To the Graduate Council:

I am submitting herewith a thesis written by Tony Emile Sawan entitled "Development and Automation of a Procedure for Post-Layout Timing Analysis of Integrated Circuits." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald W. Bouldin

Dr. Donald W. Bouldin, Major Professor

We have read this thesis
and recommend its acceptance:

Robert E. Bodenheimer

Robert W. Rochelle

Accepted for the council:

Leo Winkel

Associate Vice Chancellor
and Dean of the Graduate School

## STATEMENT OF PERMISSION TO USE

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

Permission for extensive quotation from or reproduction of this thesis may be granted by major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any coping or use of the material in this thesis for financial gain shall not be allowed without my permission.

Signature Tony Sauve

Date 11-12-93

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

# DEVELOPMENT AND AUTOMATION OF A PROCEDURE FOR POST-LAYOUT TIMING ANALYSIS OF INTEGRATED CIRCUITS

A Thesis
Presented for the
Masters of Science
Degree
The University of Tennessee, Knoxville

**TONY EMILE SAWAN**

December 1993

Copyright © Tony Emile Sawan, 1993
All rights reserved

## DEDICATION

To the Whole Body of Christ so That They All Be One!

My Wife Vicky Lynn Sawan

Edouard Emile Sawan
(My beloved departed brother, God bless his soul)

My Father Emile Cheiban Sawan

My Mother Josephine Hanna Sawan

My Brother Jacques Emile Sawan, His Wife Maggie, and Their Son Toni

My Baby Sister Samira Emile Sawan

and

My Grand Mother Samira Roustom Sawan

## ABSTRACT

VLSI complex circuit designs have an increasing need of complete timing analysis tools for circuit verification. Timing constraints of post-layout information are becoming extremely vital for accomplishing a successful design. Hence, designers prefer that timing delays be computed concurrently with functional evaluations by working with estimated gate propagation times. This thesis describes ways of bridging the gap between the functional and timing simulations. It proposes a new simulation methodology, and outlines some important delay modeling methods. One procedure for post-layout timing analysis, ***CKTIM***, is developed and automated by writing the program, ***Sim2Bat***. This software was written in ***awk***, ***sed***, and *C* languages for the Viewlogic/Lager design and layout tools for a standard-cell library. However, the procedure has been generalized to be extendible to other libraries, such as the data-path library, and other design methodologies, such as analog design methods. The developed software executes on Sun Microsystems UNIX workstations.

## ACKNOWLEDGEMENT

First and before all, I thank the Lord for bestowing upon me his grace of mercy in order to accomplish this thesis.

I would like to extend my gratitude to Dr. Donald W. Bouldin for his great support, patience, help, advice, and guidance throughout my graduate studies and especially throughout the progress of my thesis.

I am thankful to Dr. Donald W. Bouldin for serving as the major professor on my committee. Thanks goes to Dr. Robert E. Bodenheimer for his friendship, counseling, believing in me, and serving on my committee. I would like to also thank Dr. Robert W. Rochelle for his enthusiasm, advice, friendship, challenge for a better future, and serving on my committee.

I am very thankful to my wife, Vicky Lynn Sawan, for her help in the preparation of my thesis, her extreme patience, and great support, which made this thesis possible.

I would like to thank my parents for their support and encouragement to succeed and be the person who I am. I would like to thank my whole family, especially my uncle John C. Sawan, for their support throughout my educational path. Special thanks goes to my great friends Jean-Pierre Abi-Karam and Charbel Haber for their unity and help in special times.

## TABLE OF CONTENTS


CHAPTER

I. INTRODUCTION ..... 1
- Introduction ..... 1
- New Simulator - At The Architectural/Structural Level ..... 2
- Motivation: The true story behind our Goal ..... 5
- ICGEN Global Router ..... 5
- First Scenario ..... 7
- Second Scenario ..... 9
- The New Scenario ..... 11

II. PRESENT SITUATION AND BACKGROUND ..... 13
- Prelude ..... 13
- Pre_Sim2Bat ..... 13
- What Is Standard Cell Architecture? ..... 13
- Circuit Design and Viewlogic Environment ..... 15
  - Logic Synthesis ..... 15
  - Manual Schematic Entry ..... 15
- Building Random Logic Standard-Cells ..... 23
  - Generating a ViewLogic Schematic ..... 23
  - Tips and Conventions on Creating Schematics ..... 25
  - Schematic simulation ..... 25
- Lager Tools ..... 25
  - Layout Generation Via Lager ..... 25
  - Oct database ..... 29
  - Standard Cell Layout Generation ..... 31
  - Layout simulation ..... 36
  - DMOCT Design Manager For The OCT Database ..... 36
  - DMpost- post processor for the LagerIV system ..... 38
  - IRSIM ..... 38
  - Summary ..... 40
- Simulation Methodology ..... 48
  - Traditional Methodology ..... 48
  - New Methodology ..... 50
- Facing The Problem ..... 50
  - Background ..... 50
  - Delay Modeling ..... 52
  - DELTIM ..... 53
  - CKTIM ..... 53

III. DEVEIOPMENT AND AUTOMATION OF THE SIM2BAT PROG. ..... 58
- Introduction ..... 58
- Features ..... 58
- Operating Environment ..... 59
- Running Sim2bat ..... 59

User Interface ....59
Multiple Design Simulations and Simulation Display ....59
Global Variables....60
Programming Language....60
The awk Tool ....60
The sed Stream Editor ....61
The Shell program ....61
Sim2Bat: The Run File ....62

IV. RESULTS AND DISCUSSION ....99
Checking The Results ....99
The Preprocessor (Pre-Layout) View2Sim ....99
The Post-Processor (Post-Layout) Bat2sim ....100
Method of Comparison ....100
Transitional and Gate Delay Comparison ....100
Experimental Results ....112
Test Case #1: EXAMP1 ....112
Graphical Results ....114
Transitional Delay Results ....114
Gate and Net Delays Results ....122
Test Case #2: DFTESTWIN ....122
Graphical Results ....122
Transitional Delay Results ....129
Gate and Net delays Results ....129
Test Case #3: DEC3TO8 ....129
Graphical Results ....129
Transitional Delay Results ....143
Gate and Net Delays ....143
Test Case #4: ADDER4WIN ....143
Graphical Results ....143
Transitional Delay Results ....163
Gate and Net Delays Results ....163
Accuracy and Error Calculation ....163
Types of Errors ....163
Intentional Errors ....177
Unintentional Errors ....177
Human Intervention ....177
Experimental Error ....178
Error Compensation Program ....179

V. SUMMARY AND CONCLUSIONS ....181
Future Work ....181
Contributions ....183
Conclusion ....184

BIBLIOGRAPHY ....185
APPENDICES ....188

APPENDIX A -- CKTIM PROGRAM ..... 189
Pre-Sim2Bat PROGRAM ..... 189
Sim2Bat PROGRAM ..... 191
APPENDIX B -- View2Sim PROGRAM ..... 203
APPENDIX C -- Sim2Bat PROGRAM ..... 209
APPENDIX D -- FIX PROGRAM ..... 215
EXPAND.NAWK PROGRAM ..... 215
TPHL.NAWK PROGRAM ..... 216
APPENDIX E -- COMP.C PROGRAM ..... 217

VITA ..... 219

## LIST OF FIGURES


Figure

1.1 Levels of Design and Simulation ....2
1.2 Overview of ICGEN ....6
1.3 First Scenario "ICGEN" ....8
1.4 Second Scenario "Standard Cell" ....10
1.5 New Scenario "Stdcell - CKTIM" ....12
2.1 Standard-Cell Architecture ....14
2.2 Pre-Sim2Bat Structured Flow-Chart ....17
2.3 Example User Directory Flow Diagram ....24
2.4 examp1 - Schematic Entry Via Viewsim ....26
2.5 Exampl.cmd - Command File ....27
2.6 examp1.plt - Waveform results ....28
2.7a Vb2oct Sample Run on examp1 Example ....30
2.7b List of examp1/ Current Project Directory ....30
2.8 examp1.log - Creating a SIV View ....32
2.9 Unexpended View of examp1 Circuit ....33
2.10 Expanded View of examp1 Circuit ....34
2.11 Stdcell - Standard Cell Layout Generator Outline ....35
2.12 Generating The Simulation Netlist ....37
2.13 examp1.sim - Simulation Netlist ....39
2.14 examp1.ext - Electrical Circuit Description ....41
2.15 examp1.al - Labels Present In Layout ....42
2.16 examp1.nodes - Nodes Present In Layout ....42
2.17 examp1.mag - Layout Netlist ....43
2.18 examp1.cmds - Command File for IRSIM ....46
2.19 examp1.ps - IRSIM Simulation Results ....47
2.20 Traditional Simulation Methodology ....49
2.21 New Simulation Methodology ....51
2.22 DELTIM - Net Delays Only Method Example ....54
2.23 CKTIM - Gate and Net Delays Method Example ....56
2.24 Back-annotated Gate Delays ....57
3.1 examp1.001p - Circuit Schematic ....63
3.2 Structured Flow-Chart for The Sim2Bat Algorithm ....64
3.3 examp1.nodes ....73
3.4 examp1magnod ....73
3.5 examp1.vsm ....74
3.6 examp1.net ....75
3.7 examp1.pkg ....75
3.8 examp1.dvf ....76
3.9 examp1.fnc ....76
3.10 examp1.drc ....76
3.11 examp1.net ....78
3.12 examp1.nettemp ....78
3.13 List of program FIVE. ....79

3.14 examp1.nettemp1 ....80
3.15 examp1nettemp2 ....80
3.16 labels.nodes ....81
3.17 examp1.magcompterm ....81
3.18 examp1.lognet ....82
3.19 examp1h2l.lognet ....86
3.20 examp1l2h.lognet ....86
3.21 examp1.hltemp ....87
3.22 examp1.lhtemp ....87
3.23 examp1.hltab ....88
3.24 examp1.lhtab ....88
3.25 List of Threemod Program ....89
3.26 examp1.hltab2 ....90
3.27 examp1.lhtab2 ....90
3.28 examp1.hltabdel ....92
3.29 examp1.lhtadel ....92
3.30 List of TWO Program ....93
3.31 examp1.hltabdel1 ....94
3.32 examp1.lhtabdel1 ....94
3.33 examp1.tphval ....94
3.34 examp1.tplhval ....94
3.35 List of FOUR Program ....95
3.36 examp1.tphltemp ....96
3.37 examp1tplhtemp ....96
3.38 examp1.tphltemp1 ....96
3.39 examp1.tplhtemp1 ....96
3.40 examp1.tphlh ....97
3.41 examp1-new.bat ....97
3.42 examp1-new.vsm ....98
4.1 examp1.1 - Wire Netlist ....101
4.2 examp1.trc - Transitional Delays Captured Via Viewsim ....103
4.3 examp1.trchltab2 - High-to-Low Delay Values ....105
4.4 examp1.trclhtab2 - Low-to-High Delay Values ....105
4.5 examp1pass1.bat - Back Annotation File ....105
4.6 examp1.trc - Generated Using New Delay Attributes ....106
4.7 examp1.trchltab2 - High-to-Low Delay Values ....108
4.8 examp1.trclhtab2 - Low-to-High Delay Values ....108
4.9 examp1pass2.bat - Back Annotation File ....108
4.10 First Path ....109
4.11 Second Path ....110
4.12 Third Path ....111
4.13 examp1.001p - Circuit Schematic ....113
4.14 examp1.plt - Original Pre-Layout Simulation ....115
4.15 examp1.ps - Layout Simulation ....116
4.16 examp1-new.plt - Post-Layout Simulation ....117
4.17 examp1.trchltab2 - Pre-Layout Transitional Delays (1st path) ....118
4.18 examp1.hltab2 - Layout Transitional Delays (2nd path) ....118

4.19 examp1.trchltab2 - Post-Layout Transitional Delays (3rd path) .....118
4.20 examp1.trclhtab2 - Pre-Layout Transitional Delays (1st Path) .....120
4.21 examp1.lhtab2 - Layout Transitional Delays (2nd Path) .....120
4.22 examp1.trclhtab2 - Post-Layout Transitionl Delays (3rd Path) .....120
4.23 examp1pass1.bat - Pre-Layout Backannotation Gate & Net Delays ....123
4.24 examp1-new.bat-Layout Backannotation Gate & Net Delays .....123
4.25 examp1pass2.bat - Post-Layout Backannotation Gate & Net Delays ...123
4.26 dftestwin.001p - Circuit Schematic .....125
4.27 dftestwin.plt - Original Pre-Layout Simulation (1st Path) .....126
4.28 dftestwin.ps - Layout Simulation (2nd Path) .....127
4.29 dftestwin-new.plt - Post-Layout Simulation (3rd Path) .....128
4.30 dftestwin.trchltab - Pre-Layout Transitional Delays .....130
4.31 dftestwin.hltab2 - Layout Transitional Dealys .....131
4.32 dftestwin.trchltab2 - Post-Layout Transitional Delays .....131
4.33 dftestwin.trclhtab2 - Pre-Layout Transitional Delays .....133
4.34 dftestwin.lhtab2 - Layout Transitional Delays .....133
4.35 dftestwin.trclhtab2 - Post-Layout Transitional Delays .....134
4.36 dftestwinpass1.bat - Pre-Layout Backannotation Gate & Net Delays ...135
4.37 dftestwin-new.bat - Layout Backannotation Gate & Net Delays .....136
4.38 dftestwinpass2.bat - Post-Layout Backannotation Gate & Net Delays .137
4.39 dec3to8.001p - Circuit Schematic .....139
4.40 dec3to8.plt - Original Pre-Layout Simulation (1st Path) .....140
4.41 dec3to8.ps - Layout Simulation (2nd Path) .....141
4.42 dec3to8-new.plt - Post-Layout Simulation (3rd Path) .....142
4.43 dec3to8.trchltab2 - Pre-Layout Transitional Delays .....144
4.44 dec3to8.hltab2 - Layout Transitional Delays .....145
4.45 dec3to8.trchltab2 - Post Layout Transitional Delays .....146
4.46 dec3to8.trclhtab2 - Pre-Layout Transitional Delays .....148
4.47 dec3to8.lhtab2 - Layout Transitional Delays .....149
4.48 dec3to8.lhtab2 - Post-Layout Transitional Delays .....150
4.49 dec3to8pass1.bat - Pre-Layout Backannotation Gate & Net Delays .....152
4.50 dec3to8-new.bat - Layout Backannotation Gate & Net Delays .....154
4.51 dec3to8pass2.bat Post-Layout Backannotation Gate & Net Delays .....156
4.52 adder4win.001p - Circuit Schematic .....159
4.53 adder4win.plt - Original Pre-Layout Simulation (1st Path) .....160
4.54 adder4win.ps - Layout Simulation (2nd Path) .....161
4.55 adder4win-new.plt - Post-Layout Simulation (3rd path) .....162
4.56 adder4win.trchltab2 - Pre-layout Transitional Delays .....164
4.57 adder4win.trchltab2 - Layout Transitional Delays .....165
4.58 adder4win.trchltab2 - Post-Layout Transitional Delays .....166
4.59 adder4win.trclhtab2 - Pre-Layout Transitional Delays .....168
4.60 adder4win.lhtab2 - Layout Transitional Delays .....169
4.61 adder4win.trclhtab2 - Post-Layout Transitional Delays .....170
4.62 adder4winpass1.bat Pre-Layout Backannotation Gate & Net Delays ..172
4.63 adder4win-new.bat - Layout Backannotation Gate and Net Delays .....73
4.64 adder4winpass2.bat Post-Layout Backannotation Gate & Net Delays .174

## LIST OF TABLES


Table

2.1 Standard Cell Functions Supported by ITD .....16
2.2 Viewsim and IRSIM Common Commands .....45
4.1 examp1 - High-to-Low Transitional Delays Summary .....119
4.2 examp1 - Low-to-High Transition delays Summary .....121
4.3 examp1 TPHL & TPLH Transitional Gate and Net Delays Summary .124
4.4 dftestwin High-to-Low Transition Delays Summary .....132
4.5 dftestwin - Low-to-High Transitional Delays Summary .....134
4.6 dftestwin TPHL & TPLH Transitional Gate & Net Delays Summary .138
4.7 dec3to8 - High-to-Low Transitional Delays .....147
4.8 dec3to8 - Low-to-High Transitionl Delays Summary .....151
4.9 dec3to8 TPHL & TPLH Transitional Gate & Net Delays Summary .....158
4.10 adder4win High-to-Low Transitional Delays Summary .....167
4.11 adder4win Low-to-High Transitional Delays Summary .....171
4.12 adder4win TPHL&TPLH Transitional Gate & Net delays Summary ..175
4.13 ERROR Calculation and Total Performance Results .....176

# CHAPTER I

## DEVELOPMENT AND AUTOMATION OF A PROCEDURE FOR POST-LAYOUT TIMING ANALYSIS OF INTEGRATED CIRCUITS

### *Introduction*

Recent advances in integrated circuit fabrication technology have allowed larger and more complex designs to form complete systems on single VLSI chips. Design complexities equivalent to 100,000 to 250,000 transistors have created a combinatorial explosion of details that are a major limitation for the designer to foresee all the consequences and cost of each design decision. To minimize the likelihood of costly errors, the designs are increasing in need of complete timing analysis simulation tools in order to verify the design before committing to the costly silicon fabrication.

Conventional IC design methodology flows from a set of ideas to an architectural specification, then a logic design, and then to a circuit design that is in turn converted into a layout topology design (see Figure 1.1). To verify the equivalence of lower level design with the architectural specification the same set of test vectors would be used to run a gate level simulation from the logic design or a switch-level simulation from the circuit design. To verify the equivalence between the layout topology and the circuit schematics, a layout extraction program would be used to convert the layout into a transistor net-list format that is either compared to the circuit design or used to run a switch-level simulation. These traditional steps require much cross checking and rechecking at various levels.

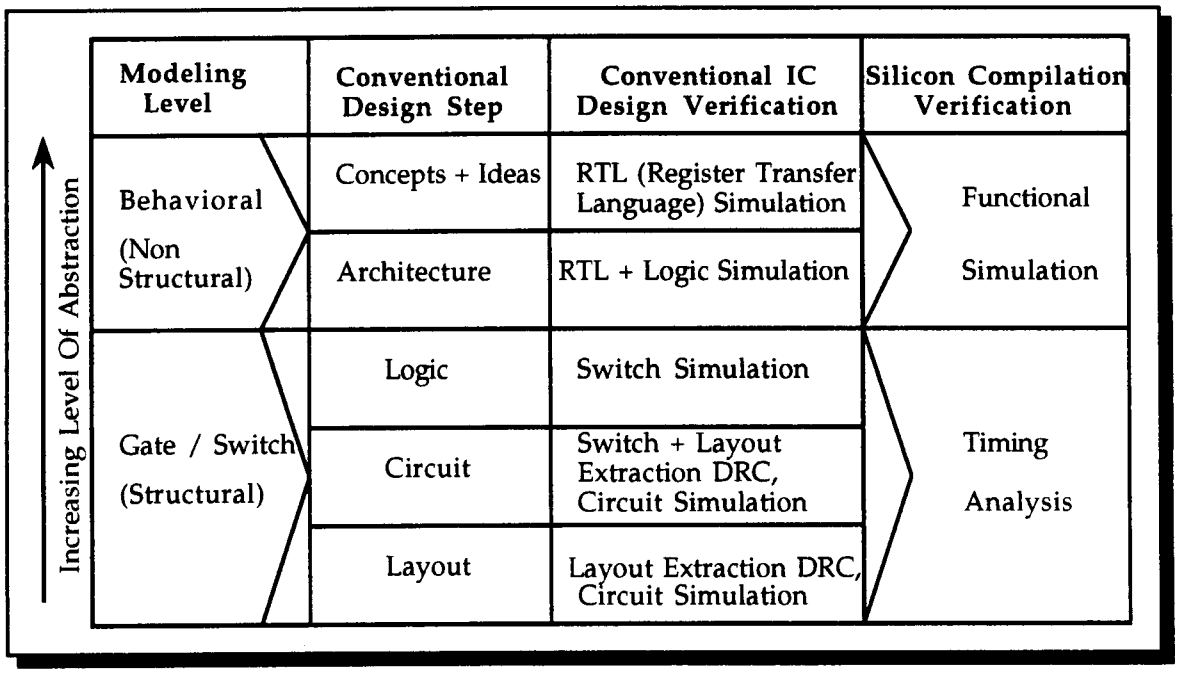



**Figure 1.1 Levels of Design and Simulation**

Most of the verification efforts in conventional IC design are concentrated on checking the equivalence between the various design levels; however, since silicon compilation replaces all the manual steps between architecture design and layout design, the associated equivalence checking becomes obsolete. Hence, only functional simulation at the architectural level, to verify the architectural design, and timing analysis at the layout level, to verify the timing constraint and performance, are needed. [Chen]

We conclude from this scenario that post-layout information is becoming extremely vital for accomplishing a successful design with working silicon. In other words, post-layout timing analysis not only is required to accompany the functional simulation but goes further to become a must and plays a greater role in reducing the design time and overall design process.

One common design practice that functional timing designs are handled separately, even though there is some intermingling of the two activities in the iteration loop that consists of *do functional design*, then *do timing analysis*, and *iterate*. When working with synchronous circuits, engineers like to treat functional and timing designs as separable processes; however, designers prefer that timing delays to be computed concurrently with functional evaluations by working with estimated gate propagation times.

Although the majority of the design is modeled at the highest level that can be most efficiently simulated, timing analysis must be done at the lowest physical level, including parasitic capacitances and resistances.

This gap that exists between the functional and timing simulations needs to disappear, and a bridge needs to be built. Hence our goal is to replace the pre-layout functionality timings with more accurate post-layout better silicon timings. The designer needs to use only one simulator, the functional simulator, which is transformed into a complete simulation tool (function and timing), thus maintaining a constant user interface.

*New Simulator - At The Architectural/Structural Level*

The proposed simulator needs to be considerably efficient in terms of run-time and computer resources. Its circuit models are generated by block compilers and use high-level logic primitives such as gates, flip-flops, adders and decoders, and they are encoded without any delay information. The compiled circuit can be simulated at any level of the design hierarchy, since the system supports hierarchical signal naming. Signal names that are completely internal to a module are invisible to other modules and will not cause name conflicts. Signal names that are used as connectors at the module level are visible to other blocks and modules at the same level. The simulator is controlled using a menu and form-driven interface such that input test vectors may be entered interactively at the terminal or read from a file in batch mode. ***Viewsim*** and ***IRSIM*** simulators combined can provide such a simulator with some changes in order to accomplish our goal.

The new environment needs to account for the net delays connecting the modules, which is a great factor in the timing analysis process of a design. The new simulator should allow the designer to use a single high level user interface for both pre-layout and post-layout timing. The logical choice is to choose the ***Viewsim*** simulator since it runs from within the Viewlogic environment where the design schematics are captured. One advantage of using ***Viewsim*** is the existence of many similarities with the layout simulator ***IRSIM*** such as similar commands and eventually similar command files. ***IRSIM*** itself cannot be chosen to be the user interface since it acts on a extracted net-list of the layout and does not interface with the schematic directly.

*Motivation: The True Story Behind Our Goal*

Any CMOS design, whether it is digital or analog, needs the capability to use variable-sized transistors in addition to any other modules such as named gates, decoders, and state machines. These transistors do not exist in our Lager libraries. Although they can be created, a complete library will be endlessly long since a transistor is controlled by a width parameter (W) and a length parameter (L) which are numbers that could range from 0 to 1,000. To have a 1000x1000 combinations of transistors is not very practical. However, if these combinations could be generated automatically, than the need of a transistor library would cease to exist. Given that the sized transistors can be created automatically, the need of a special router arises. One such router exists and is called "***ICGEN***".

*ICGEN Global Router*

***ICGEN***, developed at the University of Illinois (Urbana-Champaign), is a global router for SOG (Sea of Gates) and custom logic arrays using the Zero-to-One optimization program called ***ZOOM***.[Thaik] It performs "over the cell" channelless SOG routing without any net-ordering. A global routing information extraction step to capture relevant routing information and filter out extraneous details precedes the actual global routing. The nets are then decomposed into different kinds of path segments: vertical, horizontal, and unrestricted and handled separately in three routing phases. In each phase, possible path segments are identified and the optimal paths that maximize the chosen benefit function are selected. ***ICGEN*** can handle very large SOG routing problems with superior packing density and computational efficiency. An overview of ***ICGEN*** is shown in Figure 1.2.

***ICGEN*** interfaces to ***TimberWolf***, a place and route software that is part of the LagerIV tool set currently available at the University of Tennessee.

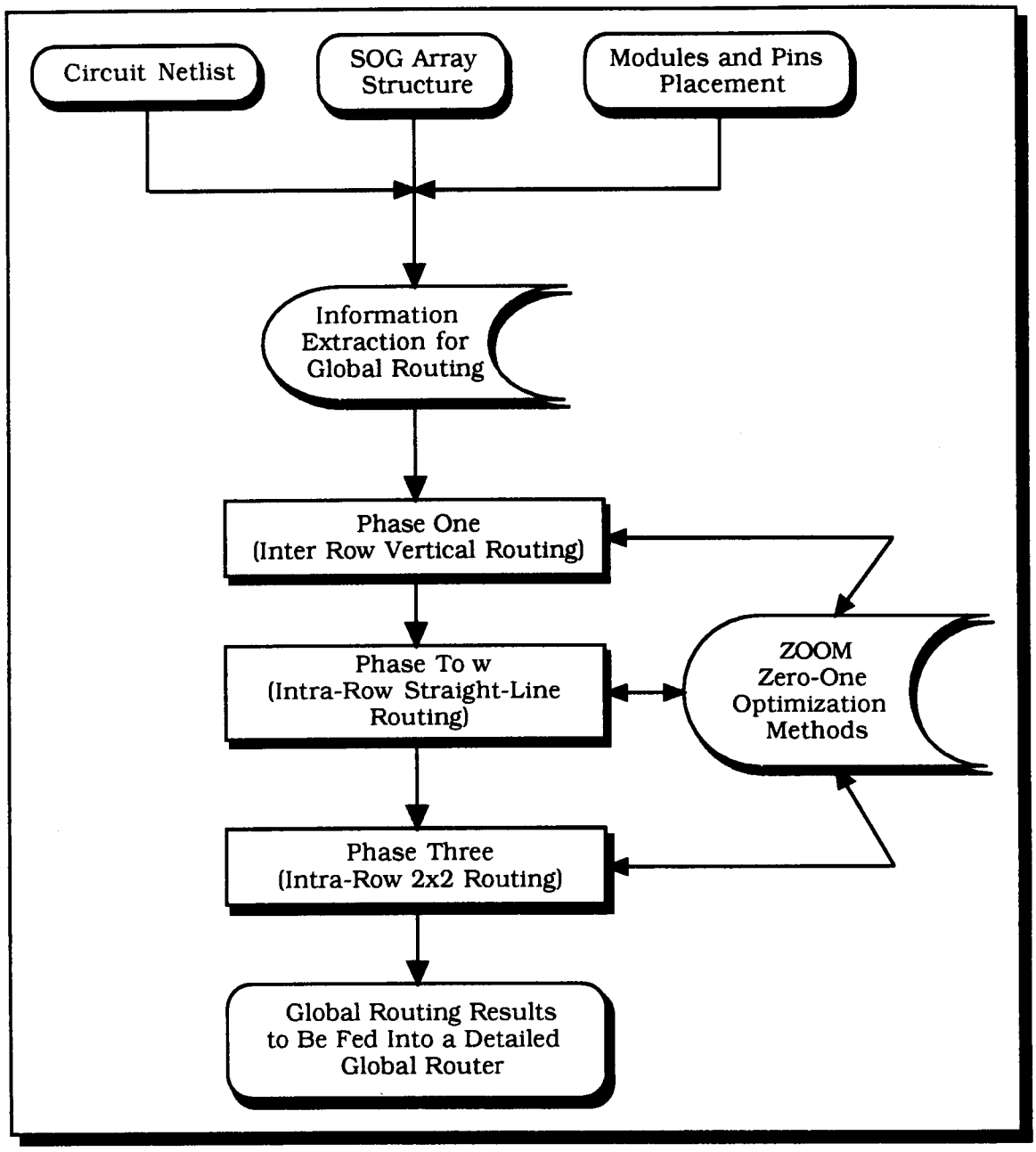



**Figure 1.2 Overview of ICGEN**

*First Scenario*

One of the initial approaches was to develop a basic component (cell) generator, where the generated cells are composed of FETs (Field Effect Transistors) of variable sizes and can be employed in any CMOS (Complementary MOSFET) analog design. Next, these basic components would be fed into ***ICGEN*** since it can alter the sizes of the cells by modifying the transistor sizes and re-routing the internal connections. The results of ***ICGEN*** would then be piped into a place and route program (e.g. ***TimberWolf)*** that globally routes the cell-to-cell pins. An extraction of the circuit net-list would then be done using a layout extractor such as the ***Magic*** extractor. Then the design would be simulated using ***IRSIM*** after the simulation net-list was extracted via an extractor such as ***Ext2sim***. The final step would be to write a filter that back-annotates the layout simulation results to the design in either a schematic or a net-list format. This scenario had high hopes but was not possible due to the unavailability of the program ***ZOOM*** which plays a major role in the ***ICGEN*** program. Another hindering obstacle against following the original approach was that most of the interfaces between all the proposed programs such as ***ICGEN*** and ***TimberWolf*** are not available. These obstacles deterred us from taking this approach and thus another scenario was outlined (see Figure 1.3).

This first scenario exists commercially and is provided by Cadence Design Systems, Inc. as a software product named **Analog** ***Artist***.[Cadence] This package includes ***cdsSpice*** (a version based on University of California, Berkeley ***SPICE2***), ***HSPICE***, ***Spectre*** (contains algorithms that aim at eliminating DC convergence problems and can handle very large circuits). It has the capability of integrating other simulators handled on a custom basis and provide DC, transient and AC simulation noise analysis. Also, ***Artist*** provides extensive results analysis. First, its graphic output has many features such as post-simulation plots, real-time (marching) waveforms, easy

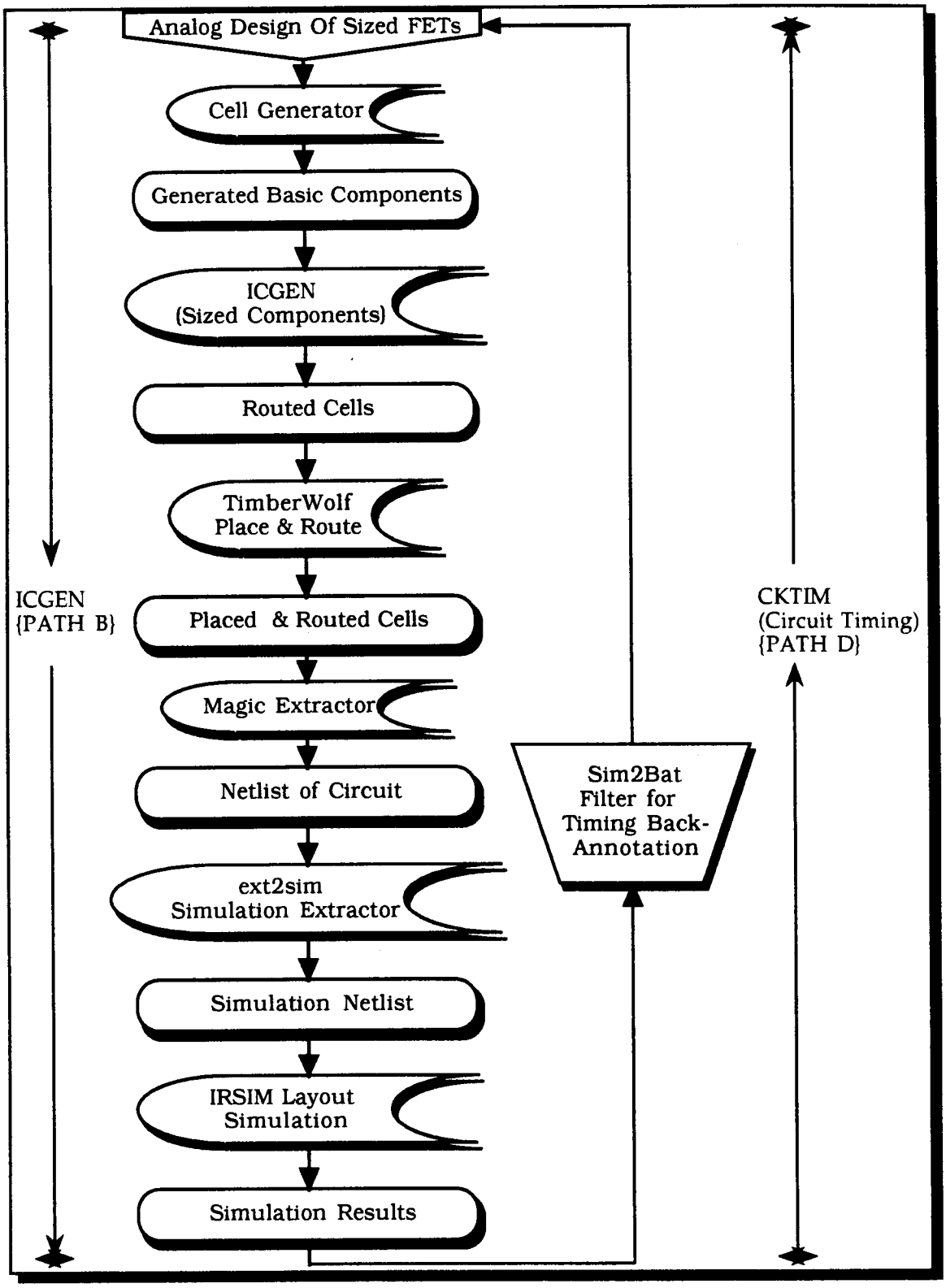



**Figure 1.3 First Scenario "ICGEN"**

recovery of old simulation results for comparison, and any two curves can be plotted against each other. Second, ***Artist*** incorporates a post-processor that allows the user to create complex expressions for both calculation and plotting. Third, the results can be back-annotated onto the schematic to show DC voltages, node numbers, operating points and other information.

***Artist*** is a great program that provides many features for the user; however, it is a very expensive software package that requires other Cadence tools like ***cds*** (similar to Viewlogic's environment) in order to become effective. The University of Tennessee does not currently possess any Cadence tools, which form a challenge to develop similar tools with similar capabilities in the Viewlogic environment. The new tool would be a low-cost version with similar without the (though not as comprehensive) capabilities of ***Artist***.

*Second Scenario*

Given the challenges to be faced in accomplishing the "Analog design" scenario, a "standard cell design" Scenario was developed. First the components are standard cell library cells that do not change, thus there is no need for ***ICGEN***. The standard cells are then placed and routed using the ***TimberWolf*** program. After the layout is generated, a special extractor has to be developed in order to extract nets only; these nets will have attached delays provided by a layout simulator (e.g. ***IRSIM***). Next the net-delay timing information is back-annotated into the design in order to be viewed with a schematic logic simulator or a net-list format. This scenario is depicted in Figure 1.4.

The standard cells and their place and route program are available via the Lager tools; however, the extractor needs to be written since it needs to be tailored to extract nets information only. This task is a major task since it involves modifying the Lager tools set. Thus, the third scenario is chosen which is a combination of both the first and second scenarios.

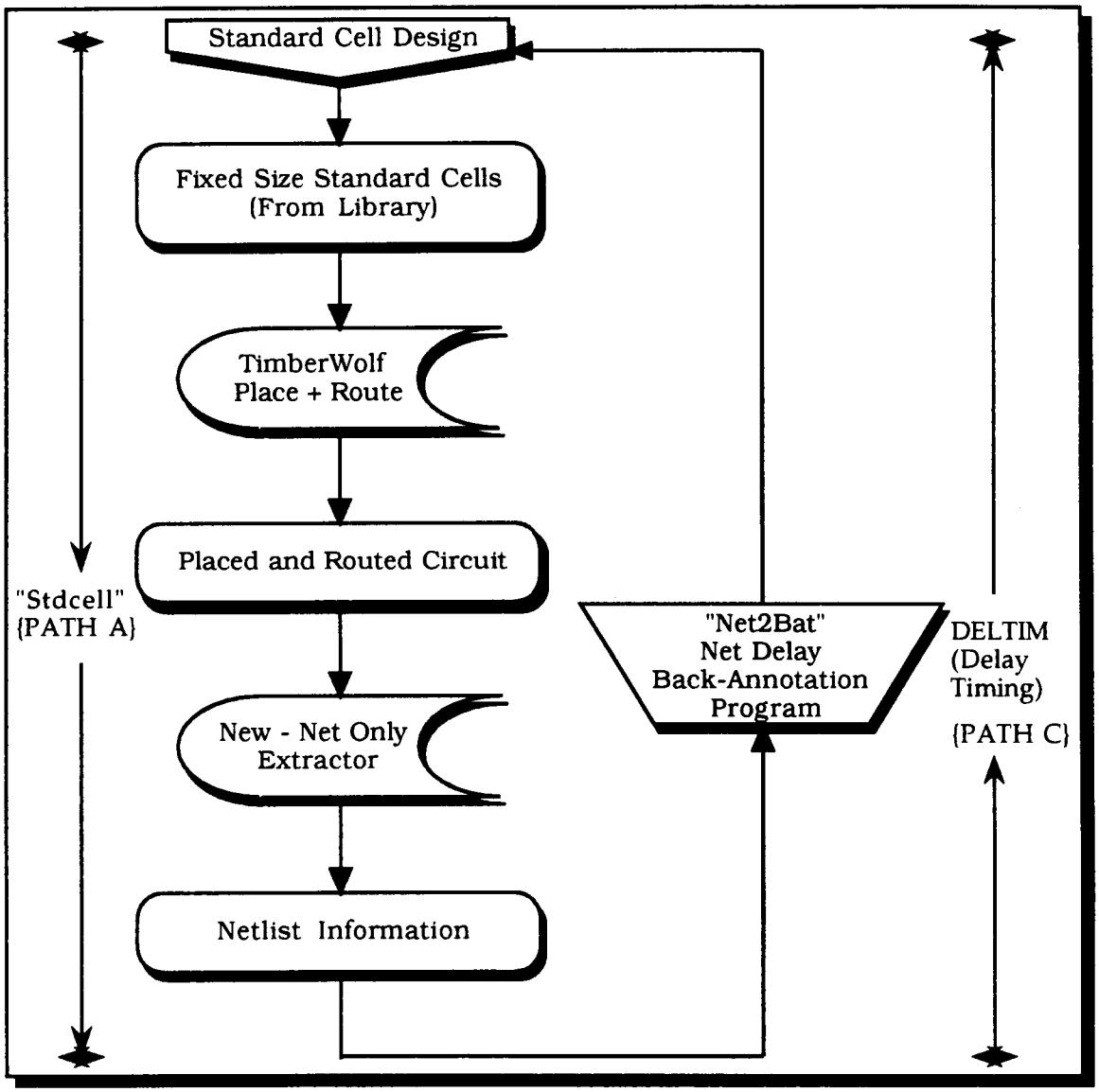



**Figure 1.4 Second Scenario "Standard Cell"**

*The New Scenario*

Given the difficulties listed in the previous two scenarios which are the pairs of paths **B** and **D** (***ICGEN*** and ***CKTIM***) (see Figure 1.3) and paths **A** and **C** (***Stdcell*** and ***DELTIM***) (see Figure 1.4), the new goal was formalized to accomplish path **B** by developing path **D** and using path **A**. The new strategy is outlined in Figure 1.5. Thus, the structural design is translated into physical net-list (path **A**) that is back-annotated into the schematic in the form of timing delays (path **D**). The extracted circuit delays are modified in order to incorporate net and gate delays.

In conclusion, given that ***ICGEN*** is provided with the ***ZOOM*** program and a Components generator, by accomplishing ***CKTIM***, the Analog Design of sized FETS scenario becomes easily attainable.

This thesis is divided into five chapters, beginning with the introduction. Chapter II describes each step outlined by ***Pre-Sim2Bat***, which includes describing briefly the circuit and layout generation methods, defining a new simulation methodology, and outlining the two delay modeling methods ***DELTIM*** and ***CKTIM***.

Chapter III discusses the development of the ***Sim2Bat*** program and outlines the various steps involved by providing Input/Output files scenarios. The goals and motivations of this utility are discussed in chapter I.

Chapter IV discusses the results of ***Sim2Bat*** and provides four test cases in order to measure the success of the methodology. Tables of results are given and the sources of errors in the process are discussed.

Chapter V summarizes the automated procedure for improved timing analysis of circuits at the schematic level. This chapter also presents the pros and cons of ***Sim2Bat*** and the overall contributions.

The Appendix contains the codes for ***CKTIM*** programs and any other information needed by the curious readers.

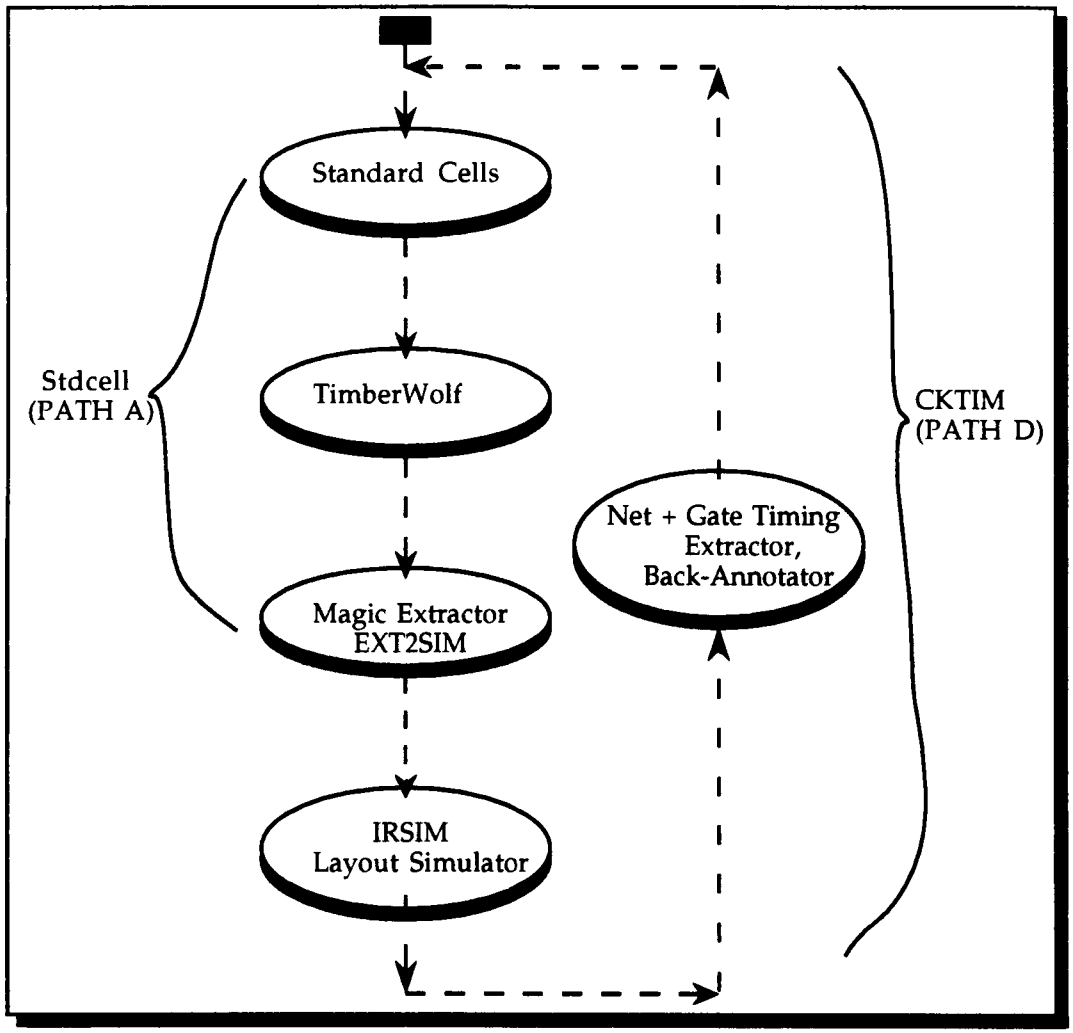



**Figure 1.5 New Scenario "Stdcell - CKTIM"**

# CHAPTER II

## PRESENT SITUATION AND BACKGROUND

### *Prelude*

Currently, the Lager/Viewlogic interface (Lager 3.0) supports schematic-driven methodology, standard-cell and datapath libraries, VHDL models, and many other features; however, it does not support accurate timing models and switch-level simulation done through ***IRSIM***.

### *Pre_Sim2Bat*

This section is a description of the ***Pre-Sim2Bat*** procedure; it should provide a good insight on the various steps, through which the design progresses. This includes details on how to proceed from entering the schematics into Viewlogic to generating the layout though ***Vb2oct*** and ***DMOCT*** programs, in other words, the essence of capturing a logic schematic and transforming it into silicon layout. Circuit simulation is also explained with helpful details for using the logic simulator ***Viewsim*** and the layout simulator ***IRSIM***.

The following material in this chapter will refer to an example called *"examp1"* unless otherwise specified.

### *What Is Standard Cell Architecture?*

The standard cell architecture, presently used in most popular ASIC circuits, divides the layout area into a set of parallel rows separated by routing channels, as shown in Figure 2.1. Each row consists of a number of predefined leaf cells from the standard cell library, which contains at least one standard cell for each component in the circuit schematic. All leaf cells have

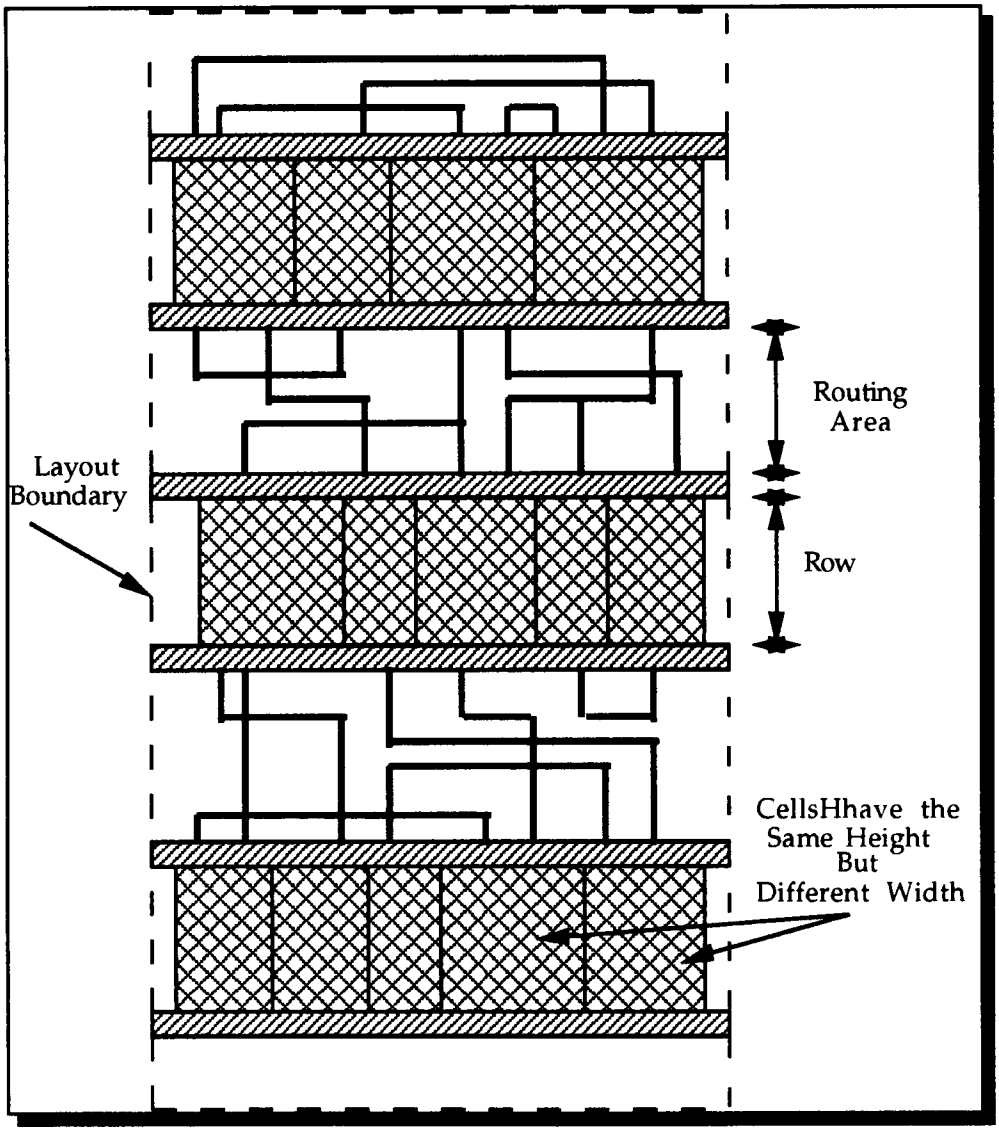



Figure 2.1 Standard-Cell Architecture

the same fixed height but may have different widths. Power connections are on the left and right boundaries so that cells can be abutted.

The current standard cell library (v2.3) supported by ITD (Institute of Technology Development) supports the functions depicted in Table 1.1. A structured flow-chart of ***Pre-Sim2Bat*** is best illustrated in Figure 2.2.

### *Circuit Design And Viewlogic Environment*

#### *Logic Synthesis*

Viewlogic tools allow the designer to specify circuits using two methods. On one hand, a circuit can be described indirectly through a behavioral description structured (BDS) language. The net-list then is synthesized, which produces a logic-level (state machine, decoders, random logic blocks) schematic. The main advantage of logic synthesis via BDS is that logic minimization and library mapping is done automatically. However, this method is not good for memory elements or arithmetic operations mapping since the results produced by the synthesizer might differ greatly from the desired goal. Another disadvantage of the BDS language is that the designer must first synthesize the net-list and then use the ***viewsim*** logic simulator in order to simulate the results. Moreover, the BDS language cannot be simulated from within ViewLogic environment.

#### *Manual Schematic Entry*

The second method of generating a circuit is by manually specifying the exact cells and interconnects via Viewlogic's ***viewdraw***. The main advantage of this method is that what you see is what you get and simulation via ***Viewsim*** is possible; also, a designer specifies and decides which symbol of

Table 2.1 Standard Cell Functions Supported by ITD

| Cell Name | Cell Description |
|---|---|
| aof2201 | 2,2 AND/OR Mux |
| aof2301 | 2,3 AND/OR Mux |
| aof3201 | 3,2 AND/OR Mux |
| aof4201 | 4,2 AND/OR Mux |
| aoif2201 | 2,2 AND/OR Mux |
| blf00001 | 2,1 AND/OR Mux |
| blf00101 | 2,1 OR/NAND Mux |
| buff100 | High Impedance Buffer |
| buff101 | Non-Inverting Buffer |
| buff121 | Tri-State Buffer |
| delf011 | Delay Cell |
| dfbf311 | D-Flip Flop with S, R, Q & QB |
| dfnf311 | D-Flip Flop with Q & QB |
| dfrf301 | D-Flip Flop with Asynchronous R & Q |
| dfrf311 | D-Flip Flop with Asynchronous R, Q, & QB |
| faf001 | Full Adder |
| invf100 | High Impendance Inverter |
| invf101 | Inverter |
| invf103 | 3X Inverting Buffer |
| invf104 | 4X Inverting Buffer |
| invf121 | Tri-State Inverter |
| invf201 | Dual Inverter |
| labf111 | NAND Latch |
| labf211 | NOR Latch |
| larf310 | Clocked Latch |
| lrbf201 | Logic Reference Cell |
| ldf001 | RC Load Cell |
| muxf201 | Data Select |
| nanf201 | 2 Input NAND |
| nanf211 | 2 Input NAND/AND |
| nanf251 | A or B-not Decoder |
| nanf301 | 3 Input NAND |
| nanf311 | 3 Input NAND/AND |
| nanf401 | 4 Inuput NAND |
| nanf411 | 4 Input NAND/AND |
| norf201 | 2 Input NOR |
| norf211 | 2 Input OR/NOR |
| norf251 | A-not AND B Decoder |
| norf301 | 3 Input NOR |
| norf311 | 3 Input OR/NOr |
| norf401 | 4 Input NOR |
| oaif2201 | 2,2 OR/NAND MUX |
| orf401 | 4 Input OR |
| pudf000 | Pull-Down |
| puuf000 | Pull-Up |
| slrf310 | Scan Latch W/Reset |
| swcf020 | Transmission Gate |
| swcf120 | Transmission Gate |
| xnorf201 | Exclusive NOR |
| xorf201 | Exclusive OR |

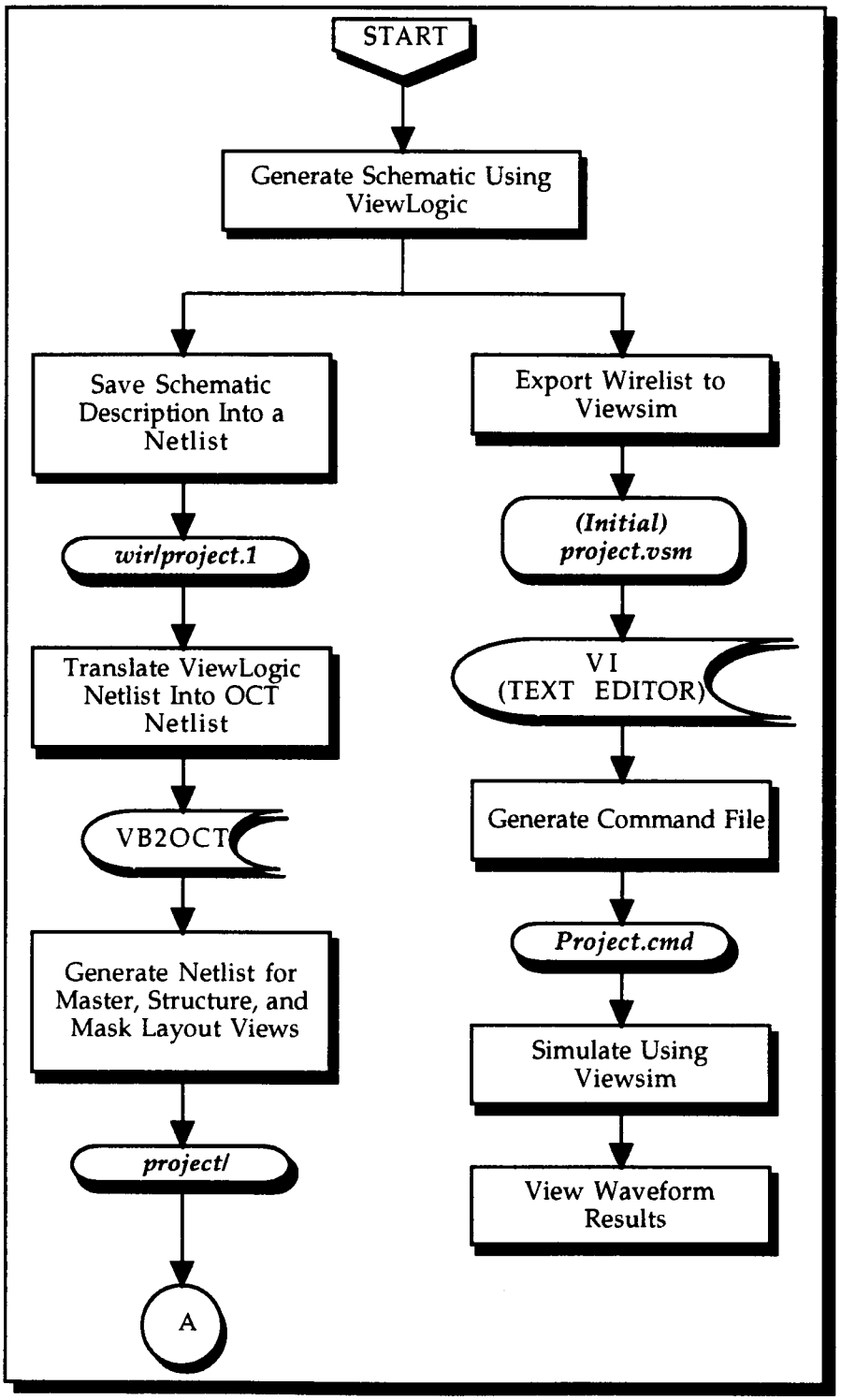
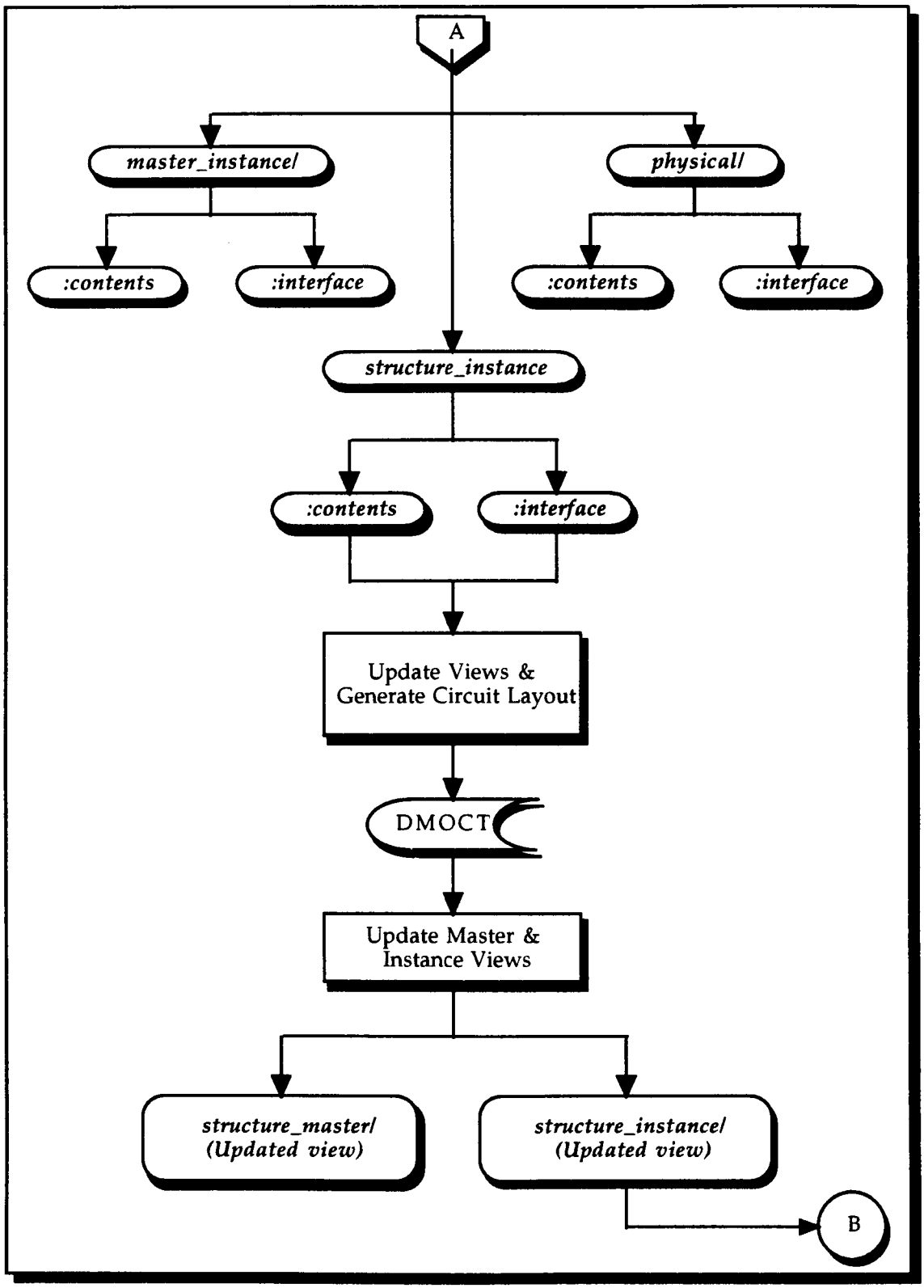
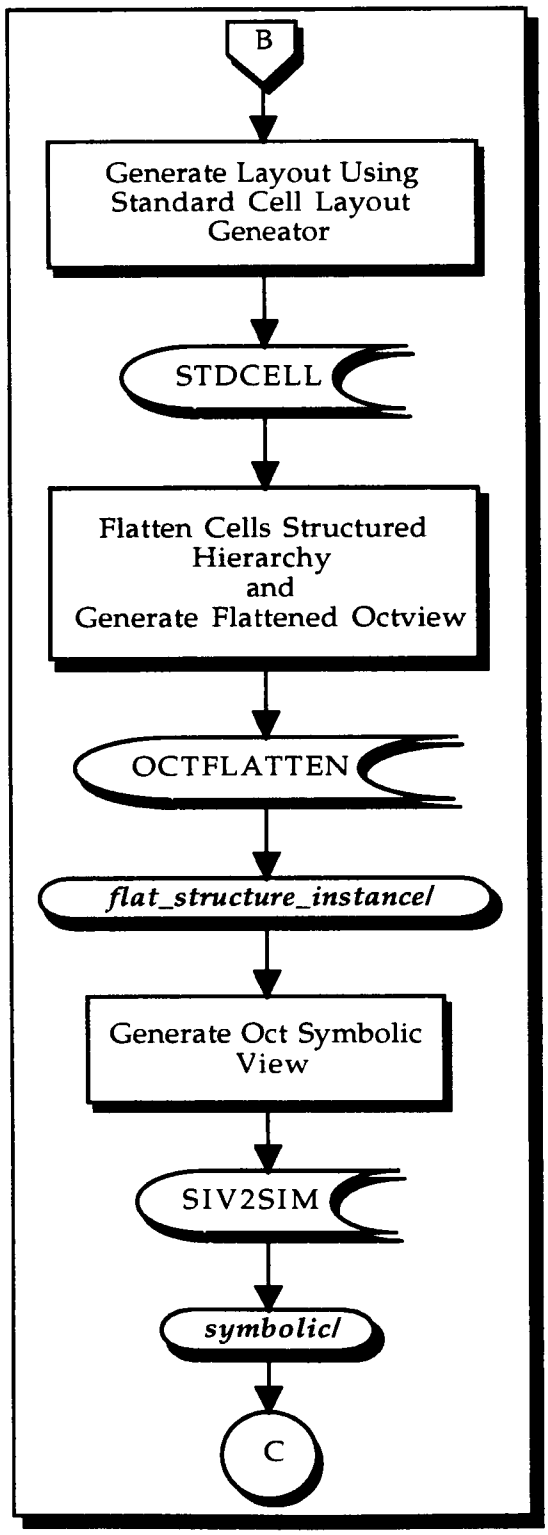


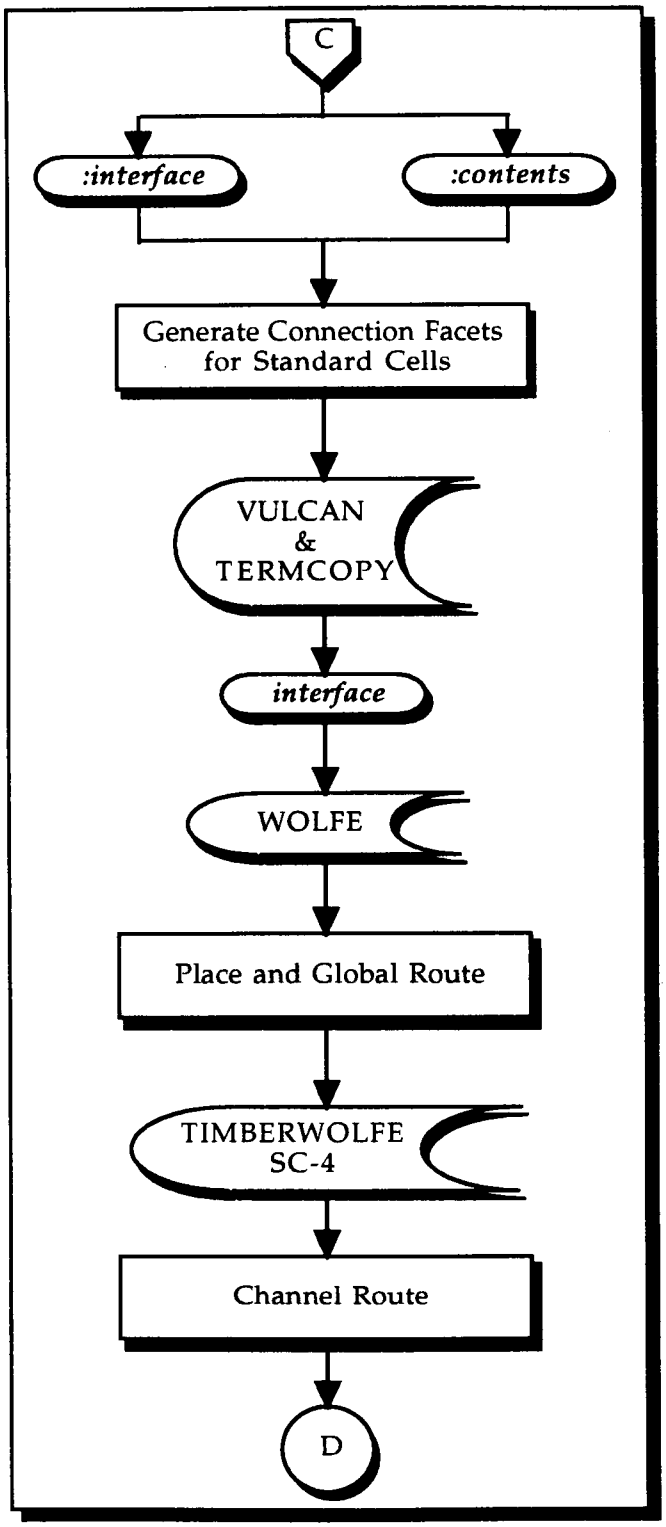

Figure 2.2 Pre-Sim2Bat Structured Flow-Chart


A
master_instance/
physical/
:contents
:interface
:contents
:interface
structure_instance
:contents
:interface
Update Views &
Generate Circuit Layout
DMOCT
Update Master &
Instance Views
structure_master/
(Updated view)
structure_instance/
(Updated view)
B

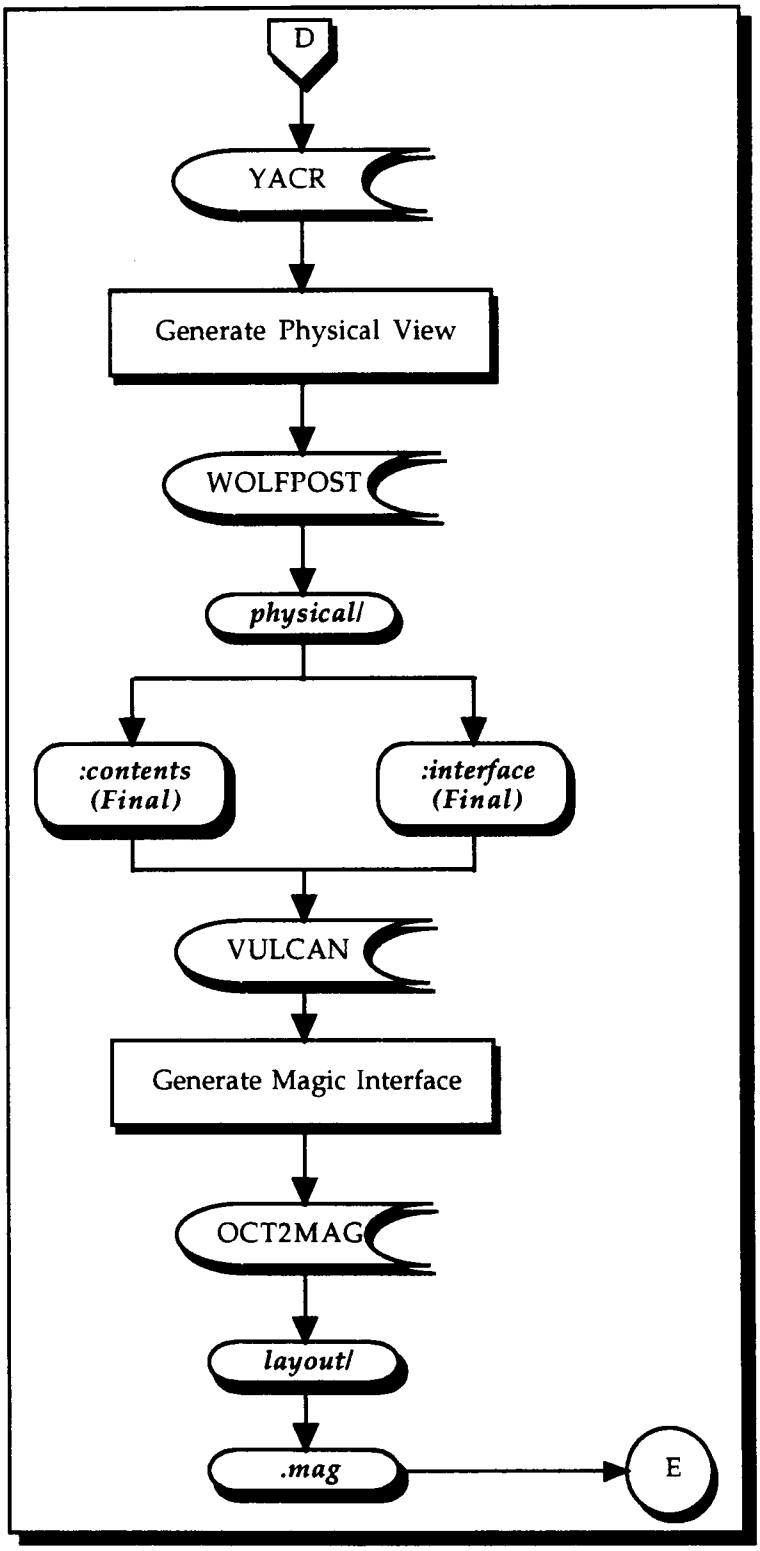
Continued


B
Generate Layout Using Standard Cell Layout Geneator
STDCELL
Flatten Cells Structured Hierarchy and Generate Flattened Octview
OCTFLATTEN
flat_structure_instance/
Generate Oct Symbolic View
SIV2SIM
symbolic/
C

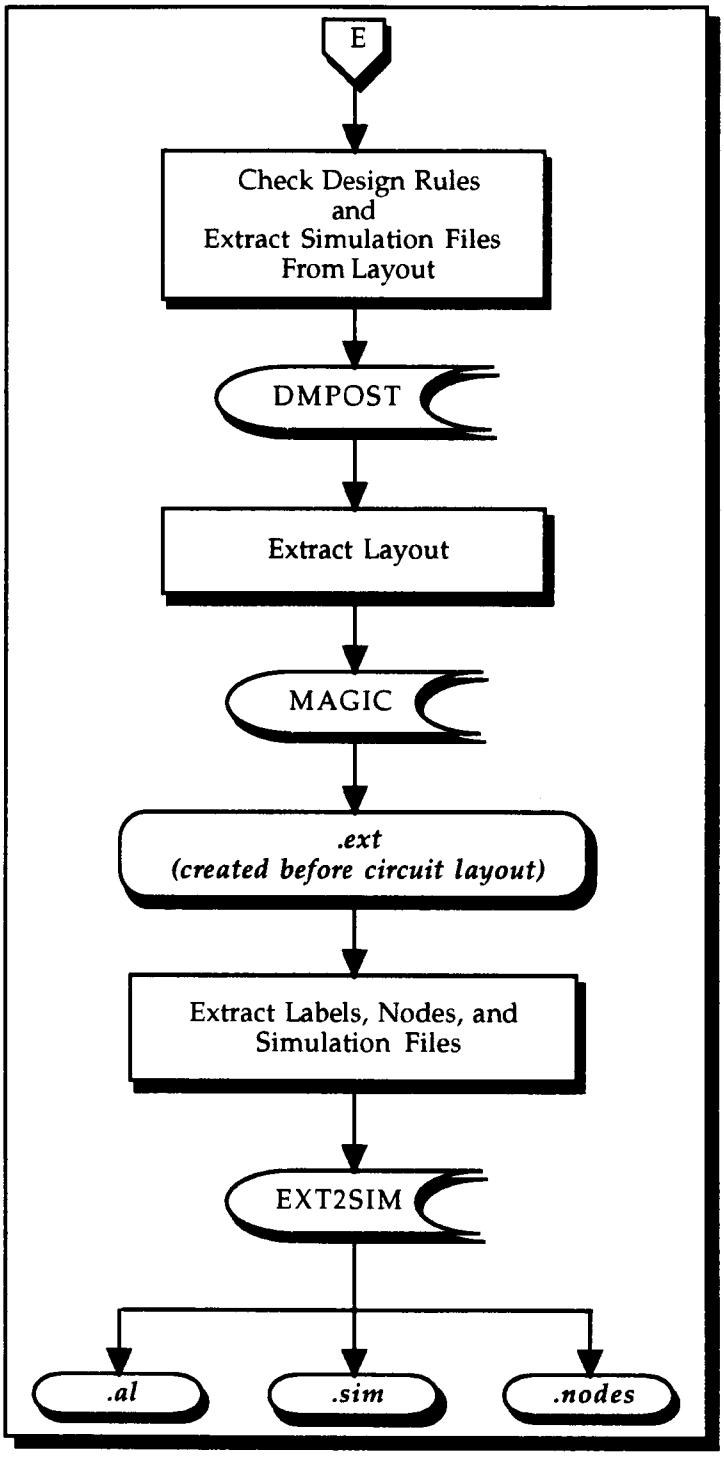
Continued


C
:interface
:contents
Generate Connection Facets for Standard Cells
VULCAN & TERMCOPY
interface
WOLFE
Place and Global Route
TIMBERWOLFE SC-4
Channel Route
D

Continued


D
YACR
Generate Physical View
WOLFPOST
physical/
:contents
(Final)
:interface
(Final)
VULCAN
Generate Magic Interface
OCT2MAG
layout/
.mag
E

Continued


E
Check Design Rules
and
Extract Simulation Files
From Layout
DMPOST
Extract Layout
MAGIC
.ext
(created before circuit layout)
Extract Labels, Nodes, and
Simulation Files
EXT2SIM
.al
.sim
.nodes

Concluded

which library is needed very easily. This method is advantageous especially in debugging since the circuit is organized in groups and blocks of which the user defines the location and placement. This is very difficult to accomplish via BDS which throws the synthesized logic into a sea of gates with no specific logical or sequential order.

### *Building Random Logic Standard-Cells*

#### *Generating a ViewLogic Schematic*

The first time a designer brings up ***Workview***, three directories get created. They are the symbol files directory (sym), the schematic files directory (sch), and the net-list files directory (wir).[Workview] The *schematic* file is the pictorial representation of the design. It contains the components, nets, text, attributes and the graphical placement information. The *symbol* file is the pictorial representation of the interface to a schematic. In order to use a schematic as a component, a symbol is created with defined pin connections that match the I/O's of the schematic. The *wir* file provides an ASCII text description of the schematic sheet and contains any error messages that were reported during the check; furthermore, it describes the components and net connections without any graphical information. A wir file is used by ***Viewsim*** in order to produce the simulation net-list (see Figure 2.3). The files in the *sch*, *sym*, and *wir* directories would be named *examp1.1*, *examp1.1*, *examp1.1* for a design named *examp1*; these files are in ASCII format which facilitates the task of manipulating the database. Other schematic capture programs such as Cadence "***opus***" tools store the information in binary format which executes faster but hinders third-party software development for the Cadence environment.

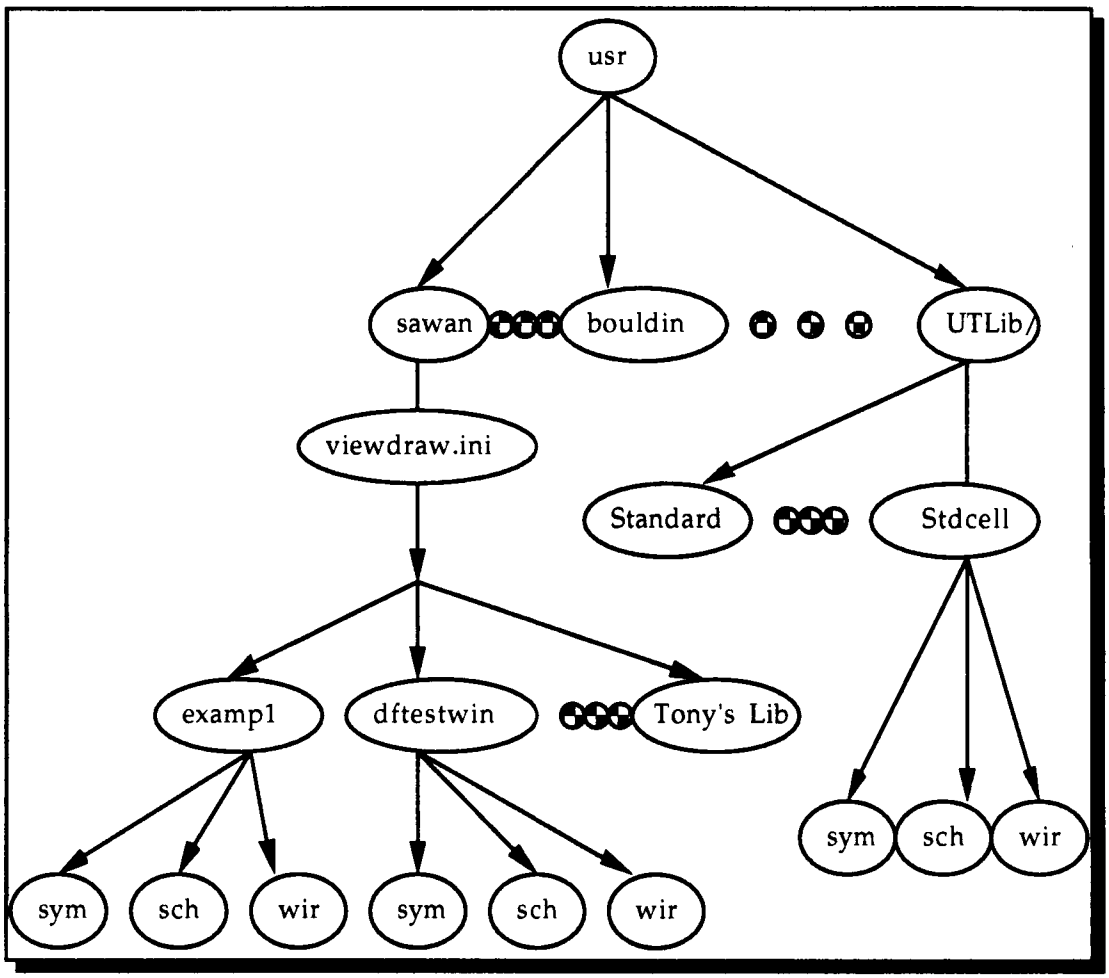



Figure 2.3 Example User Directory Flow Diagram

### *Tips and Conventions on Creating Schematics*

In order for the "***Icongen***" program to generate symbols automatically for schematics, formal terminals must be attached to the I/O pins such as fti (formal terminal input), fto (formal terminal output) and other formal terminals like input and output busses (ftiN and ftoN), where N stands for the buss size (see Figure 2.4). These terminals MUST be labeled (***Sim2Bat*** requires labeling too) and the internal nodes should be labeled despite the fact that our program handles Viewlogic-generated names. Once the design is entered using ***Viewdraw***, it must always be saved. This creates the *wir* and *sch* net-lists which are needed by the programs such as ***Vb2oct*** and ***Sim2Bat***. One symbol that is required is the *smv* symbol. This is the component, not an instance, that passes the needed information (attributes) to the Lager tool-set.

### *Schematic Simulation*

The simulator ***Viewsim*** provided by Viewlogic can be a stand-alone simulation tool and does not have to go through the Lager interface; however, the current VHDL models which are provided for the standard cells are functional only and do not support accurate timing information. For this reason it is important to use the Lager tools in order to get accurate timings. After exporting the wirelist to ***Viewsim*** which creates a "*vsm*" simulation file, the user has to control and direct the simulation by creating a command file "*cmd*" (see Figure 2.5). Using ***Viewvave*** the designer can look at the waveform (see Figure 2.6) normally saved in a "*wfm*" file.

## *Lager Tools*

### *Layout Generation Via Lager*

After the designer has verified the functionality of the design using ***Viewsim***, Lager interface tools are used to generate the circuit's physical

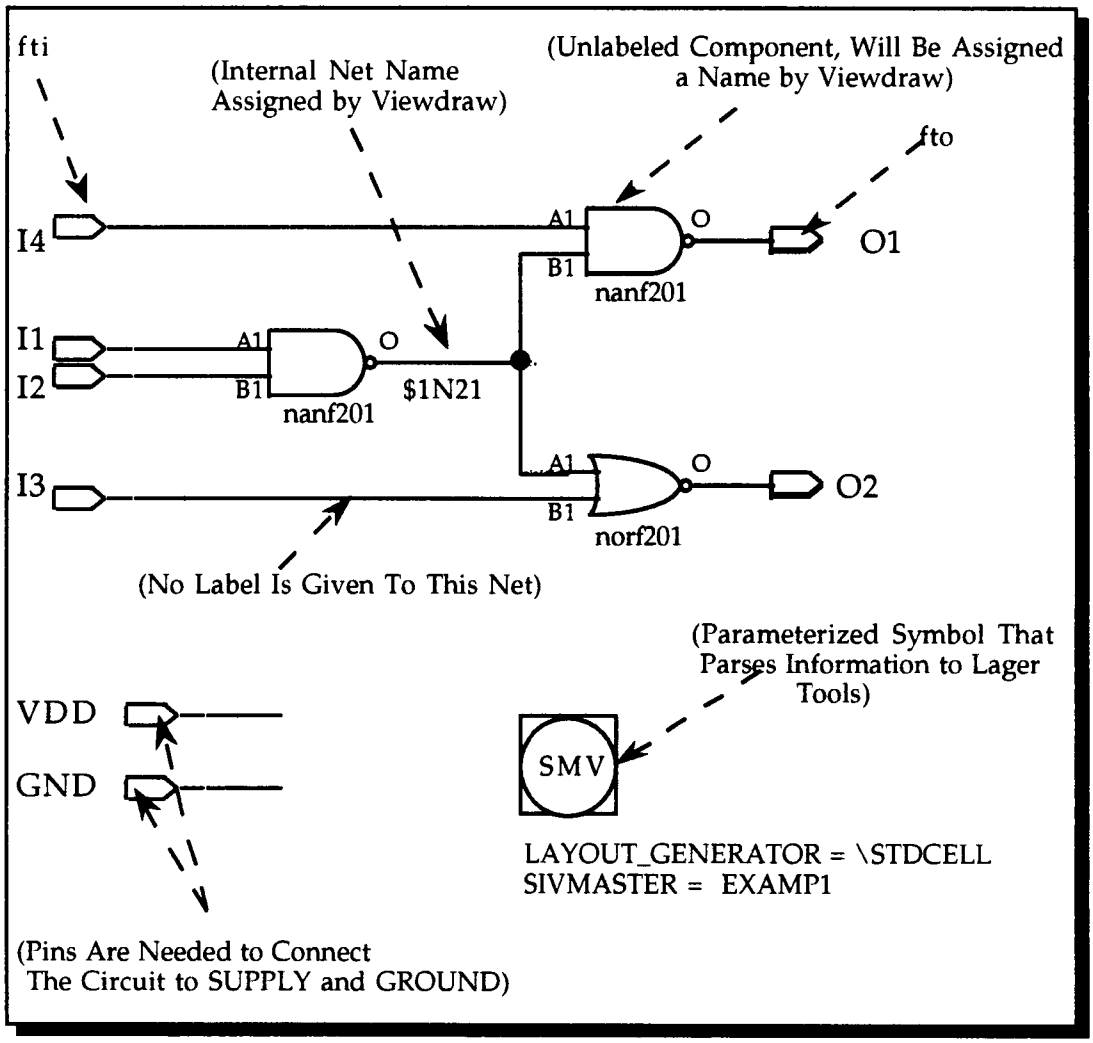



Figure 2.4 examp1 - Schematic Entry Via Viewsim

```
| This is a command file for "examp1"
|
| Display the following signals and LOG Viewsim output
| waveform stream into the file examp1.wfm

wave examp1.wfm I1 I2 $1N21 I4 O1 I3 O2

| Set the simulation step

stepsize 20ns

| Set a Clock of 50% Duty Cycle that will have a period
| of 40ns

clock I1 1 0

| Write a set of test vectors or patterns. A 100% coverage
| is needed in order to try to toggle all of the internal
| nodes from High-to-Low and Low-to-High voltage levels.

pattern I2 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
pattern I3 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
pattern I4 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

| Monitor the primary inputs and primary outputs

watch I1 I2 $1N21 I4 O1 I3 O2

| Trace and record the node and vector transitions and
| simulation times of the listed I/Os any time the
| the nodes/vectors values change. The collected data is
| saved in the file "examp1.trc"

trace I1 I2 $1N21 I4 O1 I3 O2 > examp1.trc

| Run the simulation

run
```

Figure 2.5 Exampl.cmd - Command File

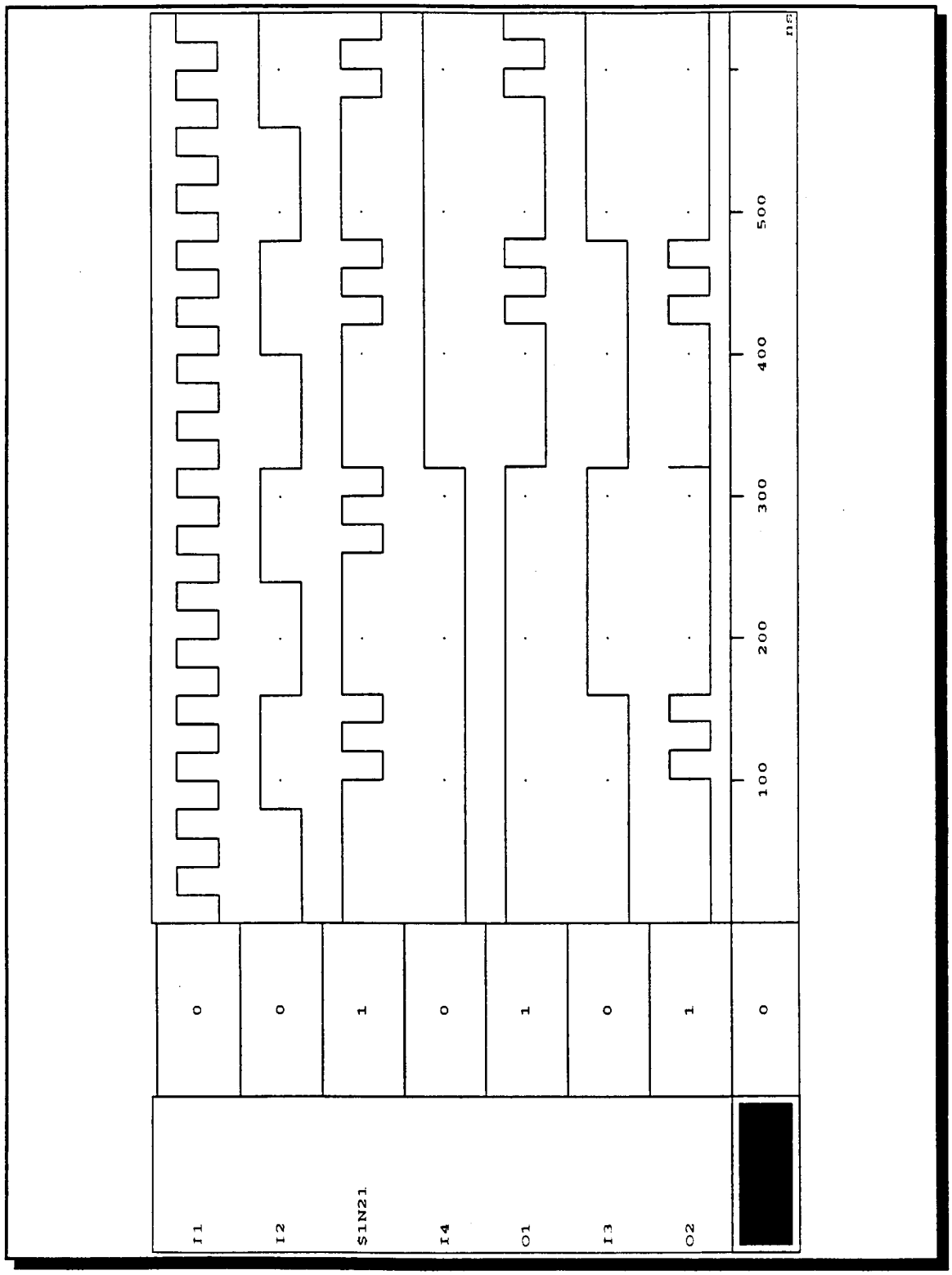



Figure 2.6 examp1.plt - Waveform Results

layout. The parameter needed to be included on the *smv* schematic symbol to generate the layout is LAYOUT_GENERATOR (refer to Figure 2.4). In our case this attribute is set to "STDCELL" denoting the standard-cell layout generator program.

*Oct Database*

One of the powerful database tools is the OCT tool-set. It is used by the Lager tool-set for storing all net-list and physical geometry of a design. One of its tools is ***Vb2oct***. This program translates a Viewlogic schematic in a schematic hierarchy into an OCT view called the *Structure_Master* view or SMV. This view is a generic net-list view. For parameterized designs the SMV view contains items whose parameters have not been bound to any particular value (e.g. an N-bit wide bus or register). ***Vb2oct*** operates hierarchically and translates the design schematic by converting each schematic to a SMV. In our case, ***Vb2oct*** is run from a X-term/Shell window.

In OCT database terminology a design is stored as a cell, a view, and a facet. A cell can contain different views; i.e., a cell can contain a net-list view (*structure_instance* view) and a layout or mask-level view (*physical* view). Cells and views are implemented in UNIX as directories which would be named, for instance for our design *examp1*, exactly *"structure_instance"* and *"physical"*.

On the other hand, each view can contain facets which contain the objects such as nets, terminals, and instances that define the design. Lager facets are the *"contents* "and *"interface* " facets; the *contents* facet contains the complete design; the *interface* facet contains the formal terminals and bounding box information. Facets in UNIX are represented as files under the view directories. Unfortunately, facets are binary files and cannot be examined via text editors (see Figure 2.7a). Following the run of ***Vb2oct*** the current directory will resemble Figure 2.7b.

After generating the OCT database, a Lager tool called *"**DMOCT**"* is used to produce two additional views from the *structure_master* view; they are

```
vb2oct> ***** vb2oct version 1.7.2 *****
vb2oct> Loading examp1vb...
(0 Comps, 0 Nets) Reading EXAMP1VB sheet 1 ...
(0 Comps, 1 Nets) (0 Comps, 2 Nets) (0 Comps, 3 Nets)
(1 Comps, 3 Nets) (1 Comps, 4 Nets) (1 Comps, 5 Nets)
(1 Comps, 6 Nets) (2 Comps, 6 Nets) (2 Comps, 7 Nets)
(3 Comps, 7 Nets) (3 Comps, 8 Nets) (3 Comps, 9 Nets)
(4 Comps, 9 Nets) (4 Comps, 9 Nets)
Reading complete.
vb2oct> (examp1vb
vb2oct>       (up-to-date relative to nanf201 MODULE)
vb2oct>       (up-to-date relative to norf201 MODULE)
vb2oct>       (examp1vb up-to-date relative to symbol)
vb2oct> examp1vb done)
vb2oct> Cell: examp1vb generated...
```

Figure 2.7a Vb2oct Sample Run on examp1 Eample

```
sawan> ls
examp1/      examp1.vsm  spcurins.ins
examp1.cmd   examp1.wfm          sym/
examp1.vb2oct       sch/   wir/
examp1.trc
sawan> ls examp.1
structure_master
examp1/stucture_master     - structure view created by vb2oct.
contents;
```

Figure 2.7b List of examp1/ Current Project Directory

*structure_instance* view, "SIV", and *physical* view, "PHV". The SIV is the net-list of the design with all parameters bound to real values (e.g 8-bit bus or 32-bit register). The PHV view is the layout view of the design. The LAYOUT_GENERATOR is a program defined in the *SMV* symbol in a Viewlogic schematic and is provided for the designer. The SIV, on the other hand, and the output will be a physical view. Figure 2.8 illustrates a DMOCT run of *examp1*.

To view the layout (physical view) of *examp1* the layout editor ***Magic*** can be used, but ***Magic*** is not needed in our application since the designer needs not deal with the layout details. The expanded and unexpanded views for *examp1* are presented in Figures 2.9 and 2.10.

***Standard Cell Layout Generation***

***Stdcell*** is a layout generator for the LagerIV VLSI design tools which carries out and routes the layout using the standard cell library (see Figure 2.11). The net-list information for the design is defined in the *structure_instance* view which produces a *symbolic* view using ***Siv2sym***. This latter is followed by ***Vulcan*** to generate the *interface* facets for the standard cells from the library. The actual place and route is performed by ***Wolfe*** which itself uses ***TimberWolf*** to preform the global place and route operations. ***Wolfepost*** then adds on the Power/Ground connections to produce the complete macro-cell. Stdcell should only be used as the layout generator at the very top level of the hierarchial standard design. To generate a standard cell layout successfully, the designer must add a flatten flag which causes the hierarchy to be flattened before the layout is generated.

In addition, it is imperative that VDD and GND pins be added to the schematic (without being attached to symbols) in order for ***Stdcell*** to connect the supply and ground to the circuit.

```
DMoct> Command line parsed successfully: Master name declared: ``examp1'';
DMoct> Instance name declared: ``examp1''
DMoct> No parameter value file specified.
DMoct>
DMoct>
DMoct> CREATING/READING STRUCTURE MASTER VIEWS (SMV's) ..
DMoct>
DMoct> (examp1 (examp1:structure_master is up to date))
DMoct>
DMoct>
DMoct> CREATING STRUCTURE INSTANCE VIEWS (SIV's)..
DMoct>
DMoct> Successfully opened SMV for root ``examp1''
DMoct> (./examp1 Using SMV of cell ``./examp1'' to generate SIV for cell ``.\
DMoct>     /examp1''.
DMoct>
DMoct>     Processing subcell instance : ``V1I1'' of parent cell ``./examp1''
DMoct>      Successfully opened SMV for subcell ``/u4/cad/Lager/common/LagerIV\
DMoct>      /cellib/stdcell/leafcells/nanf201''
DMoct>       (/u4/cad/Lager/common/LagerIV/cellib/stdcell/leafcells/nanf201
DMoct>         (read-only))
DMoct>     Processing subcell instance : ``V1I18'' of parent cell ``.\
DMoct>     /examp1''
DMoct>      Successfully opened SMV for subcell ``/u4/cad/Lager/common/LagerIV\
DMoct>      /cellib/stdcell/leafcells/norf201''
DMoct>       (/u4/cad/Lager/common/LagerIV/cellib/stdcell/leafcells/norf201
DMoct>         (read-only))
DMoct>     Processing subcell instance : ``V1I2'' of parent cell ``./examp1''
DMoct>      Successfully opened SMV for subcell ``/u4/cad/Lager/common/LagerIV\
DMoct>      /cellib/stdcell/leafcells/nanf201''
DMoct>     (the SIV `./examp1' is up to date.)
DMoct> )
DMoct>
DMoct>
DMoct> RUNNING LAYOUT GENERATORS
DMoct>
DMoct> (./examp1
DMoct>     Generating layout for master of cell V1I1
DMoct>       (/u4/cad/Lager/common/LagerIV/cellib/stdcell/leafcells/nanf201
DMoct>         CELLCLASS property with value LEAF found
DMoct>     )
DMoct>     Generating layout for master of cell V1I18
DMoct>       (/u4/cad/Lager/common/LagerIV/cellib/stdcell/leafcells/norf201
DMoct>         CELLCLASS property with value LEAF found
DMoct>     )
DMoct>     Generating layout for master of cell V1I2
DMoct>      (up-to-date))
```

Figure 2.8 examp1.log - Creating a SIV View

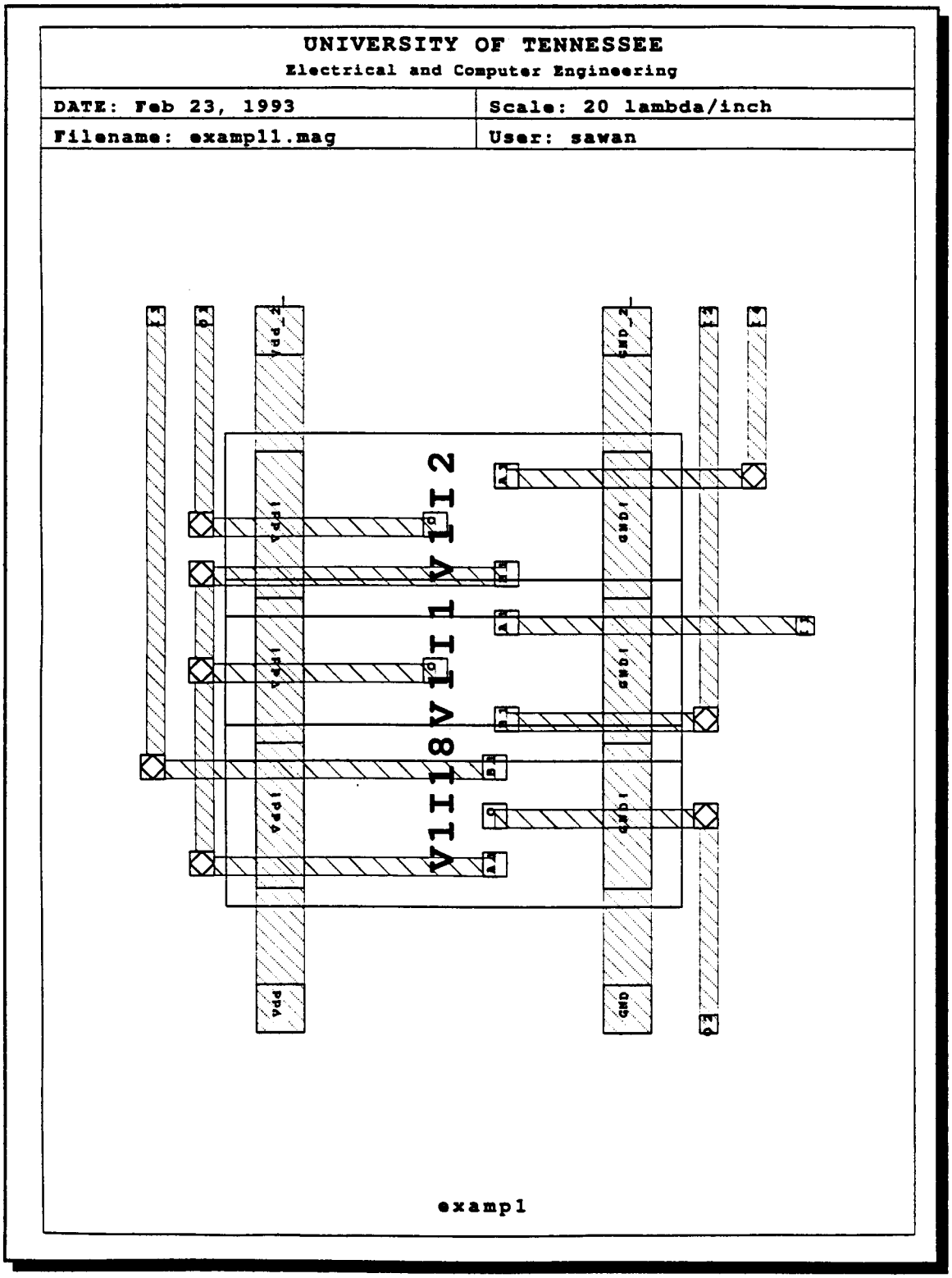



Figure 2.9 Unexpanded View of examp1 Circuit

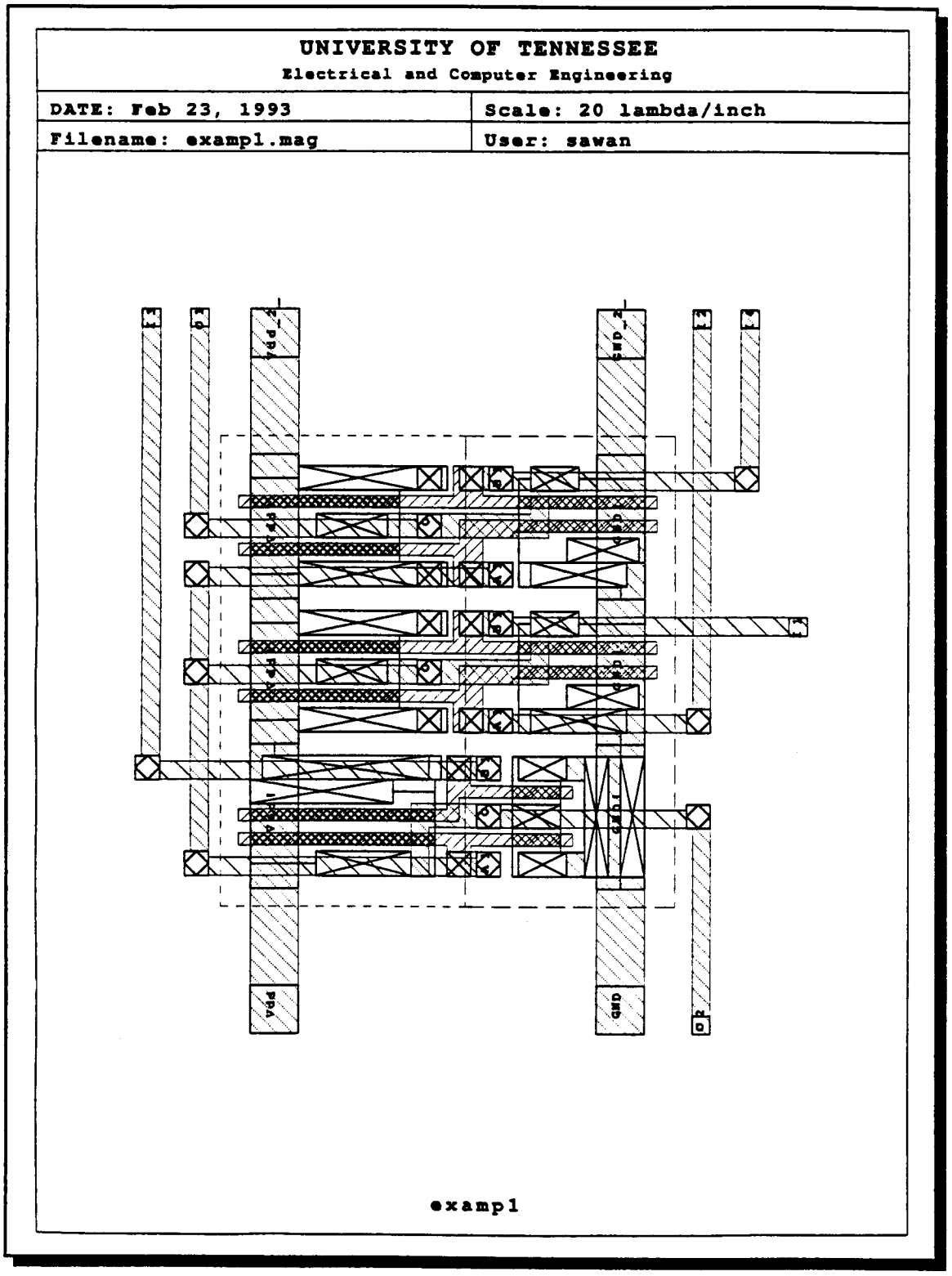



Figure 2.10 Expanded View of examp1 Circuit

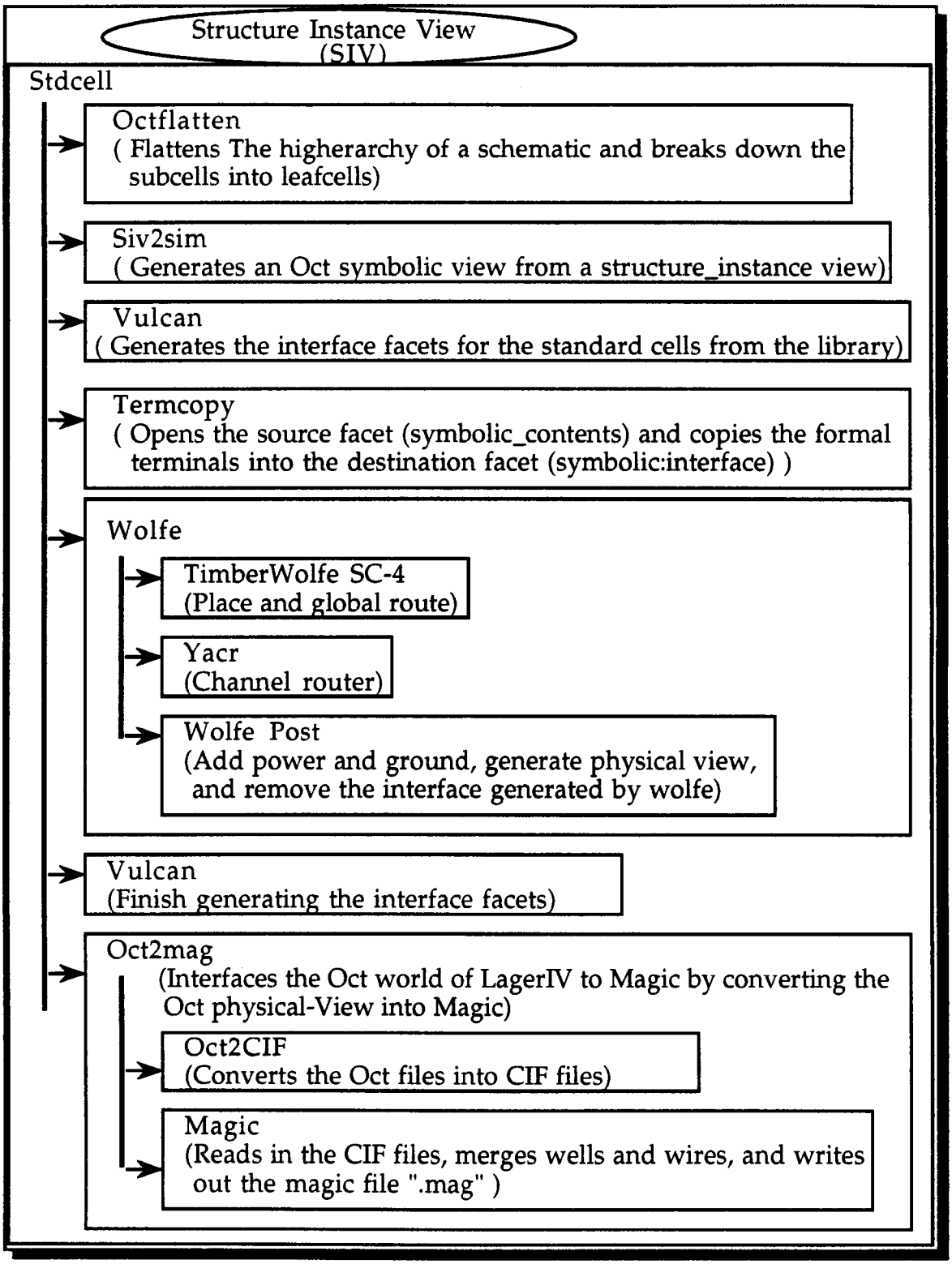



Figure 2.11 Stdcell - Standard Cell Layout Generator Outline

*Layout Simulation*

In our program the layout simulation is done (automatically) transparently to the user; however, it is important for the designer to be aware of the main issues involved with the layout simulation. As mentioned earlier, the ***Viewsim*** simulator can be used to simulate the design (schematic), but the timing information in the standard cell models is not very accurate. This is why there would be a need to use a more accurate event-driven, switch-level simulator for MOS circuits such as ***"IRSIM"***. This simulator operates off a circuit transistor net-list extracted form the physical view. The designer need not worry about having any knowledge of this simulator since ***Sim2Bat*** provides a few easy instructions in order to modify ***Viewsim's*** command file (e.g *examp1.cmd*) and generate an ***IRSIM*** command file (e.g *examp1.cmds*). Both ***Viewsim*** and ***IRSIM*** have very similar and sometimes identical commands.

Before starting the circuit layout simulation, a layout must be present! To generate a layout, the wire-list net-list generated by Viewlogic is translated into an OCT view net-list by the program ***Vb2oct*** which will produce the *contents* and *interface* net-lists for each of the *master, structure,* and *physical* views (see Figure 2.12). These views are then used by ***DMOCT*** (OCT design manager) which in turn uses ***Stdcell*** (standard cell layout generator) in order to produce the layout. Once the layout is ready, the program ***DMPOST*** is used to generate the necessary simulation files e.g. *examp1.sim* to be used by ***IRSIM***.

*DMoct: Design Manager for The OCT Database*

***DMoct*** is a design manager utility for the LagerIV silicon assembly system. It provides an interface between the user, the OCT database and the LagerIV layout generation tools. ***DMoct*** carries out the layout generation of an instance of a cell by first creating a cell *structure_master* view, second a *structure_instance* view, and last a cell instance layout.

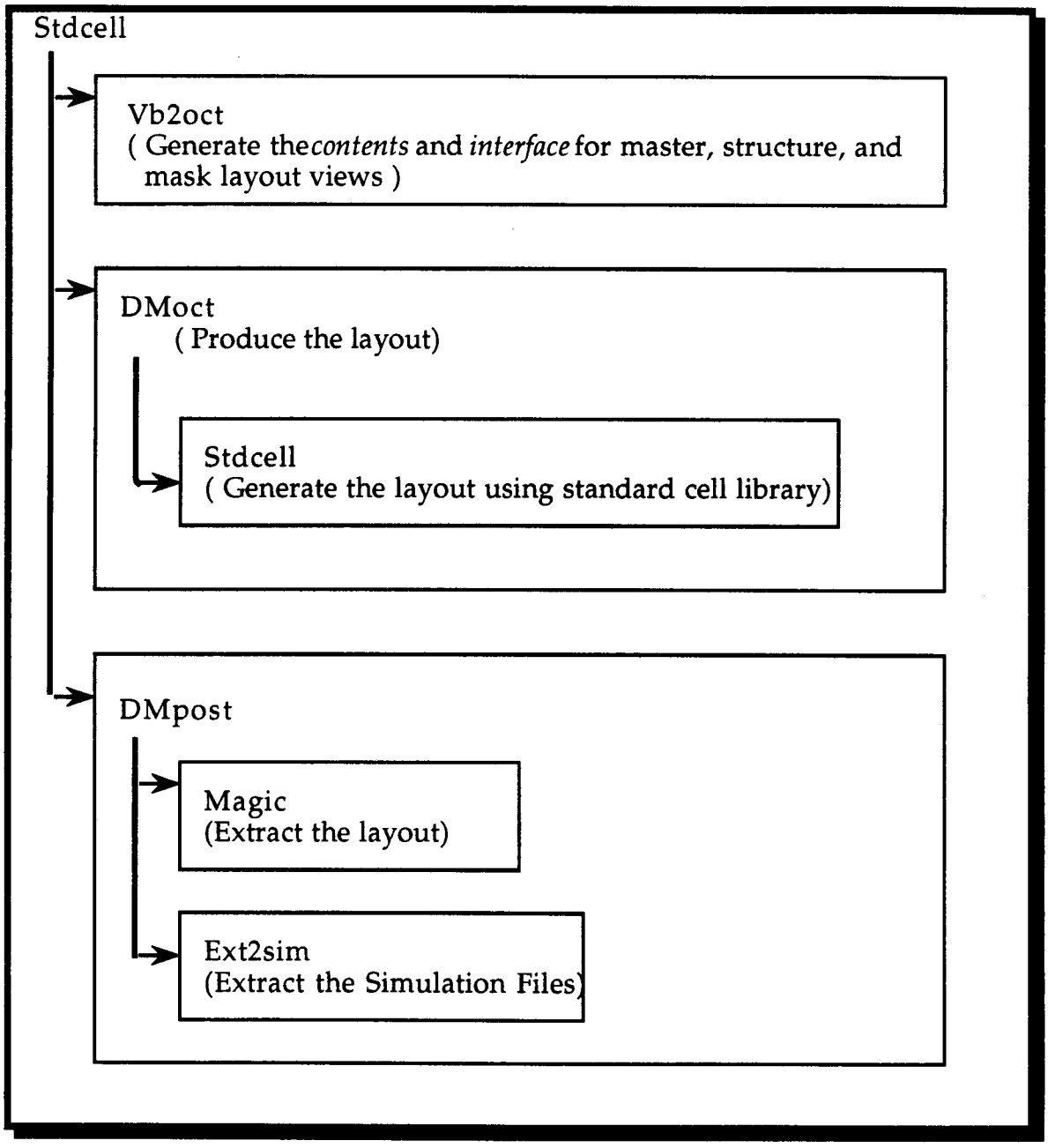



Figure 2.12 Generating The Simulation Net-list

The structure master view for the cell and for all its subcells is created by ***DMoct*** which demands a structural description of the cell be provided by the user; in our case the library contains that type of description for each leaf-cell. The structure instance view for the desired instance of the cell as well as for the instances of its sub-cells are then generated from the *structure-master* view. Finally, ***DMoct*** runs a user-defined layout generation program (e.g. ***Stdcell***) to create the layout and OCT *physical* view for the cell instance (and its sub-cell instances). The layout generator program name must be included in the schematic (please refer to Figure 2.4) as a LAYOUT_GENERATOR property which will be passed on to the OCT database of the cell.

***DMpost: Post-Processor for The LagerIV System***

***DMpost*** is a command file that provides tools to do all the necessary steps for shipping a design to fabrication including design rule checking, CIF file generation which interfaces the OCT database to ***Magic***, and simulation extraction. ***DMpost*** generates the files which reside in the "layout" directory. It uses ***Octtocif*** generated magic files (physical view) then calls the ***Ext2sim*** program to produce the *.sim* file from *.ext* files (e.g. layout/*examp1.sim*). Since ***DMpost*** is called from within a shell script command file the ***Ext2sim*** program was not successful in producing the simulation net-list file *.sim*, therefore, ***Sim2Bat*** asks the user to type the command ***Ext2sim***, then the project name (e.g. *ext2sim examp1)* (see Figure 2.13) in order to fix this bug which is a std-error described in the UNIX manual.

***IRSIM***

After generating the layout during ***DMpost*** using the layout file *.mag*, ***Magic***, an interactive editor for VLSI layouts, extracts the layout producing one or more hierarchical *ext* files that describe the electrical circuit, including the circuit parameters such as parasitic resistances and capacitances.

```
| units: 100   tech: scmos
p I4 Vdd O1 2 25 72 60
p V1I1/O O1 Vdd 2 25 64 60
n I4 GND V1I2/6_26_6 2 21 68 19
n V1I1/O V1I2/6_26_6 O1 2 21 64 19
p I1 Vdd V1I1/O 2 25 48 60
p I2 V1I1/O Vdd 2 25 40 60
n I1 GND V1I1/6_26_6 2 21 44 19
n I2 V1I1/6_26_6 V1I1/O 2 21 40 19
p V1I1/O O2 V1I18/6_18_76 2 31 15 54
p I3 V1I18/6_18_76 Vdd 2 31 19 54
n V1I1/O GND O2 2 8 15 33
n I3 O2 GND 2 8 23 33
C V1I18/6_18_76 GND 18
C O2 GND 76
C I3 GND 15
C V1I1/6_26_6 GND 12
C I2 GND 15
C I1 GND 13
C V1I2/6_26_6 GND 12
C O1 GND 87
C Vdd GND 252
C V1I1/O GND 107
C I4 GND 15
```

Figure 2.13 examp1.sim - Simulation Net-list

In addition, two more files are generated, *.al* (aliases) file and *.nodes* (node names file). These files will be used in our research to generate the correct database (see Figures *.ext, .al, .nodes,* and *.mag* 2.14 through 2.17). One of the advantages of using ***IRSIM*** is that it has the capability of modeling a transistor not only as a voltage-controlled switch but also as a resistor in series with a voltage- controlled switch where each node has a capacitance node value and transition times are computed from the resulting RC network. Another advantage of using ***IRSIM*** is that it resembles the Viewlogic schematic simulator ***Viewsim*** in many ways; one great and beneficial resemblance between both simulators is that they have similar commands which allow the user to write one simulation file and translate it very easily into the other simulator command file format. Table 2.2 provides a listing of the similarities between ***Viewsim*** and ***IRSIM***.

After generating the simulation file *.sim* via ***Ext2sim***, the simulation can be run after the designer has prepared a *.cmds* command file. Given the file *examp1.cmds* (see Figure 2.18). The simulation results would look like Figure 2.19.

***Summary***

The ***Viewsim*** simulator provides a functional capability while the ***IRSIM*** simulator provides a switch-level physical simulation capability. ***Viewsim*** is based on VHDL models provided with the cell libraries (e.g. standard-cell) and acts as a powerful debugging tool for finding functional problems. On the other hand, ***IRSIM*** is not as easy to use as ***Viewsim*** but can provide very critical timing information. Given the dilemma of using both simulators, a simulation methodology is needed which will lead us into developing a very special and powerful simulation tool.

```
timestamp 726298080
version 5.0
tech scmos
scale 1000 1000 100
resistclasses 25300 130400 38400 49 33 3000000 2776000 25300
use norf201 V1I18 1 0 8 0 1 16
use nanf201 V1I1 -1 0 56 0 1 16
use nanf201 V1I2 -1 0 80 0 1 16
node "I1" 1 2 50 -8 m2 0 0 0 0 0 0 0 0 153 108 0 0 0 0 0 0
node "I4" 1 4 74 0 v 0 0 0 0 0 0 0 94 68 117 84 0 0 0 0 0 0
node "I2" 1 6 34 8 v 0 0 0 0 0 0 0 214 148 93 68 0 0 0 0 0 0
node "O2" 1 4 -16 8 m1 0 0 0 0 0 0 0 118 84 99 72 0 0 0 0 0 0
node "GND_2_" 1 23 -16 19 m1 0 0 0 0 0 0 0 960 256 0 0 0 0 0 0 0 0
equiv "GND_2_" "GND"
node "Vdd_2_" 1 23 -16 77 m1 0 0 0 0 0 0 0 960 256 0 0 0 0 0 0 0 0
equiv "Vdd_2_" "Vdd"
node "O1" 1 4 66 92 v 0 0 0 0 0 0 0 118 84 111 80 0 0 0 0 0 0
node "7_20_184#" 2 10 10 92 v 0 0 0 0 0 0 0 168 116 399 284 0 0 0 0 0 0
node "I3" 2 8 26 100 v 0 0 0 0 0 0 0 238 164 165 116 0 0 0 0 0 0
cap "Vdd_2_" "7_20_184#" 3
cap "I4" "GND_2_" 1
cap "Vdd_2_" "O1" 1
cap "I1" "GND_2_" 1
cap "O2" "GND_2_" 1
cap "Vdd_2_" "I3" 1
cap "I2" "GND_2_" 1
cap "V1I18/A1" "V1I18/O" 1
merge "V1I18/B1" "I3" 0 0 0 0 0 0 0 0 0 -6 -4 0 0 0 0 0 0
merge "V1I2/A1" "I4" 0 0 0 0 0 0 0 0 0 -6 -4 0 0 0 0 0 0
merge "V1I1/B1" "I2" 0 0 0 0 0 0 0 0 0 -6 -4 0 0 0 0 0 0
merge "V1I2/Vdd!" "V1I1/Vdd!" -13 0 0 0 0 0 0 -576 -200 0 0 0 0 -492 -188 0 0
merge "V1I1/Vdd!" "V1I18/Vdd!"
merge "V1I18/Vdd!" "Vdd_2_"
merge "V1I2/GND!" "V1I1/GND!" -13 0 0 0 0 0 0 -576 -200 0 0 -420 -164 0 0 0 0
merge "V1I1/GND!" "V1I18/GND!"
merge "V1I18/GND!" "GND_2_"
merge "V1I2/O" "O1" 0 0 0 0 0 0 0 0 0 -6 -4 0 0 0 0 0 0
merge "V1I18/O" "O2" 0 0 0 0 0 0 0 0 0 -6 -4 0 0 0 0 0 0
merge "V1I2/B1" "V1I1/O" 0 0 0 0 0 0 0 0 0 -18 -12 0 0 0 0 0 0
merge "V1I1/O" "V1I18/A1"
merge "V1I18/A1" "7_20_184#"
merge "V1I1/A1" "I1" 0 0 0 0 0 0 0 0 0 -6 -4 0 0 0 0 0 0
```

Figure 2.14 examp1.ext - Electrical Circuit Description

```
= O2 V1I18/O
= I3 V1I18/B1
= I2 V1I1/B1
= I1 V1I1/A1
= GND GND
= GND GND_2_
= O1 V1I2/O
= Vdd Vdd
= Vdd Vdd_2_
= V1I1/O V1I18/A1
= V1I1/O 7_20_184
= V1I1/O V1I2/B1
= I4 V1I2/A1
```

Figure 2.15 examp1.al - Labels Present In Layout

```
94 V1I18/6_18_76 1700 5400 brown;
94 O2 -1600 800 m1;
94 I3 1900 5000 p;
94 V1I1/6_26_6 4200 1900 green;
94 I2 3400 800 v;
94 I1 5000 -800 m2;
94 V1I2/6_26_6 6600 1900 green;
94 GND -1600 1900 m1;
94 O1 7000 6000 brown;
94 Vdd 5300 5200 nsd;
94 V1I1/O 6400 1700 p;
94 I4 7400 0 v;
```

Figure 2.16 examp1.nodes - Nodes Present In Layout

```
magic
tech scmos
timestamp 726298080
use nanf201 V1I1
timestamp 638648608
transform -1 0 56 0 1 16
box -3 -2 27 74
use norf201 V1I18
timestamp 638648608
transform 1 0 8 0 1 16
box -3 -2 27 74
use nanf201 V1I2
timestamp 638648608
transform -1 0 80 0 1 16
box -3 -2 27 74
<< metal1 >>
rect 101 8 104 11
rect 101 100 104 103
rect -16 8 -13 11
rect 101 0 104 3
rect 101 92 104 95
rect -16 19 -8 27
rect 96 19 104 27
rect -16 77 -8 85
rect 96 77 104 85
rect 26 103 30 104
rect 26 100 104 103
rect 66 95 70 96
rect 58 95 62 96
rect 42 95 46 96
rect 10 95 14 96
rect 66 92 104 95
rect 10 92 62 95
rect -16 77 104 85
rect -16 19 104 27
rect 34 11 38 12
rect 18 11 22 12
rect 34 8 104 11
rect -16 8 22 11
rect 74 3 78 4
rect 74 0 104 3
<< metal2 >>
rect 50 -8 53 -5
rect 26 100 30 104
rect 26 45 29 100
rect 66 92 70 96
rect 58 92 62 96
rect 42 92 46 96
rect 10 92 14 96
rect 66 55 69 92
rect 58 43 61 92
```

Figure 2.17 examp1.mag - Layout Net-list

```
rect 42 55 45 92
rect 10 45 13 92
rect 18 12 21 45
rect 74 4 77 43
rect 50 -8 53 43
rect 34 12 37 43
rect 34 8 38 12
rect 18 8 22 12
rect 74 0 78 4
<< m2contact >>
rect 26 100 30 104
rect 66 92 70 96
rect 58 92 62 96
rect 42 92 46 96
rect 10 92 14 96
rect 34 8 38 12
rect 18 8 22 12
rect 74 0 78 4
<< labels >>
rlabel metal2 50 -8 53 -5 0 I1
rlabel metal1 101 8 104 11 0 I2
rlabel metal1 101 100 104 103 0 I3
rlabel metal1 -16 8 -13 11 0 O2
rlabel metal1 101 0 104 3 0 I4
rlabel metal1 101 92 104 95 0 O1
rlabel metal1 -16 19 -8 27 0 GND
rlabel metal1 96 19 104 27 0
GND_2_
rlabel metal1 -16 77 -8 85 0 Vdd
rlabel metal1 96 77 104 85 0 Vdd_2_
<< end >>
```

Continued

Table 2.2- Viewsim and IRSIM Common Commands

| Viewsim | IRSIM | Description |
|---|---|---|
| > | > | Write curent network state ti file |
| \| | \| | Comment |
| watch | analyzer | Display nodes in analysis window |
| cycle [n] | cycle (c[n]) | Simulate for n clock cycles |
| clock[node[val]] | clock [node[val]] | Define value sequence for clock node |
| high (h) | high (h) | Make node logic high (1) input |
| low (l) | low (l) | Make node logic low (0) input |
| logfile[filename] | logfile [filename] | Start/Stop log file |
| print comment... | print comment... | Print specified cycles |
| run | R [n] | Simulate for n cycles |
| sim | S [n] | Set simulation stepsize to n ns |
| trace | t [-] wnode... | Start/Stop tracing of specified nodes |
| vector | vector | Define bit vectoe |
| . . . . etc. | . . . . etc. | . . . . etc. |

```
| This is a command file for example circuit "examp1", very similar
| to examp1.cmd command file of Viewsim.
| Open a log file in which all the standard output of the commands
| used in this file will be saved

logfile  layout/examp1.lognet

| Set the clock to be a 50% duty cycle and 40ns period

clock I1 0 1
stepsize 20ns
|  Declare the input vectors/terminals and define the test vectors
| set.  A 100% coverage of the internal nodes is needed in order to
| toggle all the nodes from the high state to the low state and vice
| versa.

vector I1 I2 I3 I4
V I1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
V I2 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
V I3 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
V I4 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

| Declare the I/Os to monitor in a graphical waveform format

analyzer I1 I2 V1I1/O I4 O1 I3 O2

| Declare the formal terminal to be traced and saved in
| examp1.lognet

t I1 I2 V1I1/O I4 O1 I3 O2

| Run the simulation

R
logfile
```

Figure 2.18 examp1.cmds - Command File for IRSIM

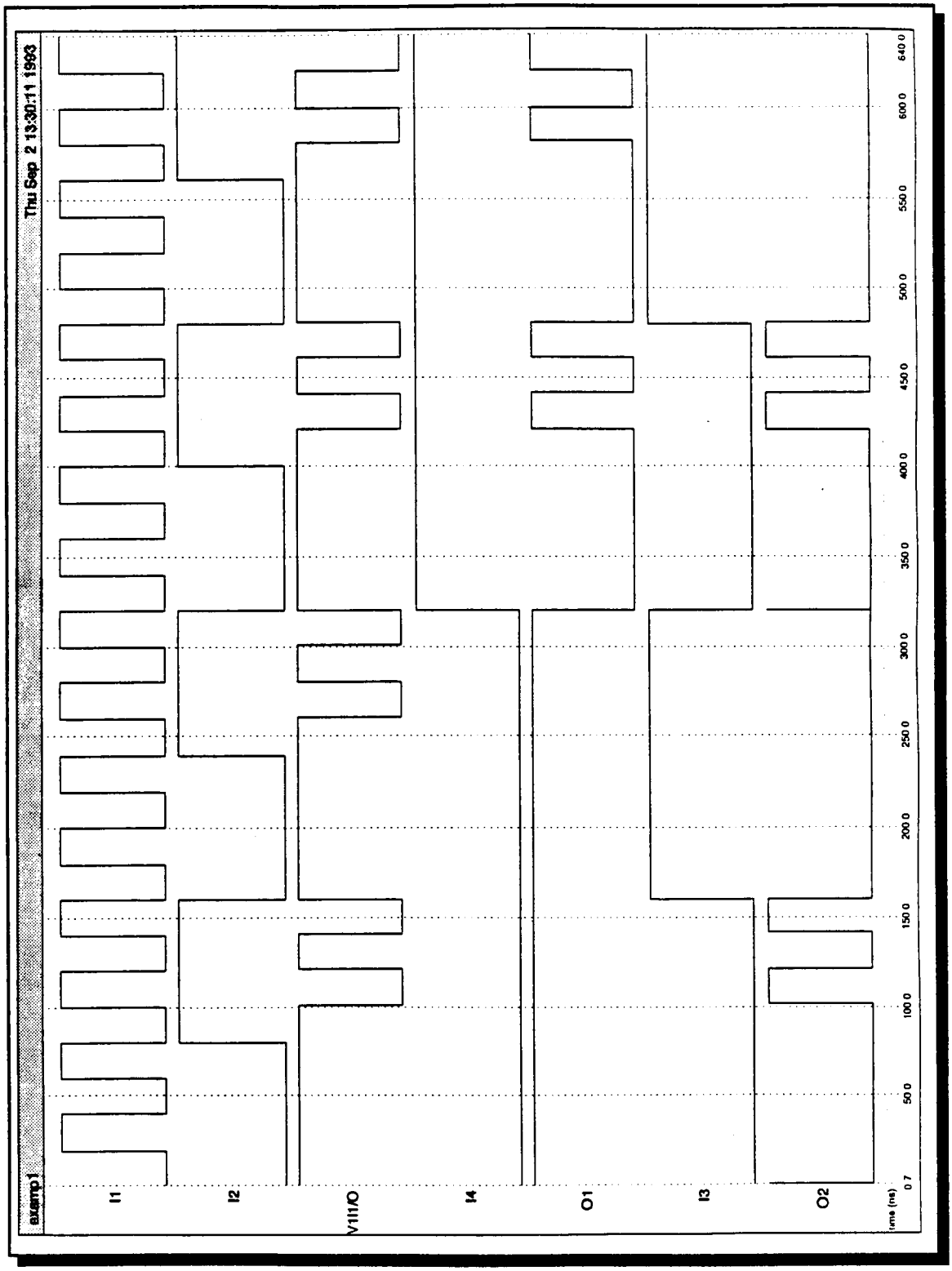



Figure 2.19 examp1.ps - IRSIM Simulation Results

## *Simulation Methodology*

### *Traditional Methodology*

Traditionally, a design is first generated using Viewlogic schematic capture, then the circuit is verified using the ***Viewsim*** logic-level simulator. These two steps can be repeated as many times as possible in order to refine the design. Layout generation follows ***Viewsim*** using OCT tools, then the layout simulation is done using ***IRSIM***. The whole procedure can be repeated starting with schematic entry and finishing with the ***IRSIM*** switch-level simulation (see Figure 2.20).

***IRSIM*** simulation can be a painful and long process; for instance, tracing nets (e.g. using ? and ! commands) demands a considerable degree of expertise from the user to decipher. In addition, the user has to consult the leaf-cell diagrams in the library (e.g. standard-cell) documentation to trace the nets through the library elements. Another problem with the ***IRSIM*** simulation is that internal nets cannot be observed unless they are brought out in the SDL (structure description language) net-list as a terminal. This process is very time-consuming since once a change is introduced in a SDL net-list, the ***DMoct*** tool must be run again in order to recreate the design.

The obvious solution to these problems is to examine the ***Viewsim*** simulation results and ignore the timing delay information which could be obtained from ***IRSIM;*** this is not a real solution since it still does not allow accurate timing results. The other alternative is to edit the layout itself and add the labels needed on each node, then re-run ***DMpost*** to re-extract the *.sim* file. This requires familiarity with the ***Magic*** layout editor, and does not allow back-annotation of the new labels into the schematic since they will be ignored by the OCT tools. Moreover, the back-annotation program needed to accomplish this test simply does not exist (yet!) and the user has to search for the internal node names in Viewlogic which exist for the circuit but do not exist for the internal nets of the library cells. Thus, this methodology proves to be ineffective in accomplishing our goal, that's why a new methodology

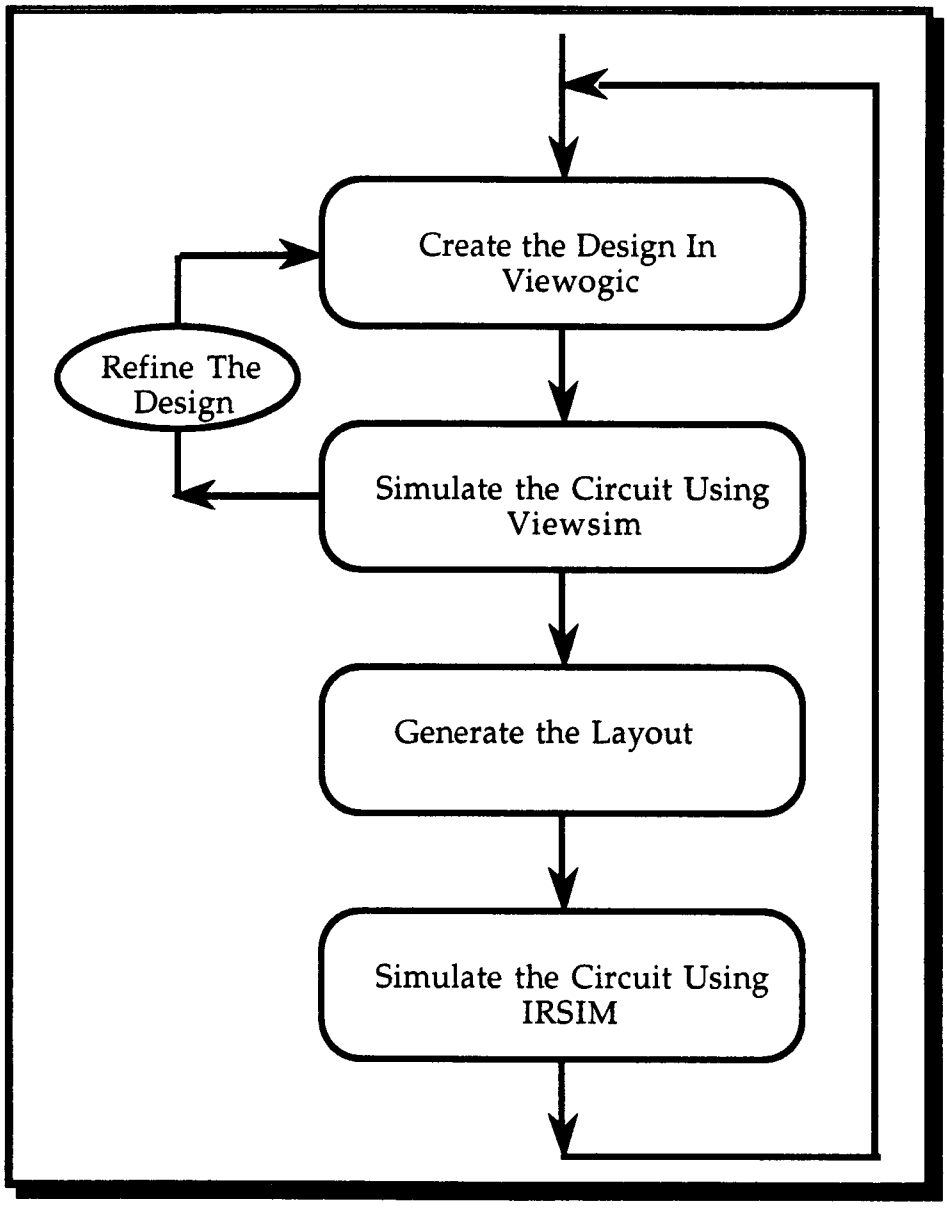



Figure 2.20- Traditional Simulation Methodology

has been devised.

### *New Methodology*

Similar to the old simulation methodology, the new method will start by entering the design in Viewlogic and verifying the design using ***Viewsim***. After the design is error free, the designer will run the back-annotation program which captures the layout simulation (including generation) results and provides them to the designer to observe them with ***viewsim*** (see Figure 2.21). This program, ***Sim2Bat*** allows the designer to avoid the whole layout process (generation and simulation); for instance there is no need to look at the transistor net-list, consult the leaf-cells diagrams, trace the nets, change the SDL net-list, or even edit the layout. The new methodology works for many libraries such as standard cell and data path libraries given that they are supported by Lager and Viewlogic tools. The challenge does not stop here but extends to the analog library where each transistor is represented as an independent component which eliminates the internal net delay back-annotation dilemma. Last but not least, ***Sim2Bat*** can be used to detect any ***Vb2oct*** transistor-sizing translation errors.

## *Facing The Problem*

### *Background*

In the process of developing a layout extractor filter that back-annotates the net timing delays, two procedures were uncovered to be possible candidates towards a delay simulation on a high level of abstraction. The procedures are ***DELTIM*** (Delay Timing) and ***CKTIM*** (Circuit Timing) (refer to chapter I). In order to understand these procedures better one has to consider delay modeling.

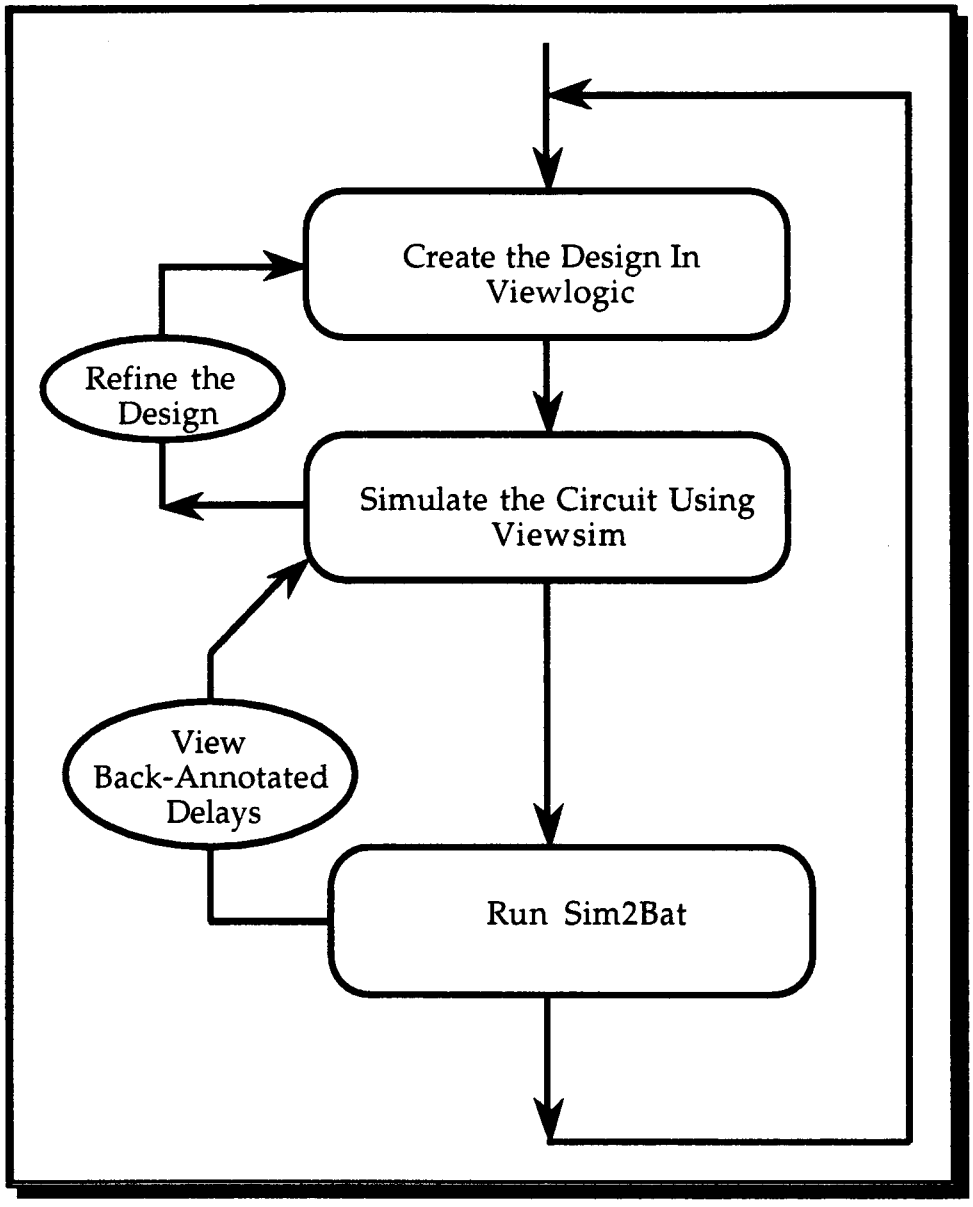



Figure 2.21 - New Simulation Methodology

*Delay Modeling*

There are primarily two types of delays that are of concern to circuit designers: delay through "active components" and delay through wires. The exact definition of active component varies with the level of simulation; it might be a single transistor for a switch-level simulation, or it might be an entire ALU in a function simulation.[Huber] Circuit-level simulators inherently calculate active-component and wire delay, since they are normally taken into account in the equations that are solved. For efficiency, logic-level simulators often assume a simplified model of delay, such as using a unit delay through active components and assigning zero delay through wires. Many variations are possible. For example a functional simulator might use a table of delays for various functional blocks. The table could be based on actual observed delays for parameterized library components if appropriate.

In the case in which a simple delay model, such as a unit-delay or table-driven approach, is used, the simulator needs to know only the connectivity of the circuit and the type of each node. However, much of the value of simulation lies in the ability to determine circuit delays accurately. When this detail is required, a more complex delay model is used. Timing simulators fit this type of requirement. These simulators use relatively complex (compared to most logic-level simulators) delay models combined with simplification provided in assuming a fixed set of logic levels. For example, ***RSIM*** (the basis of ***IRSIM***)[Terman] uses a linear model of transistors to determine delay and final logic values for each mode. The model takes into account the capacitance at each node and the drive capabilities, and is carefully tuned for each technology so that accurate results are obtained.

*DELTIM*

This Delay-Only Timing method suggests capturing the delays incurred by the net connections between components and back-annotating these delays into the schematic. Thus, each wire in the schematic will be a true representation of its pair in the switch-level layout. This method assumes that the original supplied models of the library components are accurate thus the component delays need not be calculated. Once the net delays are determined they are back-annotated into the circuit (see Figure 2.22). In order to reflect these delays in a design a special simulator that must possess net delay recognition capability might be needed. In addition, a new set of equations has to be introduced in order to represent the wires, thus making each wire a component!

In our case, the present VHDL delay timing models of the library components are not accurate. These models can be modified easily; however, in case of any modification to the VHDL models of the library components, these components need to be reinstalled and be reintegrated into the library structure. One stumbling block to our efforts was the unavailability of the *"Mega"* utility needed to accomplish the reinstallation process at the time this research was conducted. The whole task becomes fairly easy to develop *DELTIM* once a installing filter is provided.

This method ignores the component delays, thus a new library timing characterization is needed. Cell characterization requires many man power hours and requires special tools. One disadvantage of this method is that it ignores such factors as fanout and input loading. However, one cannot ignore the advantage of multilevel schematic simulation which DELTIM can support.

*CKTIM*

This Circuit Timing method captures both net and gate delays reported by the layout simulator and back-annotates them into the schematic

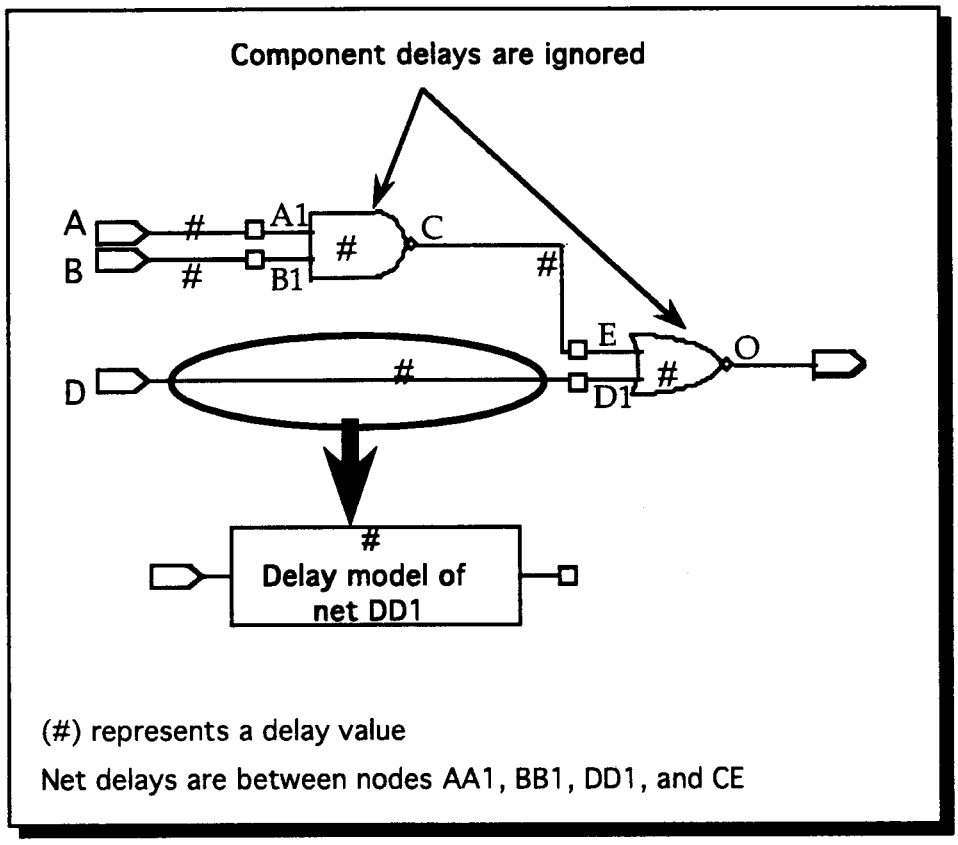



Figure 2.22 DELTIM - Net Delays Only Method Example

simulation net-list. This delay calculation process estimates the actual delays of each component in the design based on factors such as fanout, input loading, and drive strength. In order to back-annotate the delay information in a ***Viewsim*** simulation data structure, gate delays are first estimated by measuring the path delays of each component (from the output of one component to the output of another component) (see Figure 2.23). Then an average of the delays is calculated which represents the new gate delay. The interconnect delay calculation is embedded into the path delays. Therefore, the new gate delay would represent both gate and net delays, thus there is no need for making the wires in the Viewlogic schematic into components (see Figure 2.24).

This method is feasible and extendable to other symbols, thus it can be used to generate models for new libraries that will have updated delays and can be interfaced to the installation program "Mega", which would lead to accomplishing ***DELTIM***. However, this ***CKTIM*** method is not as accurate as ***DELTIM*** method since it incorporates some delay averaging errors (10 %); these errors result mainly from the ***IRSIM*** layout simulator which depends greatly on the designer's input stimulis. Another disadvantage is that ***CKTIM*** cannot handle multi-level schematics. This is not an impossible task but it requires another lengthy research.

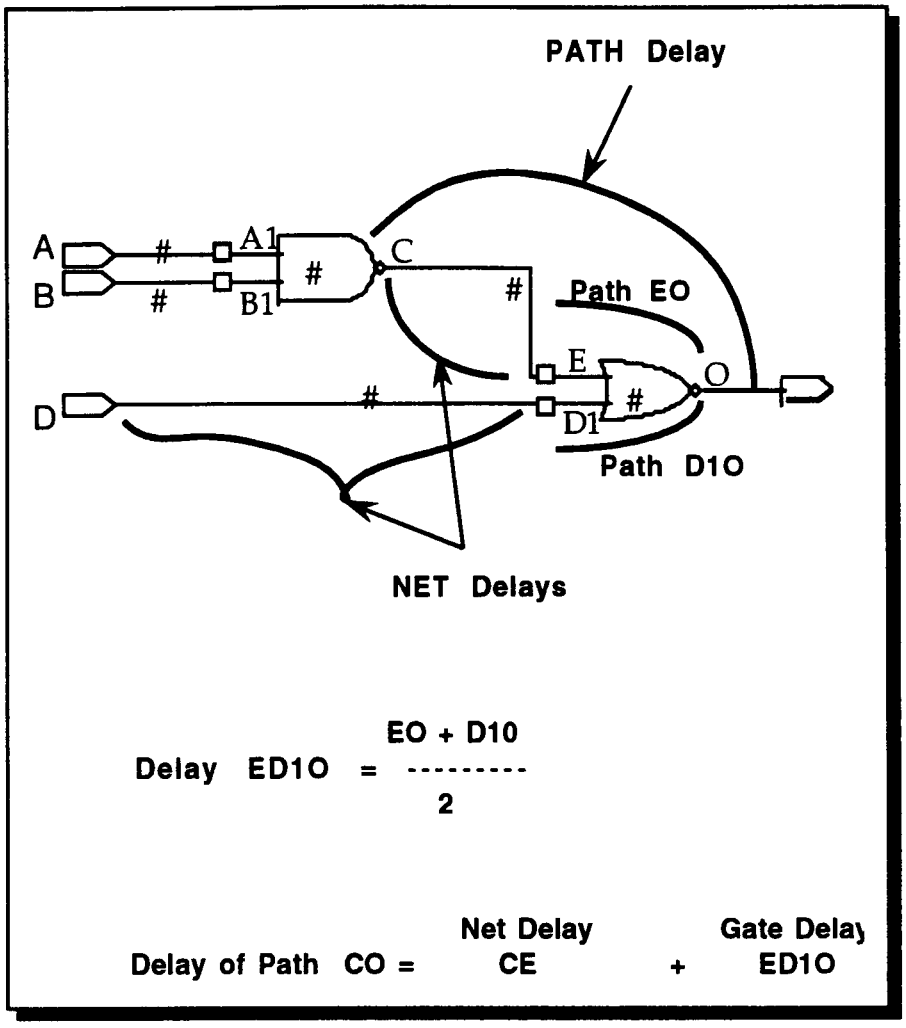



Figure 2.23 CKTIM - Gate and Net Delays Method Example

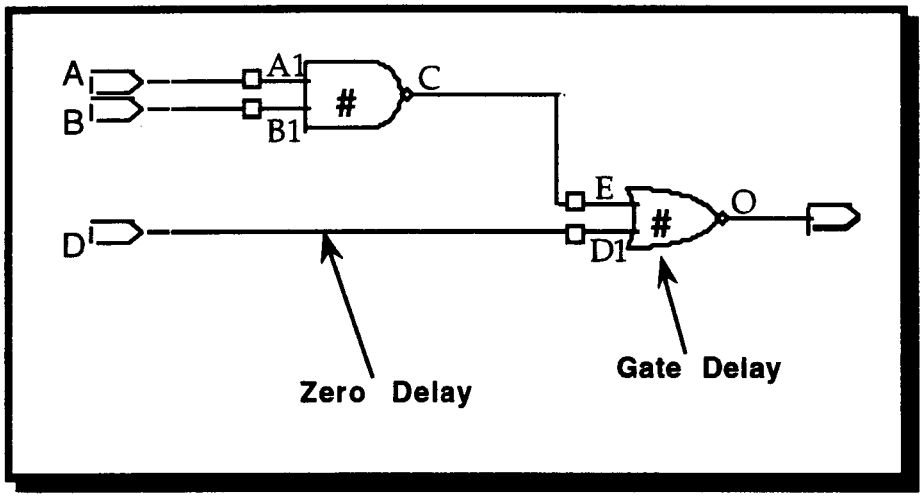



**Figure 2.24 Back-annotated Gate Delays**

# CHAPTER III

## DEVEIOPMENT AND AUTOMATION OF THE SIM2BAT PROGRAM

### *Introduction*

The ***Sim2Bat*** program is a software tool for the automation of post layout timing analysis of integrated circuits. The circuit layout timing delays are extracted and then back-annotated into the circuit in order to provide the designer with accurate timing analysis at the schematic level. The delay information is presented in the form of lumped gate and net delays in addition to delays caused by fanout.

### *Features*

- Integrated with the Lager and Viewlogic tools
- No special memory needs to be allocated for ***Sim2Bat***
- **C, awk** &***sed*** compilers need to be present in order to execute ***Sim2Bat***
- Provides instructions on converting a ***Viewsim*** command file into an ***IRSIM*** command file
- Contains many useful routines for further developments
- Input/Output ASCII files for ease of debugging
- Shell commands provide tight control over ***Sim2Bat***
- A special directory with addon name *backit* is created, where all the generated files are stored and can be manipulated by the user
- Fast computing time (on the order of seconds)

*Operating Environment*

The minimum hardware requirements for the ***Sim2Bat*** program include:

o Sun Sparc station with 32 Mb of RAM.

The ***Sim2Bat*** software does not require more than the minimum amount of RAM provided with any Sparc station; however, additional memory improves the execution time of ***Pre-Sim2Bat*** program which runs the Lager tools.

*Running Sim2bat*

***Sim2bat*** is run by sourcing the program ***CKTIM***: **source CKTIM <design>**. The program next asks for a design name, that is usually the circuit name. See Appendix A for a complete listing of ***CKTIM***.

*User Interface*

***Sim2Bat*** is a user friendly utility that requires some user input frequently. Command entry is not available since the object of this program is the automation of timing back-annotation. However, CTRL-C can be used to abort the program execution and a log of the run can be created using the *enscript* UNIX command.

*Multiple Design Simulations and Simulation Display*

The ***Sim2Bat*** simulator initially simulates a single design, but additional designs can be simulated using additional command shell windows. Precautions should be taken not to simulate the same circuit with modified data at the same time, this will either crash the program or report garbage results. The supplied ***Sim2Bat*** display code uses a single window.

The entire command window represents a scrollable region in which session output is displayed. The window can be sized to contain screen buffer memory that saves any number of previous lines of output, which is written to a design's scroll region (this number is controlled by setting the necessary global UNIX variables).

### *Global Variables*

Variables such as "designame" and "cycle" are global since they are repeatedly used in the program. The global variable"designame" can be any legitimate UNIX name constructed of any combination of upper or lower case letters, numbers, or the characters"-" and "_"; no special characters such as "%", "$", "@", or any control characters are accepted by ***Sim2Bat***. The "cycle" global variable can be any positive integer and naturally is in "nano_seconds" (e.g. cycle = 20 means 20 ns).

### *Programming Language*

***Sim2Bat*** is written in **C**, ***awk***, ***sed*** and shell script languages. Only a small part of the program is written in **C** due to the difficulties encountered in handling data files. Most of the software is written in ***awk*** and ***sed*** and the shell script is used to control both of them.

### *The awk Tool*

***awk*** is best described as "A Pattern Scanning and Processing Language"; it is very useful both for computation and pattern processing. The name ***awk*** is an acronym comprising the last initials of its three developers (Aho, Weinberg, and Kernighan), though some users have complained that it is awkward to use. However, ***awk*** is really a powerful programming language; its commands can do things that no other tool can handle without extreme

contortions.

The ***awk*** command scans a list of input files for lines that match a set of specified patterns. For each pattern, a specified set of actions is performed. These actions can involve field manipulations with the line or arithmetic operations on the values of the fields. The ***awk*** tool is a programming language with features of the shell programming , **bc** (calculator program) and **C** programming languages. It's completely interpreted like **bc,** contains field variables like the shell $1, $2, and $3 arguments, and contains printing and control operators similar to those of the **C** language.[Dougherty]

### *The sed Stream Editor*

The ***sed*** program is a stream editor that combines some features of both the **ed** and **grep** programs. It is a filter like **grep** but allows making changes to files. On the other hand, it is not interactive like the editor **ed**, instead it reads its input line by line and writes the lines one by one to its standard output. Therefore, there is no in-memory buffer of the entire file, which means that files of any size can be changed with **sed**, even ones that are too big for **ed** or other text editors (e.g. files larger than one megabyte).[Dougherty]

### *The Shell Program*

The ***shell*** is the software that listens to commands typed in at the terminal and translates them into instructions in the system's internal syntax.[Anderson] It is hard material that stands between the core of the system and the outside world, providing a robust user interface for the operating system. The ***shell*** includes a large number of functions and implements some of the most powerful and elegant concepts in the UNIX system. Commands are usually separate executable programs that the ***shell*** finds and executes in response to typed instructions, such **grep, *awk*** and **sed.** Therefore, the ***shell*** acts as a command interpreter that can expand and

change the commands which can take advantage of conditions in the current user environment, managed by the ***shell***, to modify the way they work.

In conclusion ***awk*** ***sed***, and ***shell*** commands will be the engine that drives ***Sim2Bat*** to be a powerful yet flexible tool.

*Sim2bat: The Run File*

In order to best illustrate ***Sim2Bat***, the design *examp1* (see Figure 3.1) is used by showing Input/Output files and explaining the steps involved. The ***Sim2Bat*** program is shown in its entirety in Appendix A and is best illustrated in the detailed flowchart depicted in Figure 3.2.

***Sim2Bat*** starts by extracting the simulation file in the layout directory using ***Ext2sim*** and produces the nodes file *examp1.nodes* (see Figure 3.3), which is then used to find all of the internal nodes used by the ***Magic*** program and stores them into a file named *examp1.magnod*; this file is then formatted and stored into the file *examp1magnod* (see Figure 3.4). A design file directory for ***Sim2Bat*** database is then created and named *examp1backit;* the added "backit" extension renders the project directory distinguishable from other directories. Assuming that the user has saved the design schematic, a wirelist is automatically generated which constitutes the simulation net-list for ***Viewsim*** *examp1.vsm* (see Figure 3.5).

In order to check the circuit for design rule violations, the program ***LDRC*** is executed. Errors such as nets without inputs or outputs, nets with multiple outputs, and partially connected components can be easily detected by the ***LDRC*** utility. This software executes in turn the ***NETLIST*** and ***LOGDRC*** programs. The ***NETLIST*** program uses the *wir* file in order to create a net names file *examp1.net*, a list of labeled gates used in the schematic *examp1.pkg*, a gate pinorder, pintype and function list *examp1.dvf*, and finally a list of all the circuit components *examp1.fnc* (see Figures 3.6 through 3.9).

Once the ***NETLIST*** program is done, the ***LDRC*** program saves all the design rule errors in the file *examp1.drc* (see Figure 3.10). These errors should be ignored if the circuit is checked and saved without any errors.

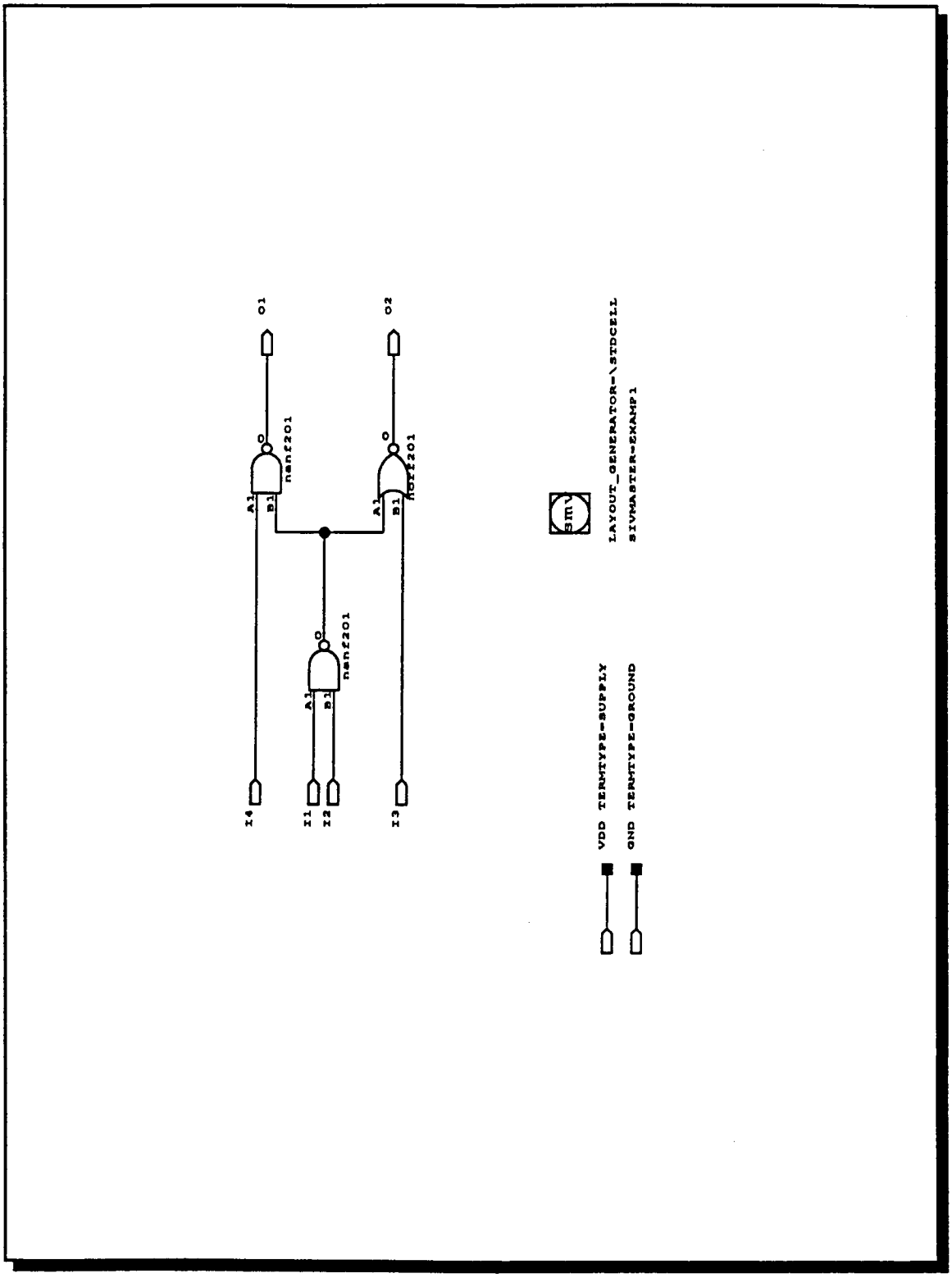



Figure 3.1 examp1.001p - Circuit Schematic

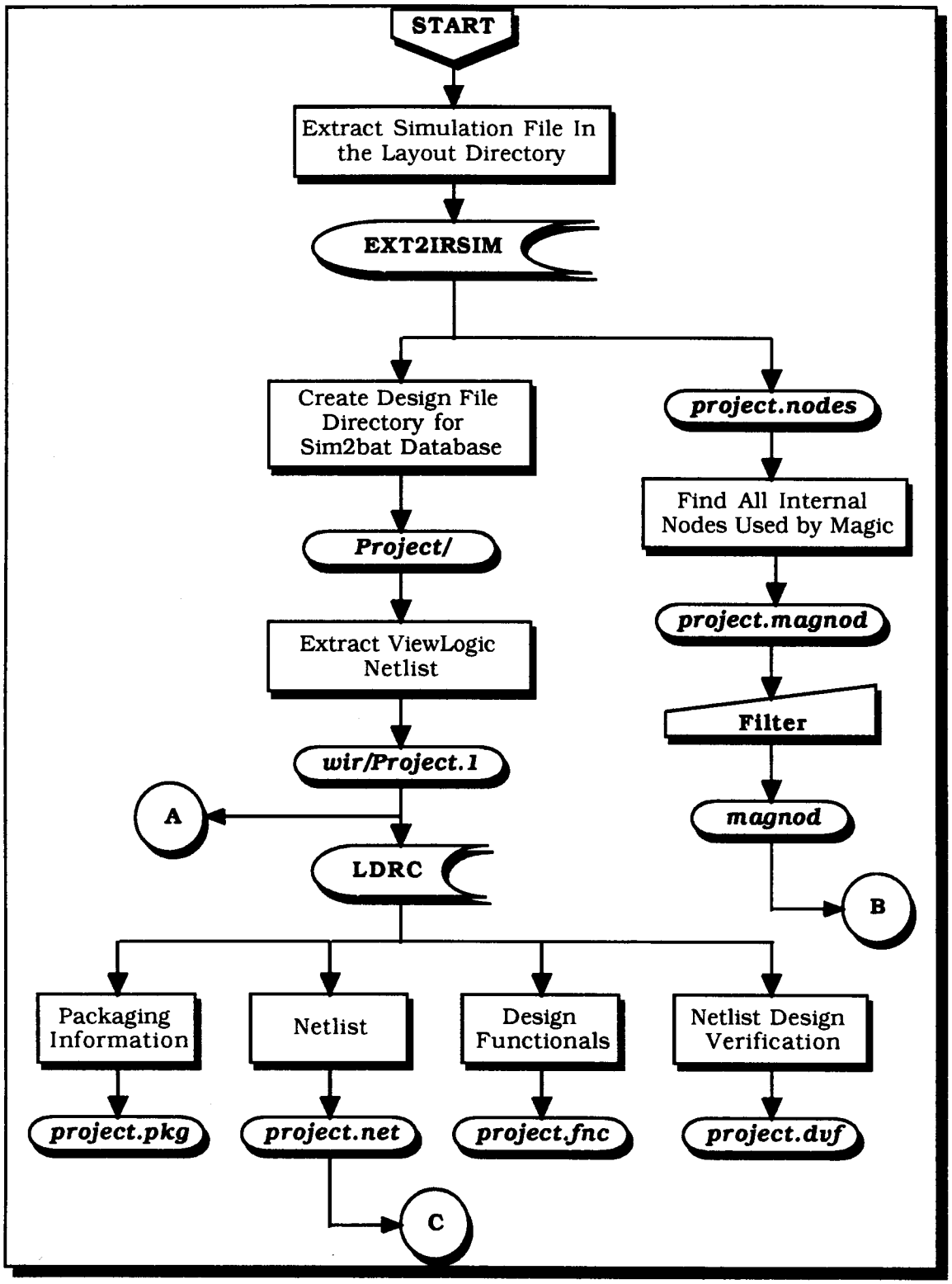
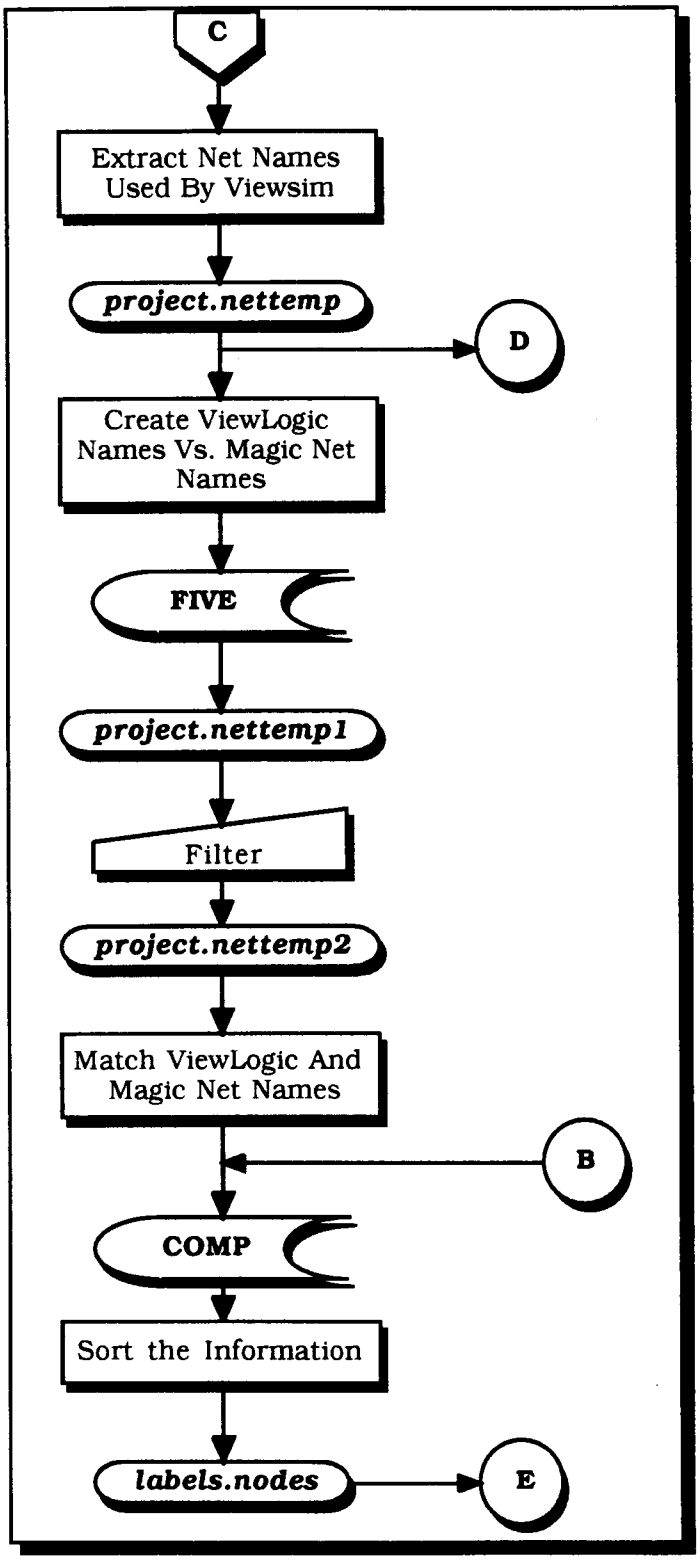
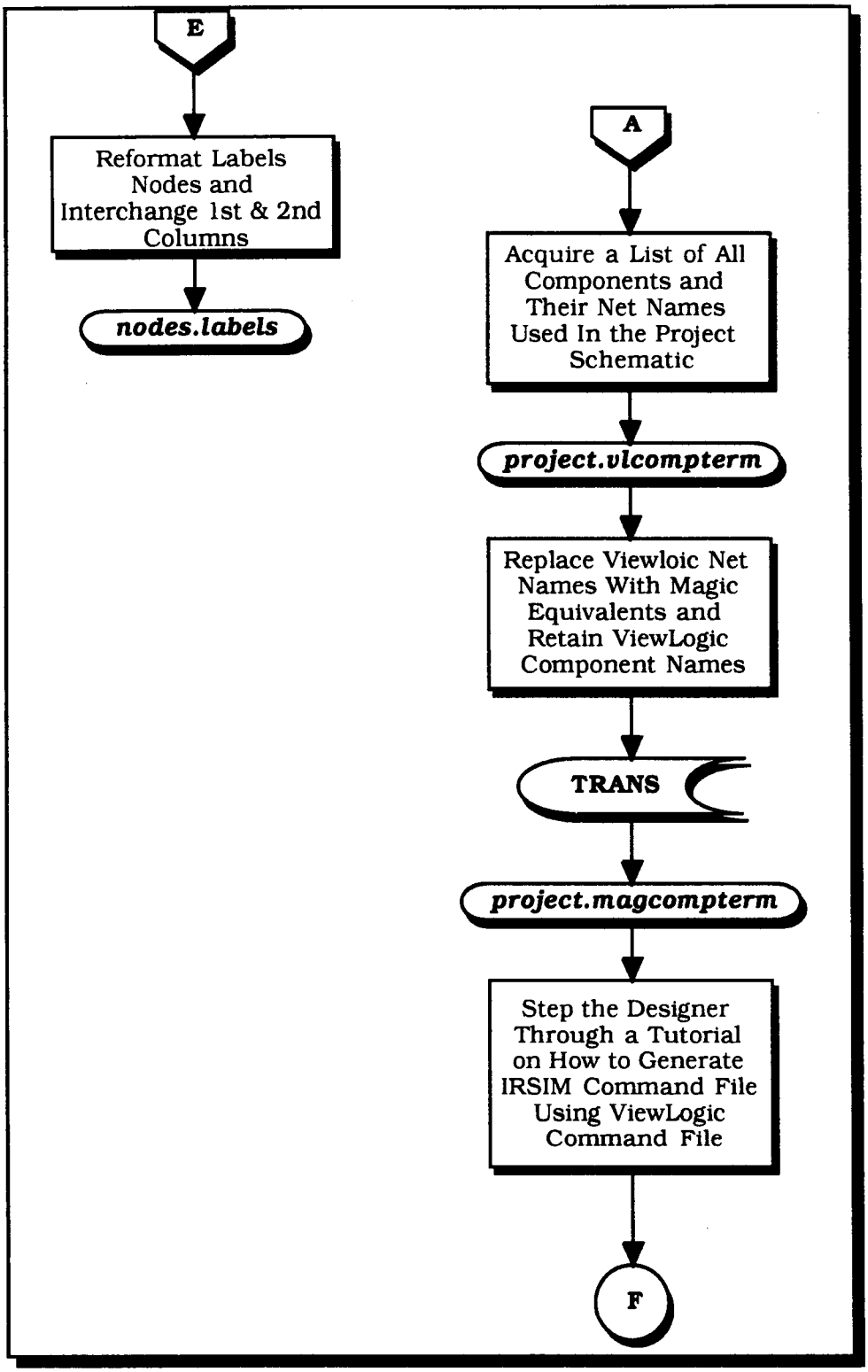


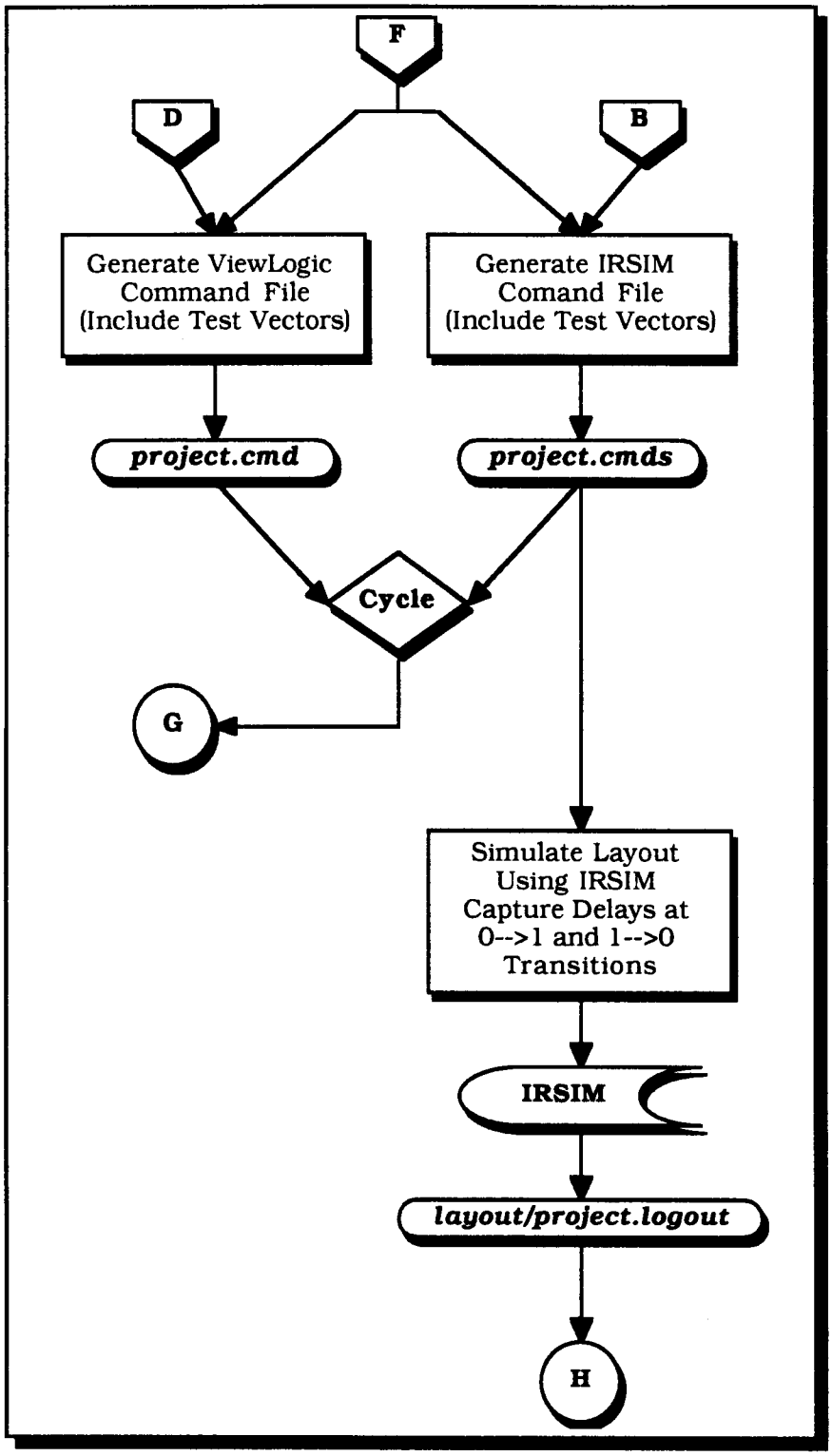

**Figure 3.2 Structured Flow-Chart for the Sim2Bat Algorithm**


C
Extract Net Names Used By Viewsim
project.nettemp
D
Create ViewLogic Names Vs. Magic Net Names
FIVE
project.nettemp1
Filter
project.nettemp2
Match ViewLogic And Magic Net Names
B
COMP
Sort the Information
labels.nodes
E

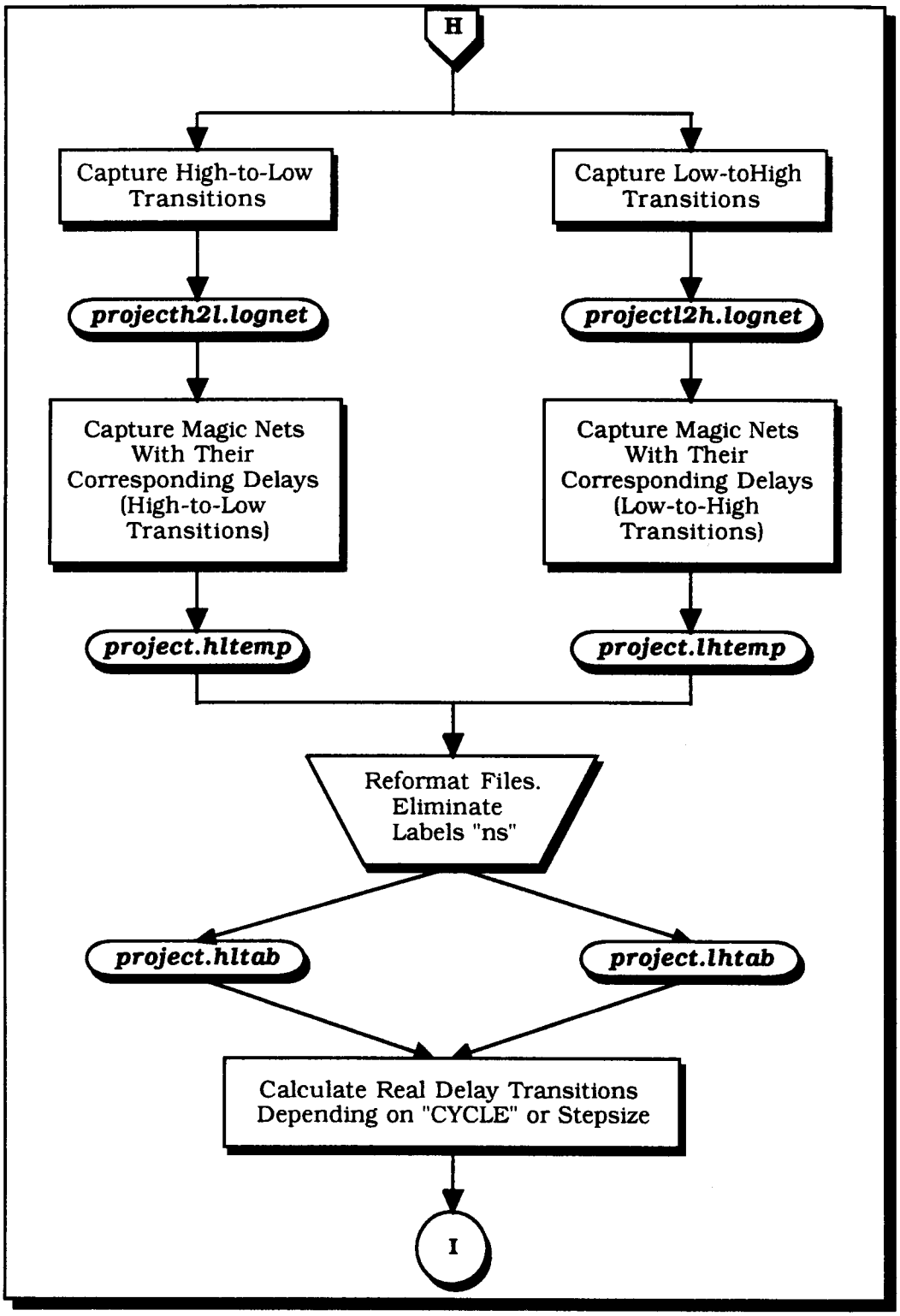
Continued


E
Reformat Labels Nodes and Interchange 1st & 2nd Columns
nodes.labels
A
Acquire a List of All Components and Their Net Names Used In the Project Schematic
project.vlcompterm
Replace Viewloic Net Names With Magic Equivalents and Retain ViewLogic Component Names
TRANS
project.magcompterm
Step the Designer Through a Tutorial on How to Generate IRSIM Command File Using ViewLogic Command File
F

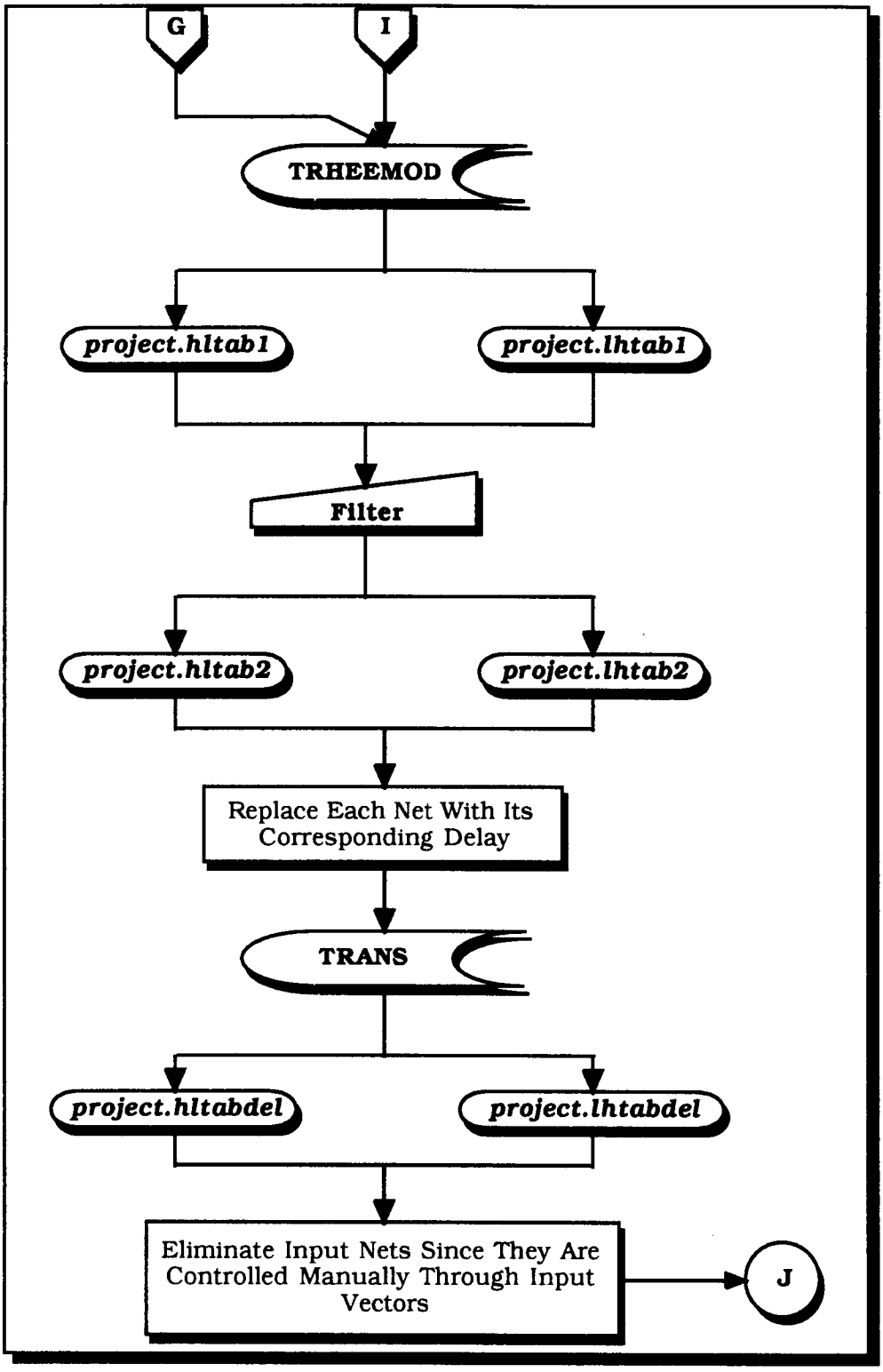
**Continued**


F
D
B
Generate ViewLogic Command File (Include Test Vectors)
Generate IRSIM Comand File (Include Test Vectors)
project.cmd
project.cmds
Cycle
G
Simulate Layout Using IRSIM Capture Delays at 0-->1 and 1-->0 Transitions
IRSIM
layout/project.logout
H

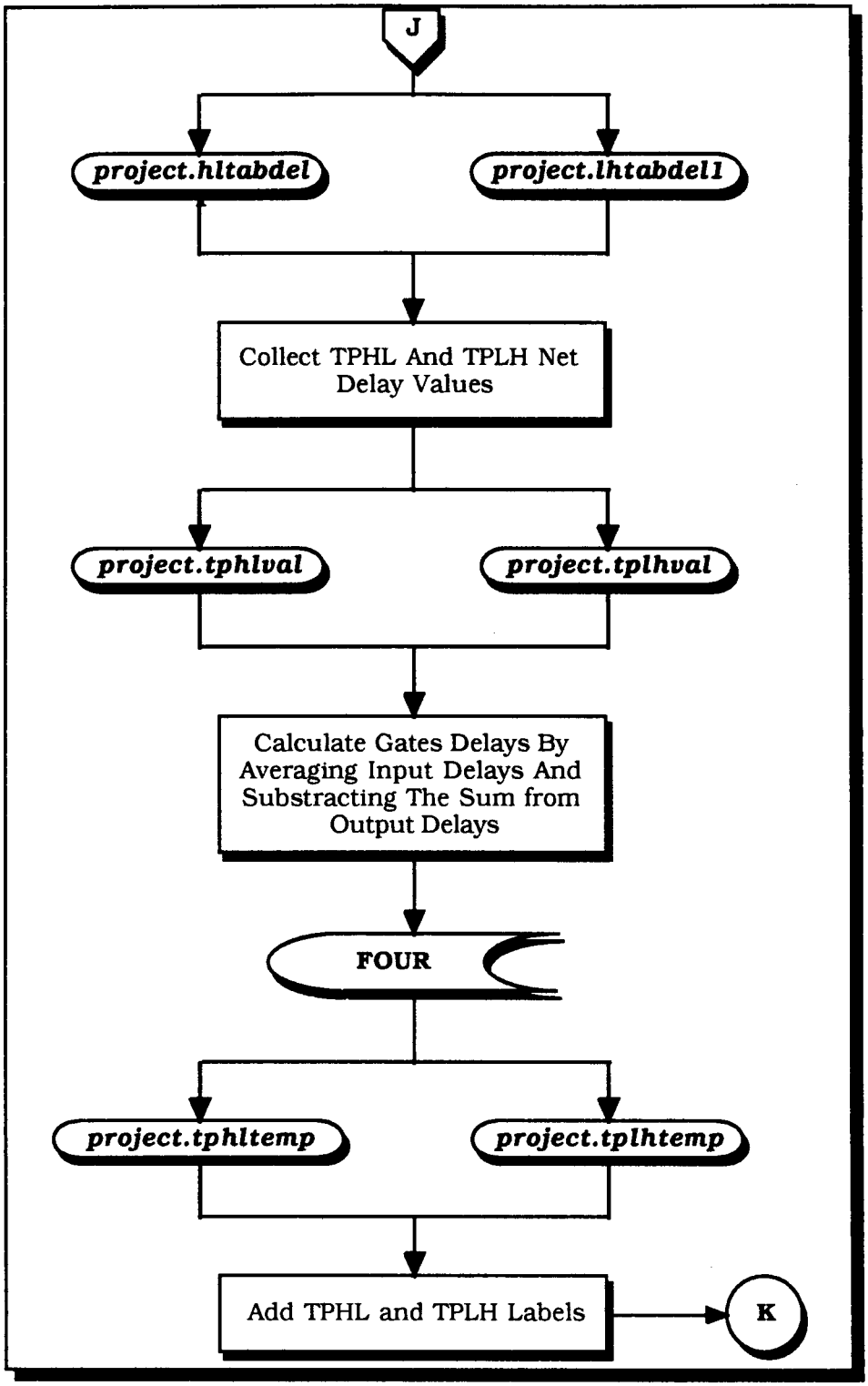
Continued


H
Capture High-to-Low Transitions
Capture Low-toHigh Transitions
projecth2l.lognet
projectl2h.lognet
Capture Magic Nets With Their Corresponding Delays (High-to-Low Transitions)
Capture Magic Nets With Their Corresponding Delays (Low-to-High Transitions)
project.hltemp
project.lhtemp
Reformat Files. Eliminate Labels "ns"
project.hltab
project.lhtab
Calculate Real Delay Transitions Depending on "CYCLE" or Stepsize
I

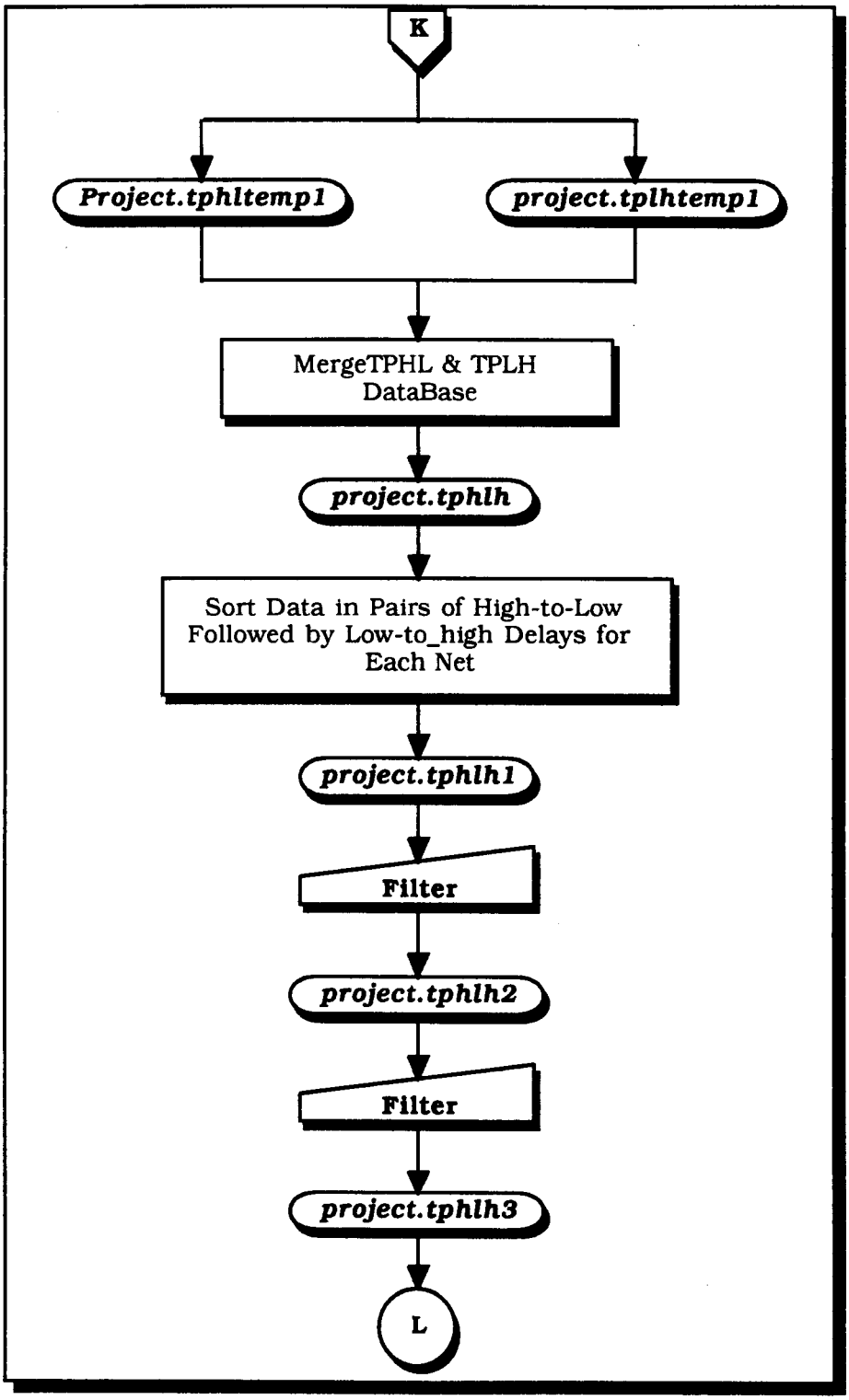
Continued


G
I
TRHEEMOD
project.hltab1
project.lhtab1
Filter
project.hltab2
project.lhtab2
Replace Each Net With Its Corresponding Delay
TRANS
project.hltabdel
project.lhtabdel
Eliminate Input Nets Since They Are Controlled Manually Through Input Vectors
J

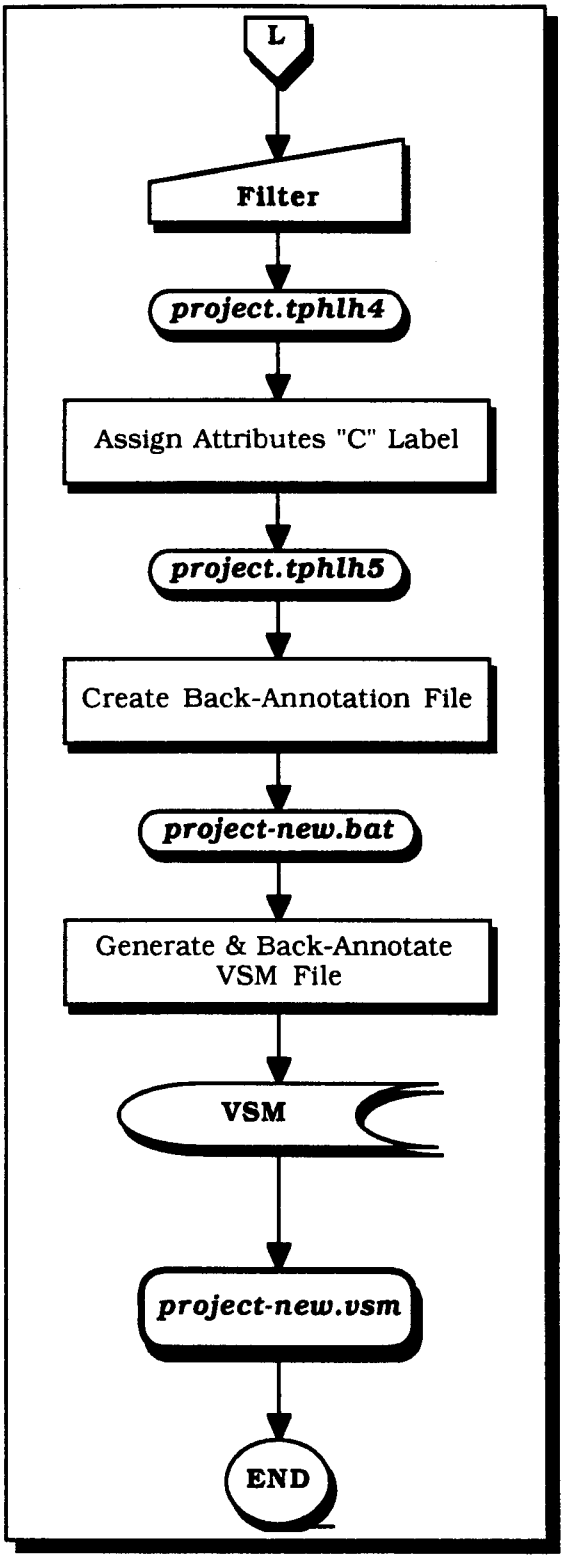
**Continued**


J
project.hltabdel
project.lhtabdel1
Collect TPHL And TPLH Net Delay Values
project.tphlval
project.tplhval
Calculate Gates Delays By Averaging Input Delays And Substracting The Sum from Output Delays
FOUR
project.tphltemp
project.tplhtemp
Add TPHL and TPLH Labels
K

**Continued**


K
Project.tphltemp1
project.tplhtemp1
MergeTPHL & TPLH
DataBase
project.tphlh
Sort Data in Pairs of High-to-Low
Followed by Low-to_high Delays for
Each Net
project.tphlh1
Filter
project.tphlh2
Filter
project.tphlh3
L

Continued


L
Filter
project.tphlh4
Assign Attributes "C" Label
project.tphlh5
Create Back-Annotation File
project-new.bat
Generate & Back-Annotate VSM File
VSM
project-new.vsm
END

Concluded

```
94 V1I18/6_18_76 1700 5400
brown;
94 O2 -1600 800 m1;
94 I3 1900 5000 p;
94 V1I1/6_26_6 4200 1900
green;
94 I2 3400 800 v;
94 I1 5000 -800 m2;
94 V1I2/6_26_6 6600 1900
green;
94 GND -1600 1900 m1;
94 O1 7000 6000 brown;
94 Vdd 5300 5200 nsd;
94 V1I1/O 6400 1700 p;
94 I4 7400 0 v;
```

**Figure 3.3 examp1.nodes**

```
O2
I3
I2
I1
O1
V1I1/O
I4
```

**Figure 3.4 examp1magnod**

```
V 4.1
|
| Wirelist created using VIEWSIM WIRELISTER 4.11
| Using initialization file '/u4/cad/viewlogic/standard/vsm.ini'
|   Using 24 attribute filter(s)
| Long format VIEWSIM file for project EXAMP1
|
DW EXAMP1 4 262995077300
M $1I1 NANF201 ( CELLIBTYPE=\STDCELL NEW_SYMBOL
SIMMODEL=VHDL VHDL=NANF201 ) +
 I1 I2 $1N21
M $1I17 SMV ( LAYOUT_GENERATOR=\STDCELL
SIVMASTER=EXAMP1 STRUCTURE_PROCESSOR )
M $1I18 NORF201 ( CELLIBTYPE=\STDCELL NEW_SYMBOL
SIMMODEL=VHDL VHDL=NORF201 ) +
 $1N21 I3 O2
M $1I2 NANF201 ( CELLIBTYPE=\STDCELL NEW_SYMBOL
SIMMODEL=VHDL VHDL=NANF201 ) +
 I4 $1N21 O1
W D
```

**Figure 3.5 examp1.vsm**

```
$1N21;$1I2^B1,
  $1I18^A1,
  $1I1^O

I1;$1I1^A1

I2;$1I1^B1

I3;$1I18^B1

I4;$1I2^A1

O1;$1I2^O

O2;$1I18^O
```

**Figure 3.6 examp1.net**

```
NORF201;;$1I18
NANF201;;$1I2,$1I1
```

**Figure 3.7 examp1.pkg**

```
NANF201;IC;0
PINORDER D1; A1,B1,O
PINTYPE D1; IN,IN,OUT
FUNCTION D1; F1; A1,B1,O
END

NORF201;IC;0
PINORDER D2; A1,B1,O
PINTYPE D2; IN,IN,OUT
FUNCTION D2; F1; A1,B1,O
END
```

**Figure 3.8 examp1.dvf**

```
$1I1
$1I18
$1I2
```

**Figure 3.9 examp1.fnc**

```
LDRC - V3.2; Workview 4.1.0 060591,
6000 Series
c Copyright 1985,1991 by Viewlogic
Systems, Inc.

ERROR: Net I1 has 0 output pins.
ERROR: Net I2 has 0 output pins.
ERROR: Net I3 has 0 output pins.
ERROR: Net I4 has 0 output pins.
ERROR: Net O1 has 0 input pins.
ERROR: Net O2 has 0 input pins.
```

**Figure 3.10 examp1.drc**

Next, the net names used internally by ***Viewsim*** are extracted from the *examp1.net* file and stored in the file *examp1.nettemp* (see Figures 3.11 and 3.12).

In order to create a database that contains Viewlogic component names versus ***Magic*** net names, the program ***FIVE*** was developed (see Figure 3.13). This program assigns *K-names* to all non-Viewlogic components which will be transformed into ***Magic*** *V-names* equivalents; the results are stored in the file *examp1.nettemp1* (see Figure 3.14) and then filtered to fit a certain format and create the file *examp1.nettemp1* (see Figure 3.15).

One of the languages used to create ***Sim2Bat*** is the **C** language; it is used to write the program ***COMP.C*** (see Appendix E). The software ***COMP*** is a utility used to search both ***Magic*** and Viewlogic databases and match the corresponding net names. The files *examp1magnod* and *examp1.nettemp2* are compared and the results are sorted to produce the *labels.nodes* file (see Figure 3.16).

Using the *examp1.wir* wire-list, ***Sim2Bat*** acquires a list of all the components and their Input/Output net names used in the *examp1backit* directory; the results are stored into the file *examp1.vlcompterm*. This latter file is then used by the ***TRANS*** Viewlogic utility in order to replace Viewlogic net names with ***Magic*** equivalents while retaining the Viewlogic component names. The file produced by this software is *examp1.magcompterm* (see Figure 3.17).

Although ***Sim2Bat*** promotes the automation of back-annotation tools; nonetheless, it requires some user intervention in order to maintain a certain level of stability and compatibility among the various programs it links. One such user interface is the step where ***Sim2Bat*** prompts and steps the designer through a small tutorial on how to translate the Viewlogic command file *examp1.cmd* into an ***IRSIM*** command file *examp1.cmds* (see Figures 2.5 and 2.18). Due to the great similarities between both formats this translation task is limited to a few substitution changes such as changing the ***Viewsim*** *watch* command into the ***IRSIM*** *analyzer* command (see chapter 2).

```
$1N21;$1I2^B1,
  $1I18^A1,
  $1I1^O

I1;$1I1^A1

I2;$1I1^B1

I3;$1I18^B1

I4;$1I2^A1

O1;$1I2^O

O2;$1I18^O
```

**Figure 3.11 examp1.net**

```
$1N21 $1I2/B1  $1I18/A1
$1I1/O
I1 $1I1/A1
I2 $1I1/B1
I3 $1I18/B1
I4 $1I2/A1
O1 $1I2/O
O2 $1I18/O
```

**Figure 3.12 examp1.nettemp**

```
{
        f1 = $1
        f2 = $2
        f3 = $3
        f4 = $4
        f5 = $5
        f6 = $6
if ( f2 ~ /^$[A-Z1-9].*\/[A-Z1-9].*/ ) f2 = "K"f2
if ( f3 ~ /^$[A-Z1-9].*\/[A-Z1-9].*/ ) f3 = "K"f3
if ( f4 ~ /^$[A-Z1-9].*\/[A-Z1-9].*/ ) f4 = "K"f4
if ( f5 ~ /^$[A-Z1-9].*\/[A-Z1-9].*/ ) f5 = "K"f5
if ( f6 ~ /^$[A-Z1-9].*\/[A-Z1-9].*/ ) f6 = "K"f6
        print f1" " f2" " f3" " f4" " f5" " f6" "
}
```

**Figure 3.13 Listing of Program FIVE**

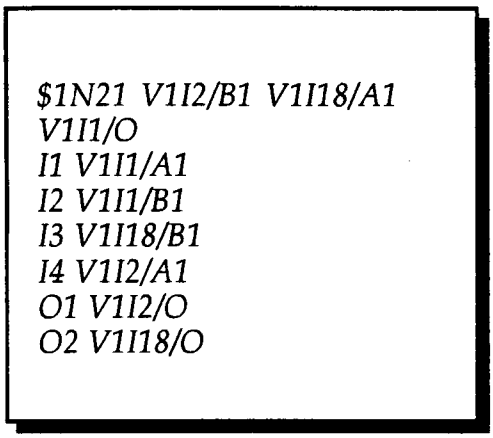



**Figure 3.14 examp1.nettemp1**

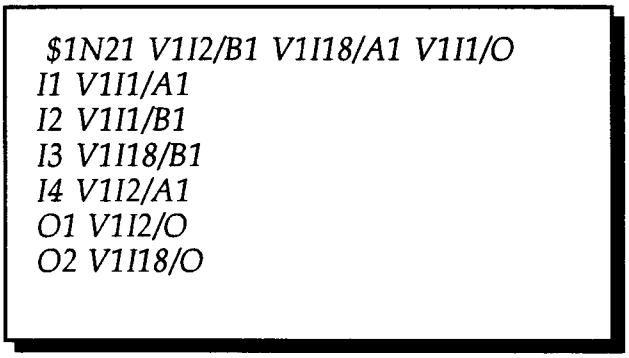



**Figure 3.15 examp1nettemp2**

```
O2 O2
I3 I3
I2 I2
I1 I1
O1 O1
V1I1/O $1N21
I4 I4
```

**Figure 3.16 labels.nodes**

```
$1I1 I1 I2 V1I1/O
$1I18 V1I1/O I3 O2
$1I2 I4 V1I1/O O1
```

**Figure 3.17 examp1.magcompterm**

```
| (./layout/examp1.cmds,4): 'I1' is a node, can't be a vector
| [event #1] input node I4: -> 0 @ 0.0ns
| [event #2] input node I3: -> 0 @ 0.0ns
| [event #3] input node I2: -> 0 @ 0.0ns
| [event #4] input node I1: -> 0 @ 0.0ns
| [event #5] node O1: X -> 1 @ 0.2ns
| [event #7] node V1I1/O: X -> 1 @ 0.3ns
| [event #9] node O2: X -> 0 @ 0.7ns
| [event #10] input node I1: -> 1 @ 20.0ns
| time = 40.0ns
| time = 40.0ns
| [event #12] input node I1: -> 0 @ 40.0ns
| [event #13] input node I1: -> 1 @ 60.0ns
| time = 80.0ns
| time = 80.0ns
| [event #14] input node I1: -> 0 @ 80.0ns
| [event #15] input node I2: -> 1 @ 80.0ns
| [event #17] input node I1: -> 1 @ 100.0ns
| [event #19] node V1I1/O: 1 -> 0 @ 100.8ns
| [event #20] node O2: 0 -> 1 @ 101.3ns
| time = 120.0ns
| time = 120.0ns
| [event #21] input node I1: -> 0 @ 120.0ns
| [event #22] node V1I1/O: 0 -> 1 @ 120.7ns
| [event #24] node O2: 1 -> 0 @ 121.2ns
| [event #25] input node I1: -> 1 @ 140.0ns
| [event #27] node V1I1/O: 1 -> 0 @ 140.8ns
| [event #28] node O2: 0 -> 1 @ 141.3ns
```

**Figure 3.18 examp1.lognet**

```
| time = 400.0ns
| time = 400.0ns
|  [event #63] input node I1: -> 0 @ 400.0ns
|  [event #64] input node I2: -> 1 @ 400.0ns
|  [event #66] input node I1: -> 1 @ 420.0ns
|  [event #68] node V1I1/O: 1 -> 0 @ 420.8ns
|  [event #69] node O1: 0 -> 1 @ 421.3ns
|  [event #70] node O2: 0 -> 1 @ 421.3ns
| time = 440.0ns
| time = 440.0ns
|  [event #71] input node I1: -> 0 @ 440.0ns
|  [event #72] node V1I1/O: 0 -> 1 @ 440.7ns
|  [event #74] node O1: 1 -> 0 @ 441.1ns
|  [event #75] node O2: 1 -> 0 @ 441.2ns
|  [event #76] input node I1: -> 1 @ 460.0ns
|  [event #78] node V1I1/O: 1 -> 0 @ 460.8ns
|  [event #79] node O1: 0 -> 1 @ 461.3ns
|  [event #80] node O2: 0 -> 1 @ 461.3ns
| time = 480.0ns
| time = 480.0ns
|  [event #81] input node I2: -> 0 @ 480.0ns
|  [event #82] input node I1: -> 0 @ 480.0ns
|  [event #83] input node I3: -> 1 @ 480.0ns
|  [event #84] node V1I1/O: 0 -> 1 @ 480.3ns
|  [event #85] node O2: 1 -> 0 @ 480.4ns
|  [event #86] node O1: 1 -> 0 @ 480.6ns
|  [event #87] input node I1: -> 1 @ 500.0ns
| time = 520.0ns
| time = 520.0ns
|  [event #88] input node I1: -> 0 @ 520.0ns
|  [event #89] input node I1: -> 1 @ 540.0ns
| time = 560.0ns
| time = 560.0ns
|  [event #90] input node I1: -> 0 @ 560.0ns
|  [event #91] input node I2: -> 1 @ 560.0ns
|  [event #93] input node I1: -> 1 @ 580.0ns
|  [event #95] node V1I1/O: 1 -> 0 @ 580.8ns
|  [event #97] node O1: 0 -> 1 @ 581.3ns
| time = 600.0ns
| time = 600.0ns
|  [event #98] input node I1: -> 0 @ 600.0ns
|  [event #99] node V1I1/O: 0 -> 1 @ 600.7ns
|  [event #101] node O1: 1 -> 0 @ 601.1ns
|  [event #102] input node I1: -> 1 @ 620.0ns
|  [event #104] node V1I1/O: 1 -> 0 @ 620.8ns
|  [event #105] node O1: 0 -> 1 @ 621.3ns
```

**Continued**

```
| time = 160.0ns
| time = 160.0ns
|  [event #29] input node I2: -> 0 @ 160.0ns
|  [event #30] input node I1: -> 0 @ 160.0ns
|  [event #31] input node I3: -> 1 @ 160.0ns
|  [event #32] node V1I1/O: 0 -> 1 @ 160.3ns
|  [event #33] node O2: 1 -> 0 @ 160.4ns
|  [event #34] input node I1: -> 1 @ 180.0ns
| time = 200.0ns
| time = 200.0ns
|  [event #35] input node I1: -> 0 @ 200.0ns
|  [event #36] input node I1: -> 1 @ 220.0ns
| time = 240.0ns
| time = 240.0ns
|  [event #37] input node I1: -> 0 @ 240.0ns
|  [event #38] input node I2: -> 1 @ 240.0ns
|  [event #40] input node I1: -> 1 @ 260.0ns
|  [event #42] node V1I1/O: 1 -> 0 @ 260.8ns
| time = 280.0ns
| time = 280.0ns
|  [event #44] input node I1: -> 0 @ 280.0ns
|  [event #45] node V1I1/O: 0 -> 1 @ 280.7ns
|  [event #47] input node I1: -> 1 @ 300.0ns
|  [event #49] node V1I1/O: 1 -> 0 @ 300.8ns
| time = 320.0ns
| time = 320.0ns
|  [event #50] input node I3: -> 0 @ 320.0ns
|  [event #51] input node I2: -> 0 @ 320.0ns
|  [event #52] input node I1: -> 0 @ 320.0ns
|  [event #53] input node I4: -> 1 @ 320.0ns
|  [event #56] node O2: 0 -> 1 @ 320.3ns
|  [event #57] node V1I1/O: 0 -> 1 @ 320.3ns
|  [event #58] node O1: 1 -> 0 @ 320.6ns
|  [event #59] node O2: 1 -> 0 @ 320.7ns
|  [event #60] input node I1: -> 1 @ 340.0ns
| time = 360.0ns
| time = 360.0ns
|  [event #61] input node I1: -> 0 @ 360.0ns
|  [event #62] input node I1: -> 1 @ 380.0ns
```

**Concluded**

This step can be automated, however, it serves as a good exercise for the user and promotes the revision of the input stimulus which plays a great role into accomplishing good results (see chapter 4).

The user needs to examine the *nodes.labels* file in order to determine the nodes which should be observed by ***IRSIM*** and ***Viewsim***. Once the command file is produced, the user needs to specify the clock speed at which the circuit is desired to operate. This clock speed is specified in the command files using the command *stepsize* and must be included in the developed program ***Threemod*** (which will be discussed later) by replacing the global variable "cycle" with the *stepsize* value, only if it is different than *20ns*.

Using the command file *"examp1.cmds"*, the design layout is simulated using ***IRSIM***. The delays are then captured at the Zero-to-One (0-->1) and One-to-Zero (1-->0) transitions and stored under the layout directory in the file *examp1.lognet* (see Figure 3.18).

The High-to-Low and Low-to-High transitional delays are then extracted from the *examp1.logout* file in order to produce *examp1h2l.lognet* and *examp1h2.lognet* files, respectively (see Figures 3.19 and 3.20). These latter files are then processed in order to capture the ***Magic*** nets with their corresponding delays. The output files are *examp1.hltemp* for High-to-Low transitions and *"examp1.lhtemp"* for Low-to-High transitions (see Figures 3.21 and 3.22). Both files are reformatted in order to prepare them for the delay calculation program; the resulting files are *examp1.hltab* and *examp1.lhtab,* respectively (see Figures 3.23 and 3.24).

Using *examp1.hltab* and *examp1.lhtab* files and depending on the simulation stepsize "cycle", the Real Delay transitions are calculated using the program ***Threemod*** (see Figure 2.26). This utility calculates the real delays by sizing-down the transition delays to be less than the "cycle" stepsize. This step is very crucial for the success of ***Sim2Bat*** and prevents later errors. The resulting files are *examp1.hltab1* and *examp1.lhtab1* which are then filtered to produce *examp1.hltab2* and *examp1.lhtab2*, respectively (see Figures 26 and 27).

```
[event #19] node V1I1/O: 1 -> 0 @ 100.8ns
[event #24] node O2: 1 -> 0 @ 121.2ns
[event #27] node V1I1/O: 1 -> 0 @ 140.8ns
[event #33] node O2: 1 -> 0 @ 160.4ns
[event #42] node V1I1/O: 1 -> 0 @ 260.8ns
[event #49] node V1I1/O: 1 -> 0 @ 300.8ns
[event #58] node O1: 1 -> 0 @ 320.6ns
[event #59] node O2: 1 -> 0 @ 320.7ns
[event #68] node V1I1/O: 1 -> 0 @ 420.8ns
[event #74] node O1: 1 -> 0 @ 441.1ns
[event #75] node O2: 1 -> 0 @ 441.2ns
[event #78] node V1I1/O: 1 -> 0 @ 460.8ns
[event #85] node O2: 1 -> 0 @ 480.4ns
[event #86] node O1: 1 -> 0 @ 480.6ns
[event #95] node V1I1/O: 1 -> 0 @ 580.8ns
[event #101] node O1: 1 -> 0 @ 601.1ns
[event #104] node V1I1/O: 1 -> 0 @ 620.8ns
```

**Figure 3.19 examp1h2l.lognet**

```
[event #20] node O2: 0 -> 1 @ 101.3ns
[event #22] node V1I1/O: 0 -> 1 @ 120.7ns
[event #28] node O2: 0 -> 1 @ 141.3ns
[event #32] node V1I1/O: 0 -> 1 @ 160.3ns
[event #45] node V1I1/O: 0 -> 1 @ 280.7ns
[event #56] node O2: 0 -> 1 @ 320.3ns
[event #57] node V1I1/O: 0 -> 1 @ 320.3ns
[event #69] node O1: 0 -> 1 @ 421.3ns
[event #70] node O2: 0 -> 1 @ 421.3ns
[event #72] node V1I1/O: 0 -> 1 @ 440.7ns
[event #79] node O1: 0 -> 1 @ 461.3ns
[event #80] node O2: 0 -> 1 @ 461.3ns
[event #84] node V1I1/O: 0 -> 1 @ 480.3ns
[event #97] node O1: 0 -> 1 @ 581.3ns
[event #99] node V1I1/O: 0 -> 1 @ 600.7ns
[event #105] node O1: 0 -> 1 @ 621.3ns
```

**Figure 3.20 examp1l2h.lognet**

*V1I1/O:100.8ns*
*O2:121.2ns*
*V1I1/O:140.8ns*
*O2:160.4ns*
*V1I1/O:260.8ns*
*V1I1/O:300.8ns*
*O1:320.6ns*
*O2:320.7ns*
*V1I1/O:420.8ns*
*O1:441.1ns*
*O2:441.2ns*
*V1I1/O:460.8ns*
*O2:480.4ns*
*O1:480.6ns*
*V1I1/O:580.8ns*
*O1:601.1ns*
*V1I1/O:620.8ns*

**Figure 3.21 examp1.hltemp**

*O2:101.3ns*
*V1I1/O:120.7ns*
*O2:141.3ns*
*V1I1/O:160.3ns*
*V1I1/O:280.7ns*
*O2:320.3ns*
*V1I1/O:320.3ns*
*O1:421.3ns*
*O2:421.3ns*
*V1I1/O:440.7ns*
*O1:461.3ns*
*O2:461.3ns*
*V1I1/O:480.3ns*
*O1:581.3ns*
*V1I1/O:600.7ns*
*O1:621.3ns*

**Figure 3.22 examp1.lhtemp**

```
V1I1/O 100.8
O2 121.2
V1I1/O 140.8
O2 160.4
V1I1/O 260.8
V1I1/O 300.8
O1 320.6
O2 320.7
V1I1/O 420.8
O1 441.1
O2 441.2
V1I1/O 460.8
O2 480.4
O1 480.6
V1I1/O 580.8
O1 601.1
V1I1/O 620.8
```

**Figure 3.23 examp1.hltab**

```
O2 101.3
V1I1/O 120.7
O2 141.3
V1I1/O 160.3
V1I1/O 280.7
O2 320.3
V1I1/O 320.3
O1 421.3
O2 421.3
V1I1/O 440.7
O1 461.3
O2 461.3
V1I1/O 480.3
O1 581.3
V1I1/O 600.7
O1 621.3
```

**Figure 3.24 examp1.lhtab**

```
BEGIN {
        cycle = 20
    }
{       f1 = $1
        f2 = $2
        f3 = $3
        f4 = $4
        f5 = $5
        f6 = $6

for ( j = 1; j = 11; j++){

        f2 = f2 - (int(f2/cycle))*cycle
        f3 = f3 - (int(f3/cycle))*cycle
        f4 = f4 - (int(f4/cycle))*cycle
        f5 = f5 - (int(f5/cycle))*cycle
        f6 = f6 - (int(f6/cycle))*cycle
        print f1" " f2" " f3" " f4" " f5" " f6" "
        break;
        }
```

**Figure 3.25 Listing of Threemod Program**

```
O1 0.6ns
O1 0.6ns
O1 1.1ns
O1 1.1ns
O2 0.4ns
O2 0.4ns
O2 0.7ns
O2 1.2ns
O2 1.2ns
V1I1/O 0.8ns
V1I1/O 0.8ns
V1I1/O 0.8ns
V1I1/O 0.8ns
V1I1/O 0.8ns
V1I1/O 0.8ns
V1I1/O 0.8ns
V1I1/O 0.8ns
```

**Figure 3.26 examp1.hltab2**

```
O1 1.3ns
O1 1.3ns
O1 1.3ns
O1 1.3ns
O2 0.3ns
O2 1.3ns
O2 1.3ns
O2 1.3ns
O2 1.3ns
V1I1/O 0.3ns
V1I1/O 0.3ns
V1I1/O 0.3ns
V1I1/O 0.7ns
V1I1/O 0.7ns
V1I1/O 0.7ns
V1I1/O 0.7ns
```

**Figure 3.27 examp1.lhtab2**

The last two delay files are used by ***Sim2Bat*** to replace each net with its corresponding delay. Hence the utility ***TRANS*** is used and the resulting files are *examp1.hltabdel* and *examp1.lhtabdel* (see Figures 3.28 and 3.29). These files are then screened using the ***awk*** utility ***Two*** (see Figure 3.30) in order to eliminate the input nets, since they are controlled manually through the input vectors. The output files are *examp1.hltabdel1* and *examp1.lhtabdel1* (see Figures 3.31 and 3.32).

Following this last action, TPHL (High-to-Low) and TPLH (Low-to-High) values used by the ***Viewsim*** simulator are collected to produce the files *examp1tphlval* and *examp1tplhval* (see Figures 3.33 and 3.34).

In order to calculate the gate delays, the program ***Four*** was devised (see Figure 3.35). This program calculates the component delays by averaging the inputs' delays and subtracting the result from the output's delay. The files become *examp1.tphltemp* and *examp1. tplhtemp* (see Figures 3.36 and 3.37).

After adding the TPHL and TPLH labels to the previous two files, they become *examp1.tphltemp1* and *examp1.tplhtemp1*. Merging both databases creates the file *examp1.tphlh* (see Figures 3.38 through 3.40). The data is then sorted in pairs of High-to-Low followed by Low-to-High delays for each net. Then three formatting filters are applied in addition to assigning the "C" attribute label needed by ***Viewsim*** to denote "component delay". The result is the file examp1-*new.bat* (see Figure 3.41). This is the back-annotation file that feeds ***Viewsim*** with the gate and net delays extracted form the layout.

The last step in ***Sim2Bat*** is to generate and back-annotate the *vsm* file; this is done using the Viewlogic ***VSM*** utility which produces the new and improved simulation file *examp1-new.vsm* (see Figure 3.42).

```
$1I1 I1 I2 0.8ns
$1I18 0.8ns I3 1.2ns
$1I2 I4 0.8ns 1.1ns
```

**Figure 3.28 examp1.hltabdel**

```
$1I1 I1 I2 0.7ns
$1I18 0.7ns I3 1.3ns
$1I2 I4 0.7ns 1.3ns
```

**Figure 3.29 examp1.lhtadel**

```
{
        f1 = $1
        f2 = $2
        f3 = $3
        f4 = $4
        f5 = $5
        f6 = $6
        if ( f2 !~ /[1-9\.][1-9\.]*.*ns/ ) f2 = ""
        if ( f3 !~ /[1-9\.][1-9\.]*.*ns/ ) f3 = ""
        if ( f4 !~ /[1-9\.][1-9\.]*.*ns/ ) f4 = ""
        if ( f5 !~ /[1-9\.][1-9\.]*.*ns/ ) f5 = ""
        {
        print f1" " f2" " f3" " f4" " f5" " f6" "
        }
}
```

**Figure 3.30 Listing of TWO Program**

```
$1I1   0.8ns
$1I18 0.8ns  1.2ns
$1I2  0.8ns 1.1ns
```

**Figure 3.31 examp1.hltabdel1**

```
$1I1   0.7ns
$1I18 0.7ns  1.3ns
$1I2  0.7ns 1.3ns
```

**Figure 3.32 examp1.lhtabdel1**

```
$1I1 0.8
$1I18 0.8 1.2
$1I2 0.8 1.1
```

**Figure 3.33 examp1.tphval**

```
$1I1 0.7
$1I18 0.7 1.3
$1I2 0.7 1.3
```

**Figure 3.34 examp1.tplhval**

```
{       f1 = $1
        f2 = $2
        f3 = $3
        f4 = $4
        f5 = $5
        f6 = $6
for ( j = 1; j <= NF; j++){
        if ( f3 == "" ) compdelay = f2
else    if ( f4 == "" ) compdelay = f3 - f2
else    if ( f5 == "" ) compdelay = f4 - (f3 + f2)/2
else    if ( f6 == "" ) compdelay = f5 -(f4 + f3 + f2)/3
else    if ( f7 =="" ) compdelay = f6 - (f5 + f4 + f3 + f2)/4
        print f1" " compdelay
        break;
        }
}
```

**Figure 3.35 Listing of FOUR Program**

```
$1I1 0.8
$1I18 0.4
$1I2 0.3
```

**Figure 3.36 examp1.tphltemp**

```
$1I1 0.7
$1I18 0.6
$1I2 0.6
```

**Figure 3.37 examp1tplhtemp**

```
$1I1 TPHL=0.8NS,
$1I18 TPHL=0.4NS,
$1I2 TPHL=0.3NS,
```

**Figure 3.38 examp1.tphltemp1**

```
$1I1 TPLH=0.7NS;
$1I18 TPLH=0.6NS;
$1I2 TPLH=0.6NS;
```

**Figure 3.39 examp1.tplhtemp1**

```
$1I1 TPHL=0.8NS,
$1I18 TPHL=0.4NS,
$1I2 TPHL=0.3NS,
$1I1 TPLH=0.7NS;
$1I18 TPLH=0.6NS;
$1I2 TPLH=0.6NS;
```

**Figure 3.40 examp1.tphlh**

```
.BA
C $1I1
A TPHL=0.8NS
A TPLH=0.7NS
C $1I18
A TPHL=0.4NS
A TPLH=0.6NS
C $1I2
A TPHL=0.3NS
A TPLH=0.6NS
.AB
```

**Figure 3.41 examp1-new.bat**

```
V 4.1
|
| Wirelist created using VIEWSIM WIRELISTER 4.11
| Using initialization file '/u4/cad/viewlogic/standard/vsm.ini'
|   Using 24 attribute filter(s)
| Long format VIEWSIM file for project EXAMP1
| Using back-annotation file '/u4/sawan/Project/xilinx1/examp1backit/examp1-
| new.bat'
|   Found 3 component, 0 pin, and 0 net records in examp1backit/examp1-new.bat
|
DW EXAMP1 4 262995077300
M $1I1 NANF201 ( TPHL=0.8NS TPLH=0.7NS CELLIBTYPE=\STDCELL
NEW_SYMBOL +
 SIMMODEL=VHDL VHDL=NANF201 ) I1 I2 $1N21
M $1I17 SMV ( LAYOUT_GENERATOR=\STDCELL SIVMASTER=EXAMP1
STRUCTURE_PROCESSOR )
M $1I18 NORF201 ( TPHL=0.4NS TPLH=0.6NS CELLIBTYPE=\STDCELL
NEW_SYMBOL +
 SIMMODEL=VHDL VHDL=NORF201 ) $1N21 I3 O2
M $1I2 NANF201 ( TPHL=0.3NS TPLH=0.6NS CELLIBTYPE=\STDCELL
NEW_SYMBOL +
 SIMMODEL=VHDL VHDL=NANF201 ) I4 $1N21 O1
W D
```

**Figure 3.42 examp1-new.vsm**

# CHAPTER IV

## RESULTS AND DISCUSSION

### *Checking The Results*

In order to check the results, the new simulation file *examp1-new.vsm* is used to simulate the circuit schematic. The waveforms produced by ***Viewwave*** should match the waveforms generated by ***IRSIM***, and the gate plus net delays can be checked *graphically* using ***Viewwave***. However, in order to show the accuracy and effectiveness of ***Sim2Bat***, the data needs to be visually (manually) compared before, during, and after ***CKTIM*** is executed. Therefore a pre-processor and post-processor routines were developed.

### *The PreProcessor (Pre-Layout) View2Sim*

This program ***View2Sim*** (see Appendix B) takes the results of the initial circuit simulation and extracts the High-to-Low and Low-to-High delay information. Initially, after the designer finishes entering the schematic in Viewlogic, the design is exported which creates a wire and a simulation net-lists (see Figures 4.1 and 3.5). The simulation command file contains a *"trace"* command that logs the transitional delays into a *"trc"* file *examp1.trc* (see Figure 4.2). This file is then processed and filtered in order to produce the two High-to-Low and Low-to-High files, *examp1.trchltab2* and *examp1.trclhtab2* respectively (see Figures 4.3 and 4.4). These files are then processed and the final result is a *"Pass1.bat"* back-annotation delay file *examp1pass1.bat* (see Figure 4.5). The intermediate files generated during the process of ***View2Bat*** are stored in a *"pass1"* directory named *examp1pass1.*

*The Post-Processor (Post-Layout) Bat2sim*

The utility ***Bat2sim*** (see Appendix C) uses the results of the new back-annotated circuit simulation and extracts the High-to-Low and Low-to-High delay information. After the designer runs ***Sim2Bat***, a *"bat"* back-annotation file *examp1-new.bat* (see Figure 3.43) is created and a new ***Viewsim*** simulation file *examp1-new.vsm* is generated (see Figure 3.44). The *vsm* simulation file is used by ***Viewsim*** and a new *"trc"* file *examp1.trc* is generated (see Figure 4.6).

This High-to-Low and Low-to-High transitional delays file is then processed and filtered producing the files *examp1.trchltab2* and *examp1.trclhtab2* (see Figures 4.7 and 4.8). Processing these files produces the *"pass2.bat"* file *examp1pass2.bat*, which contains the TPHL and TPLH delays (see Figure 4.9). The intermediate files generated during the process of ***Bat2Sim*** are stored in a *"pass2"* directory *examp1pass2*.

*Method of Comparison*

The following three graphs (see Figures 4.10, 4.11, and 4.12) depict the three paths that will be used to compare the results. The first path extracts the transitional delays produced by ***Viewsim*** using the original *"vsm"* simulation file and generates an attributes file *pass1.bat*. The second path is the extraction of the delays produced by ***IRSIM*** and generation of a real-time delay attributes file *pass2.bat*. The third path basically extracts the delays produced by ***Viewsim*** using the improved *"new.vsm"* simulation file and produces a attribute file *"-new.bat"*. The data produced by the second and the third paths will prove to be similar if not identical and the first path gives erroneous delay results originally used by the designer before the ***CKTIM*** is used.

*Transitional and Gate Delay Comparison*

In order to show that ***Sim2Bat*** back-annotates the delays correctly from ***IRSIM*** to ***Viewsim***, two comparisons are needed. First, the transitional

```
| Wirelist created using version 4.09.
V 4.09
K 262995077300 examp1
DW  examp1
AS fti TERM_RELATIVE_EDGE
AS fti TERM_EDGE
AS fti TERMTYPE=SIGNAL
AS fti DIRECTION=INPUT
AS fti PINORDER = IN
AP fti 1 TERMTYPE=SIGNAL
AP fti 1 TERM_EDGE
AP fti 1 DIRECTION=INPUT
AP fti 1 TERM_RELATIVE_EDGE
AS fto TERM_RELATIVE_EDGE
AS fto TERM_EDGE
AS fto TERMTYPE=SIGNAL
AS fto DIRECTION=OUTPUT
AS fto PINORDER = OUT
AP fto 1 TERMTYPE=SIGNAL
AP fto 1 TERM_RELATIVE_EDGE
AP fto 1 TERM_EDGE
AP fto 1 DIRECTION=OUTPUT
AS smv LAYOUT_GENERATOR
AS smv STRUCTURE_PROCESSOR
AS nanf201 NEW_SYMBOL
AS nanf201 VHDL=NANF201
AS nanf201 SIMMODEL=VHDL
AS nanf201 PINORDER=A1 B1 O
AS nanf201 CELLIBTYPE=\STDCELL
AS nanf201 LEVEL=LEAFCELL
AP nanf201 1 PINTYPE=IN
AP nanf201 2 PINTYPE=IN
AP nanf201 3 PINTYPE=OUT
```

**Figure 4.1 examp1.1 - Wire Net-list**

```
AS norf201 PINORDER=A1 B1 O
AS norf201 VHDL=NORF201
AS norf201 SIMMODEL=VHDL
AS norf201 LEVEL=LEAFCELL
AS norf201 CELLIBTYPE=\STDCELL
AS norf201 NEW_SYMBOL
AP norf201 1 PINTYPE=IN
AP norf201 2 PINTYPE=IN
AP norf201 3 PINTYPE=OUT
P fti I2
I I2 fti I2
P fti I1
I I1 fti I1
P fto O1
I O1 fto O1
M smv $1I17
I $1I17 smv
SIVMASTER=EXAMP1`LAYOUT_GENERATOR=\STDCELL`
P fto O2
I O2 fto O2
M nanf201 $1I1
I $1I1 nanf201 I1 I2 $1N21
M norf201 $1I18
I $1I18 norf201 $1N21 I3 O2
P fti I3
I I3 fti I3
M nanf201 $1I2
I $1I2 nanf201 I4 $1N21 O1
P fti I4
I I4 fti I4
E W
```

**Continued**

```
[time =     0.0ns] O1       X -> 1
[time =     0.0ns] $1N21    X -> 1
[time =     0.0ns] O2       X -> 1
[time =     0.0ns] I1       X -> 1
[time =     0.0ns] I4       X -> 0
[time =     0.0ns] I3       X -> 0
[time =     0.0ns] I2       X -> 0
[time =     0.7ns] O2       1 -> 0
[time =    20.0ns] I1       1 -> 0
[time =    40.0ns] I1       0 -> 1
[time =    60.0ns] I1       1 -> 0
[time =    80.0ns] I1       0 -> 1
[time =    80.0ns] I2       0 -> 1
[time =    83.6ns] $1N21    1 -> 0
[time =    85.1ns] O2       0 -> 1
[time =   100.0ns] I1       1 -> 0
[time =   103.0ns] $1N21    0 -> 1
[time =   103.7ns] O2       1 -> 0
[time =   120.0ns] I1       0 -> 1
[time =   123.6ns] $1N21    1 -> 0
[time =   125.1ns] O2       0 -> 1
[time =   140.0ns] I1       1 -> 0
[time =   143.0ns] $1N21    0 -> 1
[time =   143.7ns] O2       1 -> 0
[time =   160.0ns] I1       0 -> 1
[time =   160.0ns] I3       0 -> 1
[time =   160.0ns] I2       1 -> 0
[time =   180.0ns] I1       1 -> 0
[time =   200.0ns] I1       0 -> 1
[time =   220.0ns] I1       1 -> 0
[time =   240.0ns] I1       0 -> 1
[time =   240.0ns] I2       0 -> 1
[time =   243.6ns] $1N21    1 -> 0
[time =   260.0ns] I1       1 -> 0
[time =   263.0ns] $1N21    0 -> 1
[time =   280.0ns] I1       0 -> 1
[time =   283.6ns] $1N21    1 -> 0
[time =   300.0ns] I1       1 -> 0
[time =   303.0ns] $1N21    0 -> 1
[time =   320.0ns] I1       0 -> 1
```

**Figure 4.2 examp1.trc - Transitional Delays Captured Via Viewsim**

```
[time =    320.0ns]  I4       0 -> 1
[time =    320.0ns]  I3       1 -> 0
[time =    320.0ns]  I2       1 -> 0
[time =    323.6ns]  O1        1 -> 0
[time =    340.0ns]  I1       1 -> 0
[time =    360.0ns]  I1       0 -> 1
[time =    380.0ns]  I1       1 -> 0
[time =    400.0ns]  I1       0 -> 1
[time =    400.0ns]  I2       0 -> 1
[time =    403.6ns]  $1N21      1 -> 0
[time =    405.1ns]  O2        0 -> 1
[time =    406.6ns]  O1        0 -> 1
[time =    420.0ns]  I1       1 -> 0
[time =    423.0ns]  $1N21      0 -> 1
[time =    423.7ns]  O2        1 -> 0
[time =    426.6ns]  O1        1 -> 0
[time =    440.0ns]  I1       0 -> 1
[time =    443.6ns]  $1N21      1 -> 0
[time =    445.1ns]  O2        0 -> 1
[time =    446.6ns]  O1        0 -> 1
[time =    460.0ns]  I1       1 -> 0
[time =    463.0ns]  $1N21      0 -> 1
[time =    463.7ns]  O2        1 -> 0
[time =    466.6ns]  O1        1 -> 0
[time =    480.0ns]  I1       0 -> 1
[time =    480.0ns]  I3       0 -> 1
[time =    480.0ns]  I2       1 -> 0
[time =    500.0ns]  I1       1 -> 0
[time =    520.0ns]  I1       0 -> 1
[time =    540.0ns]  I1       1 -> 0
[time =    560.0ns]  I1       0 -> 1
[time =    560.0ns]  I2       0 -> 1
[time =    563.6ns]  $1N21      1 -> 0
[time =    566.6ns]  O1        0 -> 1
[time =    580.0ns]  I1       1 -> 0
[time =    583.0ns]  $1N21      0 -> 1
[time =    586.6ns]  O1        1 -> 0
[time =    600.0ns]  I1       0 -> 1
[time =    603.6ns]  $1N21      1 -> 0
[time =    606.6ns]  O1        0 -> 1
[time =    620.0ns]  I1       1 -> 0
[time =    623.0ns]  $1N21      0 -> 1
[time =    626.6ns]  O1        1 -> 0
```

**Continued**

```
I1 I1
I2 I2
I3 I3
O1 3.6ns
O1 6.6ns
O2 0.7ns
O2 3.7ns
V1I1/O 3.6ns
```

**Figure 4.3 examp1.trchltab2 - High-to-Low Delay Values**

```
I1 I1
I2 I2
I3 I3
I4 I4
O1 6.6ns
O2 5.1ns
V1I1/O 3ns
```

**Figure 4.4 examp1.trclhtab2 - Low-to-High Delay Values**

```
.BA
C $1I1
A TPHL=3.6 NS
A TPLH=3 NS
C $1I18
A TPHL=0.1 NS
A TPLH=2.1 NS
C $1I2
A TPHL=3 NS
A TPLH=3.6 NS
.AB
```

**Figure 4.5 examp1pass1.bat - Back-Annotation File**

```
[time =     0.0ns] O1       X -> 1
[time =     0.0ns] $1N21    X -> 1
[time =     0.0ns] O2       X -> 1
[time =     0.0ns] I1       X -> 1
[time =     0.0ns] I4       X -> 0
[time =     0.0ns] I3       X -> 0
[time =     0.0ns] I2       X -> 0
[time =     0.4ns] O2       1 -> 0
[time =    20.0ns] I1       1 -> 0
[time =    40.0ns] I1       0 -> 1
[time =    60.0ns] I1       1 -> 0
[time =    80.0ns] I1       0 -> 1
[time =    80.0ns] I2       0 -> 1
[time =    80.8ns] $1N21    1 -> 0
[time =    81.4ns] O2       0 -> 1
[time =   100.0ns] I1       1 -> 0
[time =   100.7ns] $1N21    0 -> 1
[time =   101.1ns] O2       1 -> 0
[time =   120.0ns] I1       0 -> 1
[time =   120.8ns] $1N21    1 -> 0
[time =   121.4ns] O2       0 -> 1
[time =   140.0ns] I1       1 -> 0
[time =   140.7ns] $1N21    0 -> 1
[time =   141.1ns] O2       1 -> 0
[time =   160.0ns] I1       0 -> 1
[time =   160.0ns] I3       0 -> 1
[time =   160.0ns] I2       1 -> 0
[time =   180.0ns] I1       1 -> 0
[time =   200.0ns] I1       0 -> 1
[time =   220.0ns] I1       1 -> 0
[time =   240.0ns] I1       0 -> 1
[time =   240.0ns] I2       0 -> 1
[time =   240.8ns] $1N21    1 -> 0
[time =   260.0ns] I1       1 -> 0
[time =   260.7ns] $1N21    0 -> 1
[time =   280.0ns] I1       0 -> 1
[time =   280.8ns] $1N21    1 -> 0
[time =   300.0ns] I1       1 -> 0
[time =   300.7ns] $1N21    0 -> 1
[time =   320.0ns] I1       0 -> 1
```

**Figure 4.6 examp1.trc - Generated Using New Delay Attributes**

```
[time =   320.0ns]  I4       0 -> 1
[time =   320.0ns]  I3       1 -> 0
[time =   320.0ns]  I2       1 -> 0
[time =   320.3ns]  O1        1 -> 0
[time =   340.0ns]  I1       1 -> 0
[time =   360.0ns]  I1       0 -> 1
[time =   380.0ns]  I1       1 -> 0
[time =   400.0ns]  I1       0 -> 1
[time =   400.0ns]  I2       0 -> 1
[time =   400.8ns]  $1N21      1 -> 0
[time =   401.4ns]  O2        0 -> 1
[time =   401.4ns]  O1        0 -> 1
[time =   420.0ns]  I1       1 -> 0
[time =   420.7ns]  $1N21      0 -> 1
[time =   421.0ns]  O1        1 -> 0
[time =   421.1ns]  O2        1 -> 0
[time =   440.0ns]  I1       0 -> 1
[time =   440.8ns]  $1N21      1 -> 0
[time =   441.4ns]  O2        0 -> 1
[time =   441.4ns]  O1        0 -> 1
[time =   460.0ns]  I1       1 -> 0
[time =   460.7ns]  $1N21      0 -> 1
[time =   461.0ns]  O1        1 -> 0
[time =   461.1ns]  O2        1 -> 0
[time =   480.0ns]  I1       0 -> 1
[time =   480.0ns]  I3       0 -> 1
[time =   480.0ns]  I2       1 -> 0
[time =   500.0ns]  I1       1 -> 0
[time =   520.0ns]  I1       0 -> 1
[time =   540.0ns]  I1       1 -> 0
[time =   560.0ns]  I1       0 -> 1
[time =   560.0ns]  I2       0 -> 1
[time =   560.8ns]  $1N21      1 -> 0
[time =   561.4ns]  O1        0 -> 1
[time =   580.0ns]  I1       1 -> 0
[time =   580.7ns]  $1N21      0 -> 1
[time =   581.0ns]  O1        1 -> 0
[time =   600.0ns]  I1       0 -> 1
[time =   600.8ns]  $1N21      1 -> 0
[time =   601.4ns]  O1        0 -> 1
[time =   620.0ns]  I1       1 -> 0
[time =   620.7ns]  $1N21      0 -> 1
[time =   621.0ns]  O1        1 -> 0
```

**Continued**

```
I1 I1
I2 I2
I3 I3
O1 0.3ns
O1 1ns
O2 0.4ns
O2 1.1ns
V1I1/O 0.8ns
```

**Figure 4.7 examp1.trchltab2 - High-to-Low Delay Values**

```
I1 I1
I2 I2
I3 I3
I4 I4
O1 1.4ns
O2 1.4ns
V1I1/O 0.7ns
```

**Figure 4.8 examp1.trclhtab2 - Low-to-High Delay Values**

```
.BA
C $1I1
A TPHL=0.8 NS
A TPLH=0.7 NS
C $1I18
A TPHL=0.3 NS
A TPLH=0.7 NS
C $1I2
A TPHL=0.2 NS
A TPLH=0.7 NS
.AB
```

**Figure 4.9 examp1pass2.bat - Back-Annotation File**

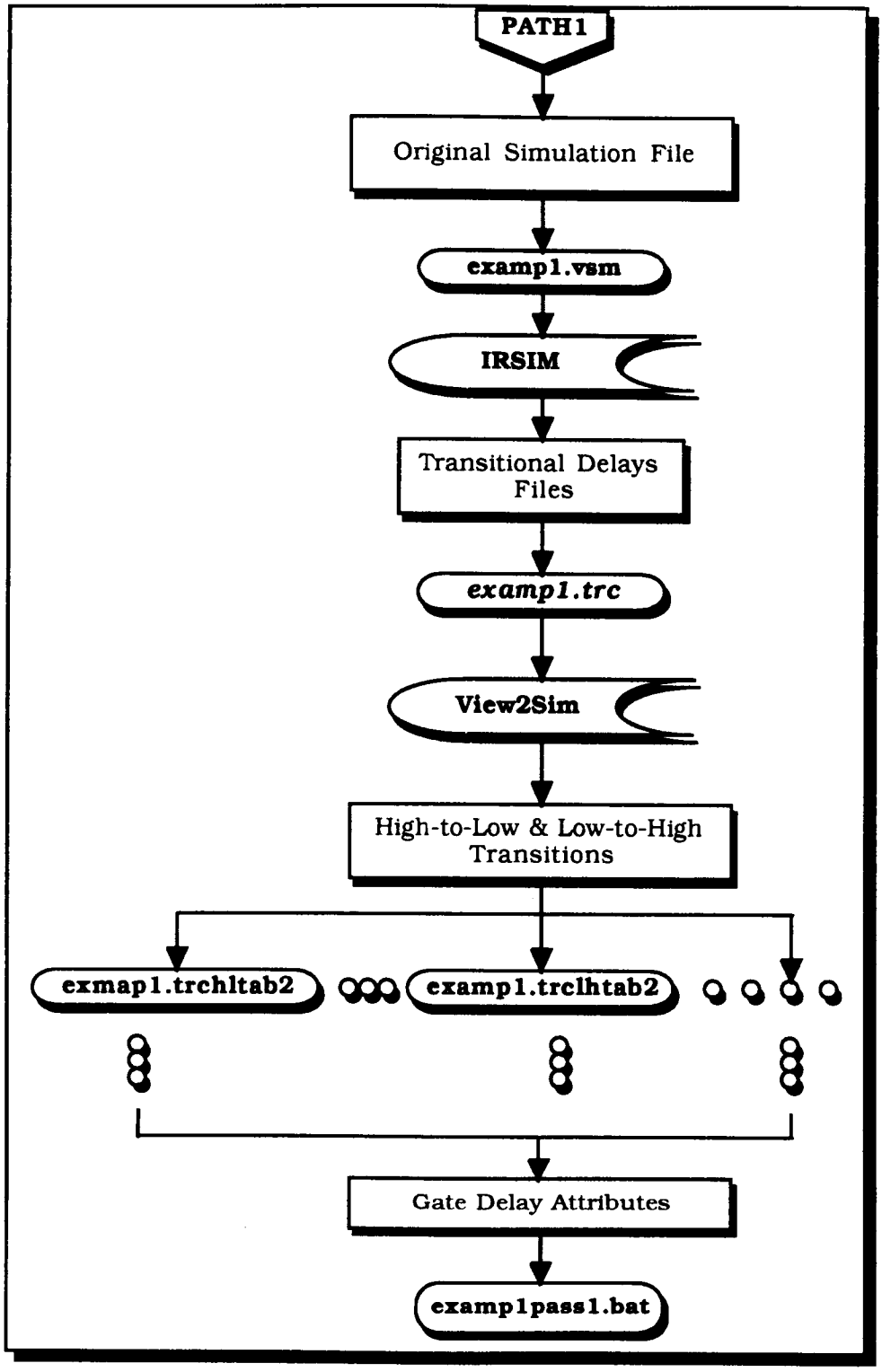



Figure 4.10 First Path

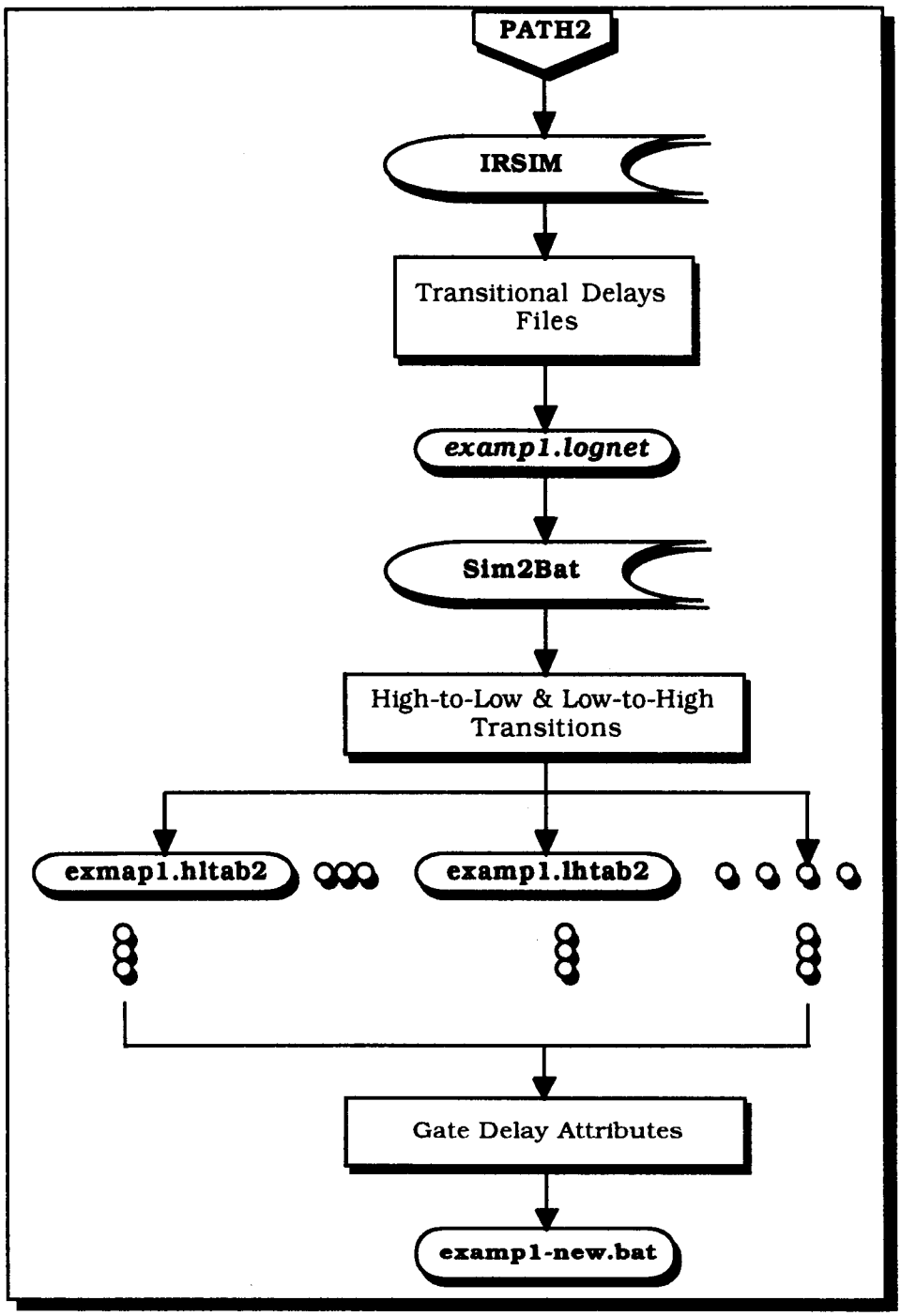



**Figure 4.11 Second Path**

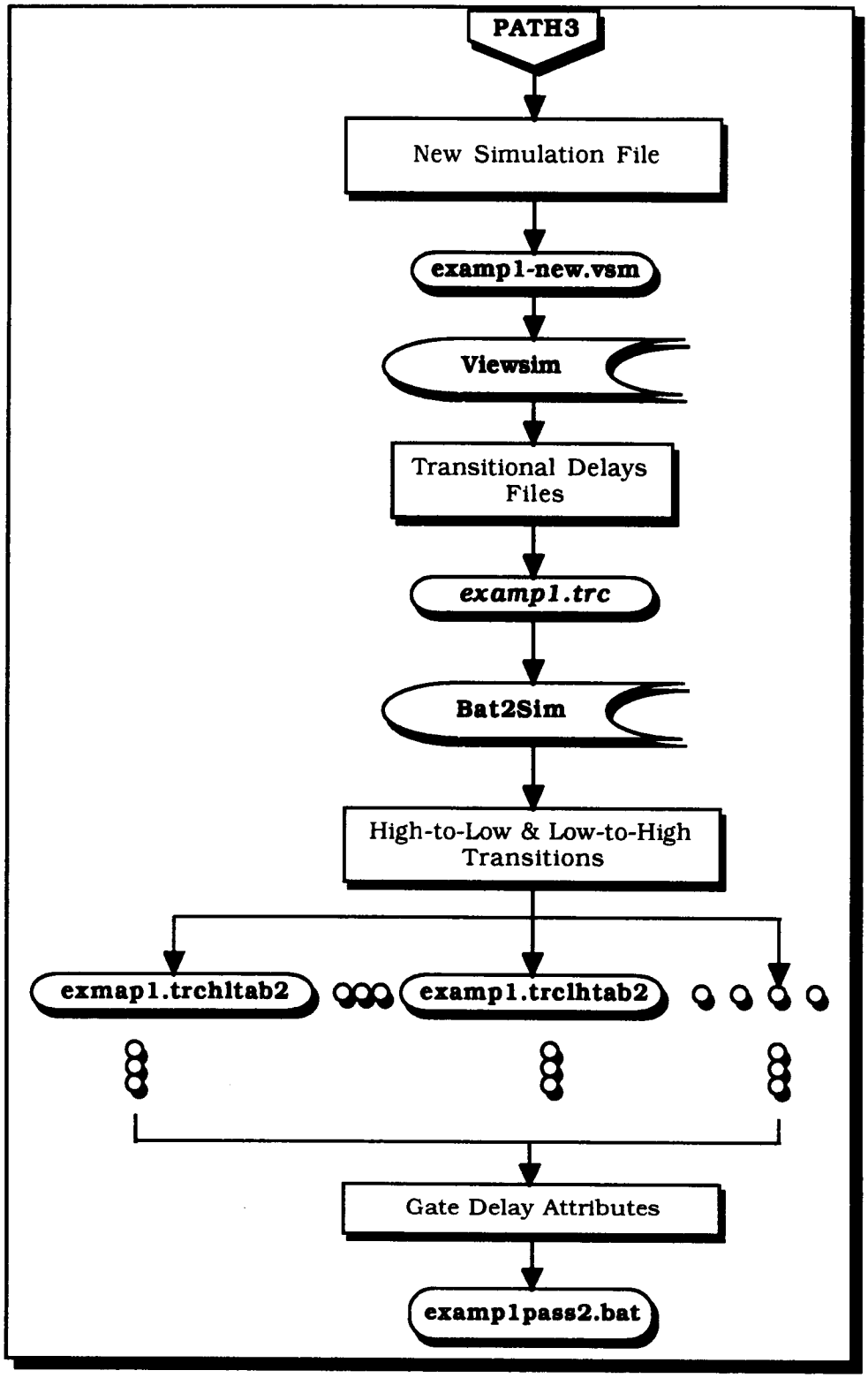



Figure 4.12 Third Path

delays are compared by comparing the *hltab2* delay files of each of the three paths. These delays show the gate input and output timings starting with the original pre-layout ***View2Sim*** path, continuing with the desired layout timings of ***Sim2Bat***, and finally ending with the new and back-annotated transitional delays of ***Bat2Sim*** path. Second, the calculated gate delays are compared by comparing the *"bat"* back-annotation files of each of the paths. These files show the gate delays represented as High-to-Low and Low-to-High transitions.

### *Experimental Results*

In order to test the ***CKTIM*** process, four test cases were created. These test cases serve as prospect applications and eventually prove the effectiveness of ***Sim2Bat*** program. As indicated previously, the results are presented in both graphical and transitional net and gate delay methods.

### *Test Case #1: EXAMP1*

*examp1* is a sample design created by the author to simulate and examine the design delay timing verification method of ***Sim2Bat***. Figure 4.13 shows the circuit design. As shown in this picture, this design includes two NAND gates (nanf201) and one NOR gate (norf201), with four inputs and two outputs. Although this design contains only three gates, it serves as a macro circuit that can be installed in the standard cell user library if the designer wishes under the name *"andor1212"*, for example. Macros are very beneficial especially if used repetitively throughout a design and help in reducing the circuit layout area and transistor count.

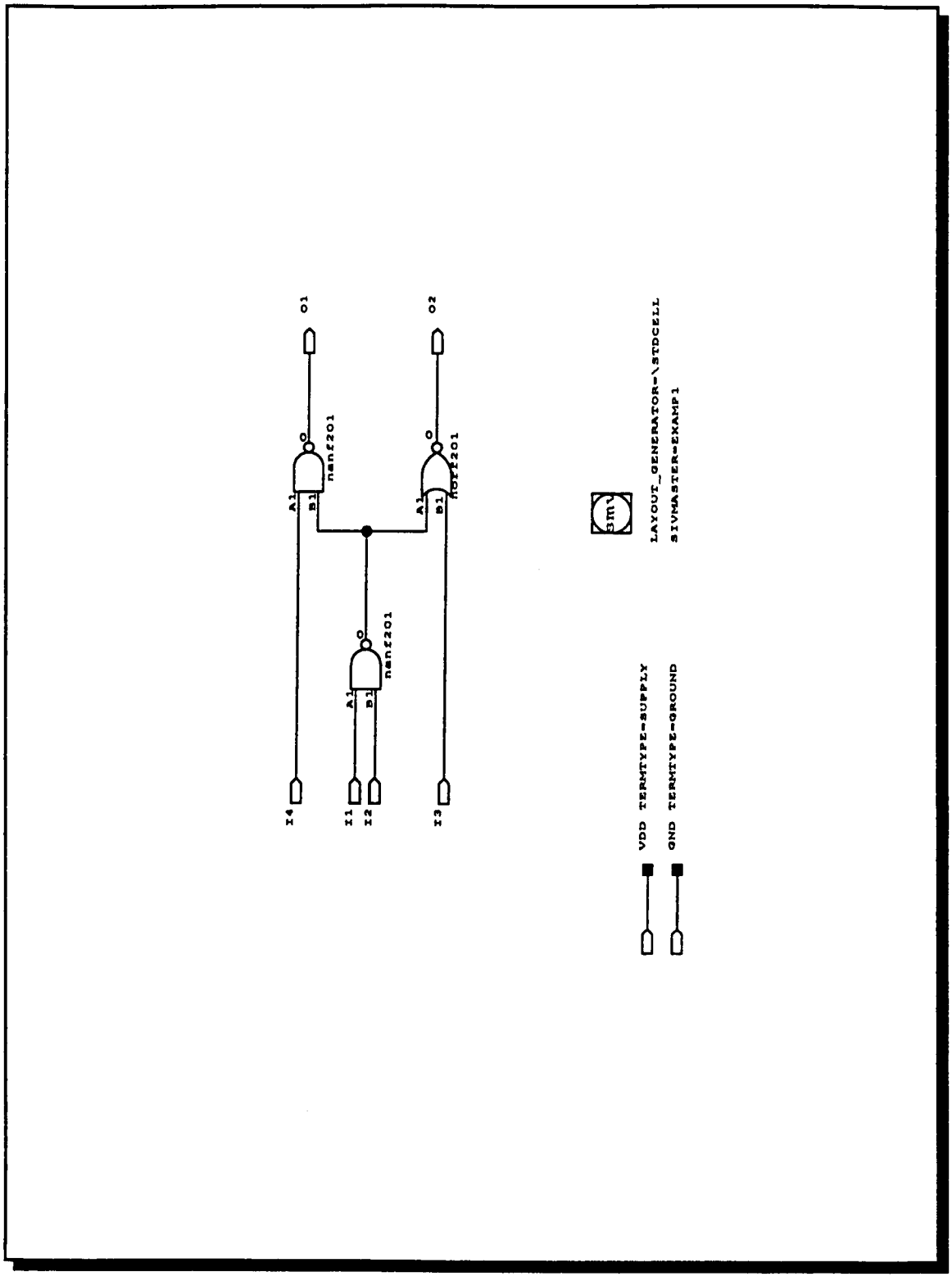



Figure 4.13 examp1.001p - Circuit Schematic

*Graphical Results*

The following Figures depict the results of pre-layout, layout, and post layout simulation waveforms of the circuit *examp1*. It is helpful to notice the matching of the second and third path waveforms (see Figures 4.14 and 4.15). The first Figure is produced by ***IRSIM*** and the second Figure is generated using ***Viewsim***. Obviously, the first path results (see Figure 4.16) are quite different from the other two path results and reflect larger delays than the layout simulations actually produce.

*Transitional Delay Results*

During the transitional delay Calculation, all input, output and intermediate nodes of the circuit are scanned and every High-to-Low or Low-to-High change of any node is captured. Using the *"cycle"* global variable defined by the designer, the transitional delays are calculated. In the first test case *"examp1"*, the High-to-Low delay files are presented in Figures 4.17 for the first path, Figure 4.18 for the second path, and Figure 4.19 for the second path. The results are summarized in Table 4.1 that contains the node names and their transitional delays, the pre-layout and post layout errors, and a total RMS error. The Low-to-High delay files are shown in Figures 4.20, 4.21, and 4.22 for the first, second, and third paths, respectively. The results are summarized in Table 4.2.

Observing the summary of percentage error in this test case, it is obvious that an improvement of about 70% is accomplished using the ***Sim2Bat*** program. On the other hand, the total run-time for ***Pre-Sim2Bat*** is less than five minutes, and the total run-time for ***Sim2Bat*** is less than one and a half minutes. This gives a total execution and user interaction time of maximum of seven minutes. This time can be cut in half if ***CKTIM*** is run on a Sparc10 Sun Workstation.

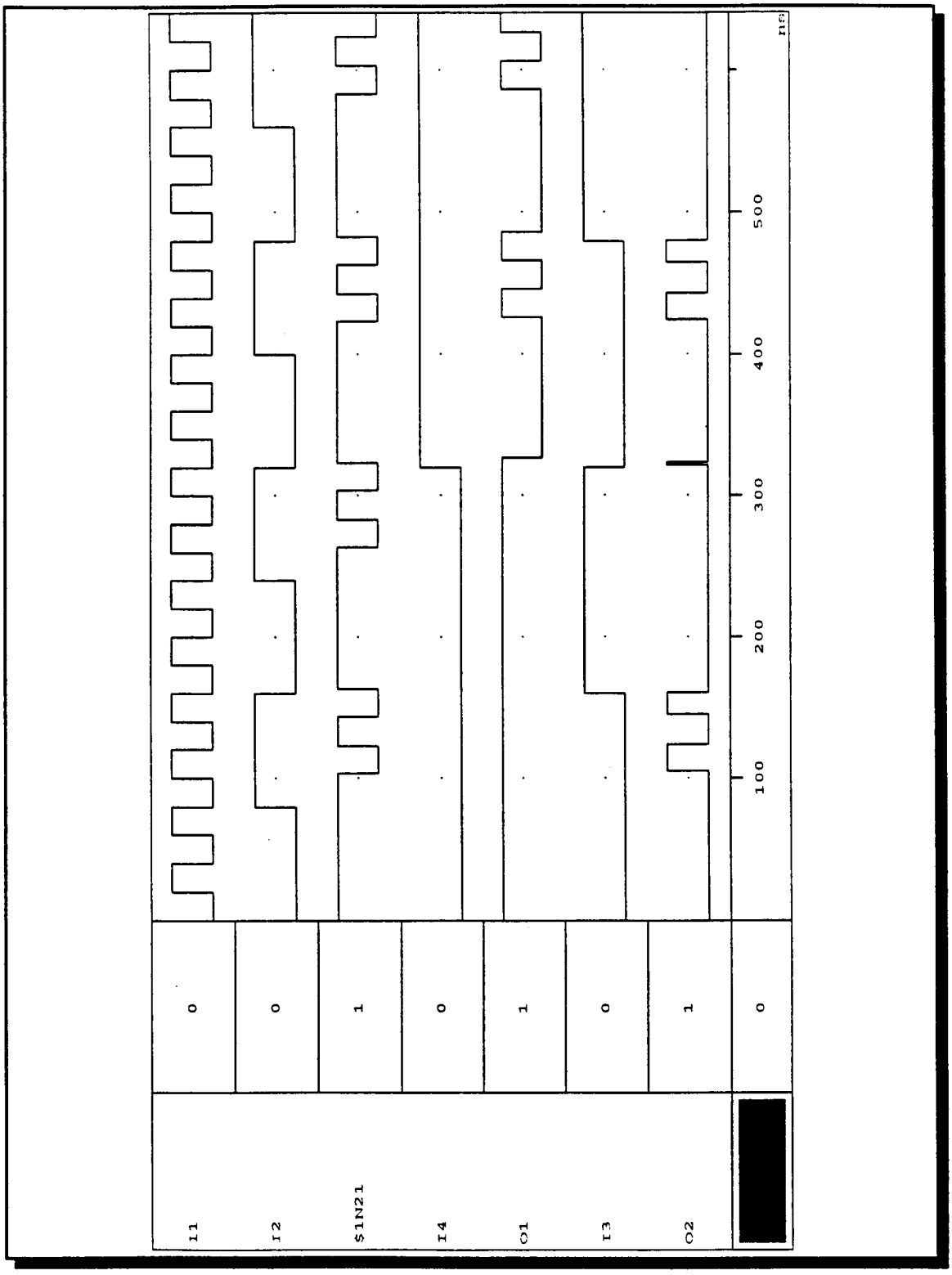



Figure 4.14 examp1.plt - Original Pre-Layout Simulation

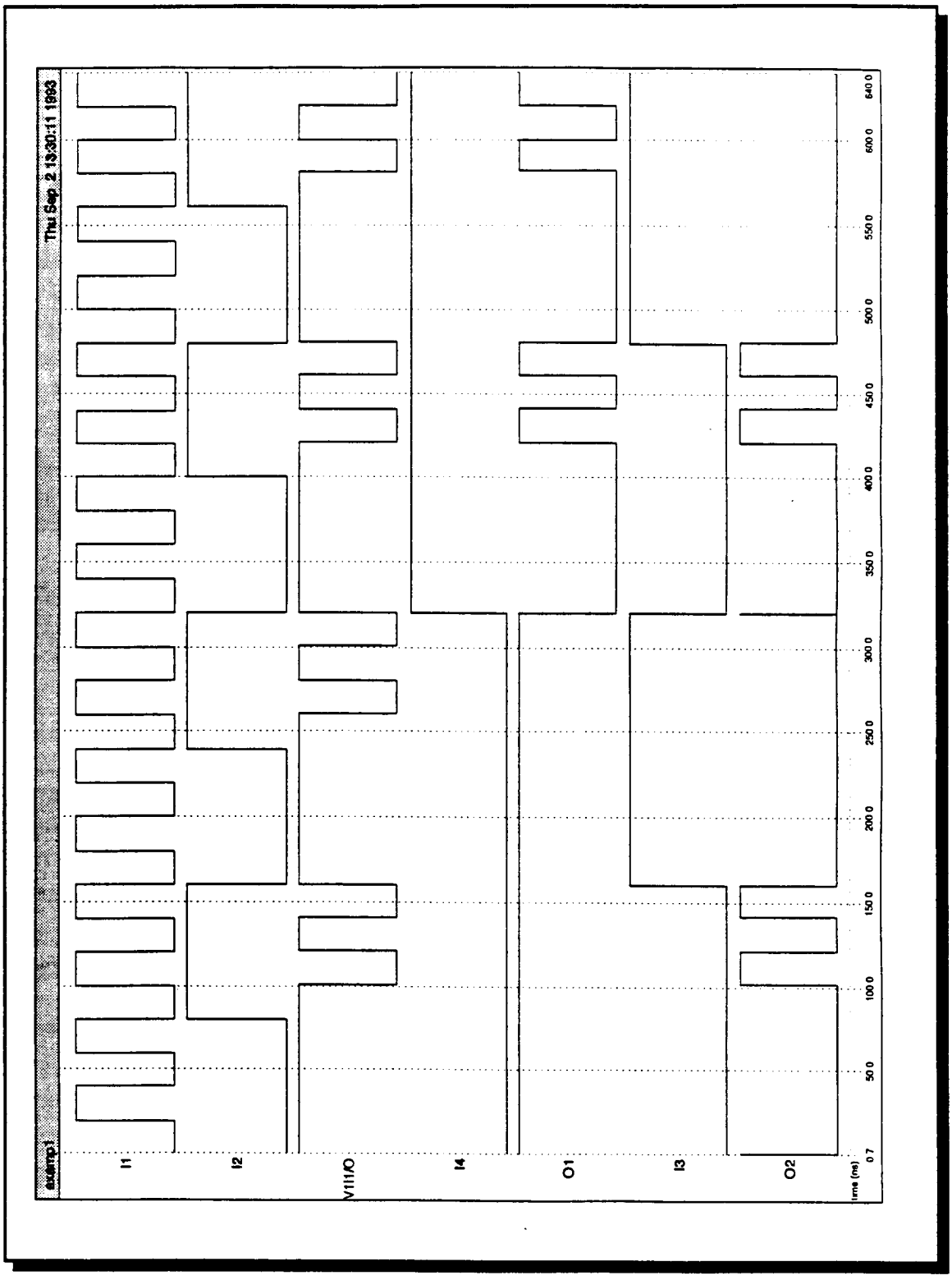



Figure 4.15 examp1.ps - Layout Simulation

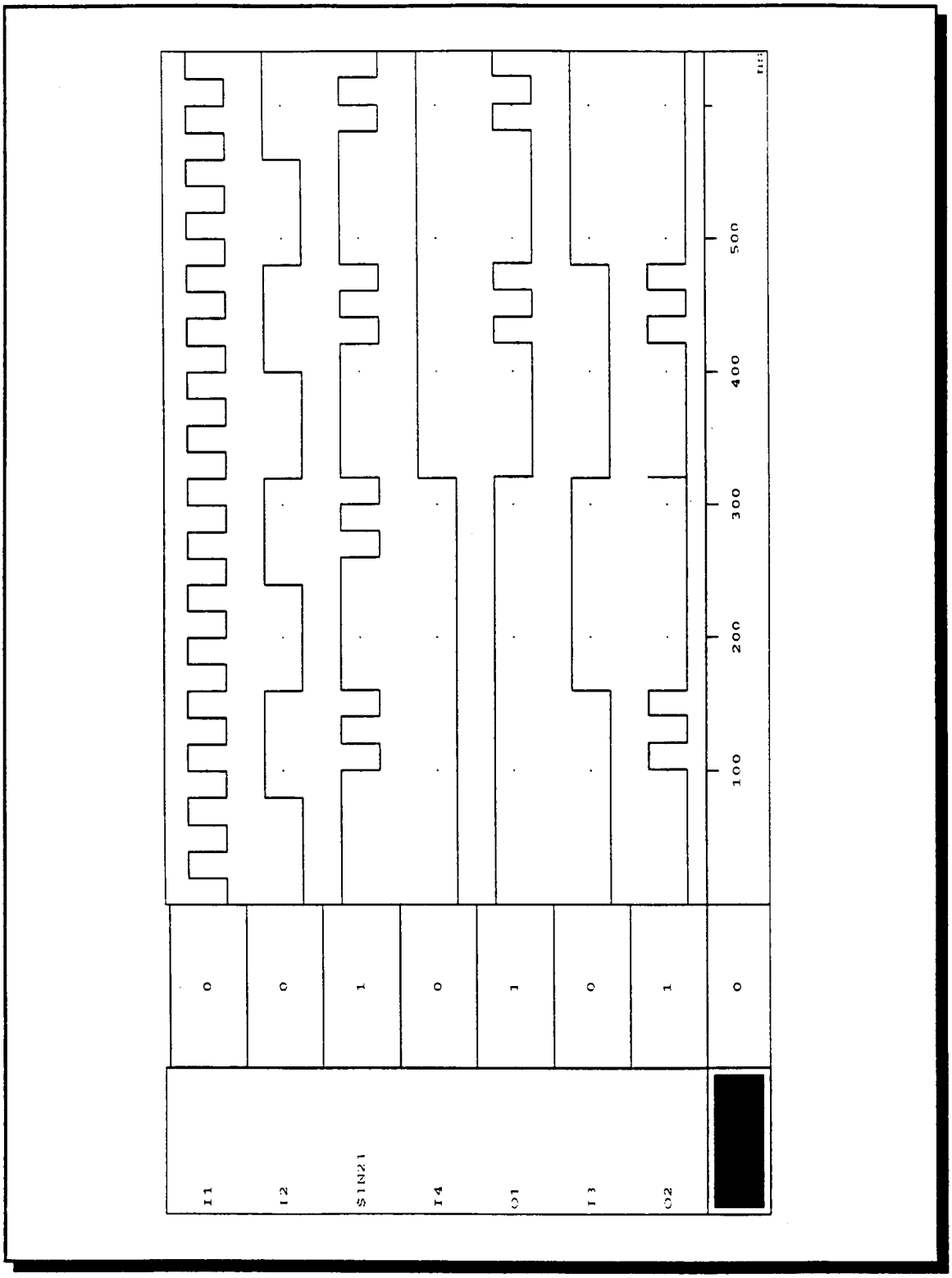



Figure 4.16 examp1-new.plt - Post-Layout Simulation

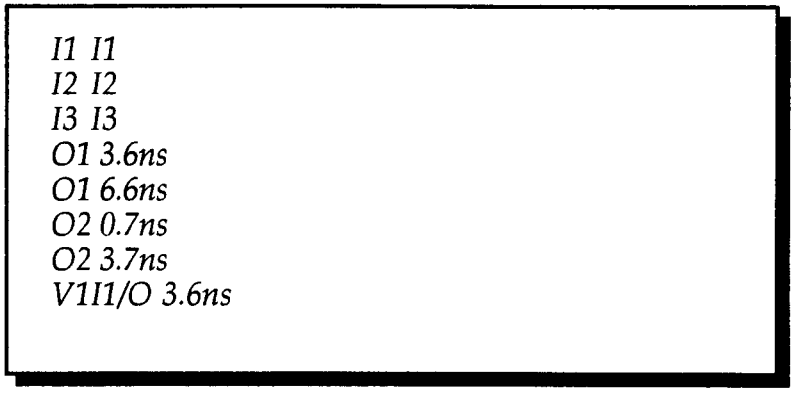



**Figure 4.17 examp1.trchltab2 - Pre-Layout Transitional Delays (1st path)**

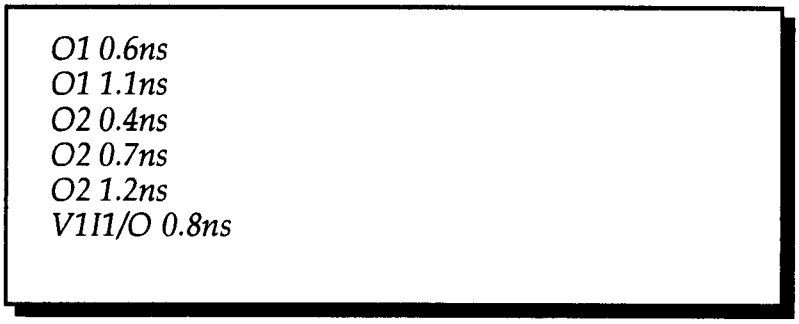



**Figure 4.18 examp1.hltab2 - Layout Transitional Delays (2nd path)**

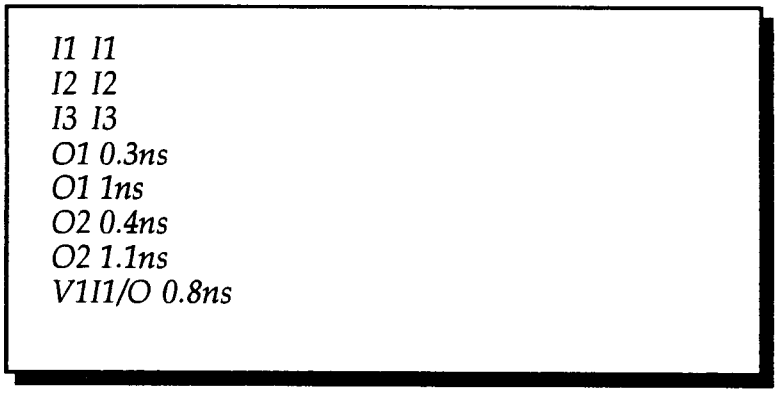



**Figure 4.19 examp1.trchltab2 - Post-Layout Transitional Delays (3rd path)**

**Table 4.1 examp1 - High-to-Low Transitional Delays Summary**

| 1 --> 0 Nodes | Transitional Delays: 1st Path View2Sim | Transitional Delays: 2nd Path Sim2Bat | Transitional Delays: 3rd Path Bat2Sim | % ERROR: PreLayout | % ERROR: PostLayout |
|---|---|---|---|---|---|
| O1 | 6.6 | 1.1 | 1.0 | 83 | 9 |
| O2 | 3.6 | 1.2 | 1.1 | 67 | 8.3 |
| V1I1/O | 3.7 | 0.8 | 0.8 | 77 | 0 |
| RMS % ERROR | | | | 76 | 5.9 |

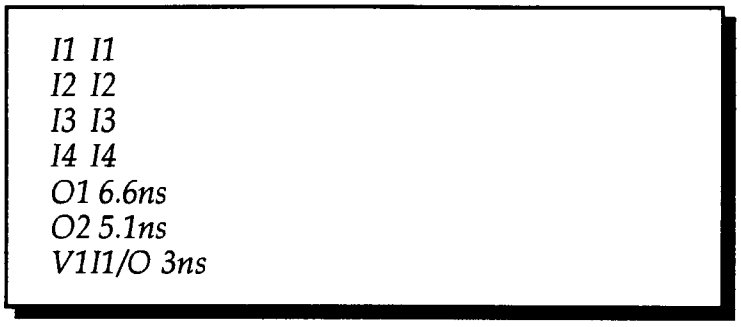



**Figure 4.20 examp1.trclhtab2 - Pre-Layout Transitional Delays (1st Path)**

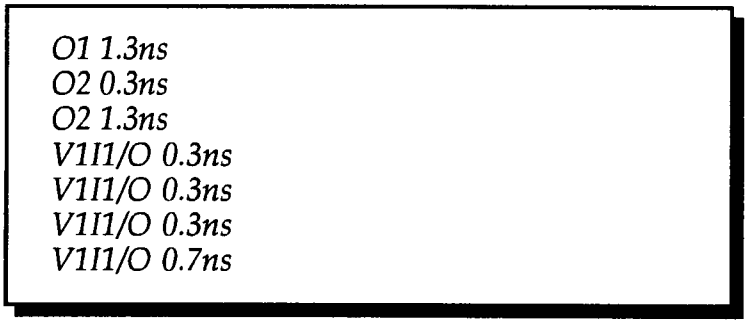



**Figure 4.21 examp1.lhtab2 - Layout Transitional Delays (2nd Path)**

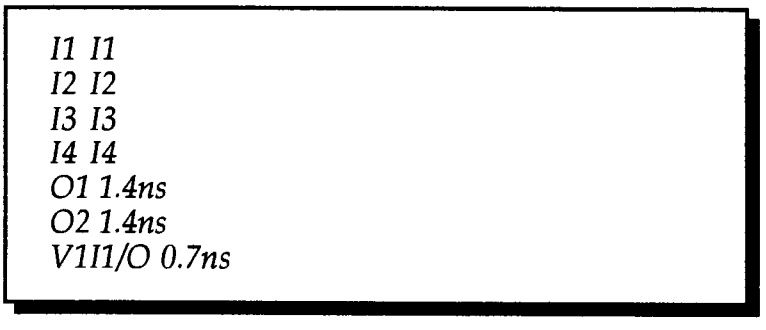



**Figure 4.22 examp1.trclhtab2 - Post-Layout Transitionl Delays (3rd Path)**

**Table 4.2 examp1 - Low-to-High Transition delays Summary**

| 0 --> 1 / Nodes | Transitional Delays: 1st Path View2Sim | 2nd Path Sim2Bat | 3rd Path Bat2Sim | % ERROR: PreLayout | PostLayout |
|---|---|---|---|---|---|
| O1 | 6.6 | 1.3 | 1.4 | 80 | 7 |
| O2 | 5.1 | 1.3 | 1.4 | 75 | 7 |
| V1I1/O | 3.0 | 0.7 | 0.7 | 76 | 0 |
| RMS % ERROR | | | | 77 | 4.6 |

*Gate and Net Delays Results*

As mentioned previously, the gate and net delays presented in the "bat" files are the calculated sum of net delays added to gate delays. The concept is to average the sum of the maximum transitional input delays and subtract the total from the maximum transitional output delays. The results are assigned as TPHL and TPLH attributes to each respective gate of design. These delay attributes are shown in Figures 4.23, 4.24, and 4.25 for the first, second, and third paths. These files are compared and the results are presented in Table 4.3 that contains gate names, TPHL and TPLH delays for each path, a percentage error, and a RMS error summary. It can be concluded from these results an improvement of 60% is obtained using the ***CKTIM*** methodology.

*Test Case #2: DFTESTWIN*

*dftestwin* is a sample design engineered by the author in order to simulate and examine the design timing verification method of ***Sim2Bat***. Figure 4.26 shows the circuit design. This circuit differs form Test Case #1 in that it incorporates many more gates, uses cross-coupled NAND gates, and constitutes a *d-flip-flop* which serves as a major and most useful macro-cell of the standard cell library. It is critical for the designer to have accurate timing results in addition to correct switching observability (0 to 1 and 1 to 0 states).

*Graphical Results*

The following Figures show the timing waveforms obtained from the simulation of pre-layout, layout, and post-layout of *dftestwin* (see Figures 4.27 through 4.29). The ***IRSIM*** generated waveforms are matched by the new ***Viewsim*** simulation results of post-layout. The pre-layout ***viewsim*** results are clearly very far from matching the desired output of ***IRSIM***.

```
.BA
C $1I1
A TPHL=3.6 NS
A TPLH=3 NS
C $1I18
A TPHL=0.1 NS
A TPLH=2.1 NS
C $1I2
A TPHL=3 NS
A TPLH=3.6 NS
.AB
```

**Figure 4.23 examp1pass1.bat - Pre-Layout Back-Annotation Gate and Net Delays**

```
.BA
C $1I1
A TPHL=0.8NS
A TPLH=0.7NS
C $1I18
A TPHL=0.4NS
A TPLH=0.6NS
C $1I2
A TPHL=0.3NS
A TPLH=0.6NS
.AB
```

**Figure 4.24 examp1-new.bat-Layout Back-Annotation Gate and Net Delays**

```
.BA
C $1I1
A TPHL=0.8 NS
A TPLH=0.7 NS
C $1I18
A TPHL=0.3 NS
A TPLH=0.7 NS
C $1I2
A TPHL=0.2 NS
A TPLH=0.7 NS
.AB
```

**Figure 4.25 examp1pass2.bat - Post-Layout Back-Annotation Gate and Net Delays**

Table 4.3 examp1 - TPHL and TPLH Transitional Gate and Net Delays Summary

| Gate | Transitional Delays (ns) | | | | | | % ERROR | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1st Path View2Sim | | 2nd Path Sim2Bat | | 3rd Path Bat2Sim | | PreLayout | | PostLayout | |
| | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH |
| $1I1 | 3.6 | 3.0 | 0.8 | 0.7 | 0.8 | 0.7 | 77 | 76 | 0 | 12.5 |
| $1I18 | 0.1 | 2.1 | 0.4 | 0.6 | 0.3 | 0.7 | 75 | 71 | 25 | 14 |
| $1I2 | 3.0 | 3.6 | 0.3 | 0.6 | 0.2 | 0.7 | 90 | 83 | 33 | 14 |
| RMS % ERROR | | | | | | | 80 | 76 | 19 | 13.5 |

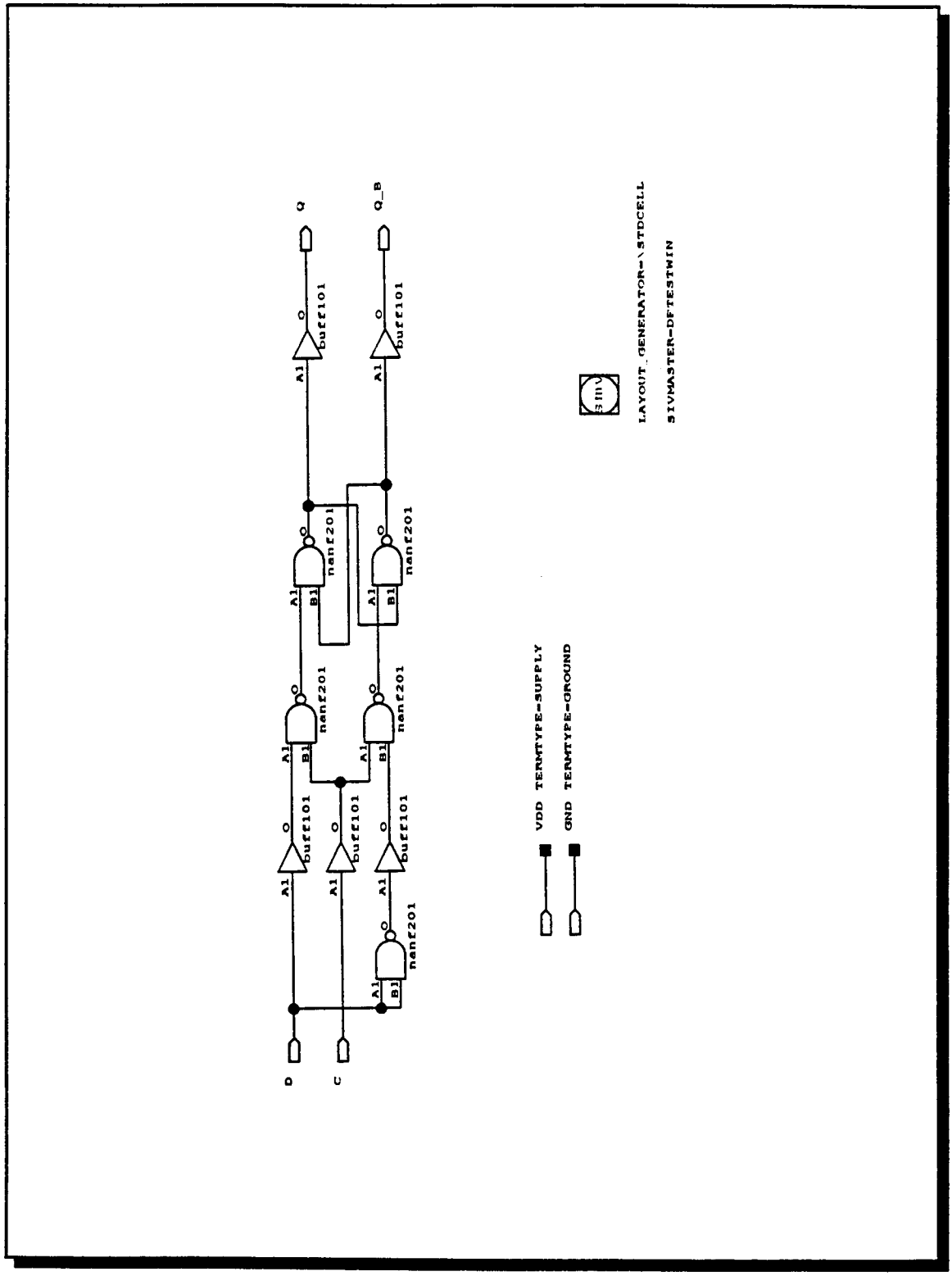



Figure 4.26 dftestwin.001p - Circuit Schematic

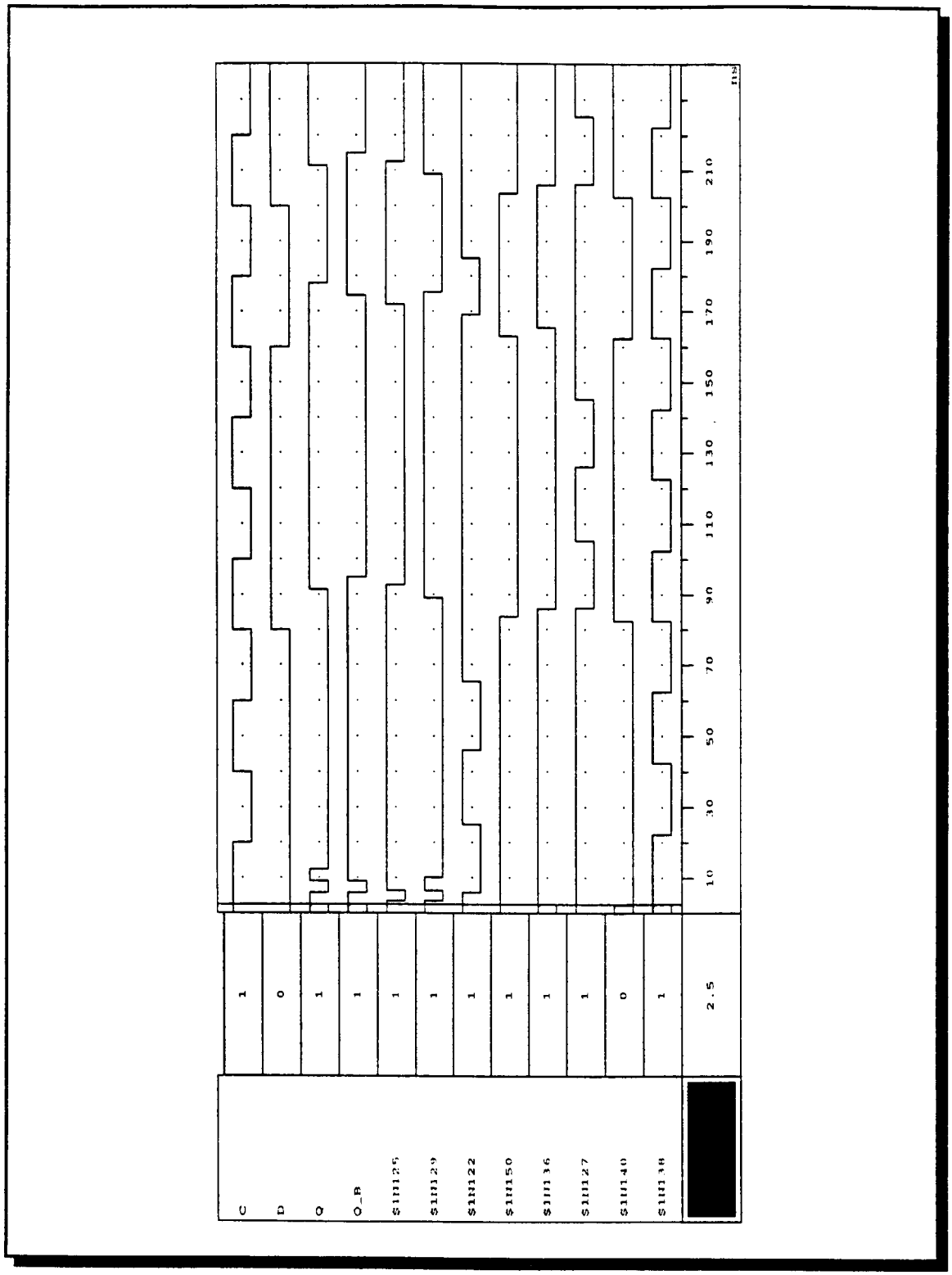



Figure 4.27 dftestwin.plt - Original Pre-Layout Simulation (1st Path)

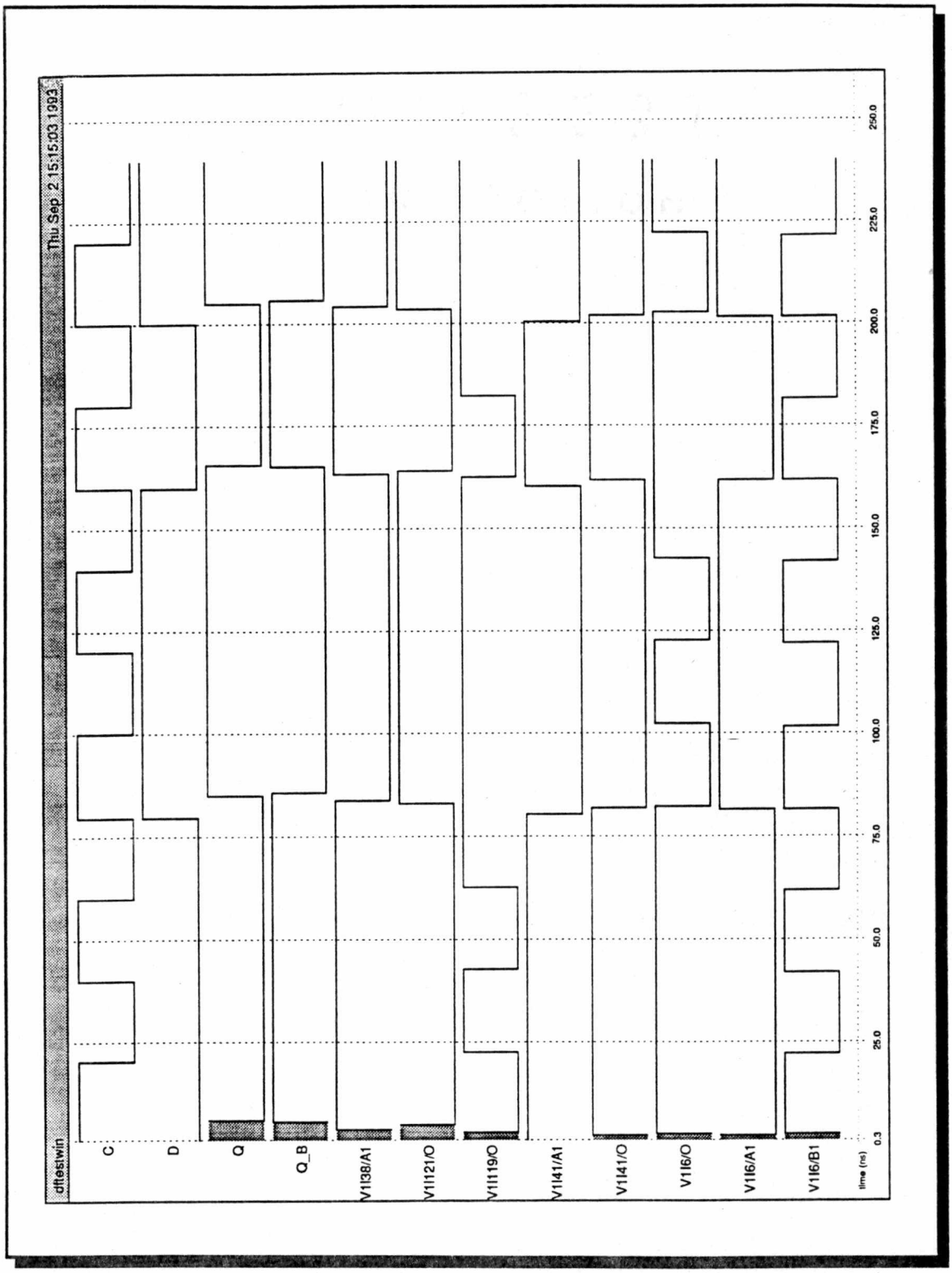



Figure 4.28 dftestwin.ps - Layout Simulation (2nd Path)

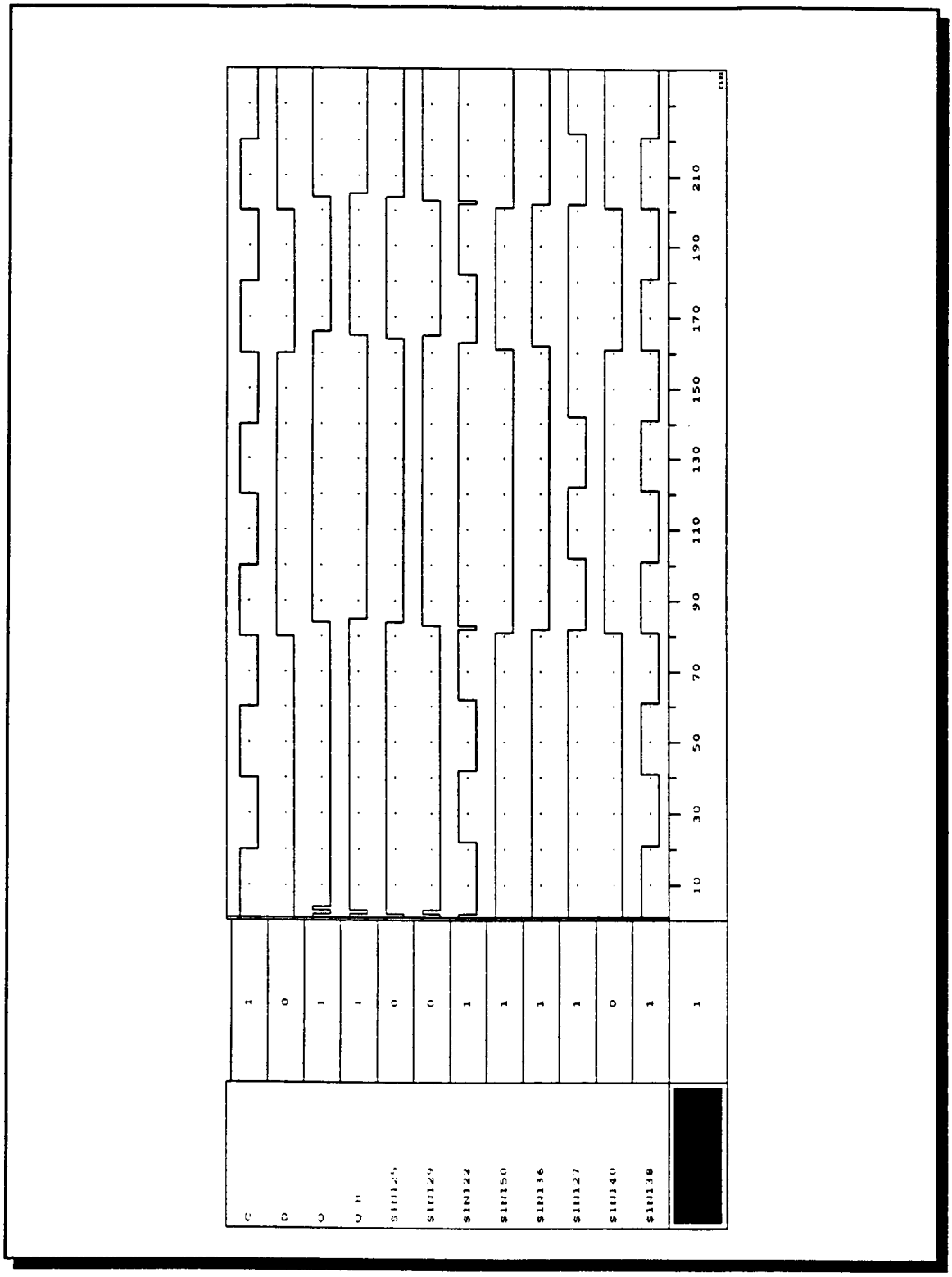



Figure 4.29 dftestwin-new.plt - Post-Layout Simulation (3rd Path)

*Transitional Delay Results*

In this second test case *dftestwin*, the High-to-Low delay files of the first, second and third paths are shown in Figures 4.30, 4.31, and 4.32. The results are summarized in Table 4.4. The Low-to-High delay files are presented in Figures 4.33, 4.34, and 4.35. Table 4.5 contains a summary of these delays and provides an RMS error calculation.

*Gate and Net delays Results*

The transitional delay attribute files for Test Case #2 are shown in Figures 4.36 (first path), 4.37 (second path), and 4.38 (third path). The results are given in Table 4.6 and include a RMS error calculation summary. The total improvement resulting from using ***CKTIM*** in this test case is 56%.

*Test Case #3: DEC3TO8*

*dec3to8* is a sample design created by the author in order to simulate and examine the design timing verification method of ***Sim2Bat***. The design is portrayed in Figure 4.39. This circuit acts as a three-to-eight decoder as the name "dec3to8" implies. It is formed of twenty-one minimum-input (1 to 2) single-output gates; this is very typical in a design of a standard macro-cell. Hence, Test Case #3 differs from the previous test cases in the number of inputs and outputs (4 to 8) and the number of utilized gates.

*Graphical Results*

The pre-layout, layout and post-layout simulation results are shown in Figures 4.40, 4.41, and 4.42. One can notice the great similarity between the output waveforms of ***IRSIM*** and the post-layout waveforms of ***Viewsim***. On the other hand there are many differences between ***IRSIM***'s waveforms and the pre-layout ***Viewsim*** output waveforms.

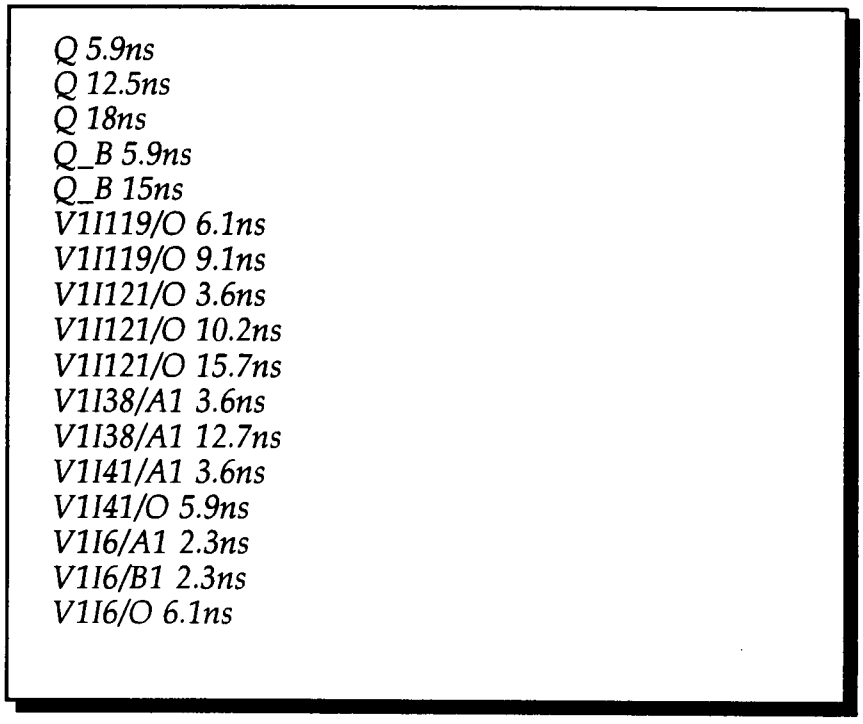



**Figure 4.30 dftestwin.trchltab - Pre-Layout Transitional Delays**

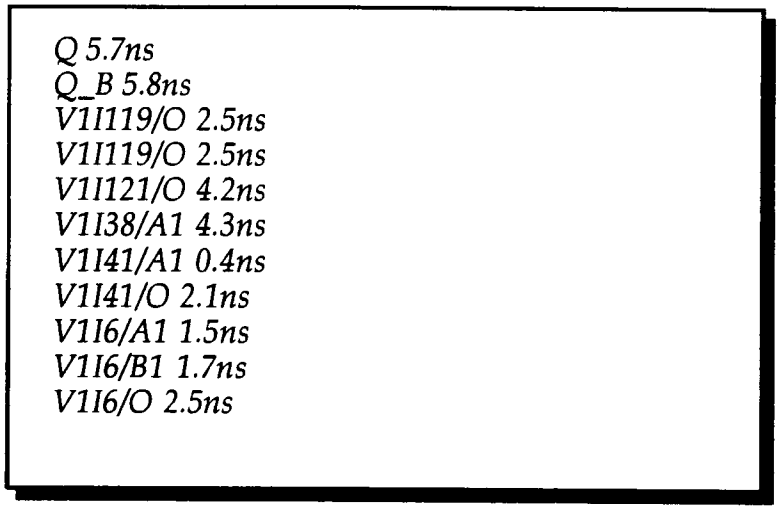



**Figure 4.31 dftestwin.hltab2 - Layout Transitional Dealys**

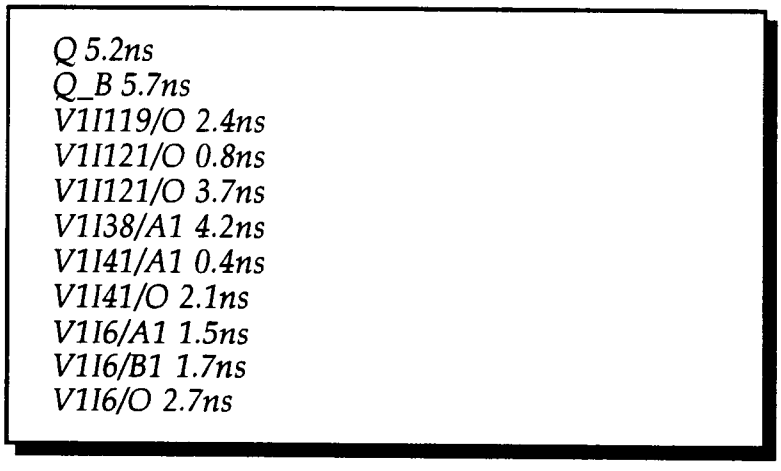



**Figure 4.32 dftestwin.trchltab2 - Post-Layout Transitional Delays**

## Table 4.4 dftestwin High-to-Low Transitional Delays Summary

| 1 --> 0 Nodes | Transitional Delays | | | % ERROR | |
|---|---|---|---|---|---|
| | 1st Path View2Sim | 2nd Path Sim2Bat | 3rd Path Bat2Sim | PreLayout | PostLayout |
| Q_B | 15 | 5.8 | 5.7 | 61 | 1.7 |
| V1I119/O | 9.1 | 2.5 | 2.4 | 72 | 4 |
| V1I121/O | 15.7 | 4.2 | 3.7 | 73 | 12 |
| V1I38/A1 | 12.7 | 4.3 | 4.2 | 66 | 2 |
| V1I41/A1 | 3.6 | 0.4 | 0.4 | 89 | 0 |
| V1I41/O | 5.9 | 2.1 | 2.1 | 64 | 0 |
| V1I6/A1 | 2.3 | 1.5 | 1.5 | 35 | 0 |
| V1I6/B1 | 2.3 | 1.7 | 1.7 | 26 | 0 |
| V1I6/O | 6.1 | 2.5 | 2.7 | 59 | 7 |
| Q | 18 | 5.7 | 5.2 | 68 | 8.7 |
| RMS % ERROR | | | | 61.3 | 3.5 |

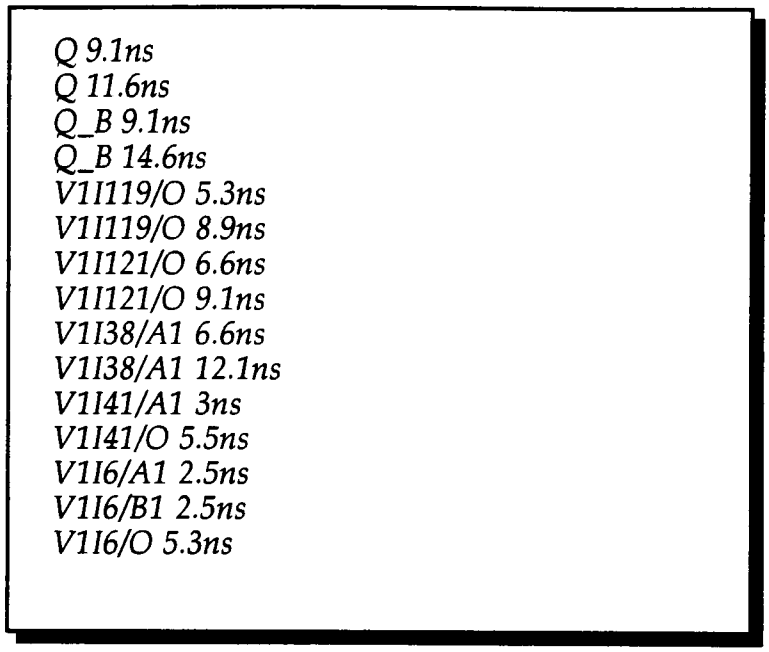



**Figure 4.33 dftestwin.trclhtab2 - Pre-Layout Transitional Delays**

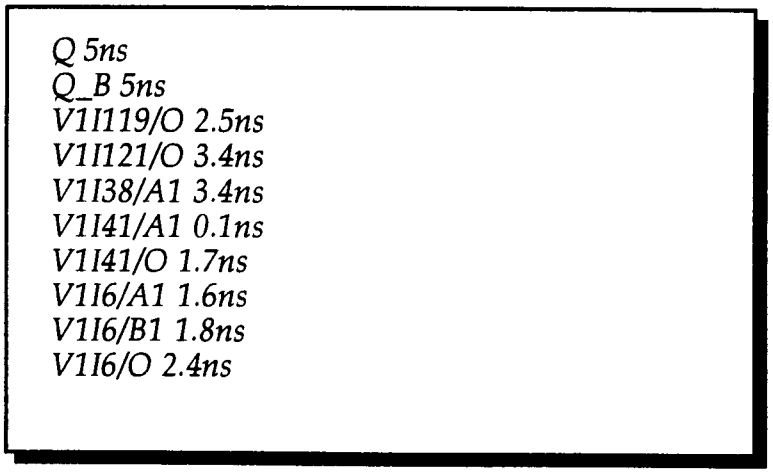



**Figure 4.34 dftestwin.lhtab2 - Layout Transitional Delays**

*Q 4.8ns*
*Q_B 4.5ns*
*V1I119/O 2.5ns*
*V1I121/O 3.2ns*
*V1I38/A1 2.9ns*
*V1I41/A1 0.1ns*
*V1I41/O 1.7ns*
*V1I6/A1 1.6ns*
*V1I6/B1 1.8ns*
*V1I6/O 2.4ns*

**Figure 4.35 dftestwin.trclhtab2 - Post-Layout Transitional Delays**

**Table 4.5 dftestwin - Low-to-High Transitional Delays Summary**

| 0 --> 1 Nodes | Transitional Delays | | | % ERROR | |
|---|---|---|---|---|---|
| | 1st Path View2Sim | 2nd Path Sim2Bat | 3rd Path Bat2Sim | PreLayout | PostLayout |
| Q_B | 14.6 | 5.0 | 4.5 | 66 | 10 |
| V1I119/O | 8.9 | 2.5 | 2.5 | 72 | 0 |
| V1I121/O | 9.1 | 3.4 | 3.2 | 62 | 6 |
| V1I38/A1 | 12.1 | 3.4 | 2.9 | 72 | 14.7 |
| V1I41/A1 | 3.0 | 0.1 | 0.1 | 96 | 0 |
| V1I41/O | 5.5 | 1.7 | 1.7 | 69 | 0 |
| V1I6/A1 | 2.5 | 1.6 | 1.6 | 36 | 0 |
| V1I6/B1 | 2.5 | 1.8 | 1.8 | 28 | 0 |
| V1I6/O | 5.3 | 2.4 | 2.4 | 55 | 0 |
| Q | 11.6 | 5.0 | 4.8 | 57 | 4 |
| RMS % ERROR | | | | 61.3 | 3.5 |

```
.BA
C $1I119
A TPHL=5 NS
A TPLH=4.9 NS
C $1I120
A TPHL=0.3 NS
A TPLH=3.1 NS
C $1I121
A TPHL=6.3 NS
A TPLH=0.4 NS
C $1I134
A TPHL=2.3 NS
A TPLH=2.5 NS
C $1I144
A TPHL=2.3 NS
A TPLH=2.5 NS
C $1I149
A TPHL=3.6 NS
A TPLH=3 NS
C $1I38
A TPHL=2.3 NS
A TPLH=2.5 NS
C $1I40
A TPHL=2.3 NS
A TPLH=2.5 NS
C $1I41
A TPHL=2.3 NS
A TPLH=2.5 NS
C $1I6
A TPHL=3.8 NS
A TPLH=2.8 NS
.AB
```

**Figure 4.36 dftestwinpass1.bat - Pre-Layout Back-Annotation Gate and Net Delays**

```
.BA
C $1I119
A TPHL=0.6NS
A TPLH=0.75NS
C $1I120
A TPHL=0.95NS
A TPLH=0.45NS
C $1I121
A TPHL=0.8NS
A TPLH=0.5NS
C $1I134
A TPHL=1.5NS
A TPLH=1.6NS
C $1I144
A TPHL=1.7NS
A TPLH=1.8NS
C $1I149
A TPHL=0.4NS
A TPLH=0.1NS
C $1I38
A TPHL=1.5NS
A TPLH=1.6NS
C $1I40
A TPHL=1.5NS
A TPLH=1.6NS
C $1I41
A TPHL=1.7NS
A TPLH=1.6NS
C $1I6
A TPHL=0.9NS
A TPLH=0.7NS
.AB
```

**Figure 4.37 dftestwin-new.bat - Layout Back-Annotation Gate and Net Delays**

```
.BA
C $1I119
A TPHL=0.5 NS
A TPLH=0.75 NS
C $1I120
A TPHL=1.15 NS
A TPLH=0.05 NS
C $1I121
A TPHL=0.25 NS
A TPLH=0.55 NS
C $1I134
A TPHL=1.5 NS
A TPLH=1.6 NS
C $1I144
A TPHL=1.7 NS
A TPLH=1.8 NS
C $1I149
A TPHL=0.4 NS
A TPLH=0.1 NS
C $1I38
A TPHL=1.5 NS
A TPLH=1.6 NS
C $1I40
A TPHL=1.5 NS
A TPLH=1.6 NS
C $1I41
A TPHL=1.7 NS
A TPLH=1.6 NS
C $1I6
A TPHL=1.1 NS
A TPLH=0.7 NS
.AB
```

**Figure 4.38 dftestwinpass2.bat - Post-Layout Back-Annotation Gate and Net Delays**

## Table 4.6 dftestwin - TPHL and TPLH Transitional Gate and Net Delays Summary

| Gate | Transitional Delays (ns) | | | | | | % ERROR | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1st Path View2Sim | | 2nd Path Sim2Bat | | 3rd Path Bat2Sim | | PreLayout | | PostLayout | |
| | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH |
| $1I119 | 5.0 | 4.9 | 0.6 | 0.75 | 0.5 | 0.75 | 88 | 98 | 16 | 0 |
| $1I120 | 0.3 | 3.1 | 0.95 | 0.45 | 1.15 | 0.05 | 68 | 85 | 17 | 88 |
| $1I121 | 6.3 | 0.4 | 0.8 | 0.5 | 0.25 | 0.55 | 87 | 20 | 68 | 9 |
| $1I134 | 2.3 | 2.5 | 1.5 | 1.6 | 1.5 | 1.6 | 35 | 36 | 0 | 0 |
| $1I144 | 2.3 | 2.5 | 1.7 | 1.8 | 1.7 | 1.8 | 26 | 28 | 0 | 0 |
| $1I149 | 3.6 | 3.0 | 0.4 | 0.1 | 0.4 | 0.1 | 88 | 97 | 0 | 0 |
| $1I38 | 2.3 | 2.5 | 1.5 | 1.6 | 1.5 | 1.6 | 35 | 36 | 0 | 0 |
| $1I40 | 2.3 | 2.5 | 1.5 | 1.6 | 1.5 | 1.6 | 35 | 36 | 0 | 0 |
| $1I41 | 2.3 | 2.5 | 1.7 | 1.6 | 1.7 | 1.6 | 26 | 36 | 0 | 0 |
| $1I6 | 3.8 | 2.8 | 0.9 | 0.7 | 1.1 | 0.7 | 76 | 75 | 18 | 0 |
| RMS % ERROR | | | | | | | 56 | 55 | 12 | 10 |

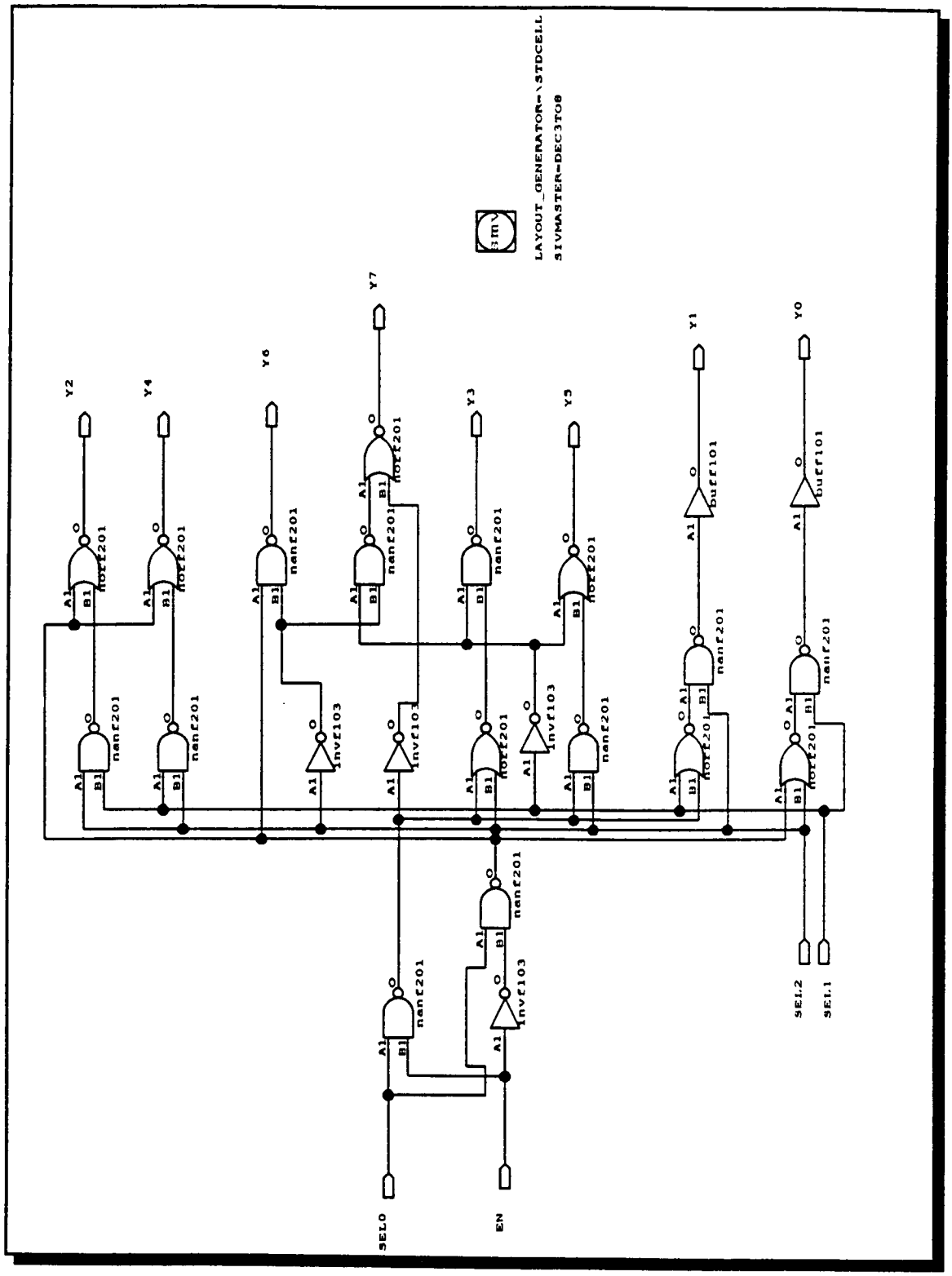



Figure 4.39 dec3to8.001p - Circuit Schematic

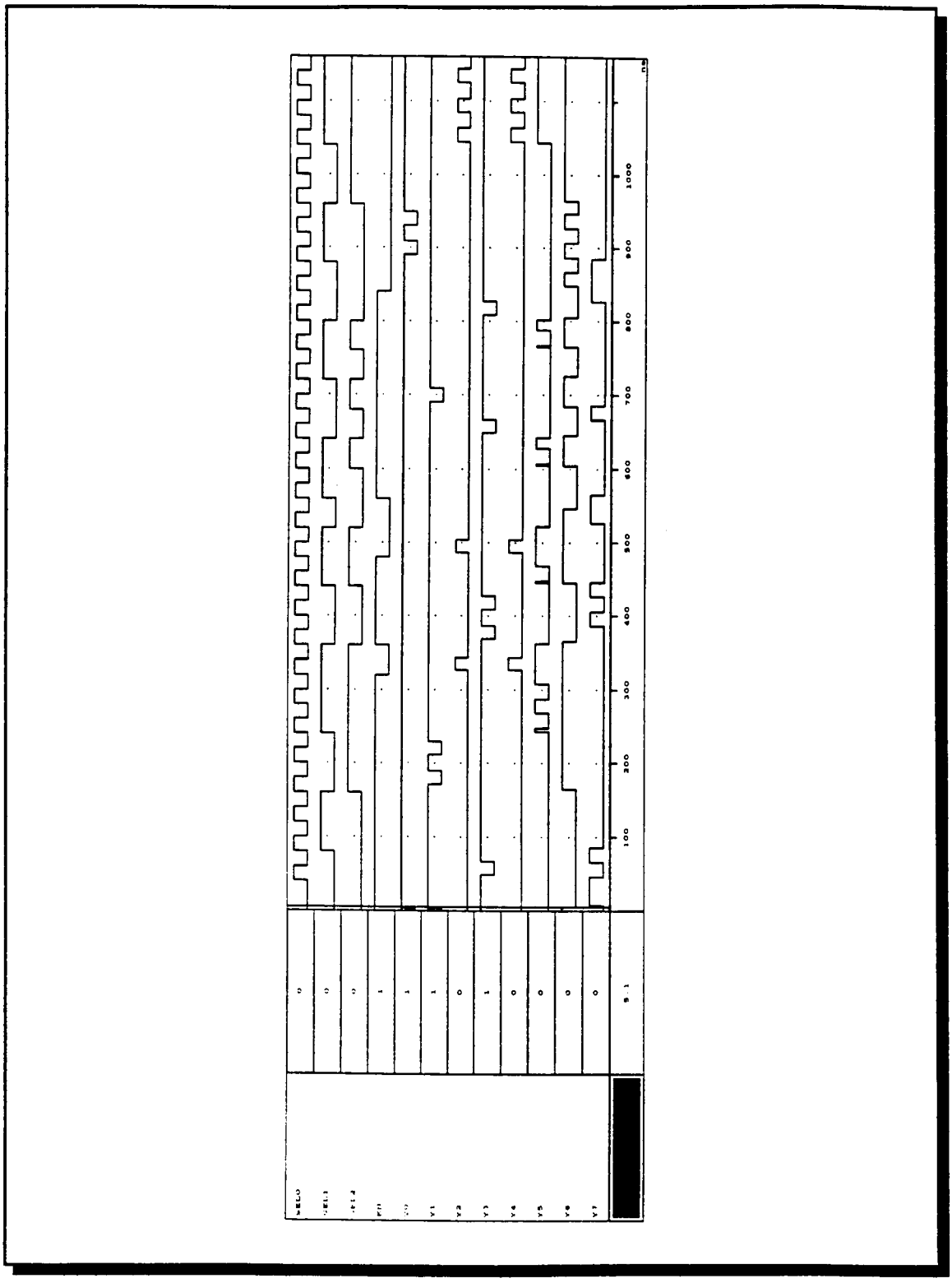


Figure 4.40 dec3to8.plt - Original Pre-Layout Simulation (1st Path)

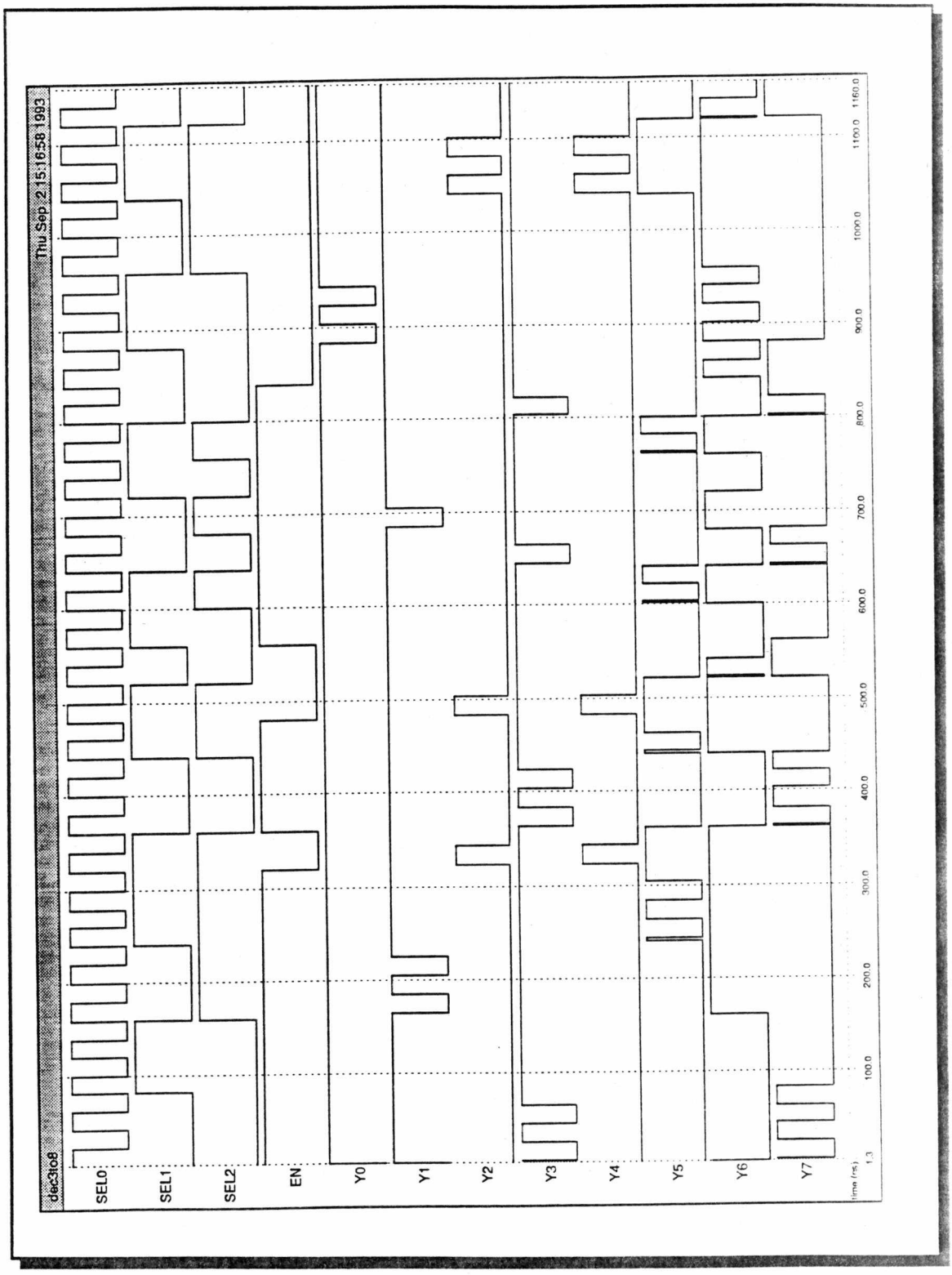



Figure 4.41 dec3to8.ps - Layout Simulation (2nd Path)

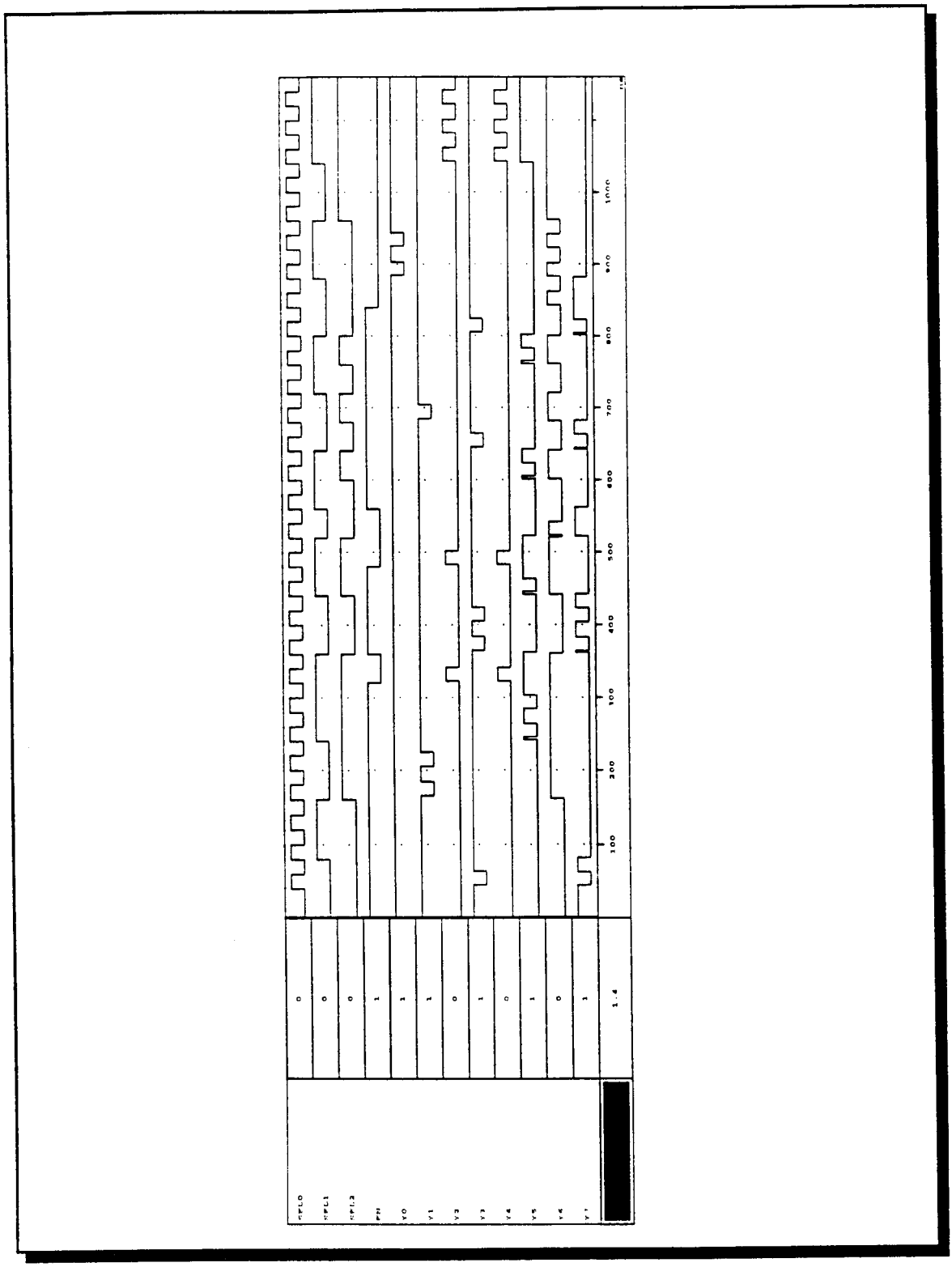


Figure 4.42 dec3to8-new.plt - Post-Layout Simulation (3rd Path)

*Transitional Delay Results*

The High-to-Low delay files of the first, second and third paths are presented in Figures 4.43, 4.44 and 4.45. A summary of the results is shown in Table 4.7. The Low-to-High delay files are shown in Figures 4.46, 4.47 and 4.48. The detailed results are summarized in Table 4.8. The result of using ***Sim2Bat*** is an improvement of almost 50% in reporting accurate delays compared to the pre-layout schematic simulation results.

*Gate and Net Delays*

The delay attribute values of TPHL and TPLH of each gate of the circuit are provided in Figures 4.49, 4.50 and 4.51. Table 4.9 shows the results of the delay comparisons of each path. ***CKTIM*** resulted an improvement of 40%.

*Test Case #4: ADDER4WIN*

*adder4win* is the last of four sample designs created by the author in order to simulate and examine the design timing verification method of ***Sim2Bat***. The design schematic is shown in Figure 4.52. This circuit represents a four-bit adder as the name *"adder4win"* implies. It differs from the other test cases since it uses complex gates such as aof2201 (4 input AND-OR function). In general a one-bit adder is sufficient to be included in the cell library; this one bit can be replicated and pipelined in order to form any number of bit adder using synthesis tools.

*Graphical Results*

The following Figures show the timing waveforms obtained form the simulation of pre-layout, layout and post-layout of adder4win (see Figures 4.53, 4.54, and 4.55). The results of ***IRSIM*** match closely the new ***Viewsim*** output waveforms. On the other hand, the pre-layout and layout waveforms are quite different.

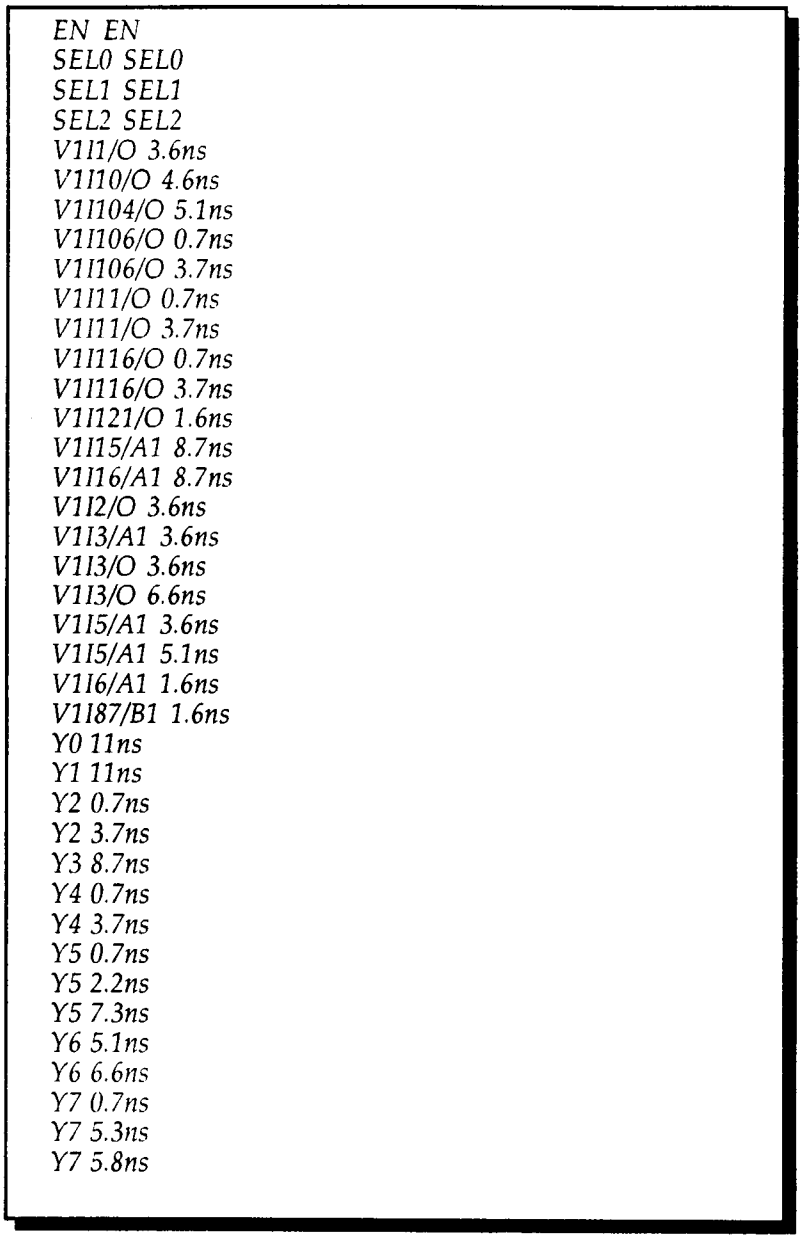



**Figure 4.43 dec3to8.trchltab2 - Pre-Layout Transitional Delays**

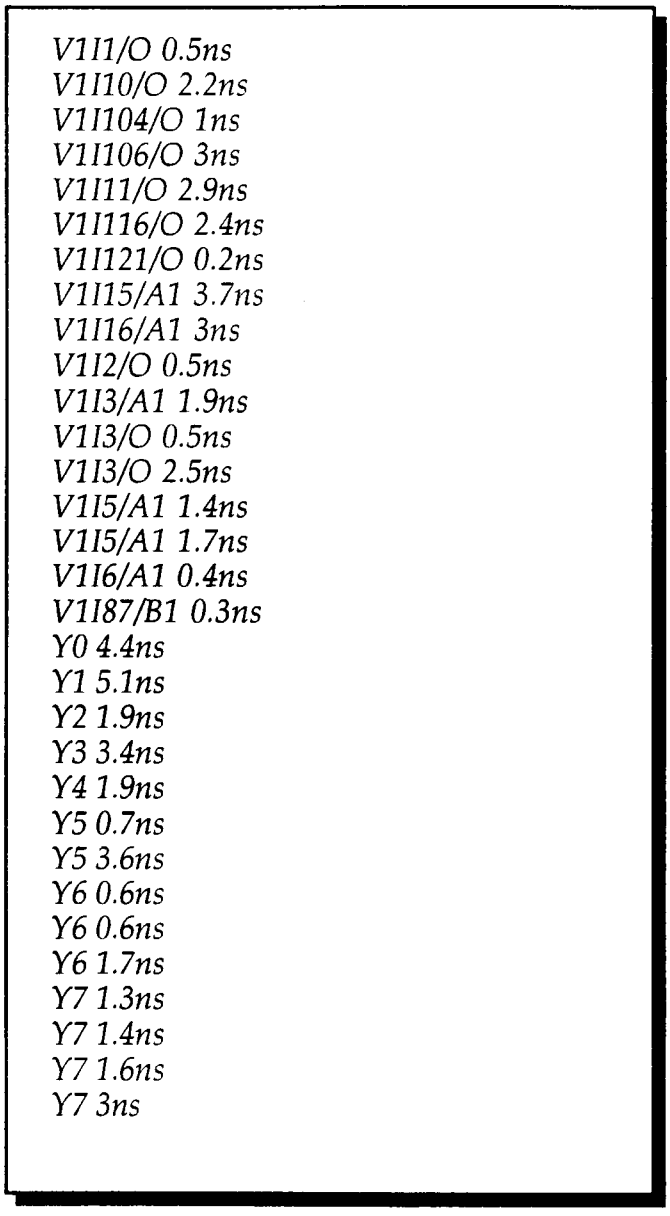



**Figure 4.44 dec3to8.hltab2 - Layout Transitional Delays**

*EN EN*
*SEL0 SEL0*
*SEL1 SEL1*
*SEL2 SEL2*
*V1I1/O 0.5ns*
*V1I10/O 1.9ns*
*V1I104/O 1ns*
*V1I106/O 1.1ns*
*V1I106/O 2.7ns*
*V1I11/O 1ns*
*V1I11/O 2.6ns*
*V1I116/O 0.8ns*
*V1I116/O 1.8ns*
*V1I121/O 0.2ns*
*V1I15/A1 4.1ns*
*V1I16/A1 3.4ns*
*V1I2/O 0.5ns*
*V1I3/A1 1.9ns*
*V1I3/O 0.6ns*
*V1I3/O 2.2ns*
*V1I5/A1 1.4ns*
*V1I5/A1 1.6ns*
*V1I6/A1 0.4ns*
*V1I87/B1 0.3ns*
*Y0 4.8ns*
*Y1 5.5ns*
*Y2 0.9ns*
*Y2 1.9ns*
*Y3 5.1ns*
*Y4 0.9ns*
*Y4 1.9ns*
*Y5 2.2ns*
*Y5 2.5ns*
*Y5 5.5ns*
*Y6 1.1ns*
*Y6 1.8ns*
*Y7 2.4ns*
*Y7 2.5ns*
*Y7 4.2ns*

**Figure 4.45 dec3to8.trchltab2 - Post Layout Transitional Delays**

## Table 4.7 dec3to8 - High-to-Low Transitional Delays

| 1 --> 0 Nodes | Transitional Delays | | | % ERROR | |
|---|---|---|---|---|---|
| | 1st Path View2Sim | 2nd Path Sim2Bat | 3rd Path Bat2Sim | PreLayout | PostLayout |
| I1/O | 3.6 | 0.5 | 0.5 | 86 | 0 |
| I10/O | 4.6 | 2.2 | 1.9 | 52 | 13 |
| I104/O | 5.1 | 1.0 | 1.0 | 80 | 0 |
| I106/O | 3.7 | 3.0 | 2.7 | 19 | 30 |
| I11/O | 3.7 | 2.9 | 2.6 | 22 | 10 |
| I116/O | 3.7 | 2.4 | 1.8 | 35 | 2.5 |
| I121/O | 1.6 | 0.2 | 0.2 | 87 | 0 |
| I15/A1 | 8.7 | 3.7 | 4.1 | 57 | 9.7 |
| I16/A1 | 8.7 | 3.0 | 3.4 | 65 | 12 |
| I2/O | 3.6 | 0.5 | 0.5 | 86 | 0 |
| I3/A1 | 3.6 | 1.9 | 1.9 | 47 | 0 |
| I3/O | 6.6 | 2.5 | 2.2 | 62 | 12 |
| I5/A1 | 5.1 | 1.7 | 1.6 | 66 | 6 |
| I6/A1 | 1.6 | 0.4 | 0.4 | 75 | 0 |
| I87/B1 | 1.6 | 0.3 | 0.3 | 81 | 0 |
| Y0 | 11 | 4.4 | 4.8 | 60 | 8 |
| Y1 | 11 | 5.1 | 5.5 | 54 | 7 |
| Y2 | 3.7 | 1.9 | 1.9 | 48 | 0 |
| Y3 | 8.7 | 3.4 | 5.1 | 61 | 31 |
| Y4 | 3.7 | 1.9 | 1.9 | 48 | 0 |
| Y5 | 7.3 | 3.6 | 5.5 | 51 | 34 |
| Y6 | 6.6 | 1.7 | 1.8 | 74 | 5.5 |
| Y7 | 5.8 | 3.0 | 4.2 | 48 | 28 |
| RMS % ERROR | | | | 59 | 10 |

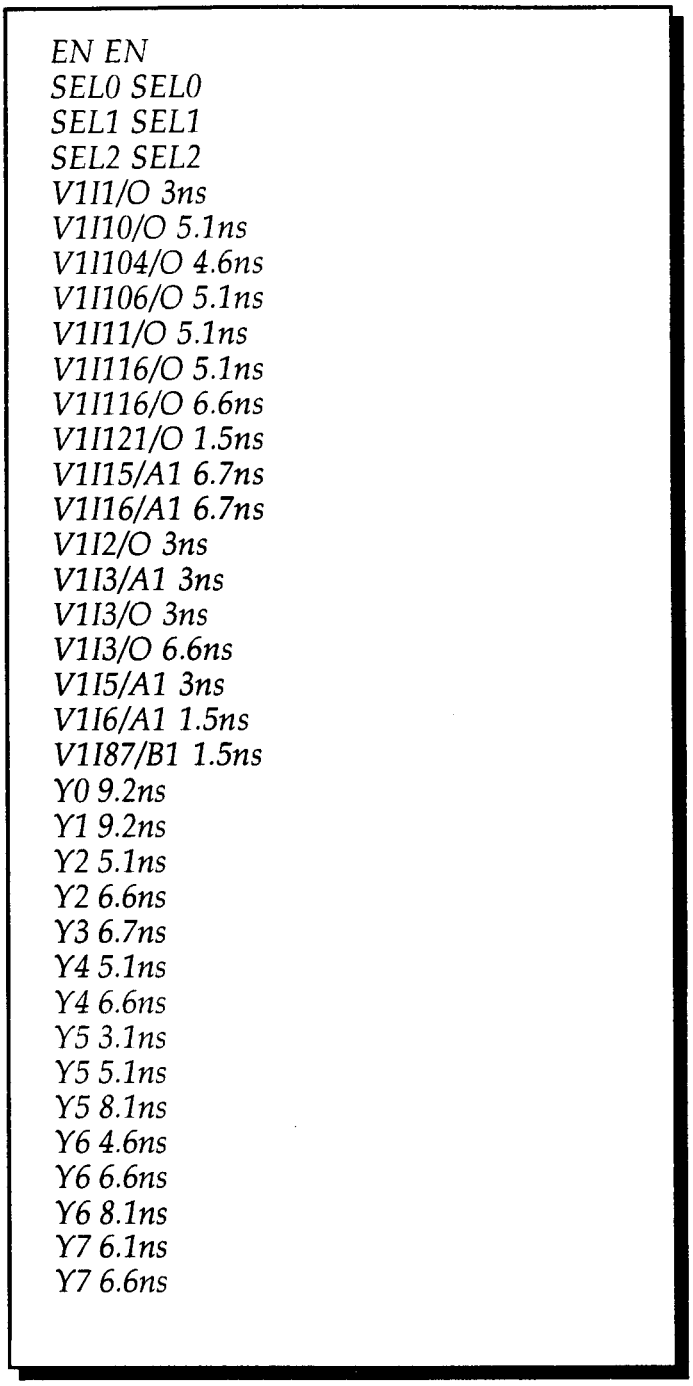



**Figure 4.46 dec3to8.trclhtab2 - Pre-Layout Transitional Delays**

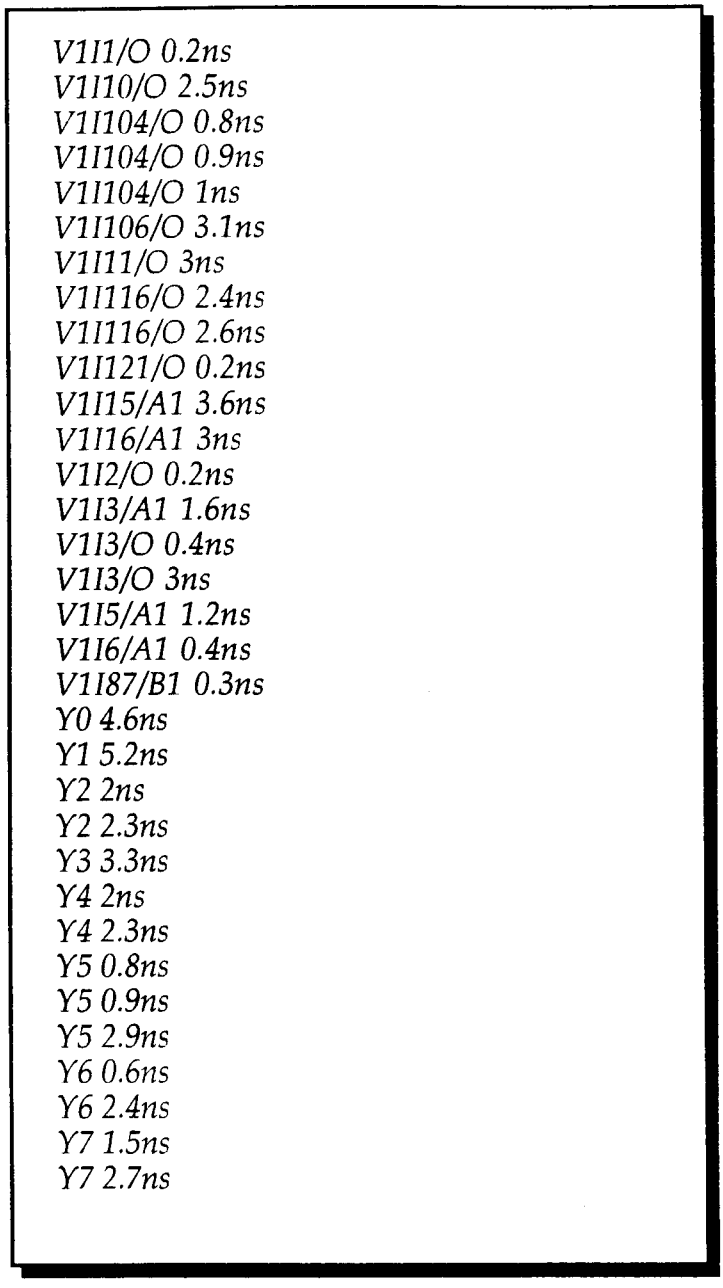



**Figure 4.47 dec3to8.lhtab2 - Layout Transitional Delays**

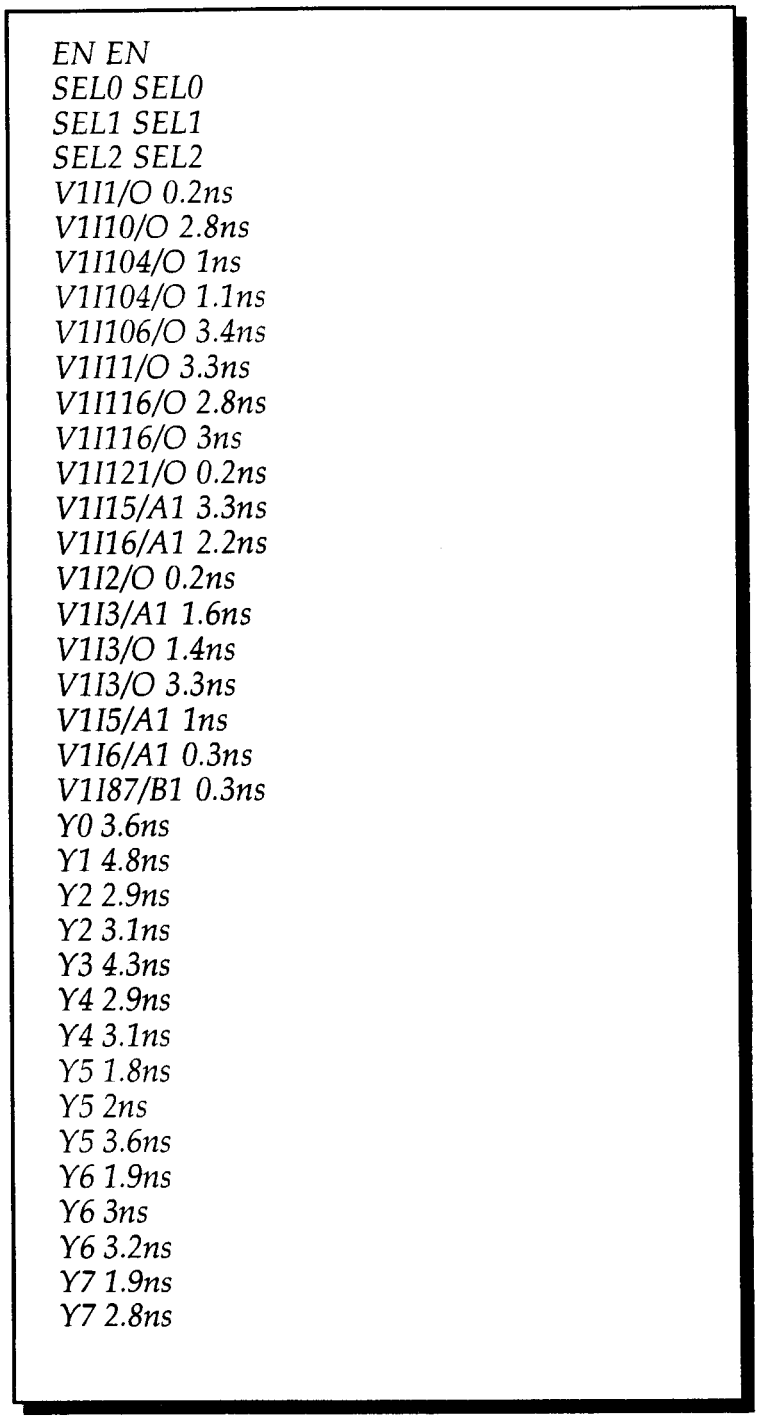



**Figure 4.48 dec3to8.lhtab2 - Post-Layout Transitional Delays**

## Table 4.8 dec3to8 - Low-to-High Transitionl Delays Summary

| 0 --> 1 / Nodes | Transitional Delays | | | % ERROR | |
|---|---|---|---|---|---|
| | 1st Path View2Sim | 2nd Path Sim2Bat | 3rd Path Bat2Sim | PreLayout | PostLayout |
| I1/O | 3.0 | 0.2 | 0.2 | 93 | 0 |
| I10/O | 5.1 | 2.5 | 2.8 | 51 | 10.7 |
| I104/O | 4.6 | 1.0 | 1.1 | 78 | 9 |
| I106/O | 5.1 | 3.1 | 3.4 | 40 | 8.8 |
| I11/O | 5.1 | 3.0 | 3.3 | 41 | 9 |
| I116/O | 6.6 | 2.7 | 3.0 | 59 | 10 |
| I121/O | 1.5 | 0.2 | 0.2 | 87 | 0 |
| I15/A1 | 6.7 | 3.6 | 3.3 | 46 | 8.3 |
| I16/A1 | 6.7 | 3.0 | 2.2 | 55 | 20 |
| I2/O | 3.0 | 0.2 | 0.2 | 93 | 0 |
| I3/A1 | 3.0 | 1.6 | 1.6 | 46 | 0 |
| I3/O | 6.6 | 3.0 | 3.3 | 55 | 0 |
| I5/A1 | 3.0 | 1.2 | 1.0 | 60 | 16 |
| I6/A1 | 1.5 | 0.4 | 0.3 | 73 | 25 |
| I87/B1 | 1.5 | 0.3 | 0.3 | 80 | 0 |
| Y0 | 9.2 | 4.6 | 3.6 | 50 | 21 |
| Y1 | 9.2 | 5.2 | 4.8 | 43 | 7.7 |
| Y2 | 6.6 | 2.3 | 3.1 | 65 | 25 |
| Y3 | 6.7 | 3.3 | 4.3 | 51 | 23 |
| Y4 | 6.6 | 2.3 | 3.1 | 65 | 25 |
| Y5 | 8.1 | 2.9 | 3.6 | 64 | 19 |
| Y6 | 8.1 | 2.4 | 3.2 | 70 | 25 |
| Y7 | 6.6 | 2.7 | 2.8 | 53 | 3.5 |
| | | | | | |
| RMS % ERROR | | | | 61.6 | 11.5 |

```
.BA
C $1I1
A TPHL=3.6 NS
A TPLH=3 NS
C $1I10
A TPHL=1 NS
A TPLH=2.1 NS
C $1I102
A TPHL=0.95 NS
A TPLH=1.75 NS
C $1I104
A TPHL=3.5 NS
A TPLH=3.1 NS
C $1I106
A TPHL=0.1 NS
A TPLH=2.1 NS
C $1I107
A TPHL=5 NS
A TPLH=1.6 NS
C $1I11
A TPHL=0.1 NS
A TPLH=2.1 NS
C $1I115
A TPHL=5 NS
A TPLH=0.1 NS
C $1I116
A TPHL=3.6NS
A TPLH=3.6NS
C $1I12
A TPHL=6.05 NS
A TPLH=3.4 NS
C $1I120
A TPHL=3.5 NS
A TPLH=1.5 NS
```

**Figure 4.49 dec3to8pass1.bat - Pre-Layout Back-Annotation Gate and Net Delays**

```
C $1I121
A TPHL=1.6 NS
A TPLH=1.5 NS
C $1I129
A TPHL=1.6 NS
A TPLH=1.5 NS C $1I134
A TPHL=1.6 NS
A TPLH=1.5 NS
C $1I15
A TPHL=2.3 NS
A TPLH=2.5 NS
C $1I16
A TPHL=2.3 NS
A TPLH=2.5 NS
C $1I17
A TPHL=3.6 NS
A TPLH=3 NS
C $1I2
A TPHL=3.6 NS
A TPLH=3 NS
C $1I3
A TPHL=3 NS
A TPLH=3.6 NS
C $1I4
A TPHL=3.6NS
A TPLH=3.6NS
C $1I5
A TPHL=3.6NS
A TPLH=3.6NS
C $1I6
A TPHL=3.2 NS
A TPLH=4.05 NS
C $1I87
A TPHL=3.25 NS
A TPLH=5.85 NS
.AB
```

**Continued**

```
.BA
C $1I1
A TPHL=0.5NS
A TPLH=0.2NS
C $1I10
A TPHL=0.3NS
A TPLH=0.9NS
C $1I102
A TPHL=1.4NS
A TPLH=0.85NS
C $1I104
A TPHL=0.65NS
A TPLH=0.7NS
C $1I106
A TPHL=1.1NS
A TPLH=1.5NS
C $1I107
A TPHL=0.7NS
A TPLH=0.6NS
C $1I11
A TPHL=1NS
A TPLH=1.4NS
C $1I115
A TPHL=0.6NS
A TPLH=0.4NS
C $1I116
A TPHL=0.8NS
A TPLH=1.4NS
C $1I12
A TPHL=1.75NS
A TPLH=1.65NS
C $1I120
A TPHL=1.4NS
A TPLH=1NS
C $1I121
A TPHL=0.2NS
A TPLH=0.2NS
C $1I129
A TPHL=0.3NS
A TPLH=0.3NS
```

**Figure 4.50 dec3to8-new.bat - Layout Back-Annotation Gate and Net Delays**

```
C $1I134
A TPHL=0.4NS
A TPLH=0.3NS
C $1I15
A TPHL=1.4NS
A TPLH=1.5NS
C $1I16
A TPHL=1.4NS
A TPLH=1.4NS
C $1I17
A TPHL=1.9NS
A TPLH=1.6NS
C $1I2
A TPHL=0.5NS
A TPLH=0.2NS
C $1I3
A TPHL=0.6NS
A TPLH=1.4NS
C $1I4
A TPHL=0.85NS
A TPLH=1.5NS
C $1I5
A TPHL=0.85NS
A TPLH=1.5NS
C $1I6
A TPHL=2.15NS
A TPLH=1.35NS
C $1I87
A TPHL=0.75NS
A TPLH=1.55NS
.AB
```

**Continued**

```
.BA
C $1I1
A TPHL=0.5 NS
A TPLH=0.2 NS
C $1I10
A TPHL=1.2NS
A TPLH=1.2NS
C $1I102
A TPHL=2.75 NS
A TPLH=0.85 NS
C $1I104
A TPHL=0.65 NS
A TPLH=0.8 NS
C $1I106
A TPHL=0.8 NS
A TPLH=1.8 NS
C $1I107
A TPHL=1.4NS
A TPLH=1.4NS
C $1I11
A TPHL=0.7 NS
A TPLH=1.7 NS
C $1I115
A TPHL=1.6NS
A TPLH=1.6NS
C $1I116
A TPHL=0.2 NS
A TPLH=2 NS
C $1I12
A TPHL=3.6 NS
A TPLH=2.5 NS
C $1I120
A TPHL=1.4 NS
A TPLH=0.8 NS
C $1I121
A TPHL=0.2 NS
A TPLH=0.2 NS
C $1I129
A TPHL=0.3 NS
A TPLH=0.3 NS
```

**Figure 4.51 dec3to8pass2.bat - Post-Layout Back-Annotation Gate and Net Delays**

```
C $1I134
A TPHL=0.4 NS
A TPLH=0.3 NS
C $1I15
A TPHL=1.4 NS
A TPLH=1.5 NS
C $1I16
A TPHL=1.4 NS
A TPLH=1.4 NS
C $1I17
A TPHL=1.9 NS
A TPLH=1.6 NS
C $1I2
A TPHL=0.5 NS
A TPLH=0.2 NS
C $1I3
A TPHL=0.3 NS
A TPLH=1.7 NS
C $1I4
A TPHL=0.85 NS
A TPLH=2.5 NS
C $1I5
A TPHL=0.85 NS
A TPLH=2.5 NS
C $1I6
A TPHL=4.2 NS
A TPLH=1.8 NS
C $1I87
A TPHL=0.85 NS
A TPLH=2.55 NS
.AB
```

**Continued**

## Table 4.9 dec3to8 TPHL and TPLH Transitional Gate and Net Delays Summary

| Gate | Transitional Delays (ns) | | | | | | % ERROR | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1st Path View2Sim | | 2nd Path Sim2Bat | | 3rd Path Bat2Sim | | PreLayout | | PostLayout | |
| | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH |
| I1 | 3.6 | 3 | 0.5 | 0.2 | 0.5 | 0.2 | 86 | 93 | 0 | 0 |
| I10 | 1.0 | 2.1 | 0.3 | 0.9 | 1.2 | 1.2 | 70 | 57 | 75 | 25 |
| I102 | 0.95 | 1.75 | 1.4 | 0.9 | 2.75 | 0.85 | 32 | 46 | 49 | 10 |
| I104 | 3.5 | 3.1 | 0.65 | 0.95 | 0.65 | 0.8 | 81 | 79 | 0 | 18 |
| I106 | 0.1 | 2.1 | 1.1 | 0.65 | 0.8 | 1.8 | 90 | 28 | 27 | 16 |
| I107 | 5.0 | 1.6 | 0.7 | 1.5 | 1.4 | 1.4 | 86 | 69 | 50 | 64 |
| I11 | 0.1 | 2.1 | 1.0 | 0.5 | 0.7 | 1.7 | 90 | 33 | 30 | 17 |
| I115 | 5.0 | 0.1 | 0.6 | 1.4 | 1.6 | 1.6 | 88 | 66 | 62 | 81 |
| I116 | 3.6 | 3.6 | 0.7 | 0.3 | 0.2 | 2.0 | 80 | 58 | 71 | 25 |
| I12 | 6.05 | 3.4 | 1.75 | 1.5 | 3.6 | 2.5 | 71 | 53 | 51 | 36 |
| I120 | 3.5 | 1.5 | 1.5 | 1.6 | 1.4 | 0.8 | 57 | 33 | 6.6 | 20 |
| I121 | 1.6 | 1.5 | 0.2 | 1.0 | 0.2 | 0.2 | 87 | 86 | 0 | 0 |
| I121 | 1.6 | 1.5 | 0.3 | 0.3 | 0.3 | 0.3 | 85 | 80 | 0 | 0 |
| I134 | 1.6 | 1.5 | 0.4 | 0.4 | 0.4 | 0.3 | 75 | 73 | 0 | 25 |
| I15 | 2.3 | 2.5 | 1.4 | 1.6 | 1.4 | 1.5 | 40 | 36 | 0 | 6 |
| I16 | 2.3 | 2.5 | 1.4 | 1.6 | 1.4 | 1.4 | 40 | 36 | 0 | 12 |
| I17 | 3.6 | 3 | 1.9 | 1.6 | 1.9 | 1.6 | 47 | 47 | 0 | 0 |
| I2 | 3.6 | 3 | 3.5 | 0.2 | 0.5 | 0.2 | 86 | 83 | 0 | 0 |
| I3 | 3 | 3.6 | 0.6 | 1.4 | 0.3 | 1.7 | 80 | 61 | 50 | 17 |
| I4 | 3.6 | 3.6 | 0.8 | 1.6 | 0.85 | 2.5 | 77 | 55 | 5.8 | 36 |
| I5 | 3.6 | 3.6 | 0.8 | 1.6 | 0.85 | 2.5 | 77 | 55 | 5.8 | 36 |
| I6 | 3.2 | 4.05 | 2.15 | 1.2 | 4.2 | 1.8 | 32 | 70 | 48 | 33 |
| I87 | 3.25 | 5.85 | 0.7 | 1.65 | 0.85 | 2.55 | 78 | 85 | 17 | 35 |
| RMS % ERROR | | | | | | | 62 | 60 | 23 | 22 |

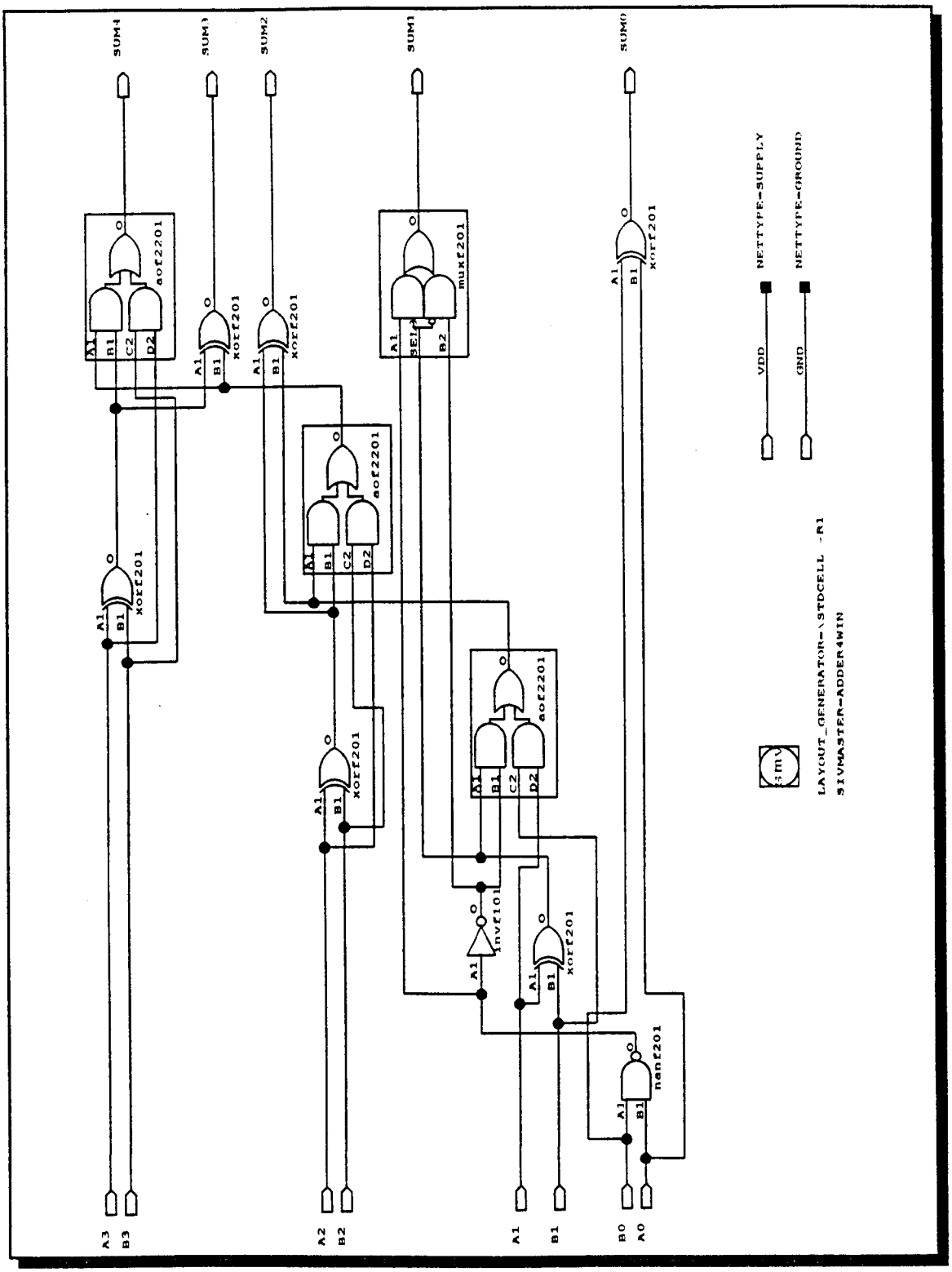



Figure 4.52 adder4win.001p - Circuit Schematic

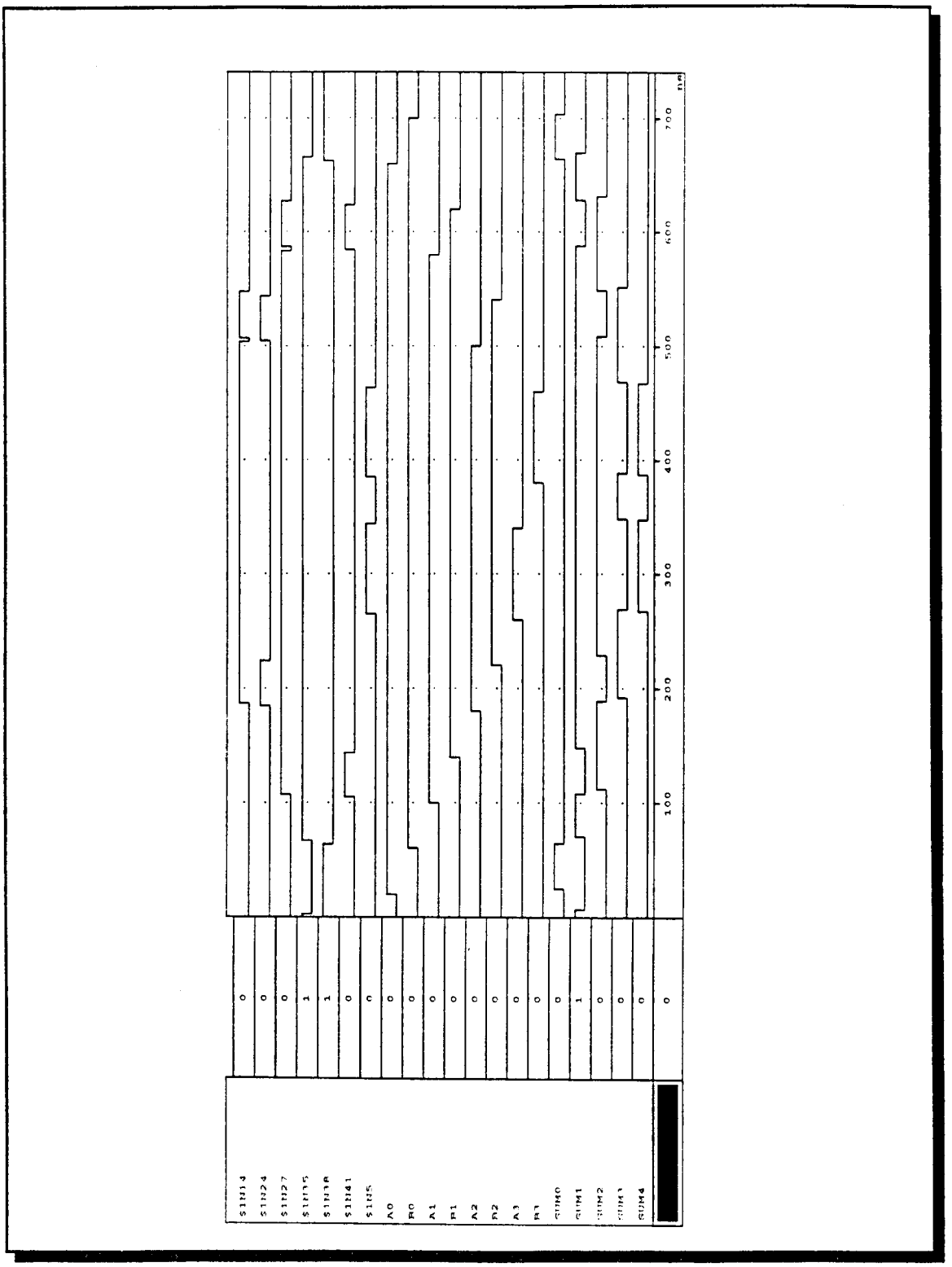



Figure 4.53 adder4win.plt - Original Pre-Layout Simulation (1st Path)

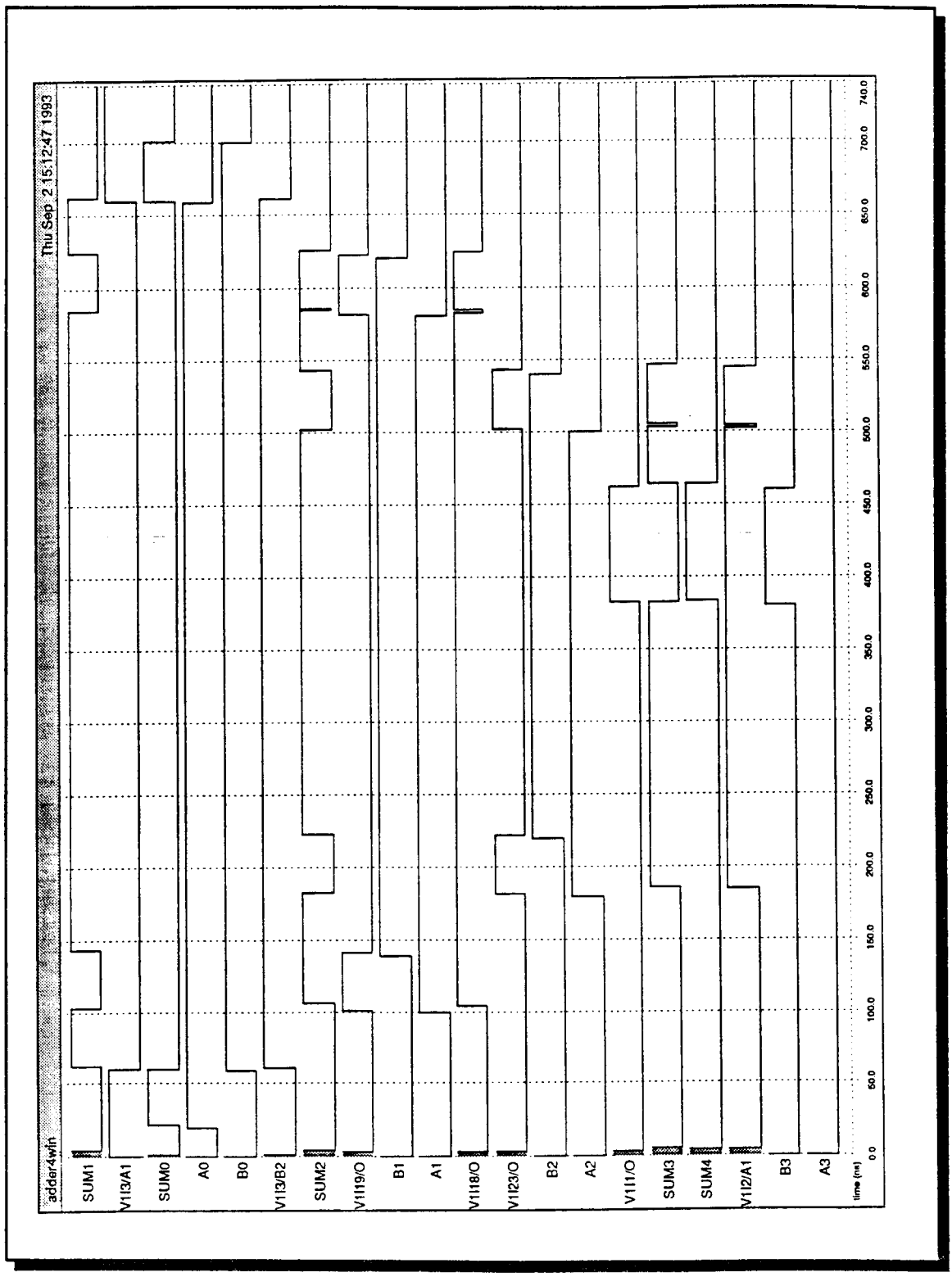



Figure 4.54 adder4win.ps - Layout Simulation (2nd Path)

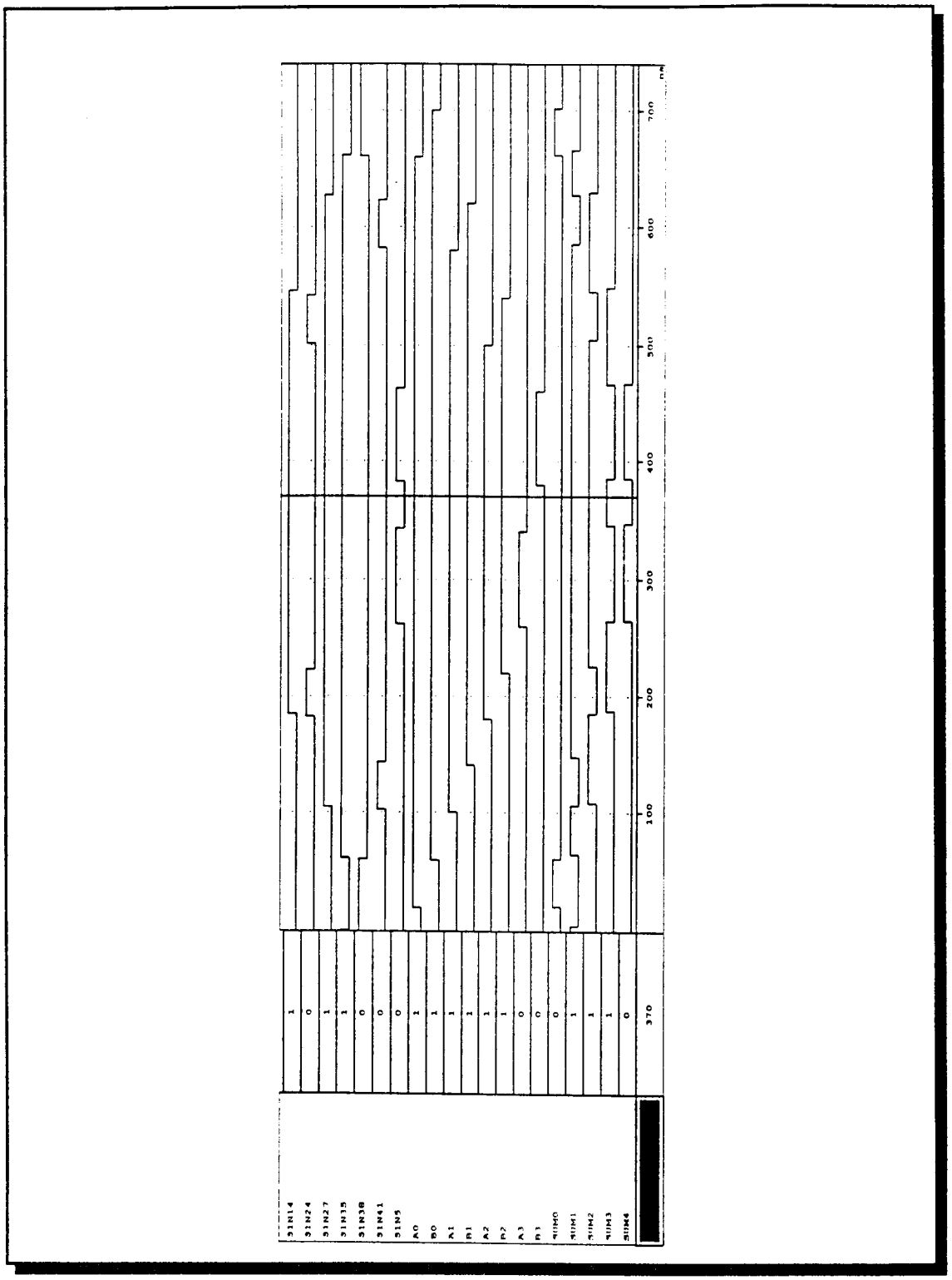



Figure 4.55 adder4win-new.plt - Post-Layout Simulation (3rd Path)

*Transitional Delay Results*

In Test Case #4, the High-to-Low delay files of the first, second, and third paths are presented in Figures 4.56, 4.57 and 4.58. Table 4.10 includes a summary of the results. Similarly, the Low-to-High delay files are shown in Figures 4.59, 4.60 and 4.61. The results are shown in Table 4.11. An average improvement of using ***Sim2Bat*** is almost 40%.

*Gate and Net Delays Results*

The delay attributes files for *"adder4win"* are presented in Figures 4.62, 4.63 and 4.64. The results are tabulated in Table 4.12. A total of 40% improvement is concluded by using ***CKTIM***.

*Accuracy and Error Calculation*

The results provided previously in this chapter are summarized in Table 4.13 This table shows the deviation of the original layout and the new post-layout TPHL and TPLH delays from the layout TPHL and TPLH delays. Moreover, Table 4.13 presents a view of the complexity of the test cases in addition to the total performance of each of them. As one can notice from this table, the consequence of using ***Sim2Bat*** gave a total improvement of 44%. An increase in the complexity of the cells under test, such as the number of gates used and the cells functionality provoked an increase of delay errors. However, it is noticed that using complex gates reduced the error by 10% compared to using expanded gates.

*Types of Errors*

The errors that spurred and accompanied our program results vary but cannot all be attributed to ***Sim2Bat***; in fact they can be divided into two categories: Intentional and Unintentional errors.

*A0 A0*
*A1 A1*
*A2 A2*
*A3 A3*
*B0 B0*
*B1 B1*
*B2 B2*
*B3 B3*
*SUM0 3.9ns*
*SUM1 7ns*
*SUM1 8ns*
*SUM1 10ns*
*SUM2 8.7ns*
*SUM2 11.5ns*
*SUM3 8.7ns*
*SUM3 11.5ns*
*SUM4 7.6ns*
*V1I1/O 3.9ns*
*V1I18/O 3.7ns*
*V1I18/O 7.6ns*
*V1I19/O 3.9ns*
*V1I2/A1 3.7ns*
*V1I2/A1 7.6ns*
*V1I23/O 3.9ns*
*V1I3/A1 3.6ns*
*V1I3/B2 3.1ns*
*V1I3/B2 6.1ns*

**Figure 4.56 adder4win.trchltab2 - Pre-layout Transitional Delays**

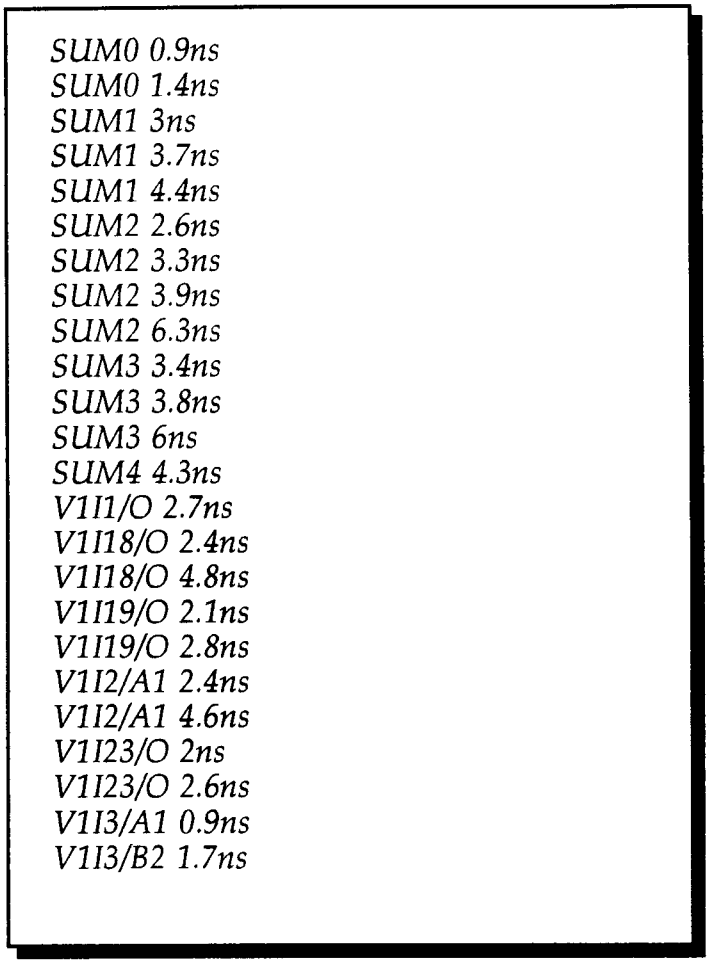



**Figure 4.57 adder4win.trchltab2 - Layout Transitional Delays**

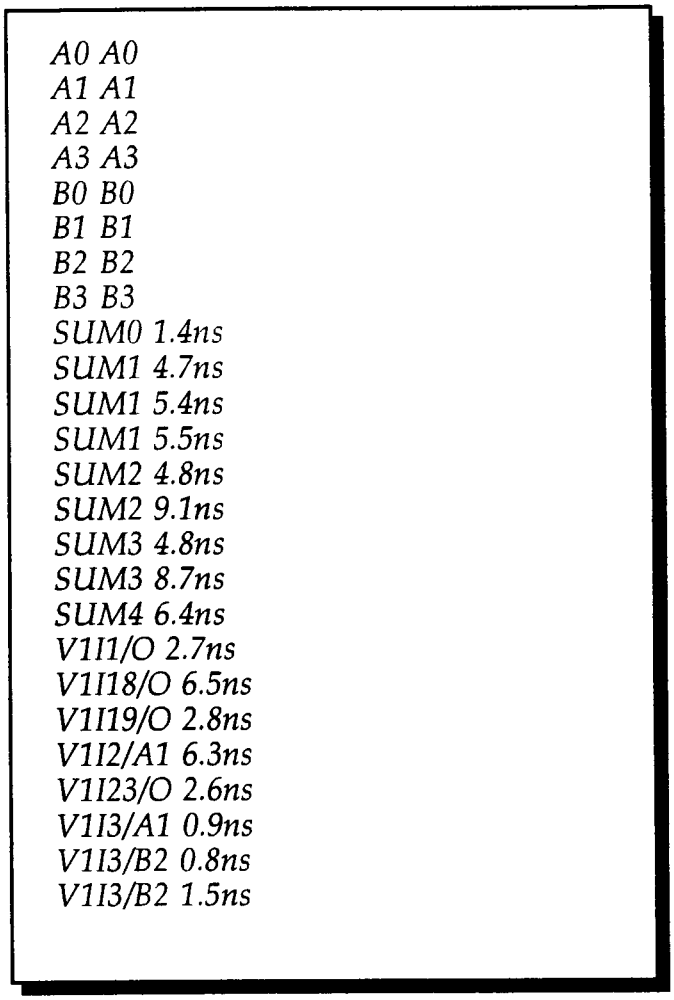



**Figure 4.58 adder4win.trchltab2 - Post-Layout Transitional Delays**

## Table 4.10 adder4win High-to-Low Transitional Delays Summary

| 1 --> 0 Nodes | Transitional Delays | | | % ERROR | |
|---|---|---|---|---|---|
| | 1st Path View2Sim | 2nd Path Sim2Bat | 3rd Path Bat2Sim | PreLayout | PostLayout |
| SUM0 | 3.9 | 1.4 | 1.4 | 64 | 0 |
| SUM1 | 10 | 4.4 | 5.5 | 56 | 20 |
| SUM2 | 11.5 | 6.3 | 9.1 | 45 | 30 |
| SUM3 | 11.5 | 6.0 | 8.7 | 48 | 31 |
| SUM4 | 7.6 | 4.3 | 6.4 | 43 | 32 |
| V1I1/O | 3.9 | 2.7 | 2.7 | 30 | 0 |
| V1I18/O | 7.6 | 4.8 | 6.5 | 37 | 26 |
| V1I19/O | 3.9 | 2.8 | 2.8 | 28 | 0 |
| V1I2/A1 | 7.6 | 4.6 | 6.3 | 40 | 27 |
| V1I23/O | 3.9 | 2.6 | 2.6 | 33 | 0 |
| V1I3/A1 | 3.6 | 0.9 | 0.9 | 75 | 0 |
| V1I3/B2 | 6.1 | 1.7 | 1.5 | 72 | 12 |
| RMS % ERROR | | | | 47.6 | 14.8 |

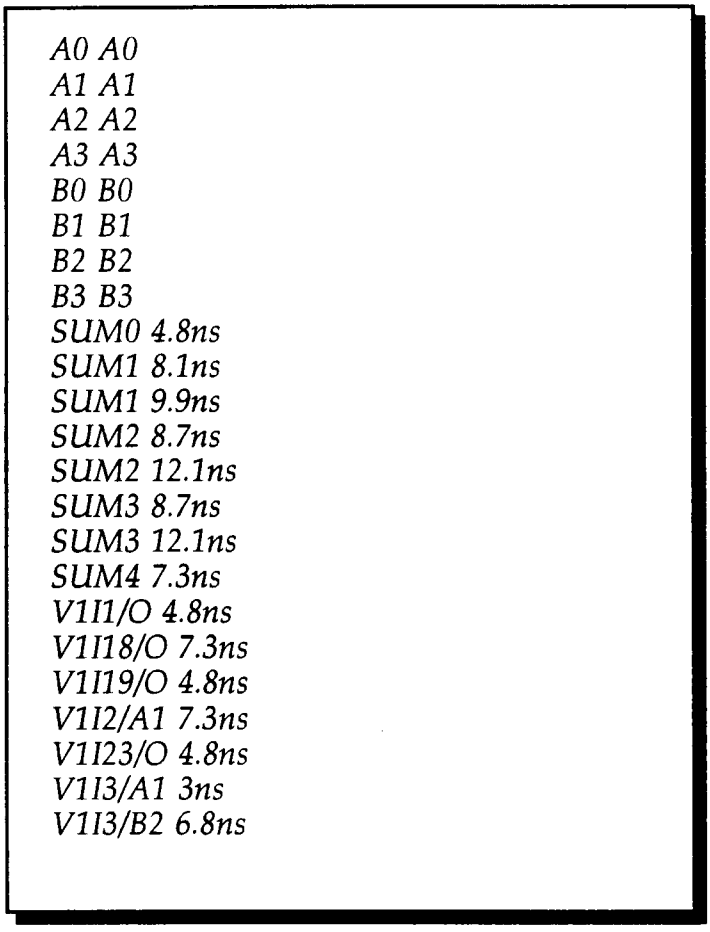



**Figure 4.59 adder4win.trclhtab2 - Pre-Layout Transitional Delays**

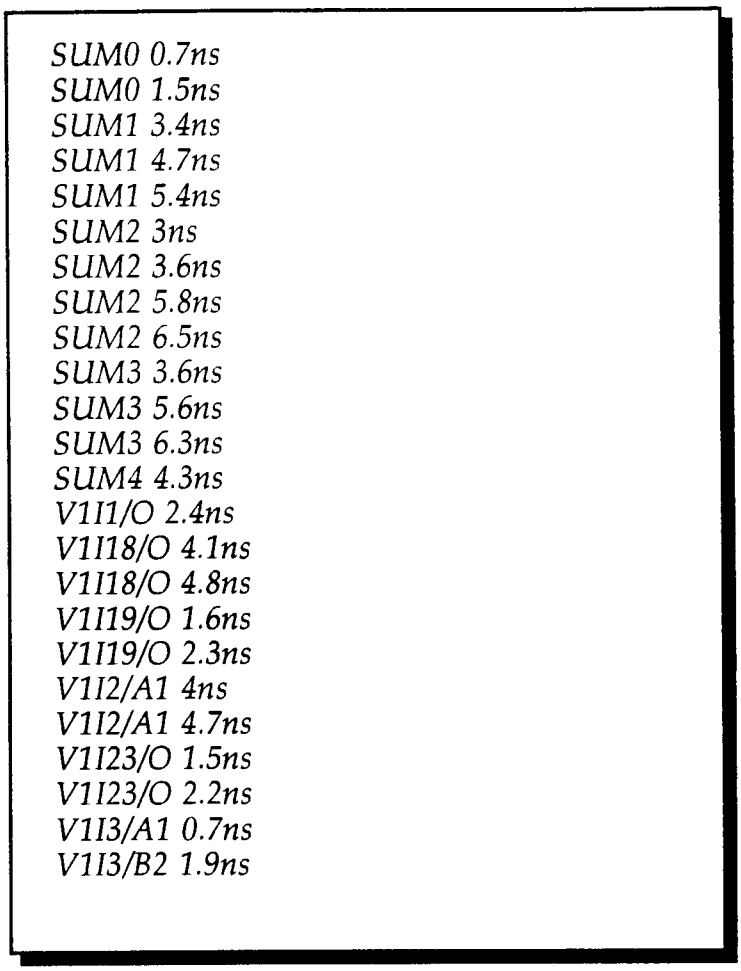



**Figure 4.60 adder4win.lhtab2 - Layout Transitional Delays**

*A0 A0*
*A1 A1*
*A2 A2*
*A3 A3*
*B0 B0*
*B1 B1*
*B2 B2*
*B3 B3*
*SUM0 1.5ns*
*SUM1 5.2ns*
*SUM1 7ns*
*SUM2 5.6ns*
*SUM2 7.8ns*
*SUM3 5.5ns*
*SUM3 7.5ns*
*SUM4 4.9ns*
*V1I1/O 2.4ns*
*V1I18/O 4.8ns*
*V1I19/O 2.3ns*
*V1I2/A1 4.7ns*
*V1I23/O 2.2ns*
*V1I3/A1 0.7ns*
*V1I3/B2 2.1ns*

**Figure 4.61 adder4win.trclhtab2 - Post-Layout Transitional Delays Summary**

## Table 4.11 adder4win Low-to-High Transitional Delays Summary

| 0--> 1 Nodes | Transitional Delays | | | % ERROR | |
|---|---|---|---|---|---|
| | 1st Path View2Sim | 2nd Path Sim2Bat | 3rd Path Bat2Sim | PreLayout | PostLayout |
| SUM0 | 4.8 | 1.5 | 1.5 | 69 | 0 |
| SUM1 | 9.9 | 5.4 | 7 | 45 | 23 |
| SUM2 | 12.1 | 6.5 | 7.8 | 46 | 16 |
| SUM3 | 12.1 | 6.3 | 7.5 | 48 | 16 |
| SUM4 | 7.3 | 4.3 | 4.9 | 41 | 12 |
| V1I1/O | 4.8 | 2.4 | 2.4 | 50 | 0 |
| V1I18/O | 7.3 | 4.8 | 4.8 | 34 | 0 |
| V1I19/O | 4.8 | 2.3 | 2.3 | 52 | 0 |
| V1I2/A1 | 7.3 | 4.7 | 4.7 | 35 | 0 |
| V1I23/O | 4.8 | 2.2 | 2.2 | 54 | 0 |
| V1I3/A1 | 3 | 0.7 | 0.7 | 76 | 0 |
| V1I3/B2 | 6.8 | 1.9 | 2.1 | 72 | 9.5 |
| RMS % ERROR | | | | 46 | 6.4 |

```
.BA
C $1I1
A TPHL=3.9 NS
A TPLH=4.8 NS
C $1I12
A TPHL=5.75 NS
A TPLH=6.05 NS
C $1I13
A TPHL=5.75 NS
A TPLH=6.05 NS
C $1I17
A TPHL=1.85 NS
A TPLH=1.25 NS
C $1I18
A TPHL=2.6 NS
A TPLH=1.5 NS
C $1I19
A TPHL=3.9 NS
A TPLH=4.8 NS
C $1I2
A TPHL=1.85 NS
A TPLH=1.25 NS
C $1I20
A TPHL=3.6 NS
A TPLH=3 NS
C $1I21
A TPHL=2.5 NS
A TPLH=3.8 NS
C $1I23
A TPHL=3.9 NS
A TPLH=4.8 NS
C $1I3
A TPHL=5.46667 NS
A TPLH=5.03333 NS
C $1I48
A TPHL=3.9 NS
A TPLH=4.8 NS
.AB
```

**Figure 4.62 adder4winpass1.bat - Pre-Layout Back-Annotation Gate and Net Delays**

```
.BA
C $1I1
A TPHL=2.7NS
A TPLH=2.4NS
C $1I12
A TPHL=2.35NS
A TPLH=2.75NS
C $1I13
A TPHL=2.6NS
A TPLH=3NS
C $1I17
A TPHL=0.9NS
A TPLH=1.2NS
C $1I18
A TPHL=2.55NS
A TPLH=2.7NS
C $1I19
A TPHL=2.8NS
A TPLH=2.3NS
C $1I2
A TPHL=0.65NS
A TPLH=0.75NS
C $1I20
A TPHL=0.9NS
A TPLH=0.7NS
C $1I21
A TPHL=0.8NS
A TPLH=1.2NS
C $1I23
A TPHL=2.6NS
A TPLH=2.2NS
C $1I3
A TPHL=2.6NS
A TPLH=3.76667NS
C $1I48
A TPHL=1.4NS
A TPLH=1.5NS
.AB
```

**Figure 4.63 adder4win-new.bat - Layout Back-Annotation Gate and Net Delays**

```
.BA
C $1I1
A TPHL=2.7 NS
A TPLH=2.4 NS
C $1I12
A TPHL=4.2 NS
A TPLH=3.95 NS
C $1I13
A TPHL=4.55 NS
A TPLH=4.3 NS
C $1I17
A TPHL=1.75 NS
A TPLH=1.2 NS
C $1I18
A TPHL=4.35 NS
A TPLH=2.6 NS
C $1I19
A TPHL=2.8 NS
A TPLH=2.3 NS
C $1I2
A TPHL=1.9 NS
A TPLH=1.35 NS
C $1I20
A TPHL=0.9 NS
A TPLH=0.7 NS
C $1I21
A TPHL=0.6 NS
A TPLH=1.4 NS
C $1I23
A TPHL=2.6 NS
A TPLH=2.2 NS
C $1I3
A TPHL=3.76667 NS
A TPLH=5.3 NS
C $1I48
A TPHL=1.4 NS
A TPLH=1.5 NS
.AB
```

**Figure 4.64 adder4winpass2.bat - Post-Layout Back-Annotation Gate and Net Delays**

## Table 4.12 adder4win TPHL and TPLH Transitional Gate and Net delays Summary

| Gate | Transitional Delays (ns) | | | | | | % ERROR | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1st Path View2Sim | | 2nd Path Sim2Bat | | 3rd Path Bat2Sim | | PreLayout | | PostLayout | |
| | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH | TPHL | TPLH |
| $1I1 | 3.9 | 4.8 | 2.7 | 2.4 | 2.7 | 2.4 | 31 | 50 | 0 | 0 |
| $1I12 | 5.75 | 6.05 | 2.35 | 2.75 | 4.2 | 3.95 | 60 | 54 | 44 | 30 |
| $1I13 | 5.75 | 6.05 | 2.6 | 3 | 4.55 | 4.3 | 55 | 50 | 43 | 30 |
| $1I17 | 1.85 | 1.25 | 0.9 | 1.2 | 1.75 | 1.2 | 51 | 4 | 48 | 0 |
| $1I18 | 2.6 | 1.5 | 2.55 | 2.7 | 4.35 | 2.6 | 2 | 44 | 41 | 3 |
| $1I19 | 3.9 | 4.8 | 2.8 | 2.3 | 2.8 | 2.3 | 28 | 52 | 0 | 0 |
| $1I12 | 1.85 | 1.25 | 0.65 | 0.75 | 1.9 | 1.35 | 65 | 40 | 66 | 51 |
| $1I20 | 3.6 | 3.0 | 0.9 | 0.7 | 0.9 | 0.7 | 75 | 76 | 0 | 0 |
| $1I21 | 2.5 | 3.8 | 0.8 | 1.2 | 0.6 | 1.4 | 68 | 79 | 25 | 14 |
| $1I23 | 4.8 | 3.9 | 2.6 | 2.2 | 2.6 | 2.2 | 46 | 43 | 0 | 0 |
| $1I3 | 5.46 | 5.03 | 2.6 | 3.76 | 3.76 | 5.3 | 52 | 25 | 30 | 29 |
| $1I48 | 3.9 | 4.8 | 1.4 | 1.5 | 1.4 | 1.4 | 64 | 69 | 0 | 0 |
| RMS % ERROR | | | | | | | 50 | 49 | 24 | 13 |

## Table 4.13 ERROR Calculation and Total Performance Results

| | Nodes Transitional Delays ERROR | | | | Gates Transitional Delays ERROR | | | |
|---|---|---|---|---|---|---|---|---|
| | High-to-Low | | Low-to-High | | High-to-Low | | Low-to-High | |
| Circuit | Pre Layout | Post Layout | Pre Layout | Post Layout | Pre Layout | Post Layout | Pre Layout | Post Layout |
| EXAMP1 | 77 | 0 | 77 | 4.6 | 80 | 19 | 76 | 13.5 |
| DFTESTWIN | 61.3 | 3.5 | 61.3 | 3.5 | 56 | 55 | 12 | 10 |
| DEC3TO8 | 59 | 10 | 61.6 | 11.5 | 62 | 23 | 60 | 22 |
| ADDER4WIN | 47.6 | 14.8 | 46 | 6.4 | 50 | 24 | 49 | 13 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Average Pre-Layout ERROR** | 61 | | 61 | | 62 | | 49 | | 58 | **Total % ERROR** |
| **Average Post-Layout ERROR** | | 7 | | 65 | | 30 | | 14 | 14 | |
| **Improve-ment** | | | | | | | | | 44 | |

| Circuit | Total Number Of Gates | Number Of Gate Inputs | Cell Complexity |
|---|---|---|---|
| EXAMP1 | 3 | 2 | Simple Circuit |
| DFTESTWIN | 10 | 1;2 | Delay Controlled Circuit |
| DEC3TO8 | 21 | 1;2 | Long Path Are Involvrd |
| ADDER4WIN | 12* | 1;2;3;4 | Complex Gates Are Used |

* If The Complex Gates Are Expanded, The Total Number Of Gates Used In The ADDER4WIN Circuit Total To 21 Gates.

*Intentional Errors*

As part of this thesis' strategy is to use the worst case scenario, which represents the designer's interest by reporting the maximum delays possible. ***Sim2Bat*** acquires the transitional delays from ***IRSIM*** and uses only the largest delay for each node. These delays depend on the gate functionality and the path of each input. One such example is a node which is driven by two outputs that have different RC delays depending on the net length. Given that the net delays are zero, the gate delays would be sufficient to simulate the circuit. This is not viable in ASIC designs since the interconnects inside and outside a cell are very important and critical for the timing considerations of a circuit.

*Unintentional Errors*

Unintentional errors are contributed to by two factors: Human and experimental errors.

*Human Intervention*

One of the largest and catastrophic errors can be attributed to the cell designer. This error characterized as a Stimulus error, which in turn produces a Sampling error. The Stimulus is developed by the designer in order to increase the confidence in the design's ability to perform to its specification. It is a non-trivial task to determine a set of test inputs that will sufficiently test a design.

Most often test inputs are generated manually by the designer and it is quite common for a human to miss or to test incompletely part of a design. Doing so, the untested part, which could be just one state of a gate, is not accounted for in the delay files. This leads to a path delay error that concludes a smaller delay for some inputs and eventually smaller gate and net delays. Moreover, this violates the need to use maximum delays in the whole circuit

delay calculations. One solution for this potential problem is to use fault tolerant techniques that guarantee the coverage of the whole design by toggling each node for the appropriate state. The most important task for a design using ***CKTIM*** is to make sure that each node in the circuit is toggled at least once from High-to-Low and once from Low-to-High states. This test input error can range from 10% to 50% and can be observed in Figure 4.57 where the node V1I2/A1 has a minimum and a maximum delay values of 2.4ns and 4.6ns, respectively. In this case, the test patterns were chosen carefully since the maximum delay value was detected. If this maximum delay transition was not covered by the input stimulis, the minimum value would have been used instead resulting an error of 25%.

*Experimental Error*

This error is a direct consequence of the measurement error present in all data that is submitted for analysis. The numerical methods used to analyze the data may have the effect of amplifying the experimental results. These measurement errors are contributed by the transitional delay errors of pre-layout simulation using ***IRSIM***. Hence, if a 10% error is introduced by the transitional delay of the layout simulation, the usage of ***Sim2Bat*** averaging routine, which subtracts the average sum of input delays from the output delay, contributes a near zero delay on a one-input gate, a 20%, 30%, and 40% error on two, three and four input gates, respectively. On the other hand these gate delay errors translate into transitional delay errors for post-layout simulation in the order of 10%, 20%, 30%, and 40% for a one, two, three and four input gates. Consider the following example: assume a three input gate, the delay is calculated as: output - (Input1+Input2+Input3)/3 = gate delay. If the inputs have a 10% error then the gate delay will have an error of 10%. Assume that this 10% error gate is used between the circuit input and another gate, the error will propagate as 10% to the output of the second gate and so forth. Thus, it is important to make sure that the data

produced by ***IRSIM*** is free of errors. In the test cases, the errors can be contributed to this type of error. On the other hand, it is very likely that ***ViewSim*** is not accurate in the delay calculation produced by the *".bat"* file and used by ***VSM***. Given this conclusion, it is clear that complex gates are superior in minimizing delay errors by a maximum of 40%.

*Error Compensation Program*

As it can be seen from the previous discussion, there are few cases that have a slight chance of occurring when using ***Sim2Bat***. The errors can translate into a zero-gate delay since the average input delay is equal to the output delay. Another error is the negative-gate delay, This is a direct result of the sampling problem since one or many of the inputs reported short path delays that can lead to a output delay that is smaller than the average sum of inputs. In order to remedy these errors, an error compensation program was developed and named ***FIX***. This program compensates for the error by making both TPHL and TPLH values the same as the positive delay value, whichever it is of the two High-to-Low or Low-to-High attributes. For example, if TPHL is negative and TPLH is positive for a certain gate, then ***FIX*** assigns the TPLH delay to TPHL. If both TPHL and TPLH in the *.bat* file are zeros, the designer is instructed to recheck his stimulus file in order to insure maximum test input coverage over the entire design.

The ***FIX*** program is a two step program. First the *.bat* file is expanded using the ***expand.nawk*** program. Second, the error correction is made using the ***tphl.nawk*** program. These programs require *"**nawk**"* utility which should exist on any new Sparc10 Sun Workstation (see Appendix D for complete listing of the ***FIX*** programs). This error correction program can compensate for the previously mentioned errors but sometimes cannot and that is when a designer needs to intervene. The designer needs to experiment with the experimental ***FIX*** program and decide when it is beneficial to use it.

In conclusion, the true percentage error is represented by the error in the transitional delays which totals about 14%. This is a minimal error that any designer can tolerate without jeopardizing the integrity of the circuit timing process.

# CHAPTER V

## SUMMARY AND CONCLUSIONS

***Sim2Bat*** and its infant programs such as ***View2Sim*** and ***Bat2Sim*** represent the conclusion of a development effort at the University of Tennessee which has lasted more than a year. During this time, many issues were addressed while expanding the capabilities of the system to meet a quality standard. As might be expected in a project of this magnitude and complexity, it is recognized that there are instances in which the software does not perform exactly as expected. For the benefit of ***CKTIM*** users, these instances are documented as follows:

*Description*: Incomplete Back-Annotation File.

*Solution*: Examine the file "**magnod**", this file contains the extracted magic nodes.

*Description*: Zero Delays in Back-Annotation File.

*Solution:* Examine your input stimulus, which lacks some test patterns in order to test all of your circuit.

*Description*: Empty *"nodes"* file.

*Solution*: Return the command *ext2irsim* in the layout directory since the system ignores this command occasionly.

*Description*: Huge delay values are reported.

*Solution*: Modify the "cycle" global variable in the program ***Threemod*** according to the stepsize used in the *"cmd"* file.

*Description:* Smaller than microseconds units.

*Solution*: ***Sim2Bat*** mandates the use of nano seconds; if the user uses micro seconds, the program will crash and report garbage results.

***Future Work***

Although ***CKTIM*** is an adequate process that fits the needs of the current tools at the University of Tennessee, few improvements can be made that would enhance the capabilities of ***Sim2Bat***. On one hand ***Sim2Bat*** can be used to extract the gate delays for each of the library cells and attach the TPHL and TPLH attributes to the components of those cells in Viewlogic. Once the designer builds the design, ***Sim2Bat*** can be used once more to extract the net delays by comparing the circuit delays versus the delays of the gates used in the design. In other words a filter needs to be developed that masks-off the net delays only. This should lead to accomplishing an analog process back-annotation filter where the components would be N-FETs and P-FETs with variable sizes (W/L). A transistor-based layout generator such as ***ICGEN*** can be used to place and route the transistors. ***Sim2Bat*** can be modified as suggested previously which would allow the user to check if the sizing of specific transistors were done correctly by checking the timing reports.

Next ***Sim2Bat*** cannot handle flip-flops since they possess additional attributes other than TPHL and TPLH. For instance the D-flip-flop *dbf311* has attributes such as CLKTOQLH, CLKTOQHL, STOQHL and many more (please consult the VHDL library for more details). Hence, in order to back-annotate those delays to ***Viewsim***, ***Sim2Bat*** needs to have a table of all the necessary information specific to each flip-flop. A challenge would be to develop a utility that handles each component of the library when used in a design depending on a certain pre-stored database.

Another task for the ambitious designer is to install all of the VHDL models in the built-in library as stand-alone symbols and not as multi-level

symbols. ***Viewsim*** does not ignore the lower level schematics delays when overwritten by the *.bat* file. It uses each of the low-level gate delays to recompute the input-to-outputs delays. This is a big disadvantage for ***Sim2Bat*** and can be solved if the ***Mega*** installation utility is available with a few modifications to ***Sim2Bat.***

*Contributions:*

The author has developed and automated the ***CKTIM*** process, which requires ***Sim2Bat*** (developed by the author) and needs ***ICGEN*** in order to accomplish ***DELTIM,*** the original goal of analog circuit back-annotation delay timing analysis automation. On the other hand, while using ***TimberWolf*** to test ***CKTIM,*** the author identified the need of ***DELTIM*** and a new extractor that extracts only the net delays. As it can be concluded from the results, ***Sim2Bat*** is poised for an ***ICGEN***-type of FET layout generator for analog circuit design.

***Sim2Bat*** is a user-friendly program that provides the user with gate and net delays in a single environment. It constitutes a powerful computational tool that runs in less than thirty seconds of CPU time. A third contribution of ***Sim2Bat*** is the flexibility of the style and the advantages of using **awk** and **sed** programs where modifications can be easily introduced. These are the main advantages of ***Sim2Bat***.

Other contributions include the development of a pre-processor ***View2Sim*** and a post-processor ***Bat2Sim*** that enable the designer to check the results of ***Sim2Bat***.

*Conclusion*

A procedure for post-layout timing analysis of integrated circuits has been developed and automated. An algorithm to determine the gate and net delays was written. The simulation procedure in ***Viewsim*** was not interrupted but enhanced in order to reflect the layout net and gate delays graphically (on screen) and manually (text files stored in directories created by ***Sim2Bat***). The process keeps track of the transitional delays and compensates for many errors using an error correction routine. The input files are formed of transitional delays, while the end result file is a *"bat"* data file that generates a *"vsm"* file using the ***VSM*** Viewlogic program. This new file can be used by the designer to verify the functionality and timing performance of the circuit under test. Overall, this project met its goal and proved to be successful in various test cases.

# BIBLIOGRAPHY

## BIBLIOGRAPHY


[Dougherty] Dale Dougherty, sed & awk. O'reilley & Associates, Inc., Sebastopol, CA, March 1991.

[Bouldin-1] Bouldin, Donald W., Computer-Aided Hierarchical Design of Digital VLSI SYSTEMS I. ECE 651 Class Notes, University of Tennessee, 1991

[Bouldin-2] Bouldin, Donald W., Computer-Aided Design of VLSI Systems II. ECE 652 Class Notes, University of Tennessee, 1992

[Cadence] Cadence Services Group, Training Manual. Analog Artist Simulation and Analysis. April 7, 1992.

[Chen] C. F. Chen, C-Y. Lo, H. N. Nham, and P. Subramaniam, The Second Generation MOTIS Mixed-Mode Simulator. Proceedings 21st Design Automation Conference, 10-17, June 1984.

[Huber ] J. P. Huber and M. W. Rosneck, Successful ASIC Design the First Time Through. New York, NY, Van Nostrand Reinhold.

[Anderson] G. Anderson and P. Anderson, The Unix C Shell Field Guide. Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[Thaik] R. W. Thaik and S-M. Kang, A New Router Using Zero-One Integer Linear Programming Techniques for Sea-Of-Gates and Custom Logic Arrays. IEEE Transactions on Computer-Aided Design. Vol 11. No. 12, December 1992.

[Terman] Terman, Christopher J. RSIM--A Logic-level Timing Simulator. Proceedings IEEE International Conference on Computer Design, 437-440, October 1983.

[Weste ] N. H. E. Weste and K. Eshraghian, Principles of CMOS VLSI Design. Addison-Wesley, 1985.

[Workview] Viewlogic Systems, Inc. Reference Manuals Volumes 1,2,3,4,5, and 6. Marlboro, MA : Viewlogic Systems, Inc., 1991.

# APPENDICES

# APPENDIX A

## CKTIM PROGRAM

```
#
# This program takes a MODEL name and proceeds to run the ViewLogic net-list
# which has been copied to MODELname.pin thru the various steps to get
# a final routed output in the user's root which will be an OCT
# database directory called MODELname... The argument is the MODELname.
# Also, it will extract the layout information necessary for CKTIM.
# "$designname" is your schematic file name depicted by the SIVMASTER attribute
# attached to the SMV symbol.
# Another feature is that it reports elapsed User, Real, Machine time it took
# CKTIM to run!
#
# Print the time at point of program starts .
time
```

***PreSim2Bat***

```
#
#
# The user is prompted to enter a designname(the SIVMASTER file name)
#
echo  "     ********************************************************"
echo  "     *                                                      *"
echo  "     *     Welcome to the University Of Tennessee           *"
echo  "     *          Back Annotation Delay Extractor!            *"
echo  "     *                  CKTIM                               *"
echo  "     *          (PreSim2Bat + Sim2Bat)                      *"
```

```
echo "      *                                                    *"
echo "      *              By: Tony E. Sawan                     *"
echo "      *              All Rights Reserved  (c)              *"
echo "      *                                                    *"
echo "      ******************************************************"
echo "                                                            "
echo "      ******************************************************"
echo -n "      Please Enter a Filename<Design Name>:"
set designname = $<
#
echo "      *.....PreSim2Bat Now Begins.....*"
echo "      *                                                    *"
echo "      *                                                    *"
#
#  vb2oct provides LagerIV users with the means to enter
#  schematics graphically using Viewlogic's Workview (described
#  below). After the schematics have been entered, executing
#  vb2oct will create a structure_master_view. When vb2oct
#  successfully completes its task, running DMoct(1) using the
#  "-s" and the "-l" flags will generate the final layout.
#
vb2oct $designname
#
DMoct -P $designname $designname
#
#
DMpost -sim $designname -w pwell -l 1.0
#
#
# generate project.nodes file which is not done using DMpost switch -sim;
# The user is prompted to run the extraction command and hit return to proceed.
```

```
# ext2irsim is an alias: ext2sim -B -R -t\! -t#
#
cd layout
#
echo  "      *.....PreSim2Bat Now Is Ending.....*"
echo  "      *                                  *"
echo  "      *                                  *"
```

**Sim2Bat**

```
#
echo " ************************************************************"
echo "      * Please type the following command in the "layout" directory  *"
echo "      * then press RETURN when done:                                 *"
echo " * ext2irsim $designname                                      *"
echo " ************************************************************"
echo " "
echo -n "      *** Please Press RETURN When Done:"
$<
#
#
cd ../
#
# Make directory for all the files needed by CKTIM
#
echo  "      *.....Sim2Bat Now Begins.....*"
echo  "      *                            *"
echo  "      *                            *"
time
```

```
#
mkdir $designname"backit"
#
# LDRC checks the design for violations, fanout/fanin errors, nets without input
# s/outputs, nets with multiple outputs, and partially connected components.  It
# executes the NETLIST program and the LOGDRC program.
# The NETLIST program uses the wir files as input and creates the following
#files:  project.net - Contains Net Equivalent Labels of the scematic
#                              used in the wir file.
#               project.pkg - Contains a list of each Gate and its ViewLogic
#                              Name
#               project.dvf - Contains IC/Gate, Pinorder, Pintype, & Function
#               project.fnc - Contains all the Components names
#
#  project.net is the file of interest since it will be used in our program
# The LOGDRC program creates a project.drc file which contains various error
# messages which SHOULD BE IGNORED by the user if his design CHECH resulted
in
# 0 errors.
#
        ldrc $designname
#
#
# InputFile:    project.nodes - this is generated by the irsim utility ext2irsim
# OutputFile:  project.magnod - Contains all the internal nodes used by MAGIC
# The user should use thes nodes(labels) of project.magnod in the "t" command of
# project.cmds file in order to generate the necessary delay values of EACH
# internal node of the layout.
#
awk '($2 !~ /[A-Za-z].*\/[1-9_][1-9_]*/)&&($2 !~ /Vdd/)&&($2 !~ /GND/)
{print $2}' layout/$designname.nodes >!
```

```
$designname"backit"/$designname.magnod
#
#
# delete the empty lines at begining of file project.magnod; these may create
# severe errors if found.
#
sed 's/  //g' $designname"backit"/$designname.magnod | sed '/^$/d' >!
$designname"backit"/$designname"magnod"
#
#
# Extract the Net Names used by ViewLogic Utilities; e.g. viewsim, viewdraw, etc
# InputFile:    project.net - this is generated by the Viewdraw utility LDRC
# OutputFile:  project.nettemp - contains all the internal nodes used by
#                                Viewsim
# The user should use thes nodes(labels) of project.nettemp in the "trace"
# command of project.cmd file in order to generate the necessary delay values of
# EACH internal node of the schematic.
#
awk '{ FS = "[;, ]"; RS = ""}{print $1 $2 $3 $4}' $designname.net | sed 's/;/ /;s/,//;
s/,//; s/\^/\//'g >!  $designname"backit"/$designname.nettemp
#
#
# Program:     FIVE - Assigns Kname to all non ViewLogic Components names
#                     and those Knames will become Magic names by assigning
#                     Vnames.
# InputFile:    project.nettemp - Contains ViewLogic Components Vs. Nets
# OutputFile:  project.nettemp1 - Contains ViewLogic Component names Vs. Magic
#                                 Net names.
awk -f five $designname"backit"/$designname.nettemp |sed 's/K\$/V/g' >!
$designname"backit"/$designname.nettemp1
#
```

```
#
#
sed 's/  //g' $designname"backit"/$designname.nettemp1 | sed '/^$/d' >!
$designname"backit"/$designname"nettemp2"
#
#
cp comp $designname"backit"/.
#
#
cd $designname"backit"
#
#
comp $designname"magnod" $designname"nettemp2" | awk '{print $1" " $2}' >!
labels.nodes
#
#
cd ../
#
#
awk '{print $2" " $1}' $designname"backit"/labels.nodes >!
$designname"backit"/nodes.labels
#
#
awk '($1 == "I")&&($3 !~ /fti/)&&($3 !~ /fto/)&&($3 !~ /smv/) {print $2"
"$4" "$5" "$6" "$7" "$8" "}' wir/$designname.1 >!
$designname"backit"/$designname.vlcompterm
#
#
trans $designname"backit"/nodes.labels -i
$designname"backit"/$designname.vlcompterm -o
$designname"backit"/$designname.magcompterm
```

```
#
#
echo " ************************************************************"
echo "        * Now, I am going to ask help you simulate your circuit! * "
echo "        * Look at the following example of examp1.cmds file; this      * "
echo "        * file is needed by IRSIM(Layout Simulator) to simulate the   * "
echo "        * transistor level of your design(e.g. examp1)                * "
echo " *                                                  * "
echo " ..... Press RETURN for more .....                  * "
$<
echo "        * logfile layout/examp1.lognet                               * "
echo "        * clock I1 0 1                                    * "
echo "        * stepsize 20ns                                   * "
echo "        * vector I1 I2 I3 I4                              * "
echo "        * V I1 0 1 0 1 0 1 0 1 0 1 0 1 0 1                * "
echo "        * V I2 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1           * "
echo "        * V I3 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1           * "
echo "        * V I4 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1           * "
echo "        * analyzer I1 I2 V1I1/O I4 O1 I3 O2                      * "
echo "        * t I1 I2 V1I1/O I4 O1 I3 O2                             * "
echo "        * R                                               * "
echo "        * logfile                                         * "
echo " ..... Press RETURN for more .....                  * "
$<
echo " *                                                  * "
echo " * 1- Notice that you need to log your test patterns simulation   * "
echo " *   results in a project.lognet file.                * "
echo " * 2- Choose one of your signals to be a system clock with the    * "
echo " *    "clock" label.                                  * "
echo " * 3- Choose a Step Size for your circuit cycle; It is Critical   * "
echo " *    that you include your cycle value in "threemod" program,     * "
```

```
echo " *    (cycle = 20) ONLY if it differs from 20ns                    * "
echo " * 4- Assign the Input Stimuli(names) to the "vector" attribute.  * "
echo " *    This includes All Inputs and Control signals including the   * "
echo " *    clock(e.g. I1)                                               * "
echo " ..... Press RETURN for more .....                                 * "
$<
echo " * 5- Assign your favorite and comprehensive test patterns to      * "
echo " *    each node/vector denoted by "V" name other than the clock    * "
echo " * 6- Include the nodes labels - the first column(field) of the     * "
echo " *    file nodes.labels -in both the "analyzer" and the trace "t"*"
echo " *    commands; you can include the nodes you wish to look at      * "
echo " *    graphically in the "analyzer" command but HAVE to include    * "
echo " *    all of them in the "t" command.                              * "
echo " * 7- Don't forget to end the file with "R"(repetitive mode) and   * "
echo " *    "logfile" to end the log file(examp1.lognet)                 * "
echo " *                                                                 *"
echo " * 8- An equivalent file of examp1.cmds is examp1.cmd; this file  *"
echo " *    is needed by Viewsim in order to check the simulation        * "
echo " *    results of the schematic.                                    * "
echo " *    Fortunately, both IRSIM and VIEWSIM have similar commands;   *"
echo " *    they have in common: STEPSIZE, CLOCK, RUN or "R",  also     * "
echo " *    TRACE or "T", WAVE is same as ANALYZER, PATTERN  is same as*"
echo " *    V, WATCH is same as VECTOR.                                  * "
echo " ..... Press RETURN for more .....                                 * "
$<
echo " *                                                                 * "
echo " * This is an example of a .cmd file:                             * "
echo " *                                                                 * "
echo " * wave examp1.wfm I1 I2 $1N21 I4 O1 I3 O2                        * "
echo " * stepsize 20ns                                                  * "
echo " * clock I1 1 0                                                  * "
```

```
echo " * pattern I2 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1                    * "
echo " * pattern I3 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1                    * "
echo " * pattern I4 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1                    * "
echo " * watch I1 I2 $1N21 I4 O1 I3 O2                                    * "
echo " * trace I1 I2 $1N21 I4 O1 I3 O2 > examp1.trc                       * "
echo " * run                                                              * "
echo " *                                                                  * "
echo " * Here, the logfile is examp1.trc and examp1.wfm denotes the       * "
echo " * analyzer option.                                                 * "
echo " ..... Press RETURN for more .....                                  * "
$<
echo " *                                                                  * "
echo " *                                                                  * "
echo " * SO, please create project.cmds and store it in the layout        * "
echo " * directory; then create project.cmd and store it in the design     * "
echo " * (top level directory).                                          * "
echo " * Next, Press RETURN to continue.                                  * "
echo " *                                                                 *"
echo " ***************************************************************"
echo -n "      PRESS RETURN PLEASE(AFTER YOU'VE CREATED THE
COMMAND FILES:"
$<
irsim scmos100.prm layout/$designname.sim -layout/$designname.cmds
#
#
grep '1 -> 0' layout/$designname.lognet >!
$designname"backit"/$designname"h2l".lognet
#
#
grep '0 -> 1' layout/$designname.lognet >!
$designname"backit"/$designname"l2h".lognet
```

```
#
#
awk '{print $5 $10}' $designname"backit"/$designname"h2l".lognet >!
$designname"backit"/$designname.hltemp
#
#
awk '{print $5 $10}' $designname"backit"/$designname"l2h".lognet >!
$designname"backit"/$designname.lhtemp
#
#
sed 'y/:/ /;s/ns//' $designname"backit"/$designname.hltemp >!
$designname"backit"/$designname.hltab
#
#
sed 'y/:/ /;s/ns//' $designname"backit"/$designname.lhtemp >!
$designname"backit"/$designname.lhtab
#
#
awk -f threemod $designname"backit"/$designname.hltab | sort >!
$designname"backit"/$designname.hltab1
#
#
awk -f threemod $designname"backit"/$designname.lhtab | sort >!
$designname"backit"/$designname.lhtab1
#
#
awk '{print $1" " $2"ns"}' $designname"backit"/$designname.hltab1 >!
$designname"backit"/$designname.hltab2
#
#
awk '{print $1" " $2"ns"}' $designname"backit"/$designname.lhtab1 >!
```

```
$designname"backit"/$designname.lhtab2
#
#
trans $designname"backit"/$designname.hltab2 -i
$designname"backit"/$designname.magcompterm -o
$designname"backit"/$designname.hltabdel
#
#
trans $designname"backit"/$designname.lhtab2 -i
$designname"backit"/$designname.magcompterm -o
$designname"backit"/$designname.lhtabdel
#
#
awk -f two $designname"backit"/$designname.hltabdel >!
$designname"backit"/$designname.hltabdel1
#
#
awk -f two $designname"backit"/$designname.lhtabdel >!
$designname"backit"/$designname.lhtabdel1
#
#
sed 's/ns//g' $designname"backit"/$designname.hltabdel1 | awk '{print $1" " $2"
" $3" " $4" " $5" " $6}' >! $designname"backit"/$designname.tphlval
#
#
sed 's/ns//g' $designname"backit"/$designname.lhtabdel1 | awk '{print $1" " $2"
" $3" " $4" " $5" " $6}' >! $designname"backit"/$designname.tplhval
#
#
#awk -f threemod $designname"backit"/$designname.hltabdel2 >!
#$designname"backit"/$designname.tphlval
```

```
#
#
#awk -f threemod $designname"backit"/$designname.lhtabdel2 >!
#$designname"backit"/$designname.tplhval
#
#
awk -f four $designname"backit"/$designname.tphlval >!
$designname"backit"/$designname.tphltemp
#
#
awk -f four $designname"backit"/$designname.tplhval >!
$designname"backit"/$designname.tplhtemp
#
#
awk '{print $1" TPHL=" $2"NS,"}' $designname"backit"/$designname.tphltemp
>! $designname"backit"/$designname.tphltemp1
#
#
awk '{print $1" TPLH=" $2"NS;"}' $designname"backit"/$designname.tplhtemp
>! $designname"backit"/$designname.tplhtemp1
#
#
cat $designname"backit"/$designname.tphltemp1
$designname"backit"/$designname.tplhtemp1 >!
$designname"backit"/$designname.tphlh
#
#
sort $designname"backit"/$designname.tphlh >!
$designname"backit"/$designname.tphlh1
#
#
```

```
cat twolines $designname"backit"/$designname.tphlh1 >!
$designname"backit"/$designname.tphlh2
#
#
awk '{FS =",\n"; RS =";\n"}{print $1 $2" " $4}'
$designname"backit"/$designname.tphlh2 >!
$designname"backit"/$designname.tphlh3
#
#
awk '$1 ~/\$[A-Z1-9].*/ {print $1" " $2" " $4" "}'
$designname"backit"/$designname.tphlh3 >!
$designname"backit"/$designname.tphlh4
#
#
awk '{print "C "$1"\n" "A "$2"\n" "A "$3}'
$designname"backit"/$designname.tphlh4 >!
$designname"backit"/$designname.tphlh5
#
#
cat heads $designname"backit"/$designname.tphlh5 tails >!
$designname"backit"/$designname"-new".bat
nawk -f expand.nawk $designname"backit"/$designname"-new".bat >!
$designname"backit" /$designname"-new".bat-exp
 nawk -f tphl.nawk $designname"backit"/$designname"-new".bat-exp >!
$designname"backit"/$designname"-new".bat
#
#
vsm $designname -f $designname"-new" -d $designname"backit"/$designname"-
new".bat
#
echo "       *.....Sim2Bat Now IS Ending.....*"
```

```
echo  "      *                                                    * "
echo  "      *                                                    * "
#
echo
"*****************************************************************"
echo "* Generation of Back-Annotated Circuit Delay Times Is DONE!      *"
echo "* You Can Now Resimulate Your Circuit Using Your $designname.cmd *"
echo "* file and the $designname-new.vsm ...                          *"
echo "*                                                               *"
echo "*                THANKS FOR USING  CKTIM(c) !!                  *"
echo  "*                              BYE FOR NOW !!                   *"
echo
"*****************************************************************"
time
#
#
#
```

## APPENDIX B

## View2Sim PROGRAM

```
echo "  ********************************************************"
echo "  Welcome to View2Sim: viewsim to IRSIM Pass Evaluation  "
echo "                                                         "
echo -n " Before running this program, simulate the project in Viewsim using "
echo -n " the back-annotated  project.vsm file!                "
echo "                                                         "
time
#
echo -n "  Please Enter a Filename<Design Name>:"
set designname = $<
#
mkdir $designname"pass1"
#
cp $designname.trc $designname"pass1"/.
#
#
grep '1 -> 0' $designname"pass1"/$designname.trc >!
$designname"pass1"/$designname"h2l".trc
#
#
grep '0 -> 1' $designname"pass1"/$designname.trc >!
$designname"pass1"/$designname"l2h".trc
#
#
awk '{print $4" " $3" "}' $designname"pass1"/$designname"h2l".trc | sed 's/]//g'
>! $designname"pass1"/$designname"h2l".trc1
```

```
#
#
awk '{print $4" " $3" "}' $designname"pass1"/$designname"l2h".trc | sed 's/]//g'
>! $designname"pass1"/$designname"l2h".trc1
#
#
#awk '{print $2" " $1}' $designname"backit"/nodes.labels >!
#$designname"pass1"/labels.nodes
#
#
trans $designname"backit"/nodes.labels -i
$designname"pass1"/$designname"h2l".trc1 -o
$designname"pass1"/$designname"h2l".trcomp
#
#
trans $designname"backit"/nodes.labels -i
$designname"pass1"/$designname"l2h".trc1 -o
$designname"pass1"/$designname"l2h".trcomp
#
#
sort $designname"pass1"/$designname"h2l".trcomp >!
$designname"pass1"/$designname"h2l".trcomp1
#
#
sort $designname"pass1"/$designname"l2h".trcomp >!
$designname"pass1"/$designname"l2h".trcomp1
#
#
sed 's/ns//' $designname"pass1"/$designname"h2l".trcomp1 >!
$designname"pass1"/$designname.trchltab
#
```

```
#
sed 's/ns//' $designname"pass1"/$designname"l2h".trcomp1 >!
$designname"pass1"/$designname.trclhtab
#
#
awk -f threemod $designname"pass1"/$designname.trchltab | sort +0 -1 +1n -2n |
uniq >! $designname"pass1"/$designname.trchltab1
#
#
awk -f threemod $designname"pass1"/$designname.trclhtab | sort +0 -1 +1n -2n |
uniq >! $designname"pass1"/$designname.trclhtab1
#
#
awk -f six $designname"pass1"/$designname.trchltab1 >!
$designname"pass1"/$designname.trchltab2
#
#
awk -f six $designname"pass1"/$designname.trclhtab1 >!
$designname"pass1"/$designname.trclhtab2
#
#
trans $designname"pass1"/$designname.trchltab2 -i
$designname"backit"/$designname.magcompterm -o
$designname"pass1"/$designname.trchltabdel
#
#
trans $designname"pass1"/$designname.trclhtab2 -i
$designname"backit"/$designname.magcompterm -o
$designname"pass1"/$designname.trclhtabdel
#
#
```

```
awk -f two $designname"pass1"/$designname.trchltabdel >!
$designname"pass1"/$designname.trchltabdel1
#
#
awk -f two $designname"pass1"/$designname.trclhtabdel >!
$designname"pass1"/$designname.trclhtabdel1
#
#
sed 's/ns//g' $designname"pass1"/$designname.trchltabdel1 | awk '{print $1" " $2"
" $3" " $4" " $5" " $6}' >! $designname"pass1"/$designname.trctphlval
#
#
sed 's/ns//g' $designname"pass1"/$designname.trclhtabdel1 | awk '{print $1" " $2"
" $3" " $4" " $5" " $6}' >! $designname"pass1"/$designname.trctplhval
#
#
awk -f four $designname"pass1"/$designname.trctphlval >!
$designname"pass1"/$designname.trctphltemp
#
#
awk -f four $designname"pass1"/$designname.trctplhval >!
$designname"pass1"/$designname.trctplhtemp
#
#
awk '{print $1" TPHL=" $2"NS,"}' $designname"pass1"/$designname.trctphltemp
>! $designname"pass1"/$designname.trctphltemp1
#
#
awk '{print $1" TPLH=" $2"NS;"}' $designname"pass1"/$designname.trctplhtemp
>! $designname"pass1"/$designname.trctplhtemp1
#
```

```
#
cat $designname"pass1"/$designname.trctphltemp1
$designname"pass1"/$designname.trctplhtemp1 >!
$designname"pass1"/$designname.trctphlh
#
#
sort $designname"pass1"/$designname.trctphlh >!
$designname"pass1"/$designname.trctphlh1
#
#
cat twolines $designname"pass1"/$designname.trctphlh1 >!
$designname"pass1"/$designname.trctphlh2
#
#
awk '{FS =",\n"; RS =";\n"}{print $1 $2" " $4}'
$designname"pass1"/$designname.trctphlh2 >!
$designname"pass1"/$designname.trctphlh3
#
#
awk '$1 ~/\$[A-Z1-9].*/ {print $1" " $2" " $4" "}'
$designname"pass1"/$designname.trctphlh3 >!
$designname"pass1"/$designname.trctphlh4
#
#
awk '{print "C "$1"\n" "A "$2"\n" "A "$3}'
$designname"pass1"/$designname.trctphlh4 >!
$designname"pass1"/$designname.trctphlh5
#
#
cat heads $designname"pass1"/$designname.trctphlh5 tails >!
$designname"pass1"/$designname"pass1".bat
```

```
nawk -f expand.nawk $designname"pass1"/$designname"pass1".bat >!
$designname"pass1"/$designname"pass1".bat-exp
 nawk -f tphl.nawk $designname"pass1"/$designname"pass1".bat-exp >!
$designname"pass1"/$designname"pass1".bat
#
#
#
echo
"*****************************************************************"
echo  "* Generation of FIRST PASS Back-Annotation File ".bat" DONE!      *"
echo  "* You Can Now Resimulate The Circuit Using Your $designname.cmd   *"
echo  "* File and The $designname-new.vsm File, Using Bat2Sim(c)     *"
echo  "*                                                               *"
echo  "*      THANK YOU FOR USING  View2Sim  (c) !!                    *"
echo  "*            BYE FOR NOW !!                                     *"
echo
"*****************************************************************"
echo  "                                                               "
time
#
#
#
```

## APPENDIX C

## Bat2Sim PROGRAM

```
echo
"*****************************************************************"
echo   " Welcome to Bat2Sim: Back-Annotated Delays From viewsim to IRSIM   "
echo   " Pass Evaluation                                                   "
echo   "                                                                  "
echo -n " Before running this program, simulate the project in Viewsim using"
echo -n " the back-annotated  Project-new.vsm file!                        "
echo   "                                                                  "
time
#
echo -n "       Please Enter a Filename<Design Name>:"
set designname = $<
#
mkdir $designname"pass2"
#
cp $designname.trc $designname"pass2"/.
#
#
grep '1 -> 0' $designname"pass2"/$designname.trc >!
$designname"pass2"/$designname"h2l".trc
#
#
grep '0 -> 1' $designname"pass2"/$designname.trc >!
$designname"pass2"/$designname"l2h".trc
#
#
```

```
awk '{print $4" " $3" "}' $designname"pass2"/$designname"h2l".trc | sed 's/]//g'
>! $designname"pass2"/$designname"h2l".trc1
#
#
awk '{print $4" " $3" "}' $designname"pass2"/$designname"l2h".trc | sed 's/]//g'
>! $designname"pass2"/$designname"l2h".trc1
#
#
#awk '{print $2" " $1}' $designname"backit"/nodes.labels >!
#$designname"pass2"/labels.nodes
#
#
trans $designname"backit"/nodes.labels -i
$designname"pass2"/$designname"h2l".trc1 -o
$designname"pass2"/$designname"h2l".trcomp
#
#
trans $designname"backit"/nodes.labels -i
$designname"pass2"/$designname"l2h".trc1 -o
$designname"pass2"/$designname"l2h".trcomp
#
#
sort $designname"pass2"/$designname"h2l".trcomp >!
$designname"pass2"/$designname"h2l".trcomp1
#
#
sort $designname"pass2"/$designname"l2h".trcomp >!
$designname"pass2"/$designname"l2h".trcomp1
#
#
sed 's/ns//' $designname"pass2"/$designname"h2l".trcomp1 >!
```

```
$designname"pass2"/$designname.trchltab
#
#
sed 's/ns//' $designname"pass2"/$designname"l2h".trcomp1 >!
$designname"pass2"/$designname.trclhtab
#
#
awk -f threemod $designname"pass2"/$designname.trchltab | sort +0 -1 +1n -2n |
uniq >! $designname"pass2"/$designname.trchltab1
#
#
awk -f threemod $designname"pass2"/$designname.trclhtab | sort +0 -1 +1n -2n |
uniq >! $designname"pass2"/$designname.trclhtab1
#
#
awk -f six $designname"pass2"/$designname.trchltab1 >!
$designname"pass2"/$designname.trchltab2
#
#
awk -f six $designname"pass2"/$designname.trclhtab1 >!
$designname"pass2"/$designname.trclhtab2
#
#
trans $designname"pass2"/$designname.trchltab2 -i
$designname"backit"/$designname.magcompterm -o
$designname"pass2"/$designname.trchltabdel
#
#
trans $designname"pass2"/$designname.trclhtab2 -i
$designname"backit"/$designname.magcompterm -o
$designname"pass2"/$designname.trclhtabdel
```

```
#
#
awk -f two $designname"pass2"/$designname.trchltabdel >!
$designname"pass2"/$designname.trchltabdel1
#
#
awk -f two $designname"pass2"/$designname.trclhtabdel >!
$designname"pass2"/$designname.trclhtabdel1
#
#
sed 's/ns//g' $designname"pass2"/$designname.trchltabdel1 | awk '{print $1" " $2"
" $3" " $4" " $5" " $6}' >! $designname"pass2"/$designname.trctphlval
#
#
sed 's/ns//g' $designname"pass2"/$designname.trclhtabdel1 | awk '{print $1" " $2"
" $3" " $4" " $5" " $6}' >! $designname"pass2"/$designname.trctplhval
#
#
awk -f four $designname"pass2"/$designname.trctphlval >!
$designname"pass2"/$designname.trctphltemp
#
#
awk -f four $designname"pass2"/$designname.trctplhval >!
$designname"pass2"/$designname.trctplhtemp
#
#
awk '{print $1" TPHL=" $2"NS,"}' $designname"pass2"/$designname.trctphltemp
>! $designname"pass2"/$designname.trctphltemp1
#
#
awk '{print $1" TPLH=" $2"NS;"}' $designname"pass2"/$designname.trctplhtemp
```

```
>! $designname"pass2"/$designname.trctplhtemp1
#
#
cat $designname"pass2"/$designname.trctphltemp1
$designname"pass2"/$designname.trctplhtemp1 >!
$designname"pass2"/$designname.trctphlh
#
#
sort $designname"pass2"/$designname.trctphlh >!
$designname"pass2"/$designname.trctphlh1
#
#
cat twolines $designname"pass2"/$designname.trctphlh1 >!
$designname"pass2"/$designname.trctphlh2
#
#
awk '{FS =",\n"; RS =";\n"}{print $1 $2" " $4}'
$designname"pass2"/$designname.trctphlh2 >!
$designname"pass2"/$designname.trctphlh3
#
#
awk '$1 ~/\$[A-Z1-9].*/ {print $1" " $2" " $4" "}'
$designname"pass2"/$designname.trctphlh3 >!
$designname"pass2"/$designname.trctphlh4
#
#
awk '{print "C "$1"\n" "A "$2"\n" "A "$3}'
$designname"pass2"/$designname.trctphlh4 >!
$designname"pass2"/$designname.trctphlh5
#
#
```

```
cat heads $designname"pass2"/$designname.trctphlh5 tails >!
$designname"pass2"/$designname"pass2".bat
nawk -f expand.nawk $designname"pass2"/$designname"pass2".bat >!
$designname"pass2"/$designname"pass2".bat-exp
 nawk -f tphl.nawk $designname"pass2"/$designname"pass2".bat-exp >!
$designname"pass2"/$designname"pass2".bat
#
#
#
echo   "*****************************************************************"
echo "* Generation of SECOND PASS Back-Annotation File ".bat" DONE!   *"
echo "* You Can Now Compare The Results of View2Sim vs. Sim2Bat       *"
echo "* Vs. Bat2Sim...                                        *"
echo "*                                                       *"
echo "*        THANK YOU FOR USING  Bat2Sim  (c) !!           *"
echo "*                BYE FOR NOW !!                         *"
echo   "*****************************************************************"
echo "                                                                "
#
time
#
```

# APPENDIX D

## FIX PROGRAM

*EXPAND.NAWK*

```
BEGIN{str="";flag=0}
{
        if(substr($1,1,1) == ".")
        {
                flag = flag + 1
                if(flag=2)
                {
                print str
                }
           print $0
        }
        else if( $1 == "C" )
            {
             print str
             str = $0
         }
        else
       {
         gsub("=","  ")
         gsub("NS","  NS")
          str = str "  " $0
         }
}
```

*TPHL.NAWK*

```
BEGIN{}
        { if(substr($1,1,1) == ".")
                { print $0 }
        else    if($1 == "C")
         {      if( $5 < 0 && $9 < 0 && substr($1,1,1) != ".")
        {       print  $1"  "$2"\n"$3"  "$4"="substr($5,2)"NS\n"$7"
"$8"="substr($9,2)"NS"}
        else if ( $5 < 0 && $9 > 0 && substr($1,1,1) != ".")
         {      print  $1"  "$2"\n"$3"  "$4"="$9"NS\n"$7"  "$8"="$9"NS"   }
        else if ( $5 > 0 && $9 < 0 && substr($1,1,1) != ".")
      {          print  $1"  "$2"\n"$3"  "$4"="$5"NS\n"$7"  "$8"="$5"NS"  }
        else if ( $5 == 0 && $9 > 0 && substr($1,1,1) != ".")
      {          print  $1"  "$2"\n"$3"  "$4"="$9"NS\n"$7"  "$8"="$9"NS"  }
        else if ( $9 == 0 && $5 > 0 && substr($1,1,1) != ".")
      {         print  $1"  "$2"\n"$3"  "$4"="$5"NS\n"$7"  "$8"="$5"NS"   }
        else if (substr($1,1,1) != ".")
      {    print $1"  "$2"\n"$3"  "$4"="$5"  NS\n"$7"  "$8"="$9"  NS"        }
    else
              print   "***************************************"
           print "WARNING!! ZERO ERRORS RESULTED..... PLEASE CHECK"
            print "YOUR CIRCUIT TO PROVIDE "TOTAL!" COVERAGE OF"
           print " TEST PATTERNS"
                    print"***************************************"
        {print $0   }
}
```

## APPENDIX E

## COMP.C PROGRAM

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


void main(int argc, char **argv)
{
  FILE *f1, *f2;
  char *file1[512], *file2[512], temp[512], inLine[512];
  char *tmpToken, *keyToken;
  int count1, count2, count3, i, j;
  count1 = count2 = count3 = 0;
  if(argc < 3){
    printf("use: %s file1 file2\n", argv[0]);
    exit(-1);
  }


  if(!(f1 = fopen(argv[1], "r"))){
    printf("Error opening file: %s\n", argv[1]);
    exit(-1);
  }

  if(!(f2 = fopen(argv[2], "r"))){
    printf("Error opening file: %s\n", argv[2]);
    fclose(f1);
    exit(-1);
  }

  while(fgets(inLine, 512, f1)){
    file1[count1] = (char *)malloc(strlen(inLine) + 5);
    strcpy(file1[count1], inLine);
    file1[count1][strlen(inLine)-1] = NULL;
    count1 = count1+1;
  };

  while(fgets(inLine, 512, f2)){
    file2[count2] = (char *)malloc(strlen(inLine) + 5);
    strcpy(file2[count2], inLine);
    file2[count2][strlen(inLine)-1] = NULL;
```

```
  count2 = count2+1;
 };

 for(i = 0; i < count1; i++){
  for(j = 0; j < count2; j++){
    strcpy(temp, file2[j]);
    tmpToken = strtok(temp,  "\n \t\0");
    do{
       if(!strcmp(tmpToken, file1[i])){
         tmpToken = strtok(temp, "\n \t\0");
         printf("%s\t\t%s\n", file1[i], tmpToken);
         break;
       }
       tmpToken = strtok(NULL, "\t \n\0");
    } while(tmpToken != NULL);
  }
 }
 fclose(f1); fclose(f2);
}
```

# VITA

Tony Emile Josephine Sawan was born in Kaferhabou/Tripoli, Lebanon on May 10, 1968. He attended the private schools of the Italian Carmelite Fathers. After graduating with first place from "Ecole Des Peres Carmes" High School in 1986 he set his goals to venture in the new land of the United States of America. He attended a special English program at Maryville College under Professor Franklin and finished the complete 8 levels in 2 months. In Spring 1987, he entered the University of Tennessee at Knoxville for his undergraduate studies and acquired a Bachelor of Science degree with honor in Electrical Engineering in August 1990. During the same month, he rejoined the University of Tennessee for the graduate program in Electrical Engineering. He was awarded a Masters of Science degree in Electrical Engineering in December of 1993.

During his graduate years, he served as a teaching assistant for the Electrical Engineering department mostly under Dr. Robert R. Bodenheimer. He developed course materials and lab experiments, in addition to assisting in developing VLSI tools workshops under Dr. Donald W. Bouldin. His educational and technical background allowed him an opportunity of joining the Paging Group of Motorola Inc. He is currently working in the VLSI IC design area developing communication chips.

The author's current mailing address is 1720 E. Woodward St. #104. Austin, Texas 78741.

This thesis was typed by the author using the WordPerfect, Microsoft Works, and Macdraw software available for Macintosh computers.