
  i --> z
U1059
  a2 --> z
U215
  a1 --> z
U1074
  a2 --> z
U1076
  a2 --> z
U1089
  a1 --> z
U1091
  a2 --> z
```

	tr	total	incr	[ramp]	cap	cell
START-OF-PATH	R	.00	.00	[.0]	2.52	
----->U271_Z						
U134						
cp --> q	R	6.97	6.97	[5.0]	1.79	dfbtnb
U282						
i --> z	R	12.47	5.50	[5.1]	1.81	ni01d1
U1059						
a2 --> z	R	13.30	.84	[.5]	.15	an02d1
U215						
a1 --> z	R	16.81	3.50	[2.5]	.79	or03d1
U1074						
a2 --> z	R	17.66	.85	[.5]	.15	an02d1
U1076						
a2 --> z	R	20.80	3.14	[1.8]	.58	or03d1
U1089						
a1 --> z	R	21.63	.83	[.5]	.15	an02d1
U1091						
a2 --> z	R	24.75	3.11	[1.8]	.58	or03d1

U1104							
	a1 --> z	R	25.58	.84	[.5]	.15	an02d1
U1106							
	a2 --> z	R	28.55	2.97	[1.6]	.53	or03d1
U1119							
	a1 --> z	R	29.38	.83	[.5]	.14	an02d1
U1121							
	a2 --> z	R	32.53	3.15	[1.8]	.59	or03d1
U1134							
	a1 --> z	R	33.39	.86	[.5]	.15	an02d1
U1136							
	a2 --> z	R	36.41	3.02	[1.7]	.55	or03d1
U1149							
	a1 --> z	R	37.26	.84	[.5]	.15	an02d1
U1151							
	a2 --> z	R	40.92	3.67	[2.3]	.75	or03d1
U1164							
	a1 --> z	R	41.76	.84	[.5]	.15	an02d1
U1166							
	a2 --> z	R	45.33	3.56	[2.2]	.72	or03d1
U1179							
	a1 --> z	R	46.15	.82	[.5]	.14	an02d1
U1181							
	a2 --> z	R	49.53	3.38	[2.0]	.66	or03d1
U1194							
	a1 --> z	R	50.35	.82	[.4]	.14	an02d1
U1196							
	a2 --> z	R	53.47	3.12	[1.8]	.58	or03d1
U1209							
	a1 --> z	R	54.32	.85	[.5]	.15	an02d1
U1211							
	a2 --> z	R	57.42	3.10	[1.8]	.57	or03d1
U1224							
	a1 --> z	R	58.25	.83	[.5]	.14	an02d1
U1226							
	a2 --> z	R	62.02	3.78	[2.4]	.79	or03d1
U1239							
	a1 --> z	R	62.87	.85	[.5]	.15	an02d1
U1241							
	a2 --> z	R	65.90	3.03	[1.7]	.55	or03d1
U1254							
	a1 --> z	R	66.75	.84	[.5]	.15	an02d1
U1256							
	a2 --> z	R	70.35	3.61	[2.3]	.73	or03d1
U1269							
	a1 --> z	R	71.19	.84	[.5]	.15	an02d1
U1271							
	a2 --> z	R	74.16	2.97	[1.6]	.53	or03d1
U1284							
	a1 --> z	R	74.98	.82	[.5]	.14	an02d1
U1286							

	a2 --> z	R	51.11	3.48	[2.1]	.69	or03d1
U791	a1 --> z	R	51.98	.87	[.5]	.16	an02d1
U793	a2 --> z	R	55.53	3.55	[2.2]	.71	or03d1
U806	a1 --> z	R	56.37	.84	[.5]	.15	an02d1
U808	a2 --> z	R	60.56	4.19	[2.9]	.92	or03d1
U821	a1 --> z	R	61.37	.81	[.4]	.14	an02d1
U823	a2 --> z	R	65.58	4.21	[2.9]	.93	or03d1
U836	a1 --> z	R	66.41	.83	[.5]	.14	an02d1
U838	a2 --> z	R	69.67	3.26	[1.9]	.62	or03d1
U851	a1 --> z	R	70.50	.83	[.5]	.14	an02d1
U853	a2 --> z	R	75.43	4.93	[3.6]	1.16	or03d1
U866	a1 --> z	R	76.29	.87	[.5]	.16	an02d1
U868	a2 --> z	R	79.34	3.05	[1.7]	.55	or03d1
U881	a1 --> z	R	80.17	.83	[.5]	.15	an02d1
U883	a2 --> z	R	83.20	3.03	[1.7]	.55	or03d1
U896	a1 --> z	R	84.07	.87	[.5]	.16	an02d1
U898	a2 --> z	R	88.92	4.84	[3.5]	1.13	or03d1
U911	a1 --> z	R	89.79	.87	[.5]	.16	an02d1
U913	a2 --> z	R	92.86	3.08	[1.7]	.56	or03d1
U918	i --> zn	F	93.36	.50	[.4]	.16	in01d1
U919	a1 --> z	F	94.82	1.46	[.4]	.15	an03d1
U924	a3 --> z	F	96.27	1.45	[.6]	.24	or04d1
U924_Z		F	97.40	1.13	[.0]	.00	
	----->SETUP-TIME						

REFERENCES

- [1] D. D. Gajski, F. Vahid, S. Narayan, J. Gong, *Specification and Design of Embedded Systems*, Englewood Cliffs, N.J., Prentice-Hall, 1994.
- [2] F. J. Hill, "Introducing AHPL," *IEEE Computer*, December 1974.
- [3] M. Masud, "Modular Implementation of a Digital Hardware Design Automation System," Ph.D. Dissertation, University of Arizona, Tucson, 1981.
- [4] Z. Navabi, "VLSI Design Automation Using A Hardware Programming Language," Ph.D. Dissertation, University of Arizona, Tucson, 1981.
- [5] F. J. Hill, et al, "Hardware Compilation from an RTL to a Storage Logic Array Target," *IEEE Trans. on CAD*, July 1984.
- [6] G. L. Chen, "Performance and Area Optimization in Sea-of-Wires Array Synthesis," Ph.D. Dissertation, University of Arizona, Tucson, 1994.
- [7] M. M. Massoumi, "Structuring VHDL Synthesis Using The AHPL Paradigm," Ph.D. Dissertation, University of Arizona, Tucson, 1994.
- [8] M. M. Al-Sharif, "Functional Level Simulator for Universal AHPL," M.S. Thesis, University of Arizona, Tucson, 1983.
- [9] J. X. Chen, "A Hardware Compiler for VLSI Synthesis Applications," M.S. Thesis, University of Arizona, Tucson, 1992.
- [10] H. Lougee, *AENT*, Computer Software, Unpublished, 1990.
- [11] Y. L. Lim, "A Package Efficient PC Based AHPL to EDIF Translator," M.S. Thesis, University of Arizona, Tucson, 1990.
- [12] B. Jepperson, "Framework for CMOS Standard Cell Realization of AHPL," M.S. Thesis, University of Arizona, Tucson, 1991.
- [13] Electronic Industries Association, *Electronic Design Interchange Format Version 2 0 0*, ANSI/EIA-548-1988, March 1988.

- [14] F. J. Hill, G. R. Peterson *Digital Systems - Hardware Organization and Design*, 3rd Edition, John Wiley and Sons, New York, 1987.
- [15] Xilinx Inc., *Xilinx XNF Netlist Specification*, Version 4.00b, Xilinx Inc., September 2, 1992.
- [16] P. B. Cohen, *The Design of a 4 BCD Digit Clock*, Massachusetts Microelectronics Center, 1990.
- [17] T. I. Wang, G. L. Chen, *The EDIF 1 1 0 to EDIF 2 0 0 Format Translator*, Software Tool, University of Arizona, Tucson, 1994.
- [18] VLSI Technology, Inc. *1-Micron High Density Standard Cell Library (VSC370)*, VLSI Technology, Inc., Nov. 1991.