1

NEC MICRON

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**A Study of Common Logic Design Errors
and Methods for their Detection**

by

GORDON F. MEIN, B.A.Sc.

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Engineering.**

**Ottawa-Carleton Institute for Electrical Engineering
Department of Electronics
Faculty of Engineering
Carleton University
Ottawa, Ontario
April 1988**

**© copyright
1988, Gordon F. Mein**

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-54395-7

The undersigned recommend to the faculty of Graduate Studies
and Research acceptance of the thesis:

"A Study of Common Logic Design Errors
and Methods for their Detection"

submitted by Gordon F. Mein in partial fulfillment of the
requirements for the degree of Master of Engineering.

Thesis Supervisor

Chairman, Dept. of Electronics

Carleton University

April 1988

Abstract

This thesis provides a study of errors made in VLSI logic designs, many of which are not revealed by present day logic simulation or timing verification. Manual circuit audits are one way to detect them, but are time consuming, and prone to human error. Automating this process will greatly improve the audit efficiency.

As a step in this direction, a preliminary rule set for detecting these errors has been developed. It is also necessary, in performing an evaluation, for the "protocol" of signal timing to be specified. A method of describing timing protocol (ATDL) has been developed. It is not only useful for specifying I/O timing, but can also be used by the evaluator to represent the node protocols, which must be derived from the input information. An evaluator program will be more efficient at error detection than simulation, and complements the simulation process.

Acknowledgements

I would like to take this opportunity to thank Dr. M. Copeland and Dr. J. Knight for their feedback and guidance which were invaluable in the preparation of this document. I would also like to acknowledge the contribution of Bell-Northern Research (BNR) which provided me with the tools necessary to do the thesis work, and the opportunity to work with and learn from experts in the area of circuit design. The design experience gained with BNR provided me with an understanding of design problems which was essential in the development of this thesis.

TABLE OF CONTENTS

1.0	INTRODUCTION	1
2.0	LOGIC DESIGN PROBLEM CLASSIFICATION	13
2.1	Circuit Logically Incorrect	16
2.2	Clock or Signal Logically Inverted	19
2.3	Hazards and Race Conditions	22
2.3.1	Gating of two signals (Hazards)	23
2.3.2	Race Conditions	28
2.4	Set-up and Hold Violations	30
2.5	Unconditionally dangerous/inferior circuits	33
2.6	Asynchronous Event Problems	33
2.6.1	Asynchronous/Synchronous Interfaces	34
2.7	Minimum Pulse Width Violations	36
2.7.1	Asynchronous Self-modifying Outputs	37
2.8	Fan-in/Fan-out Violations	40
2.9	Conditionally Dangerous Circuits	43
2.9.1	Lock-up and Invalid States	44
2.9.2	Invalid Transition States	47
2.10	Analog Mode of Operation	49
3.0	TECHNIQUES FOR DETECTING LOGIC DESIGN PROBLEMS	53
3.1	Categories of Errors	57
3.2	Detection Methods	58
3.2.1	Tcritical	60
3.2.2	Input Rules	62
3.2.2.1	Rules for All Inputs	62
3.2.2.2	Rules for Clock Inputs	64
3.2.2.3	Rules for Clocked Inputs	64
3.2.2.4	Rules for Jam Inputs	68
3.2.3	Output Rules	69
3.2.3.1	Rules for All Outputs	69
3.2.3.2	Rules for Tri-state Outputs	73
3.2.3.3	Rules for External Outputs	74
3.2.3.4	Rules for Output Timing Protocol	77
3.2.4	Cell Rules	80
3.2.4.1	Rules for All Cells	80
3.2.4.2	Rules for Lock-up	81
4.0	RULE BASED EVALUATION	86
4.1	Basic Evaluator Requirements	87
4.1.1	The NETLIST	89
4.1.2	The FANOUT List	91
4.1.3	Pathfinding	92
4.1.3.1	Entry Nodes	94
4.1.3.2	Next Nodes	95
4.1.3.3	Next Cell I/O Points	96
4.1.4	The Scheduler	96
4.1.5	Circuit Initialization	97

4.2	Evaluation Example #1	98
4.3	Evaluation Example #2	105
5.0	TIMING REPRESENTATIONS	113
5.1	A Timing Description Language	113
5.1.1	ATDL Syntax	115
5.1.2	Using the ATDL Syntax	131
6.0	DISCUSSION AND CONCLUSIONS	135
6.1	Program Considerations	135
6.2	Conclusions	139
6.3	Recommendations	141
	APPENDIX A - ATDL Short Form Summary	144
	GLOSSARY OF TERMS	146
	REFERENCES	149

LIST OF ILLUSTRATIONS

Figure 1.	CHIPBUGS Database Analysis Results	15
Figure 2.	Probable Clock Inversion Example	20
Figure 3.	Probable Signal Inversion Example	21
Figure 4.	Circuit with Clock and Signal Gated	24
Figure 5.	Gating of Clock and Signal (Alternative #1)	26
Figure 6.	Gating of Clock and Signal (Alternative #2)	27
Figure 7.	Circuit with a Clock/Reset Race	29
Figure 8.	Circuit with Set-up/Hold Violation	31
Figure 9.	Asynchronous/Synchronous Interface Circuit	35
Figure 10.	Asynchronous Self-modifying Outputs (Cct #1)	39
Figure 11.	Asynchronous Self-modifying Outputs (Cct #2)	41
Figure 12.	Alternative to Self-Modifying (Cct #2)	42
Figure 13.	Circuit with Lock-up State	45
Figure 14.	Invalid Transition States: Circuit #1	50
Figure 15.	Invalid Transition States: Circuit #2	51
Figure 16.	Circuit with Indeterminate Switching	52
Figure 17.	Example Circuit #1 for Evaluation	100
Figure 18.	Example Circuit #2 for Evaluation	109
Figure 19.	Timing Example #1	124
Figure 20.	Timing Example #2	126
Figure 21.	Timing Example #3	128
Figure 22.	Timing Example #4	129
Figure 23.	Timing Example #5	130
Figure 24.	Example Circuit #1 with ATDL Timing	133

1.0 INTRODUCTION

This thesis investigates the design of an automated logic circuit evaluation system. If implemented, this system would be able to perform audits of logic circuits by applying many of the criteria that an expert would. It will be shown that many errors made in VLSI designs can be detected by an evaluator, but may not be detected by simulation. This is not to say that they cannot be detected by simulation, but that due to human error, inadequate circuit modelling, or insufficient testing, the error may not be revealed. The evaluator, therefore, is seen as being complementary to the process of simulation. [BapS86]

As transistor geometries shrink in response to the demand for greater levels of integration, the design complexity (circuit size) has increased dramatically. "As the circuit size increases, the complexity of evaluation increases non

linearly." [MokV85] Reductions in the allowed design cycle time, and a greater sensitivity to errors are further increasing demands on design tools. Pressure to provide high quality, quick-to-market, VLSI designs means that one cannot afford to have many errors on the "first-pass." Errors can hide other errors -- "the onionskin effect" -- and lead to costly re-cycles and loss of time. "For companies competing in the general microcomputer and memory chip market, competitive pressures require the continual development of chips that are both lower in cost and higher in performance. Instead of backing away from the problem by designing non-optimized circuits (an uncompetitive approach for the general chip market), new tools are needed to verify and optimize VLSI circuit performance." [MarJ84] In short, integrated circuit designs are now required to be much more complex than in the past, with disproportionately fewer errors, and shorter design cycle times. In order to meet these demands, circuits must be designed much more efficiently than in the past. Computer aided design (CAD) tools must evolve to higher levels of sophistication to bring this about.

In addressing the general problem of reducing the number of first-pass design errors in logic circuits, it was recognized that these errors tend to fall into three categories: specification errors, logic errors, and errors in the appli-

cation of the chosen technology. The discussion in this thesis is limited to the latter two. Specification errors require different kinds of solutions than the other two because they occur before the design phase.

"Logic Simulation programs have long been one of the standard tools for design verification of logic networks." [ChaE85] It is, however, generally recognized that the capabilities of simulators are limited when it comes to timing constraint checking. Simulators are "pattern dependent; the number of possible input patterns" (to fully check a circuit) "grows exponentially with the complexity of the circuit." [JouN83] Because of these simulation limitations, a CAD area has developed called "Timing Verification."

Timing verifiers are aimed at determining critical timing paths, set-up and hold violations, and in some cases, the minimum possible clock period and duty cycle. [ChaE85, MurM85, JouN83, NgP81, HitR82] However, the bulk of the work in this area has been focussed on the problems associated with developing computers, and microprocessors, which tend to be driven by performance (speed) requirements. [MonM82, MarJ84, HitR82, MarF82] In these types of designs, faster is better. As a result, companies have tended to use their most advanced (smallest geometry) processes and have pushed the designs to the process limit to achieve a competitive advan-

tage. This has resulted in a greater need for accurate circuit modelling, and for analysis tools capable of identifying circuit areas limiting performance. This is primarily what has dictated the direction of timing verification tools over the past ten years. Not surprisingly, these tools tend to be quite specialized, optimized for synchronous circuits only [DagM86], for a particular technology [JouN83, WeiD87], or for more specialized applications with only combinational logic [BraD86]. Such timing verification tools, though varied in implementation, are all aimed at "performance circuits," those which are designed to run as fast as the process will permit.

But many ASIC (Application Specific Integrated Circuit) designs do not require the speed but the density that integration offers. It is common for the emphasis in such designs to be on minimum implementation time (i.e. first-pass success), with cost being a secondary although still important consideration. Much of the gate array and standard cell business caters to such projects. In these types of designs, there is often much less structure to the circuits being developed than in the case of computer circuit designs. This tends to result in large amounts of simulation and verification data, as compared to more structured circuits, and this must be assessed manually. Since essentially the same tools are available to the standard cell chip de-

signer as to the rest of the industry, most circuit analyses tend to rely upon simulation and timing verification as described above.

Despite these checks, many design errors go undetected until the evaluation (debugging) stage. One reason for this is that simulators only provide the answers to questions that are asked. This limits the circuit analysis to those conditions that the designer conceived of and checked for. "This means that someone, often the logic designer, must write test cases that are intended to expose all timing problems. But this assumes some a priori knowledge of where the timing problems are -- an assumption that is very tenuous." [BenL82] Although circuit simulators are the primary tools with which to verify circuit operation, the design "guru" frequently reveals design flaws which are of a fundamental nature. For this reason, manual design audits are performed, and result in fewer errors. Despite their value, manual design audits are still subject to human error, and so cannot be guaranteed to consistently uncover errors.

More recently, the needs of the general ASIC design community have been recognized, and timing verification tools have been developed which are more general in application. Technology independence [HeiS87], and consideration of the logical relationships between signals [WeiD87] are two exam-

ples of the developments in this direction. Nevertheless, timing verification tools still only examine those areas mentioned previously: critical paths, setup and hold violations, and minimum clock period and duty cycle. Even with the combination of simulation and timing verification, many errors in design are missed.

Logic simulators and timing verifiers can only be as accurate as the circuit models backing them up. Today, industry-wide attention is being given to improving model timing accuracy. However, the focus has been on modelling correct operation accurately. Non-existent, incorrect, or poor modelling of faulty conditions continues to result in design errors being missed. In addition, a number of design error types are often not discovered in either simulation or timing verification. One example of this is logical signal inversion. This type of error is common, and is often not detected prior to circuit prototyping. Even when problems are flagged during analysis, there is often so much good data, or potential-hazard reports, surrounding the one real problem that it is missed due to human error.

Another typical simulation and verification problem occurs in circuits with inputs asynchronous to the master clock. In such cases, it is not uncommon for a certain alignment of events to create error conditions. While it is true that

such problems can be confirmed with logic simulators, it is usually not practical to perform the exhaustive simulations required to uncover them in the first place. If not picked up during design, such errors may go undetected for a very long time during debug, since preliminary evaluations tend to concentrate on verifying correct operation rather than failure modes. None of the timing verifiers reviewed deal with the problem of asynchronous circuit evaluation.

The preceding examples are currently very expensive to detect prior to the prototyping of the devices. They are expensive both in computing time and manpower, and can lead to other problems that are very costly to correct. Presently, the only way to detect some of these problems is through an audit, but this process is also subject to human error. Even though the consulting designer may ask the right question, there is no guarantee that he/she will correctly evaluate the circuit and identify the problem. This is particularly true on larger circuits where the audit process can be very tedious. A computer program developed to perform the same type of analysis would eliminate the human error aspect of the process, and greatly enhance the effectiveness of the evaluation.

Static timing verifiers provide a limited amount of circuit analysis, and require no timing input. Circuit simulators

can calculate the response to virtually any stimulus, but require precise timing definitions, and extensive sets of vectors to get good circuit coverage. Between the timing verifier and the simulator is a gap that can be filled by the evaluator, which requires only a general description of the timing, and can provide extensive circuit analysis.

Before developing such a system, it is first necessary to understand what types of errors are commonly made in design. To this end, a study of a corporate database capturing chip-errors, BNR CHIPBUGS, was conducted. [MeiG87] The results of the study, given in chapter 2, show that 46% of the chip design errors resulted due to incorrect logic being implemented, while the remainder were attributable to violations of the circuit timing parameters, or inappropriate use of circuit elements. Only 10% of the problems could be attributed to misinterpreting or omitting a specified function. An additional 5% were attributed to modelling inadequacies and file manipulation errors. These problems aside, the results suggest that 85% of the errors made were detectable.

Having established a need for automatic checking of common design errors, and having identified the types of errors that can be detected, chapter 3 deals with techniques for their detection. The approach taken is to develop generic checks that will detect symptoms rather than attempt to

check for each of the error-types individually. The checks take the form of rules, and are broken down into rules for circuit inputs, outputs, and for the cells themselves. A new concept, called Tcritical, is introduced, which is key to detecting many problems. This approach to circuit analysis is not rigorous, but does tend to be technology independent. Rules are based on common practice using a conservative design approach, so circuits that are flagged may not be in error, strictly speaking, but they also may not be what the designer intended. Flagging suspect circuits, so long as the reports do not become overwhelming, results in identifying both circuits that are unquestionably in error, and also those that are worthy of a second look by the designer. Often designs are implemented that were not what was intended, simply because they happen to pass all simulations. Such designs can lead to more subtle problems later.

Just as a designer cannot perform a circuit audit without knowing the general timing for the circuit, an automated system to fulfill this function must also have access to some timing information. This has been recognized by others in the development of better timing verification tools. [BraD86, WeiD87] However, the level of timing information considered has been limited to identifying mutually exclusive signals, to reduce the path calculations, and to bounding clock waveforms, e.g. minimum and maximum rise and fall

times. This information, while valuable, is not sufficient to do a circuit evaluation. What is required is a method of specifying the format of the expected timing without having to specify all possible sets of events. One need only examine the timing diagrams generated during a circuit audit to understand how much timing information is really needed. Two features of these diagrams are that, unlike event driven simulation, the accuracy of the timing is not dependent on a sampling interval, and it is possible to capture the relationships between signals, and between parts of a signal, the signal protocol. A method for specifying this information has been developed, and is presented.

The first timing diagram attribute, accuracy, only becomes an issue in simulations that are long compared to the sampling interval. It is desirable to make this interval as short as possible to capture narrow events (glitches). However, if the major features of the timing occur over a relatively long time, a large number of samples are required. Manual audits, on the other hand, show timing on a scale to reveal its major features, and use expanded scales to consider specific areas of interest.

The second item, protocols, represents a fundamental difference between timing as it is specified for an audit versus a simulation. Simulators are concerned with timing events,

specifically, logic state transitions (1 to 0, 0 to 1, 0 to X, etc.). So in simulation, all circuit timing is handled as a series of events. In reality, the timing consists of strings of bits, or continuous clocks, or other bursts of activity, any one of which can be related to other timing activities. These relationships can be viewed as the timing "protocol." It is in these protocols that the intended function of the circuit is captured. Manual circuit audits are able to take this information into account and capture design errors that can be missed during simulation. Designers using simulation to debug their circuits, however, cannot get the same coverage of their designs, because there is presently no way to convey the missing information, the protocol, to a simulator. Even if it were possible to convey the information, simulators are not set up to operate on such data, and this must also be done in an evaluation. Timing protocols must be propagated from node to node. This is a key difference between timing representations as they are today, and what is required to satisfy the needs of an evaluation tool. A method of representing the protocol of timing has been developed and is presented in chapter 5. This method is suitable both for the user to specify the input and predicted output signals, and also for the computer to use to represent internal node timing. The latter point makes this method considerably different than any of the timing input methods discussed to this point.

A similar design aid -- "CRITTER" -- [KelV84] was developed as an exploratory project for the purpose of critiquing digital circuit designs. This is a subsystem of a larger project -- "REDESIGN" -- [SteL84] which is a design assistant. Both of these systems were developed using expert-system technology. An extensive literature search on these systems indicates that they were never taken beyond the prototype stage, however. Although the nature of the rules used has not been discussed in the literature, the system has been used on low complexity designs, up to a dozen SSI/MSI circuits, to demonstrate the concept. It is felt that this thesis offers a viable set of rules for implementing such a system, including some novel techniques not discussed in the literature.

This thesis provides the groundwork on which the development of a powerful design aid can be based. It bounds the problem of circuit evaluation, and proposes rules for use in such a tool. In addition, a new method of describing timing information has been proposed, which should result in much lower CPU times for evaluation, than debugging with simulations. Much has been written on the subject of timing verification. Only a sampling has been referred to in this thesis. Other useful references include [DagM87, MaH87, GhoS86, GleM86, HwaS86, RimC83, Tame83, UlrE83, LeiS82, and MonM82]. An introduction to design problems is presented next.

2.0 LOGIC DESIGN PROBLEM CLASSIFICATION

This chapter serves as an introduction to common logic design problems. Clearly, errors are not intentionally put into circuit designs. They are caused by human error or lack of knowledge. Many can be found during simulation or other verification means, but in some cases, identification of a given problem through simulation requires extensive work. Sometimes exhaustive simulation of an entire VLSI design may be necessary. In other cases, simulators may be incapable of finding the error.

In order to better understand the types of errors that are made in logic design, a study of a corporate error database has been conducted.[MeiG87] This database captures the symptoms, root causes, and recommendations for avoidance for all integrated circuit design problems encountered at Bell-Northern Research (BNR) over the past four years. It encom-

passes 76 chip designs. Of the over 300 "bug reports" reviewed, approximately 45% were attributed to logic design errors. The remaining 55% were composed of errors in the following areas: analog design, chip layout, and specification. For two errors, the source of the problem could not be determined.

The results of the study are presented in Figure 1 on page 15 and are in order of significance. Note that these statistics are a reflection of the errors made in the BNR VLSI design environment. They are, therefore, influenced by both the design discipline used, and the capabilities of the CAD tools. For example, set-up and hold time violations are quite common errors in general, but this study shows that only six percent of the errors were of this type. This relatively low value can be explained by the introduction of a simulator feature to check set-up and hold violations, which took place early in the reporting period. In a design environment without such a tool, it is expected that the percentage of such errors to be significantly higher. The subject of design process and its effects on design quality will not be addressed here but is discussed by T. Curtis of BNR in a recent paper.[CurT86] The following sections define each of the types of errors, provide examples and alternatives, and include rules-of-thumb for their detection.

LOGIC ERRORS MADE IN VLSI DESIGN	
Circuit logically incorrect	
-Simulation problem	
-too long, or insufficient test vectors	(16%)
-error/stress conditions	(5%)
-mixed-mode simulation needed	(5%)
-two signals interchanged and not found	(2%)
-design changed - incomplete resimulation	(1%)
-inadequate analysis of results	(1%)
-common design problem (not checked)	(1%)
	<hr/>
	31%
-Specification problem	
- specification misinterpreted	(7%)
- function not implemented	(3%)
	<hr/>
	10%
-Modelling problem	
-inadequate cell modelling/documentation	(3%)
-functional circuit model incorrect	(1%)
	<hr/>
	4%
-Wrong file was used to create the chip	1%
	<hr/>
	46%
Clock or signal logically inverted	15%
Hazards and race conditions	7%
Set-up and/or hold violations	6%
Unconditionally dangerous/inferior circuits	6%
Asynchronous event problems	5%
Minimum pulse width violations	5%
Fan-in/fan-out violations	4%
Conditionally (on timing) dangerous circuits	3%
Analog mode of operation (e.g. long rise time)	3%
	<hr/>
	100%

Figure 1. CHIPBUGS Database Analysis Results

2.1 CIRCUIT LOGICALLY INCORRECT

This category is distinctly different from the others in degree. In these cases, the design was fundamentally wrong! As can be seen from the results (Figure 1 on page 15), 31% of these problems can be attributed to difficulties in simulation. Although some of the simulation problems shown could logically be grouped together, they are split apart here to provide examples of the kind of problems that may not be detected in simulation. For example, in two percent of the cases studied two signals were interchanged on the schematic, and the designer did not discover the problem. While one might find humor in the fact that such an apparently obvious problem could go undetected, this is a serious issue, because it indicates that exhaustive simulation of the circuit did not uncover the problem. In total, 24% of all the simulation problems reported were attributed to the simulation being either too time consuming to run, or having too few test vectors to catch the problem. This grouping includes error and/or stress conditions not being sufficiently analyzed, signals interchanged and not detected, and common design problems not being checked.

There is no replacement for a complete set of test vectors when doing circuit simulations, but it is often very difficult to conceive of a test for all possible failure modes of

a chip. Also, in many cases, the prohibitive length of the simulation is responsible for an error not being detected. This observation usually leads to a call for more powerful simulation engines, but the following question is worthy of consideration. Even if the simulations were possible, would the errors be detected given the large amount of output data that results? Keep in mind that on long runs it is often not practical to predict the precise result and have the computer do the checking. Although longer simulations may be necessary for other reasons, it is not clear that significant gains in circuit evaluation would be achieved. An alternate approach such as an automated evaluation may be the only practical way to catch such errors.

In some cases, specifications are misinterpreted, or some functions are not implemented. This is a difficult area to improve in because it is a communication problem more than a design problem. There are at least two ways in which this communication can be improved: having the person specifying the design also implement it (semi-custom design); and simply applying a more disciplined approach to writing the specification. While both of these are being done within BNR, the latter addresses the base problem of poor communication. The former simply limits it by containing the problem to one person, which is not possible in multi-designer situations. A specification writing practice, which is now

common, is to break down each chip design into functional blocks, provide a brief description in paragraph form, and then to provide a point-by-point listing of every function that the block performs. This approach insures that the communication problems are minimized.

Modelling problems at the cell and functional socket do not often occur, but when they do, can result in significant problems. Typically, such problems occur when a library cell is being used in a way that either has never been characterized, or has never been used. The results can be very serious if a bad assumption about the cell has been made on the part of the designer. Being thorough is the key to success. As designers push the technology limits, and therefore the accuracy of the models, there will be more problems of this type. Keeping both circuit and cell models up to date requires great care, and a significant maintenance effort, but a disciplined design approach is the best way to insure modelling accuracy.

The final problem listed under this section has to do with using the wrong file in the creation of the chip. While it is true that the number of files needed to create a chip can become large, it is also true that there are file management systems that can be used to reduce this sort of problem. Beyond this though, it is a good practice to do post-layout

simulations of a chip design to take into account the loading effects of the actual tracking on the chip layout. Most file errors will be picked up through such simulations. A good reference on good design practices can be found in a book by W. Fletcher. [FleW80]

2.2 CLOCK OR SIGNAL LOGICALLY INVERTED

Perhaps the most common problem during design (15% of those reported) is the incorrect polarity of clocks and signals. This is particularly a problem for signals travelling between large blocks of circuitry ("macros") the joint simulation of which may not be practical. Consequently, manual checking is needed to uncover such problems. However, manual checking does remain subject to human error. Two typical examples of signal inversion follow: one in Figure 2 on page 20, and the other in Figure 3 on page 21.

The timing of Figure 2 shows a typical example of a clock inversion. Such a situation can occur on a microprocessor driven bus with a latch enable signal, where the rising edge of the clock (latch enable) signal is intended to latch the data, but the wrong polarity of flip-flop clock is used. Another example of an inverted clock can be seen in Figure 8

SCHEMATIC **TIMING**

Data is being latched on the falling edge of the clock (indicated by the 'V'). Consequently, the 'X' (unknown) state is clocked in instead of the data. This timing would, more typically, be representative of one bit of a microprocessor bus.

Figure 2. Probable Clock Inversion Example

SCHEMATIC TIMING

Normally, tri-state drivers are only active for a small percentage of the time. The control signal to this one is probably inverted.

Figure 3. Probable Signal Inversion Example

on page 31. In this example the Case 1 timing shows a set-up or hold violation occurring. Set-up and hold violations can be symptomatic of clock inversion. By comparison, the timing of Figure 3 is not obviously a problem, and indeed may not be one. The fact that the driver is active more than half of the time, though, is an indication that a problem may exist. Because the circuit is a tri-state bus driver, it is likely that it is connected to a bus shared with other such devices. Usually in such cases, each driver only takes over the bus for a short period of time. Therefore, the CONTROL signal is probably either inverted or incorrectly defined. A discussion of design considerations for "wired logic" is given in Fletcher.[FleW80 pp. 251-258]

2.3 HAZARDS AND RACE CONDITIONS

Hazards and race conditions account for seven percent of the problems in the study, and can occur in very subtle ways. Perhaps the most common error in this category is the result of reconvergent fan-out. Reconvergent fan-out occurs when two signals that were generated in time alignment at one point in the circuit arrive at the input to another part of the circuit having followed different paths, therefore having different delays. Hazards are those circuits that

involve the gating of two signals resulting in a glitch. This does not always result in a problem in circuit operation, but is, in general, an undesirable effect. A race condition occurs when two signals connected to a circuit element compete for control of the circuit. [FleW80 p. 668] There are two kinds of races: non-critical, those that don't affect the resulting state of the circuit; and critical races, those that do.

2.3.1 Gating of two signals (Hazards)

Perhaps the most common error in this category is the gating of a clock and another signal. This problem type is often encountered when a pulse narrower than one clock period is required, and will occur when a signal, which is generated by a given clock, is subsequently gated with the clock. Problems in this type of configuration are most often caused by differential delays between the clock and the signal, causing a glitch, and may or may not represent a hazard depending on the logic into which they feed. [FleW80 pp. 324-325, FriA86 pp. 198-201, MicA86 pp. 136-137] In most cases, the circuit can be re-designed such that there is no possible hazard, as is shown in the following examples.

SCHEMATIC **TIMING**

The delay between the clock going low and the 'Q' output changing state (the propagation delay of the flip-flop) results in a momentary low-low input to the NOR, which causes the glitch.

Figure 4. Circuit with Clock and Signal Gated

Figure 4, on the previous page, shows a circuit whose objective is to generate a pulse narrower than one clock period. The width of the resulting glitch will depend upon the propagation delay of the input signal through the flip-flop. [Flew80 pp. 472-475] The underlying assumption is that the "CLOCK" signal is the highest speed available. This being the case, it is probable that the output will not be clocked by a subsequent circuit. If, therefore, the OUTPUT is being used to clock or asynchronously set/reset a subsequent circuit, the glitch can easily cause the desired function to fail. Even if the signal is going to a clocked input, the glitch could still cause trouble if it were to coincide with a clock edge. To demonstrate that such a circuit can be avoided through careful design, two alternative circuits are included.

In Figure 5 on page 26, the possibility of a narrow pulse exists, but it will depend on the input pulse width, not on a race condition, and the glitch is eliminated. This solution is quite acceptable if the input pulse width can be guaranteed by another circuit. Figure 6 on page 27 is safer still. It avoids placing additional loading on the input signal, and is absolutely safe. It does, however, impose significant overhead on the circuit to achieve this.

SCHEMATIC **TIMING**

This circuit is a suitable alternative if the integrity of the input signal can be relied upon.

Figure 5. Gating of Clock and Signal (Alternative #1)

SCHEMATIC TIMING

This circuit does not rely on the INPUT signal being well controlled. OUTPUT pulse width equal CLOCK low time.

Figure 6. Gating of Clock and Signal (Alternative #2)

2.3.2 Race Conditions

An example of a race condition is shown in Figure 7 on page 29. The two timing diagrams show the effect on the output of the reset being removed slightly after the activating clock edge (Case 1), or slightly before (Case 2). Assuming the desired result in these circuits is for the data to be latched (which might not always be the case), Case 1 represents a "critical race," and Case 2 represents a "non-critical race." Note that it is possible for the same design, and even the same physical circuit, to exhibit both behaviors depending on the temperature voltage and processing of the device. So it is not always possible to determine whether a race will be critical or non-critical.

SCHEMATIC TIMING (Case 1) CRITICAL RACE TIMING (Case 2) NON-CRITICAL RACE

Figure 7. Circuit with a Clock/Reset Race

2.4 SET-UP AND HOLD VIOLATIONS

All clocked inputs have set-up and hold time requirements. [FleW80 p. 323] Violations of these specifications are one of the more common problems found in logic design. These can be symptoms of other problems. In synchronous circuits, for example, set-up violations are often the result of a critical race. Many of the examples in this chapter could result in set-up and/or hold time violations, either directly, or further along in the circuit.

Figure 8 on page 31 shows a flip-flop which could represent any clocked circuit. Two sets of input timing are considered. In case 1, either the clock is inverted or a race condition exists between the clock and input signals. Because the falling clock edge coincides with a transition of the input signal, two output scenarios are possible. Case A is the result of the input transition leading the clock slightly. The lagging case is illustrated by case B. In practice however, the situation may not be quite so clear cut. There could be slight variations in circuit delays with temperature and voltage, resulting in both case A and B timing occurring at various points in the operation of the circuit. Also, the input pulse width could be slightly modulated, which would again yield either or both of case A and B timing.

SCHEMATIC TIMING (Case 1)

The output bit position depends on whether the input pulse occurs just before or just after the clock edge.

TIMING (Case 2)

Figure 8. Circuit with Set-up/Hold Violation

The case 2 timing diagram shows a condition which is always difficult to deal with. In this case, the input pulse width is less than one clock period. The underlying assumption here is that no phase-related higher frequency clock is available with which to clock in the input pulse. Once again, two possible outputs exist. The circuit will either catch the data, or miss it, for the same reasons described in the previous case. Either or both output cases are possible during a given operation of the circuit. This problem is almost certainly due to a race condition. Unfortunately, in real life design, this design problem can sometimes not be avoided. In most cases, it is not practical to generate a higher frequency phase-locked clock to re-time the input pulse. This being the case, characterization, control, and matching of circuit delays are the best way to insure correct operation.

It is important to note for asynchronous inputs, the timing shown in either case may represent correct circuit operation. Such circuits are virtually impossible to successfully simulate with even state-of-the-art logic simulators. This usually results in areas of exposure in the design which can only be evaluated manually.

2.5 UNCONDITIONALLY DANGEROUS/INFERIOR CIRCUITS

This category captures circuit implementations that are unsafe, clearly in error, or using components that are inferior to others available. Several examples of these types of problem are presented.

- Inappropriate Circuit Connections
 - unconnected input nodes (including power rails)
 - totem-pole outputs tied together (cannot be fully fault modelled)
 - outputs connected to a power rail (almost always an error)
- Inappropriate Circuit Elements Used
 - Cells which, subsequent to their use, have been:
 - found to result in low yield
 - found to have reliability problems
 - replaced with a preferred design
- Inappropriate Use of Circuit Elements
 - output directly connected to its own async override input
 - incompatible cells used e.g. wrong clock driver used with a counter

2.6 ASYNCHRONOUS EVENT PROBLEMS

No matter how hard designers or CAD tool developers wish, the rest of the world is not synchronous to their system

clocks. This leads to signals coming into a circuit from the outside world that are not phase related to the system clock. Two types of asynchronous event problems exist: those in which data, asynchronous to the system, must be synchronized to the system clock; and those in which asynchronous events result in unforeseen problems. The latter type of problem translates into one of the other problem types: often a race, or set-up/hold violation. The former is the more general of the two, and will be discussed in further detail. A good discussion of asynchronous design is presented in Fletcher. [FleW80 p. 652]

2.6.1 Asynchronous/Synchronous Interfaces

Figure 9 on page 35 shows a very basic asynchronous to synchronous interface. The timing shows jitter on the data clock happening over a very short period of time, e.g. over several bit periods of time. On average, though, the system and data clocks are at the same frequency. This situation is typical of that resulting from a clock recovery circuit employing a digital phase-locked loop. In practice, the effect would be spread over a longer period, but the result would be the same. Note that the output data is corrupted.

SCHEMATIC TIMING

ASYNC DATA

D1	D2	D3	D4	D5	D6	D7
----	----	----	----	----	----	----

DATA CLOCK

'Q1'

X	D1	D2	D3	D4	D5	D6	D7
---	----	----	----	----	----	----	----

SYSTEM CLOCK

DATA OUTPUT

D6 clocked twice

X	X	D1	D2	D3	D4	D6	D6
---	---	----	----	----	----	----	----

Asynchronous to synchronous interface: jitter in the input data results in output errors due to inadequate buffering of the recovered data.

Figure 9. Asynchronous/Synchronous Interface Circuit

The solution to this problem is usually not simple. Typically, it involves bit or byte buffering in conjunction with circuitry to detect "clock crashes." For purposes of this discussion, a "clock crash" can be defined as that which occurs when one, or the other, of the clocks in the system occurs more than, or less than, once per bit period. The result of a clock crash is either that one bit is clocked twice, or not at all. Upon detection of a crash, control circuitry can direct the data read and write circuitry to work on the appropriate buffer, negating the effect of the crash. Such a solution can never be considered to be totally safe, because it only takes care of instantaneous phase jitter and assumes that the average frequency of the two clocks remains fixed. If one clock were to drift relative to the other, data would be lost even if such a circuit were employed. In most applications, the two clocks are frequency locked however, and one can assume that the short term average frequency is fixed. A discussion on how to do such designs is given in Fletcher. [FleW80 pp. 466-470]

2.7 MINIMUM PULSE WIDTH VIOLATIONS

This problem type represented 5% of the design errors, and was usually associated with the power-up initialization pro-

cess. During initialization, questionable assumptions are often made about the stability of the power supply, the amplitude and duty cycle of system clocks, and the stability of the circuit at the time that the reset pulse is applied or generated. These effects are difficult, if not impossible, to simulate with any accuracy, and must be given special attention in the design process. Perhaps the most common minimum pulse width problem occurs with the reset pulse width for a long ripple counter. When the worst case ripple delay is not properly taken into account, it is easy to see how a ripple counter might only be partially reset.

Another circuit type which will exhibit minimum pulse width problems is the asynchronous self-modifying output. The problem is not always obvious, particularly when the feedback path involves a long path through other logic.

2.7.1 Asynchronous Self-modifying Outputs

This problem can occur when an output signal is fed back into the circuit that created it in such a way that it is immediately negated. In order for this to be a problem, there must be a feedback path within the circuit which is

independent of any circuit clock, i.e. asynchronous feedback (see Glitch-free Output Generation [FleW80 pp. 472-475]).

Consider Figure 10 on page 39. A signal is fed back through a one or more gates to the circuit that created it. This is done in an asynchronous manner such that it is "instantaneously" modified. In some cases, this may be done intentionally (through delay elements) to create a very narrow pulse. The main problem with this circuit is that the propagation delay through the feedback element(s) tends to be variable from circuit to circuit, and is therefore somewhat unpredictable. With an excessively narrow pulse, the circuit may not function at all. Such a circuit technique is never recommended because delay control is difficult. Sometimes, however, its use cannot be avoided.

Consider the circuit of Figure 11 on page 41. As with the previous example, the duration of the RESET pulse will determine whether or not the desired effect will occur. Since a ripple reset exists, the RESET pulse must be wide enough to reset all stages of the counter. However, as soon as the outputs start to reset, the reset input signal itself will disappear. This may leave the latter stages without sufficient time to reset. Note: An asynchronous reset is assumed in this example. Figure 12 on page 42 shows a safe alternative to the circuit of Figure 11. This circuit, although

SCHEMATIC TIMING

The output pulse width is related to the delay around the loop (propagation delay through the inverter plus the delay to reset of the flip-flop).

Figure 10. Asynchronous Self-modifying Outputs (Cct #1)

safe, does impose a significant amount of overhead on the circuit. This solution guarantees that the RESET pulse will be one half of a clock cycle wide, which should be more than adequate in most counter designs.

2.8 FAN-IN/FAN-OUT VIOLATIONS

Fan-in/fan-out are defined in Fletcher. [FleW80 p. 259] These problems are not quite so common (4% of the problems reported), but are significant in that there is generally no mystery about whether or not there is a violation. The fan-in and fan-out limits of circuit elements are usually defined, and for any given circuit it is a straight forward process to calculate the margin. There are situations, however, which make the calculation of these numbers a greater problem than one might think. The bulk of fan-out violations, for example, occur when a signal enters more than one macro block, and is used to drive inputs in each. Each macro block designer may assume that he/she has the full fan-out quantity to work with. Another case, which is much more difficult to detect, occurs when one of the cells driven by a signal contains transmission gates. Some cells employ these gates to implement logic functions. Since the load at the output of this cell is seen at its input, it is

SCHEMATIC - Divide-by-13 Counter

TIMING

If the reset is not wide enough, the counter may only partially reset (it may reset to one of several states).

Figure 11. Asynchronous Self-modifying Outputs (Cct #2)

SCHEMATIC - Divide-by-13 Counter **TIMING**

To guarantee the reset, the reset pulse width must be a guaranteed minimum.

Figure 12. Alternative to Self-Modifying (Cct #2)

quite possible that the input load may be much greater than assumed.

The final difficulty involves the fan-in/fan-out calculation procedure. All inputs/outputs are not created equal. It is quite possible that one input could represent twice, or more, the load of another. Often what is done in practice is a calculation of the number of inputs times an average input load. This only gives a first order approximation of the actual input/output loading. The actual per cell input and output loading must be taken into account when calculating fan-in and fan-out.

2.9 CONDITIONALLY DANGEROUS CIRCUITS

"Conditionally dangerous circuits" refers to any circuit that can, dependant on the timing, result in undesired operation, and may be otherwise precisely what is required. Often, the timing conditions that are necessary for the malfunction to occur go beyond what is considered normal operation for the circuit. A trivial example of this would be two tri-state buffers connected on the same bus. With the proper timing there will be no conflict, but under the right conditions the buffers can be made to "fight" one another.

There are more serious types of conditionally dangerous circuits that shall be referred to as circuits with "invalid states." Some examples of these follow.

2.9.1 Lock-up and Invalid States

This class of problem is often difficult to detect. It typically will be found in circuits that implement protocols, or in state machines. The nature of the problem is that a condition ("state") exists from which a circuit will never return on its own. This usually results in the circuit being locked-up, and not performing any useful function. It may also mean that it will simply lose synchronization with external events, and have no way of getting back into step. Two areas of circuit design in which this class of problem is prevalent are communication protocol handlers, and transmission circuits. Both relate to the implementation of a protocol although for the latter this may not be as obvious.

In many synchronous transmission schemes, it is necessary to detect some form of framing from a continuous stream of data. Many circuits exist for doing this, however they are usually quite similar in implementation. Confirmation of synchronization (sync) acquisition must generally occur a

SCHEMATIC TRUTH TABLE

Memory Contents

Address	Data
0 0	1 0
1 0	0 1
0 1	0 0
1 1	1 1

NOTE:
 "Address" equals
 the last value of
 "Data."

In this case the '11' state is not part of the intended function, a divide-by-three down counter. Should the machine ever get into the state, say during power-up initialization, it will never get out of it, except via a complete reset of both memory and logic. This is a very simple example but one which could easily occur since many memories default to the logic '1' state.

Figure 13. Circuit with Lock-up State

number of times before the circuit declares itself "in-sync." Similarly, it may have to go through another number of states before sync loss is declared. In the course of designing such a "state machine," as well as other circuit types, there are frequently undecoded states, or lock-up conditions, which are inadvertently incorporated into the design.

Figure 13 on page 45 shows a generalized block diagram of a circuit which will lock up. This is a very simple example of what is quite often subtle and difficult to detect. Nevertheless, the fault mechanism is the same. In this case, a state machine is being driven by the contents of a memory. This state machine is an implementation of a divide-by-three counter, but one address of the memory has not been programmed and is at the default value of binary "11." This represents a closed loop in the state table, because the data points to its own address. Under normal operation, this circuit will not fail. However, if the circuit is ever put into this state, it will only recover through a system reset, either as part of an infrequent system action, or by removal and re-application, of the power.

The best way to avoid these problems is by the use of state diagrams during the design process. Unfortunately, state diagrams are not usually used in design. For simple circuits,

they are often viewed as unnecessary, and for complex circuits they are often seen as too time-consuming to be justified. Based on experience, those who use state diagrams tend to have fewer problems. Otherwise, the only way to avoid these problems is through very careful and extensive checking of the circuits. Often these problems are quite subtle, and will only be picked up by someone who is familiar with the application. Usually the resultant problem will only show up in "illegal" states, or in a mode that the circuit shouldn't be able to get into, making the prediction of the existence of these states even more difficult. An alternate approach is to identify possible failure modes for the circuit, and then review the circuit to determine if the necessary stimulus can be generated by the circuit. Such a technique, though somewhat "brute force" and time consuming, could be automated. The design and analysis of state machines is discussed in Fletcher. [FleW80 pp. 335-386]

2.9.2 Invalid Transition States

This is another example of a problem which is conditional on both the input timing and the surrounding circuitry. Invalid transition states are most commonly found on multiplexed busses, especially tri-stateable ones, such as on a micro-

processor interface. Typically, the sequence of events leading to an invalid state can be regarded as a delay problem. For example, prior to the correct address being present, a demultiplexer taking its address from a bus may be enabled by a microprocessor write pulse. During the time between the device being enabled and a valid address appearing on the microprocessor bus, there will be an erroneous "transition state." This is illustrated in Figure 14 on page 50. (See Critical and Non-critical Races [FleW80 p. 668], and Glitch-free Output Generation [FleW80 pp. 472-475].)

A similar situation to that just described can occur if there is a differential delay between the various bits of the address to the device. Such a case involving a multiplexer is shown in Figure 15 on page 51. In most cases, these conditions can be avoided by either latching the input address, or the output. It should be recognized that the conditions just described will often not represent a problem, in that they will not result in a functional failure of the circuit. This must be taken into account in any circuit evaluation. If a given effect can be guaranteed to not result in a functional failure of a circuit then it need not be considered as a problem.

2.10 ANALOG MODE OF OPERATION

There is an input voltage region for any logic input that will result in an indeterminate output. The output may go high, low, anywhere in between, or may even oscillate. In this mode, the gate is operating in its linear region, and can no longer be considered to be functioning as a logic circuit. In a small number of cases, logic circuits will be intentionally used in this region to make use of their analog properties, such as in the case of a ring oscillator. In the vast majority of cases though, operation in this region is an error.

Figure 16 on page 52 shows an inverter whose input signal exhibits a long risetime. While the input is in the indeterminate switching zone, the output cannot be predicted.

SCHEMATIC TIMING

OUT4 has an undesirable output state as a result of input to address differential delays.

Figure 14. Invalid Transition States: Circuit #1

SCHEMATIC TIMING

Differential delay in the address lines results in undesirable output signals.

Figure 15. Invalid Transition States: Circuit #2

SCHEMATIC TIMING

Figure 16. Circuit with Indeterminate Switching

The long input signal risetime leaves the gate's input in its linear region of operation for a long time, resulting in an unpredictable output.

3.0 TECHNIQUES FOR DETECTING LOGIC DESIGN PROBLEMS

In principle, all logic design problems can be detected by simulating under all possible input conditions. This is usually impractical due to the large volume of information processing required, and the large amount of output to evaluate -- even for a low complexity circuit. So, such exhaustive simulations are normally not attempted. Instead, designers simulate for correct operation under normal operating conditions (although not simply nominal), and then simulate for failure modes that may occur when the input signals are set beyond their specification limits. This approach provides an indication of how much timing margin exists within the design, but it is not a complete analysis of the circuit.

This provides sufficient information to detect many circuit errors, but does not provide total circuit or functional

coverage in most cases. It also does not cover all timing which goes outside of the specified operation limits, full test pattern coverage of the specified conditions, or a check on whether the circuit elements are being used correctly. This should not be confused with fault simulation with which the designer attempts to produce a minimum size test pattern to reveal circuit problems due to manufacturing defects in the device. An alternative to the process of "debugging" by simulation is a circuit evaluation performed by an unbiased expert designer.

In the detection of circuit errors, there is a fundamental difference between evaluating and simulating. In an evaluation, an expert looks for conditions that may cause trouble; the expert has knowledge of circuit problems, and evaluates the circuit to see whether these conditions exist (i.e. the expert applies his/her experience). A simulation, on the other hand, assesses a circuit's response only to the stimulus provided. So if the required conditions to cause circuit failure are not provided, the error will go undetected. Debugging a circuit by simulation, therefore, inherently requires the generation of large test patterns. This difference is a very significant factor in determining the most efficient method of circuit debugging.

It is generally accepted that in evaluating a circuit by simulating all possible timing cases, the number of vectors produced can become so large the process is simply not practical. An evaluator that has knowledge of circuit errors need not generate many vectors. Knowing the conditions that can cause problems, it has only to determine whether these conditions are possible with the given circuit topology. The only timing information that need be carried through the circuit deals with its protocol, rather than a record of every timing event required in simulation. This approach to circuit evaluation has the potential to reveal the majority of design errors, without the cost of long simulations. Since an evaluation tool will not provide a detailed stimulus/response assessment, an evaluator is complementary to a simulator, and not a replacement for it. It is intended that such a tool be utilized before simulation so that most "bugs" can be removed prior to the costly simulation phase. It is expected that this would reduce the overall cost and design time of most designs.

The balance of this thesis will provide insight into how the process of evaluation can be automated. All techniques discussed are based on the assumption that the evaluator is checking for conditions that are known to cause trouble in many cases, rather than relying on simulation which is best suited to confirming the correctness of the logic.

The rules presented in this chapter are based on conservative design practices. As such, they will flag designs that may not violate any circuit or technology parameters. Although this approach is not rigorous in determining errors, it will always flag areas of the circuit that are worth a second look. For example, asynchronous-to-synchronous interfaces will always be flagged. Such circuits require careful design, and cannot be simulated for all possible conditions with present day simulators. Flagging such a circuit can be a significant aid to the designer in narrowing the focus for further work.

The rules presented here represent only a basic set. However, these alone should detect the majority of design errors. To reduce the number of reports generated, and to provide still better coverage of the design, it is possible for the rule set to expand to consider special cases, and to provide the program with the ability to recognize and operate on circuit structures beyond the cell. This would greatly enhance the value of such a tool.

3.1 CATEGORIES OF ERRORS

To this point, it has been shown that a large number of test vectors are required to detect certain design problems. It is equally true that some design errors do not require any timing information for their detection. By grouping the errors by whether or not timing information is required for their detection, one can see that two levels of evaluation can be considered. The first, requiring no timing information, can be performed with only a netlist provided as input. The second category requires that some general description of the timing accompany the netlist. These lists are compiled based on the study of the previous chapter.

CATEGORY I

Problem detection is timing independent

1. Circuit structures which are inferior or dangerous.
2. Fan-in and fan-out violations.

CATEGORY II

Problem detection is timing dependant

1. Circuit logically incorrect.
2. Clock or signal inversions.
3. Hazards and race conditions.
4. Set-up and hold time violations (clocked inputs only).
5. Problems due to asynchronous events.
6. Minimum pulse width violations, e.g. ripple reset.

7. Circuit structures which are conditionally dangerous, depending on the stimulus.
8. Logic circuits operating in an analog mode.

The Category I problems are generally quite straightforward to detect, and can include other syntactical checks of the netlist. Detecting inappropriate circuit structures, however, is a broad requirement ranging from detecting the improper use of circuit elements, to the detection of inappropriate arrangements of circuit elements. The latter requires that the checker recognize circuit structures from interconnected circuit elements (also true for the same problem type in Category II). Techniques for achieving this level of checking go well beyond the scope of this thesis.

3.2 DETECTION METHODS

In manual circuit audits, one generally does not go through a circuit looking just for one problem type at a time. Instead, paths are taken through the circuit, and a series of checks are done along the way. The results of these checks assist in making a diagnosis of the problem. They may not be conclusive by themselves. For example, a set-up and hold violation may be an indication of a malfunctioning asynchronous-to-synchronous interface, but it could also indicate a

signal inversion, or excessive delay. Only further study of the symptoms will help to isolate which is the problem.

So, instead of directly detecting the problems of the last chapter, a set of rules to check for the symptoms is applied, and based on the results, a diagnosis is made. The balance of this chapter develops the rules to detect these symptoms, and guidelines for applying them in the circuit. An algorithm for finding paths through the circuit is also described. These rules are applied to real examples in the next chapter.

Evaluation rules fall into three categories: input, output and cell. Cell rules are those that apply to a circuit element rather than their input and output nodes. These rules have been developed by mimicking what a designer looks for, and then checking them against the identified problem areas to see whether they give sufficient coverage. A description of these rules follows the introduction of a timing definition, $T_{critical}$, which is referenced in several places.

3.2.1 Tcritical

The rules that follow are based on conservative design practices. In a conservative design, timing signals are expected to be well margined, i.e. with no tight timing. It is, therefore, possible to identify a time interval, $T_{critical}$, which is smaller than can be derived with certainty. Pulses that are narrower than $T_{critical}$, or differential signal timing delays that are less than $T_{critical}$, cannot be derived using only conservative practices, and, therefore, represent potential design problems. Although $T_{critical}$ is useful in detecting potential problems, the rules are insensitive to its precise value. This is another outcome of applying conservative design rules in the checking.

Based on experience, $T_{critical}$ is approximately 10% of the highest frequency clock period (or smallest pulse width) in the circuit, i.e. the clock that would be used to obtain the highest accuracy.

$T_{critical}$ is derived as follows: A signal that is 50% of a clock period can easily be generated by gating the clock with a one-bit-wide signal. Some distortion in the resulting signal can occur due to risetime and falltime variations as it passes through the circuit. The remaining pulse could be narrowed to, say, 30% of a clock period. Since $T_{critical}$

is used in calculations of relative timing, one must consider the effects of differential delay between two signals following different circuit paths. This could lead to perhaps another 10% of delta timing, leaving the effective pulse width at 20% of a clock period. While these numbers are only approximate, one can see a pattern which suggests that there is a limit below which the designer is either attempting some very tight timing, or is not intended. A further 10% is somewhat arbitrarily assigned to unintentional circuit path delays (long circuit delay due to the interconnect medium, e.g. polysilicon instead of metal, r being physically much longer than anticipated by the designer). So, if a clock and signal that it is interacting with, have transitions within 10% of a clock period of one another, the timing is probably in error, i.e. unintended. This statement represents a conservative approach to design, but one which is useful as a benchmark. If a circuit is flagged because of this, a second look at the timing will often result in a problem being uncovered.

Note that this definition of a problem will also flag two signals that are generated from the same clock edge, as they impinge on the same cell. The flag should be considered valid even under such a condition. Often two signals that were generated by the same clock edge can take very different paths in getting to the same destination. This can

result in sufficient differential delay to cause problems. If it is obviously not a problem to the design, it will not take long to check. If it is not obviously OK, it's worth checking. Many a designer has been fooled by assuming that two signals were phase aligned just because they were originally generated by the same clock edge.

3.2.2 Input Rules

Input rules can be broken down by the input type: all, clock, clocked, and jam (asynchronous override inputs e.g. loads, sets, resets etc.).

3.2.2.1 Rules for All Inputs

Rule #11 - Fan-in

Check that the fan-in specification for the input is not exceeded.
--

Fan-in can be checked by referring to the FANOUT list. Beside each node name is a list of all cells connected to the node. The relevant properties (e.g. input current) of each cell I/O type should be known to the program. With this information, the program can calculate whether a fan-in specification is being violated.

Rule #12 - Indeterminate Switching

Check for long rise or fall times. The definition of "long" will be technology dependent, but for current CMOS technologies in the one to three micron range will typically be anything greater than 50 nanoseconds for internal nodes.

Flagging signals with transition times longer than this should provide useful pointers to the designer for further investigation. The calculation of rise and fall time can range in complexity from a capacitive loading calculation, based on the fan-out, to a more complex analog simulation of the node. In either case, the FANOUT list holds the main source of information for this calculation. Note that even though this is an input check, the calculation need only been done once for each output on the node.

3.2.2.2 Rules for Clock Inputs

The rationale for the next rule has been discussed in the section on $T_{critical}$. Pulse width is a concern for normal clock inputs when it approaches a glitch, but for structures like PLAs a certain precharge time is required. A separate rule could be defined to check such cases, but the $T_{critical}$ rule alone will pick up many problem instances.

Rule #13 - Minimum Pulse In

Flag all hazards and narrow pulses going to clock inputs as probable errors. A narrow pulse is defined as anything shorter in duration than $T_{critical}$.

3.2.2.3 Rules for Clocked Inputs

When trying to identify a clock that is inverted, it is often useful to compare the timing to common practice. For example, the timing may show that the set-up time is exceeded in several flip-flops by only a 2 nanoseconds margin. If this circuit is running at, say, 10 MHz (100 nsec period) then, even though the timing does not violate any rules, it is likely not what the designer intended. This is so, be-

cause with a 100 nsec clock it is very difficult to guarantee such a small timing margin. The only way to have such a small number is through the use of elemental circuit delays, which can vary significantly over temperature, voltage, and process variations. It is useful to consider what is normally done in design, and flag those things which are abnormal.

Rule #14 - Clock Inversion

All input data transitions that occur within $T_{critical}$ of the activating clock edge should be flagged. (This indicates either: a clock inversion; a possible set-up or hold problem; or an excessive delay problem.) Exception: internal shift register stages (These are a special case because the clock and data delays are designed to track, and are always located nearby one another so that the design assumptions are valid.)

Although this rule represents a conservative approach to design, it will in practice point to many errors.

Set-up and hold violations are often symptomatic of other problems. Set-up time is the interval between the last transition in the input data to be clocked, and the activating clock edge. Hold is the interval from the activating clock edge to the next data transition. [FleW80pg323] These

parameters can be positive, negative, or zero; all are possible. The detection of set-up and hold violations does not require the sophistication of an evaluator. Both of these parameters can be measured directly from timing waveforms supplied in normal simulation, but this is not provided universally in simulation. It should be noted that, when flagging a timing violation, the timing sequence with which the violation is associated should be identified to the user for purposes of subsequent analysis.

Rule #15 - Set-up and hold

To guarantee detection of set-up or hold violations, the calculation must be done on every data transition.

This rule requires that the evaluator be provided information on the position of data transitions relative to the clock edge. The logic state of the data is not required for this calculation.

Rule #16 - Asynchronous Data 1

If there is no relationship specified between the clock and the input data, an asynchronous-to-synchronous interface exists. All such instances should be flagged, as they are potential problems.

In stating this rule, it is assumed that the timing relationships between signals can be specified. This is discussed in more detail in chapter 5. In many cases, merely flagging this type of circuit should be sufficient for the designer to recognize that very careful checking is required. For a computer to be able to be of further benefit, a means of identifying the entire async-to-sync interface circuit would be necessary. To achieve this, the computer would have to have an understanding of circuit structures. This capability is one that, while possible to incorporate, goes well beyond the scope of this thesis.

Another rule applies to asynchronous signals on multiple clocked inputs. If, for example, two flip-flops were to clock in a signal that was asynchronous to both of their clocks, one might catch the signal, and the other not. This can easily lead to problems, if the designer assumed that since one cell captured the signal, so would the other. Due to different path delays, and cell characteristics, the signal may not be captured by both.

Rule #17 - Asynchronous Data 2

If a signal is connected (on one node) to more than one clock input, and this signal is asynchronous to more than one of the clocks, flag the condition.

3.2.2.4 Rules for Jam Inputs

Rule #18 - Jam Inputs

Jam input data should not change state within $T_{critical}$ of the activating clock edge. (This indicates either excessive delay in the data path, or a glitch on the line.)

This rule is based on the $T_{critical}$ argument that was presented earlier. Probably the most common problem type that will be picked up by this check is the reset pulse occurring too near the activating clock edge.

3.2.3 Output Rules

There are probably more types of output cell than any other. They are designed to provide various drive capabilities, and output levels, and be inverting, non-inverting, and tri-stateable. It is very important that there be compatibility between the output characteristics of a driver, and those of the inputs to which they connect. Errors in the selection of the proper cell are easily made.

3.2.3.1 Rules for All Outputs

Rule #01 - Fan-out

The fan-out of all outputs should be checked against the specification for each cell. All violations should be flagged.

As was the case for the fan-in check, the key information required to perform this check is contained in the FANOUT list. The program is expected to have knowledge of the input and output properties (e.g. input capacitance, output impedance, output swing, input current etc.) of each of the devices on the node. With this information, the evaluator

can calculate the loading, and determine if there is a fan-out violation. Loading can be calculated to varying degrees of accuracy depending on what approximations are made. While it is always desirable to have the most accurate answer possible, the details of how to implement the check are not covered in this thesis.

"Feedback" is a potential problem that can be detected readily if "generation" information is kept about each signal -- that is, the source of each signal, and every other signal that has contributed to it along the way.

Rule #02 - Feedback

Feedback is established when the signal under consideration has its own name in its ancestry list. A potential problem exists if the total delay around the feedback path is less than $T_{critical}$. All feedback paths should be listed whether a problem is detected or not.

In addition to the tracking of timing signal relationships, the program is expected to keep a list of each signal that contributed to the creation of the present signal. This is referred to as the ancestry list. If a signal is one of its own ancestors, then a feedback loop must exist (i.e. it helped to create itself). All feedback paths should be listed because they are a necessary condition for lock-up. This is discussed further in the section on rules for lock-up.

This rule is independent of the technology used, and is based on the assumption that any combination of signals resulting in a pulse width less than $T_{critical}$ is probably not intended (conservative design).

Once feedback is established, there is a problem; there is no starting point from which to generate the initial timing.

There are two possibilities for achieving this; either a jam input will exist which can be used, or else a continuous bit stream can be assumed. If not, a bit should be assumed to have just occurred and the timing taken from there. In either case, the timing protocol for the node should be re-checked after the evaluator has gone back around the loop. If the result does not match the assumption, flag that the loop could not be evaluated -- unable to find the initial conditions.

Rule #03 - Shorts

All output connections to a power supply should be flagged as errors.

This can be checked directly from the FANOUT list.

Rule #04 - Minimum Pulse Out

Flag all outputs that produce glitches, or narrow pulses ($< T_{critical}$) and indicate whether it is an internal node or not. If not, it should be given a special warning flag because it is intended to interface with another circuit.

A distinction should be kept between a narrow pulse and a glitch (a lower threshold, typically a few gate delays) when flagging a problem. Keeping them separate can sometimes

greatly simplify the identification of the problem source. There will be instances in which "delay elements" are used to produce a minimum pulse width in designs. In general, this is something to be avoided due to the typically wide variation in the delay value over temperature, voltage and the fabrication process. In some cases, however, this technique is the correct choice. In such instances, applying the 10% bit rule will result in flags being generated to the designer. This should not be viewed as a bad thing unless there are a large number of such flags generated. To limit the number of separate reports, they should be related to the number of nodes, and not to the number of timing events.

3.2.3.2 Rules for Tri-state Outputs

Very often design errors are made at circuit boundaries. In many cases, circuits are interconnected via a bus, resulting in the need for tri-state drivers to permit multiple users of the bus. Inversion of the control signal to these drivers is a common occurrence, often rendering the circuit unusable. Due to the diverse types of application circuit that can be conceived of, it is not possible to define an absolute percentage of time that a driver should be active. Depending on the configuration, and often upon the amount of

activity, this can vary greatly. Rather, the active time is nearly always less than or the same as the time that data is presented to the driver for interface to the bus.

Rule #05 - Tri-state Control

If the active (driving) time of a tri-state driver exceeds the data time by more than 10%, flag a warning. Usually, they have a one-to-one correspondence.

3.2.3.3 Rules for External Outputs

External outputs, ones that do not terminate on another circuit, have special attributes that both require, and allow, additional checks to be done. These outputs are different in that they are generally designed with a much higher drive capability than for internal nodes. In addition, they provide a point at which logic signals are well defined, and therefore, provide a convenient reference point at which to check the protocol of the timing. Since the evaluator does not have information about what other circuitry the circuit under consideration is connected to, the best that it can do is create a list of the attributes of the external output

drivers for the designer to check manually. This, at least, reduces the amount of manual checking required.

Rule #06 - Output Driver

Create a list of the following output driver characteristics:

- unidirectional or bidirectional;
- inverting or not (as intended);
- output level compatibility (CMOS, TTL etc); and
- current drive capability.

Output drive structures have different input drive requirements depending on their internal structure. For example, a simple inverting output in a CMOS process may consist of:

- N and P channel transistors that are matched in drive capability (DC and/or AC);
- as above, but designed for unbalanced drive capability;
- an N-channel pull-down transistor, and a bipolar pull-up (either PNP or NPN);
- an N-channel pull-down with a saturated transistor load (MOS or bipolar); and
- other configurations are also possible.

Two circuit problems which can affect the drive performance should be checked: insufficient voltage at the driver input, and insufficient input current drive.

Rule #07 - Interconnect to Driver

- Flag all drivers that require a DC current to operate. The resistance between the driver input and the signal source should be checked to insure that sufficient drive current is available.
- Flag all drivers that are voltage sensitive. The voltage at the input to these must be checked to insure that there are no voltage divider effects that could result in improper operation of the device.

The manual checks indicated by the above flags can only be done after the chip layout is complete because they relate to the interconnect medium as well as the circuit structure. The extracted interconnect resistance, and the interconnect configuration, are necessary to perform these checks accurately.

To clarify the rationale for the above flags, consider the following. The output of a gate is connected to another MOS cell input, and to an output driver with a bipolar input section (not uncommon in certain CMOS output drivers). Since

the bipolar input will draw a small current, there will be a voltage drop along the interconnect medium when the gate is driving high. If the interconnect is polysilicon, instead of metal, the drop will be greater. So the interconnect appears as a voltage divider to the input of the other MOS cell. The potential problem with this circuit is that the voltage drop will be so great along the interconnect that the input level to the MOS gate may not exceed its input threshold. Hence the check.

3.2.3.4 Rules for Output Timing Protocol

Without having specific knowledge of the design intent of a circuit, it is impossible to say conclusively whether it is logically correct. There are, however, indicators that can be checked that will give an indication of the correctness of the logic. Generally, it can be said that if the output "protocol" is correct, the probability of the logic being correct is high. The "protocol" of the output means that the format of the timing is correct, i.e. the bits are where they are supposed to be. It does not insure that the bits are correct, but knowing that they are in the correct place is a good indicator that the logic is also correct.

The "protocol" of timing signals is not represented in timing description methods for present day simulators. However, as can be seen from the preceding discussion, this information can be a powerful tool in the detection of logic errors. It will be shown in the following sections that this information is valuable in the detection of other error types as well.

Signal Inversion

The detection of signal inversion without simulation is difficult. Often, however, designs are partitioned into macro functions which create many well defined interfaces within a single circuit. Defining the protocol of signals at these interfaces, for example the relationship between the clock and data, can greatly improve the chances of detecting a signal inversion. Most clock inversions occur at circuit interfaces where different polarity assumptions are made by different designers.

Data inversions are much more difficult to detect. Even in circuit audits, the polarity of data must be traced from every source to the output in order to guarantee that the polarity is correct. Checking the polarity will contribute to detecting at least some instances of data inversion. This will be especially true in cases where fixed data patterns

exist which can be used as a basis for comparison. It is appropriate to specify the protocol of signals both at the periphery of a circuit as well as at internal nodes. The greater the number of nodes with protocol specified, the greater the probability of detecting an error.

Rule #08 - Output Protocol

Check the output protocol for every output where a timing specification is provided. This is equally applicable to internal nodes.

This rule assumes that the protocol of the timing on every node in the circuit is calculated, and stored along with the relationships, and ancestry lists discussed previously. An example is provided in chapter 5 which shows a simple circuit, the associated protocol representations on every node, and the graphical equivalence. This example shows how this protocol information can be used to reconstruct signals which can be compared by a computer to determine equivalence.

3.2.4 Cell Rules

3.2.4.1 Rules for All Cells

Rule #C1 - Non-preferred Circuits

Flag the use of all circuit elements that are not on the preferred usage list. These cells are either dangerous (may be incompatible with certain other circuits), inferior (may have superior replacements available), or new (never before used/characterized).

It is sometimes the case that designers have access to cell libraries containing elements whose designs have been superceded by other cells. The changes are generally made to improve performance, but in some cases the older cells have error modes or poor electrical performance under certain conditions. An example of this would be an I/O cell that was superceded by another having improved ESD and latch-up performance. The selection of such cells is often inadvertent, and is seldom what was intended by the designer.

Rule #C2 - No-connects

Flag all cell input nodes that are left unconnected. In some technologies this may not be a problem, but in CMOS this represents a danger to circuit operation.

Note that there is a distinction between a node and a cell I/O point. The above check looks for unconnected nodes, which can have more than one cell I/O point. As long as one of the cell I/O points is connected, the node is connected. Flagging unconnected inputs may also lead the designer to discover that the circuit, as represented on the schematic, is not yet complete. As a matter of design cleanliness, it is a good practice to connect all circuit inputs, even if they are grounded.

3.2.4.2 Rules for Lock-up

A circuit is said to be locked-up if it ceases functioning according to its design intent in such a way that it can only be returned to normal operation by resetting the circuit. Resetting can vary from the application of a signal to the removal and application of power. There are two

types of problem that are commonly considered to be lock-up phenomena:

- lock-up during initialization; and
- lock-up during normal operation.

Initialization Lock-up

When trying to predict circuit failure during initialization, the question to be answered is: Is it possible for any part of the circuit to fail to reset, or not fully initialize? Generally when a circuit powers up, all circuits do not receive power at exactly the same instant. Consequently, it is quite common for clocks and other signals to be out-of-spec during power-up initialization. To combat these conditions, special precautions may be taken in the circuit design, but there are no guarantees. To detect lock-up scenarios under such conditions is extremely difficult, because none of the protocol specifications can be considered as valid.

The following rule will not guarantee that there is not a problem, but it does mimic what often occurs in reality. It should, therefore, improve the view of the circuit, and give the designer some idea of how susceptible the circuit is to such conditions. "Glitching" the clock is particularly relevant on structures that require some precharge time, like

PLAs. Providing an inadequate clock may cause them to go into undesirable states. This rule should only be run after all of the previous ones are complete. Since it may be somewhat time consuming, it should be offered as an option to the user.

Rule #C3 - Initialization Lock-up

To detect possible problems during initialization, the program should run successive evaluations with each input adjusted such that: each clock signal is narrowed to a glitch for one cycle; one cycle is missed (all high); and one cycle is missed (all low).

Normal Operation Lock-up

Lock-up during normal operation is perhaps the more subtle of the two to detect. As is implied by the heading, normal operation indicates that under most conditions the circuit functions according to the intent. Often, however, there

are modes that the circuit can be put into that result in the circuit being out of service. The detection of these states can be implemented far better by a computer than by a human, because in most cases they are detected only by exhaustive evaluation of all possible states and the implications of each.

Lock-up on its own implies two things: a memory of a past event, and a feedback path. Without both being present, lock-up cannot occur, only temporary state problems are possible. Since every sequential circuit has memory, the first criterion for potential lock-up will be met by most circuits. The second, however, is not quite as common, and can be used as an indicator of a possible problem.

Both types of feedback, direct (asynchronous) and delayed (synchronous - latched feedback) can result in lock-up, but have different characteristics. Lock-up due to asynchronous feedback is characterized by the lock-up state being entered immediately after the feedback. With delayed feedback, this does not necessarily hold true. The latter is more difficult to detect.

Rules for Operational Lock-up

The first step to detecting lock-up is to identify all feedback paths. (See Rule #02) What then remains is to determine whether each feedback path is capable of sustaining a lock-up state. This task can conceivably be implemented by a computer, but represents sufficient work to constitute a subject on its own. The identification of all feedback paths will, however, allow the designer to do a limited manual check specifically focussed on lock-up on only those paths identified.

This rule is really a subsection of the rule for feedback. It should be noted that this check will pick up certain circuits which are not problems such as cross-coupled gates forming RS latches. The question that must be answered in such a case is why did the designer choose to do this when he/she could have used a cell for the same purpose. There may be a good reason, but since it is an unusual thing to do, it is worth a quick check. If the same circuit were inside of a cell, the feedback would not be detected because the evaluator does not evaluate the circuitry within cells. The evaluator only has knowledge of the properties of cells, and checks the way they are interconnected.

4.0 RULE BASED EVALUATION

This chapter will demonstrate how the rules of the last chapter can be applied. Two examples are presented in detail. The first shows how an evaluator would go through a fairly simple circuit step by step. It includes tracking of some of the important program stacks, and shows how the circuit error is found. The second example represents a more complex circuit with less obvious problems. Again, a demonstration of how the evaluation would proceed is provided, but a compressed notation is used to highlight the path taken, and to focus on those actions which are key to the solution of the problem. As was mentioned in the last chapter, the evaluator is assumed to have access to certain information, both about the cells, and in particular about the timing.

It is important to remember that the evaluator is expected to operate on timing protocol information, and derive the

timing protocol for every node in the circuit, based on the input information. This subject is dealt with more fully in chapter 5.

4.1 BASIC EVALUATOR REQUIREMENTS

In order to be able to apply the rules, the evaluator must have similar information about the circuit, and the cells that make it up, as the designer would. This is a considerable amount of information, and is listed below.

INFORMATION NEEDED BY AN EVALUATOR

CIRCUIT

- an interconnect list
- Cells
 - which cells are used
 - Cell Characteristics
 - Classification of I/O points
 - Normal output
 - Tri-state output
 - Normal input
 - Clocked input
 - Clock input

- Jam input
- Bidirectional (with I & O classified)
- Compatibility
 - a list of compatible cells, if restrictions apply
- Logical transfer function
- Input Characteristics
 - All
 - Capacitance or a load factor
 - Static current (high and low)
 - Fan-in rating
 - Switching levels (CMOS, TTL, etc.)
 - Clock
 - Set-up and hold time
- Output Characteristics
 - Drive factor
 - Fan-out rating
 - Rise time (load assumption)
 - Fall time (load assumption)
 - Propagation delays (from each input)

TIMING

- Protocol (how many bits, relationships etc.)
- Timing detail (period, rise and fall time etc.)

This information is needed as a specification for inputs, can be optionally specified for the expected outputs, and must be derived by the evaluator for every node in the cir-

2

OF/DE

2

MICROD

cuit, based on the input information. See chapter 5 for more details.

In terms of circuit information, the previous list indicates that interconnect information is needed. Usually, this information is captured for the designer in the form of a schematic diagram. Similarly for an automated evaluation program, the essential schematic information must be presented in a machine readable form. The information required consists of the cells (circuit elements) used, and their interconnections. This is usually available to simulators in the form of a network interconnect list, "netlist" for short. A description of this, and a discussion of the program activity scheduler are provided next. An understanding of these is key to understanding how an evaluator works.

4.1.1 The NETLIST

A "netlist" is a list of all cells in a circuit which indicates to which node every I/O point on every cell is connected. This information is essential to the process of "pathfinding," described in the last chapter. With the exception of textual schematic information, the netlist con-

tains enough information to completely reconstruct the schematic diagram, logically.

In order to interpret this information, the computer must have some knowledge of the circuit elements: the logic transfer function; the number and types of I/O points; the order in which the I/O points are referenced in the netlist; and the cell electrical characteristics (e.g. propagation delay). As an example of how to interpret netlist information, consider the netlist for Example #1. This example shows a circuit with two flip-flops, an inverter, and a 2-input NOR gate (Figure 17 on page 100). Looking at one entry, "DFF" (short for D-type flip-flop), there are two instances of this cell type: G2 (gate number 2), and G3. The node list for G2 is VDD,VSS,N1,,INPUT,N2. What is not shown, but must be known by the computer, is the cell I/O order for this cell, which is VDD (power), VSS (ground), Q (main output), Q- (inverting output), D-input, and Clock-input. Note that the Q- output is not connected, and is shown in the node list as a blank. Note also that the naming of the gates is somewhat arbitrary, and bears no real significance to the circuit's function. Having access to all of this information, it is possible for the program to determine the next node for analysis, since it can easily tell what nodes are connected to what cell. It is not, on the other hand,

easy to determine which cells are connected to a given node; this is the function of the FANOUT list.

4.1.2 The FANOUT List

The fan-out list shows each node in the circuit, and all of the cells (and I/O points) that are connected to it. These are not listed in any particular order. With this information, the program can readily determine all cells that are connected to a given node, so that it can determine which cell I/O point on the node is the next to be processed. Consider the FANOUT list corresponding to the netlist of the previous example (Figure 17 on page 100). Note that there is no new information here, just a reordering and reformatting of the information. The FANOUT list, in fact, is derived from the netlist.

Anything designated "external" represents a connection at the circuit boundary. This need not be the total circuit boundary, but only the boundary of the circuit segment being analyzed. In each line of the FANOUT list, a node name is followed by a list of cell I/O points. The notation used here is somewhat arbitrary, but illustrates the idea. Consider, for example, node "N2." N2 connects four cells to-

gether: G1, G2, G3, and G4. The cell type, and the cell I/O point on the cell, are identified by the information in the brackets. G2, for example, is a D-type flip-flop, and is connected to N2 at cell I/O point 6. Cell I/O point 6 is the sixth element in the node series (found in the netlist), and this is the CLOCK input. Recall that the cell I/O list for a DFF cell goes: VDD, VSS, Q, Q-, INPUT, and CLOCK (the sixth element). It is common practice for the VDD and VSS points to be listed first, followed by the cell output(s), and then their inputs. So the inverter (INV) has the cell I/O list: VDD,VSS,OUTPUT,INPUT. Similarly, the list for the NOR2 is: VDD,VSS,OUTPUT,INPUTA,INPUTB. With this information, the computer can now determine and track both the next node, and the next cell I/O point.

4.1.3 Pathfinding

The rules described in chapter 3 can be applied at any point in a circuit, and their validity is independent of the path taken to get there. In terms of evaluation efficiency, however, there are preferred methods of going through a circuit, and these tend to be fairly independent of the circuit structure, or function. In finding paths through logic circuits, there are two questions that must be answered numer-

ous times: which node should be evaluated next (start up is a special case), and which I/O point on the node should be checked next?

A "path" is the route taken by a signal in getting from the input to a circuit to any output. In circuit evaluation, a path is followed from the original input to any output until all nodes along the way have been evaluated. Should the path evaluation not result in coverage of all circuit nodes, a new circuit entry point should be found, and the process repeated.

The rules that follow represent the basic decision-making process that a designer would go through in performing a circuit audit. These rules, while a valid approach to path-finding, are by no means the only, or necessarily the optimum, way to go through a circuit; this is a subject on its own, and has been studied extensively for applications in other CAD tools such as timing verifiers. The following rules are presented to convey to the reader that pathfinding can be a simple procedure.

4.1.3.1 Entry Nodes

Rule #P1 - Entry Node

Entry node selection is made on the basis of frequency, pulse width, and signal path type (clock, pulse, D.C., etc.) in the following way:

- If a clock exists, choose the highest frequency.
- If not, choose the highest frequency pulse.
- If more than one clock or pulse exist of the same frequency, choose the shortest high time.
- If no clock or pulse type signals exist, a timing analysis cannot be done. The evaluation will be limited to category I checks.

The highest frequency clock will have the highest potential activity, and will most often be involved in the generation of a large number of signals. Therefore, it is a logical place to start an evaluation.

4.1.3.2 Next Nodes

Rule #P2 - Next Node

- Choose the first cell in the FANOUT list for the current node, and go to the first output for that cell.
- If the next-node-search is due to insufficient information from a previous check, stack the current pointers (node, gate, cell I/O point, tests done) and determine which nodes are required to complete the check. If information about more than one node is required, choose: (1) the node with the lowest frequency signal, or (2) the first node in the list, if (1) is unknown.
- If, as a result of a previous backward search for information, there is a partially evaluated node, return and try to complete the evaluation.

4.1.3.3 Next Cell I/O Points

Cell I/O points are the I/O points for circuit elements (e.g. for a D-type flip-flop the cell I/O points would be D, Q, Q-, CK, and perhaps Reset). Each node represents the interconnection of a number of such cell I/O points. So when evaluating a node, the program is really evaluating a series of cell I/O points.

Rule #P3 - Next Cell I/O Point

- Choose the cell I/O from the FANOUT list for the current node in the order that they appear in the list, except that output cell I/O points should be evaluated first.

4.1.4 The Scheduler

The scheduler is part of the program that tracks where, in the evaluation process, the program is. This includes remembering which checks have only been partially completed, and which are finished. This same function is commonly per-

formed mentally when a person conducts a manual circuit audit. Five things must be tracked from check to check: the active node, cell I/O point, those checks remaining for the active cell I/O point, those nodes that have been fully checked, and a list of node evaluations that are not complete (i.e. due to a branch in the evaluation taking place before the node evaluation was complete). The first three are required to uniquely identify where in the circuit the evaluator is operating, and what checks remain. The latter is to track the branches that have taken place along the way, so that a traceback can be done to complete the work later.

4.1.5 Circuit Initialization

In evaluation, it is not necessary to explicitly define the start-up state of the circuit, since the evaluator assesses performance during start-up, as part of its function. If a reset is naturally used in the circuit, then this information should be provided as part of the timing input specification. If no reset is normally used, none need be specified. The evaluator considers a reset signal to be just another signal. Part of the function of the program is to propagate the input timing protocol through the circuit.

Therefore, initial operating conditions can be determined as the timing paths are analyzed. Most circuits do not require a reset to operate, but many require resetting to operate as intended. If no reset is specified, and none is required in operation, the evaluator should be able to figure out the start-up conditions from only the input timing information. If it cannot, there is either a circuit error, or missing input information. This is unlike simulation, in which one is required to initialize the circuit to reduce the simulation run time. If not performed, simulations can be forced to operate on 'X' states for large amounts of time before the circuit clears itself naturally. This usually does not yield valuable data.

4.2 EVALUATION EXAMPLE #1

The circuit and timing shown next (Figure 17 on page 100) results in a glitch being produced on the output node. Since it is not known what this circuit is connected to, it must be assumed that this glitch represents a potential problem, and should be flagged. The circuit schematic is followed by three pages of table which details the evaluation process for this circuit. An explanation of the information in the

table, and the evaluation process for this example follows the table.

SCHEMATIC

Cell Type	Cell Name	Node List	NETLIST
External	Circ1	VDD,VSS,OUTPUT,CLOCK,INPUT	
INV	G1	VDD,VSS,N2,CLOCK	
DFF	G2	VDD,VSS,N1,,INPUT,N2	
DFF	G3	VDD,VSS,N3,,N1,N2	
NOR2	G4	VDD,VSS,OUTPUT,N2,N3	

Node Name	Cell I/O List	FANOUT LIST
VDD	Circ1(External.1),G1(INV.1),G2(DFF.1), G3(DFF.1),G4(NOR2.1)	
VSS	Circ1(External.2),G1(INV.2),G2(DFF.2), G3(DFF.2),G4(NOR2.2)	
OUTPUT	Circ1(External.3),G4(NOR2.3)	
CLOCK	Circ1(External.4),G1(INV.4)	
INPUT	Circ1(External.5),G2(DFF.5)	
N2	G1(INV.3),G2(DFF.6),G3(DFF.6),G4(NOR2.4)	
N1	G2(DFF.3),G3(DFF.5)	
NOCONN	G2(DFF.4),G3(DFF.4),	
N3	G3(DFF.3),G4(NOR2.5)	

TIMING

Figure 17. Example Circuit #1 for Evaluation

ANALYSIS OF EXAMPLE CIRCUIT #1

ANALYSIS OF EXAMPLE CIRCUIT #1						
LOCATION ID		EVALUATION		CHECKS REMAINING		COMMENTS
Node	Cell I/O	Rule No.	Results	Checks Left	Incomplete	
-	G1	C1	OK	-	-	Check for non-preferred circuits.
-	G2	C1	OK	-	-	
-	G3	C1	OK	-	-	
-	G4	C1	OK	-	-	
-	-	C2	OK	-	-	Check no-connects. Choose an entry node. Choose a start cell I/O point. Check fan-in. " indeterminate switching. Choose next cell I/O point.
-	-	P1	CLOCK	-	-	
-	-	P3	G1(IN)	I1 I2	-	
-	-	I1	OK	I2	-	
CLOCK	G1	I1	OK	-	-	Choose next cell I/O point.
-	-	I2	OK	-	-	
-	-	P3	None left.	-	-	
-	-	-	-	-	-	
The node "CLOCK" is now evaluated.						
-	-	P2	N2	-	-	" next node. " " cell I/O point. Check fan-out. " for feedback. " for shorts. " for minimum pulse width. " output protocol. Choose next cell I/O point. Check fan-in. " indeterminate switching. " minimum pulse input. Choose next cell I/O point. Check fan-in. " indeterminate switching. " minimum pulse input. Choose next cell I/O point. Check fan-in. " indeterminate switching. Choose next cell I/O point.
N2	-	P3	G1(OUT)	01 02 03 04 08	-	
-	-	01	OK	02 03 04 08	-	
-	G1	OUT	-	03 04 08	-	
-	-	02	OK	04 08	-	" next node. " next cell I/O point. Check fan-out. " for feedback. Choose next node. " next cell I/O point. Check fan-out.
-	-	03	OK	-	-	
-	-	04	OK	-	-	
-	-	08	Not supplied.	-	-	
-	-	P3	G2(CLK)	I1 I2 I3	-	Choose next cell I/O point. Check fan-in. " indeterminate switching. " minimum pulse input. Choose next cell I/O point. Check fan-in. " indeterminate switching. " minimum pulse input. Choose next cell I/O point. Check fan-in. " indeterminate switching. Choose next cell I/O point.
-	G2	CLK	I1 OK	I2 I3	-	
-	-	I2	OK	I3	-	
-	-	I3	OK	-	-	
-	-	P3	G3(CLK)	I1 I2 I3	-	" next node. " next cell I/O point. Check fan-out. " for feedback. Choose next node. " next cell I/O point. Check fan-out.
-	-	I1	OK	I2 I3	-	
-	-	I2	OK	I3	-	
-	-	I3	OK	-	-	
-	-	P3	G4(INA)	I1 I2	-	" next node. " next cell I/O point. Check fan-out. " for feedback. Choose next node. " next cell I/O point. Check fan-out.
-	-	I1	OK	I2	-	
-	-	I2	OK	-	-	
-	-	P3	None left.	-	-	
Nodes "N2" and "CLOCK" are now evaluated.						
-	-	P2	OUTPUT	-	-	" next node. " next cell I/O point. Check fan-out. " for feedback. Choose next node. " next cell I/O point. Check fan-out.
OUT	-	P3	G4(OUT)	01 02 03 04 08	-	
-	G4	OUT	-	02 03 04 08	-	
-	-	01	OK	03 04 08	-	
-	-	02	Need more info	Push stack	-	" next node. " next cell I/O point. Check fan-out.
-	-	P2	N3	-	-	
-	-	P3	G3(Q)	01 02 03 04 08	-	
N3	-	Q	01 OK	02 03 04 08	-	

ANALYSIS OF EXAMPLE CIRCUIT #1

LOCATION ID		EVALUATION		CHECKS REMAINING		COMMENTS
Node	Cell I/O	Rule No.	Results	Checks Left	Incomplete	
N3	G3	Q	Need more info	Push stack	OUT, G4(OUT) 02 03 04 08 N3, G3(Q) 02 03 04 08	Check for feedback.
"	"	"	N1	-	"	Choose next node.
N1	-	P3	G2(Q)	01 02 03 04 08	"	" next cell I/O point.
"	G2	Q	OK	02 03 04 08	"	Check fan-out.
"	"	"	Need more info	Push stack	OUT, G4(OUT) 02 03 04 08 N3, G3(Q) 02 03 04 08 N1, G2(Q) 02 03 04 08	" for feedback.
"	"	"	P2	-	"	Choose next node.
INPUT	-	P3	G2(D)	I1 I2 I4 to I7	"	" next cell I/O point.
"	G2	D	I1 OK	I2 I4 I5 I6 I7	"	Check fan-in.
"	"	"	I2 OK	I4 I5 I6 I7	"	" indeterminate switching.
"	"	"	I4 OK	I5 I6 I7	"	" for clock inversion.
"	"	"	I5 OK	I6 I7	"	" set-up and hold.
"	"	"	I6 OK	I7	"	" for asynchronous data #1.
"	"	"	I7 OK	-	"	" for asynchronous data #2.
"	"	"	P3	None left.	"	Choose next cell I/O point.
Nodes "N2," "CLOCK," and "INPUT" are evaluated.						
"	"	"	P2	Pop stack	OUT, G4(OUT) 01 02 03 04 08 N3, G3(Q) 01 02 03 04 08	" next node.
If the stack contains something, and a new node is sought, it should be popped. This results in tracing back along the path requiring the information.						
N1	G2	D	OK	03 04 08	"	Check for feedback.
"	"	"	OK	04 08	"	" for shorts.
"	"	"	08	-	"	" output protocol.
"	"	"	P3	G3(D)	"	Choose next cell I/O point.
"	G3	D	I1 OK	I1 I2 I4 to I7	"	Check fan-in.
"	"	"	I2 OK	I2 I4 I5 I6 I7	"	" indeterminate switching.
"	"	"	I4 OK	I4 I5 I6 I7	"	" for clock inversion.
"	"	"	I5 OK	I5 I6 I7	"	" set-up and hold.
"	"	"	I6 OK	I7	"	" for asynchronous data #1.
"	"	"	I7 OK	-	"	" for asynchronous data #2.
"	"	"	P3	None left.	"	Choose next cell I/O point.
Nodes "N2," "CLOCK," "INPUT," and "N1" are evaluated.						
"	"	"	P2	Pop stack	OUT, G4(OUT) 01 02 03 04 08	" next node.

ANALYSIS OF EXAMPLE CIRCUIT #1

LOCATION ID		EVALUATION		CHECKS REMAINING		COMMENTS
Node	Cell I/O	Rule No.	Results	Checks Left	Incomplete	
N3	G3	Q	OK	03 04 08	"	Check for feedback.
"	"	"	OK	04 08	"	" for shorts.
"	"	"	OK	08	"	Check minimum pulse width.
"	"	"	Not supplied.	-	"	" output protocol.
"	"	"	G4(IN8)	11 12	"	Choose next cell I/O point.
"	G4	IN8	I1 OK	12	"	Check fan-in.
"	"	"	I2 OK	-	"	" indeterminate switching.
"	"	"	P3 None left.	-	"	Choose next cell I/O point.
Nodes "N2," "CLOCK," "INPUT," "N1," and "N3" are evaluated.						
"	"	"	P2 Pop stack	03 04 08	-	" next node.
OUT	G4	OUT	O2 OK	04 08	-	Check for feedback.
"	"	"	O3 OK	08	-	" for shorts.
"	"	"	O4 ***WARNING***	08	-	" minimum pulse width.
WARNING	"	"	A glitch is present at G4(OUT). See timing for more detail.	-	-	" output protocol.
"	"	"	O8 ***WARNING***	-	-	Choose next cell I/O.
WARNING	"	"	A glitch is present at G4(OUT). See timing for more detail.	-	-	
"	"	"	P3 None left.	-	-	
"N2," "CLOCK," "INPUT," "N1," "N3," and "OUT" are evaluated.						
"	"	"	P2 No more nodes	-	-	" next node.
Check #C3 for Initialization Lock-up could be run at this point. It is optional.	"	"	END OF EVALUATION	-	-	

The preceding table shows the step-by-step activities of the evaluator in checking the circuit of Figure 17 on page 100. The first three columns show the current node, cell and I/O point that are being evaluated. The Rule No. column indicates which of the rules of the last chapter is being applied. The result of the check is shown in the next column. Columns six and seven represent two program stacks: the checks that remain for the current cell I/O point; and the list of those nodes which have been partially completed.

To summarize the evaluation of Example #1, the evaluation starts with a check of the cells: whether any are non-preferred, and if any non-connected inputs exist. Next the first entry node is found. The signal "CLOCK" is chosen because it is the highest speed clock signal in the circuit (the only one in this case). After checking the input characteristics of inverter G1, the analysis moves to the output of the inverter, following the "CLOCK" signal path, where each of the cell I/O points are checked. No problems are detected so the analysis moves to the next node: "OUT," which is the end of the path started on the "CLOCK" node. "OUT" cannot be fully evaluated because no information is yet available on one of the inputs to the gate producing it, "N3." The evaluator attempts to determine the missing information by moving back to "N3" while stacking the checks not yet conducted on "OUT." "N3" cannot be checked out

either because "N1" information is not available. Moving back to "N1" the evaluator finds that the D-input of the flip-flop (G2) has not yet been checked out and so goes back to node "INPUT." At this point, checks are stacked for "OUT," "N2," and "N1." The timing information for "INPUT" is supplied by the user so the analysis continues in a forward direction again. Once "INPUT" is fully checked out, the path back toward "OUT" is followed by popping the stack on outstanding checks and analyzing each.

In this example, the problem is a glitch on the output. Since the node "OUTPUT" is at the end of the path, the detection of the problem is at the end of the evaluation. Note that it is detected by two checks: 04, the minimum pulse width check; and 08, the protocol check. These should only result in a single warning to the user, but with reference to both tests that failed. In general, an evaluation can go to completion even if a problem is detected at the beginning.

4.3 EVALUATION EXAMPLE #2

The second example is considerably more complex than the first. It demonstrates problem detection on a circuit with

feedback, as well as for two independent errors. to de-emphasize the details of the program stack tracking, a compressed notation is used. If all checks for a cell I/O point are passed, there is only a single line entry stating ALL rules checked OK.

In this simple data scrambler circuit, Figure 18 on page 109, it is shown how the evaluator can deal with feedback. The same initial process of cell checking, used in the previous example, is done here. The entry node chosen is, once again, "CLOCK." After evaluating the "CLOCK" node the next node chosen corresponds to the output of the first cell in the FANOUT list for node "CLOCK." This is a settable D-type flip-flop (G1). Since the timing at the input to this flip-flop is not known, the analysis of node "N1" cannot be completed. Therefore, the remaining "N1" checks are stacked, and the next node choice is made. "LOAD" is chosen because it is the JAM input, with the lowest frequency (being the only one). The selection is made on the basis that the lowest frequency JAM input is likely to be an initialization, or reset, signal. When feedback exists, resets are often the only way to start up the loop. Checking the "LOAD" node at both cell I/O points, it is found that the set/reset are being released within $T_{critical}$ of the activating clock edge. One flag should be raised to the user citing two instances of violations for node "LOAD." The

clock timing, in such a case, should always be assumed to be dominant over a reset, or other JAM input, falling within $T_{critical}$ of its activating edge. That is usually what is intended in the design. This assumption is made in the timing shown in the figure (Figure 18). With "LOAD" checked, the stack is popped to see whether there is sufficient information to evaluate "N1." There is not, so the 'D' input timing is sought. This process of looking backward around the loop continues through "N2," and "N3" until the evaluator reaches "N1" again. At this point the feedback path is identified by comparing the current node to those in the "incomplete" stack. Given that a loop has been established, the evaluator now must start up to do the timing analysis. This is done by looking for a JAM input ("LOAD" in this case) to cause a reset. The timing for the first node is then assumed to be that provided by the reset, and on this basis the reset of the loop is evaluated (by popping the nodes back off the stack). When the loop has been completed, the resulting timing must be compared to the initial assumption to insure that there is no inconsistency. If there is one, a flag must be raised warning the user that the loop could not be initialized.

Once the loop has been evaluated, a new entry node is sought ("INPUT"). This information is then combined with the derived loop timing to generate the "OUTPUT" timing. At this

point the last problem is detected, that of an incorrect protocol match. Here, it is assumed that the user has provided the program with the expected timing protocol. Although it is not totally obvious from the diagram, one can see that the output consists of some pulses that are half clock widths, and some that are full clock cycles wide. The reason for the half widths is because the "INPUT" timing, and the scrambler timing are shifted in time by one half clock cycle. If the user only specified that the output signal was related to the clock signal on the rising edge, and nothing more, the evaluator would detect the problem and flag it.

THE NETLIST (usually generated from the schematic)

Cell Type	Cell Name	Node List
External	Circ2	VDD, VSS, OUTPUT, CLOCK, LOAD, INPUT
SDF	G1	VDD, VSS, N1, N2, CLOCK, LOAD
SREG3	G2	VDD, VSS, N3, N4, N1, CLOCK, LOAD
EXOR2	G3	VDD, VSS, N2, N3, N4
EXOR2	G4	VDD, VSS, OUTPUT, N2, INPUT

THE FANOUT LIST (derived from the NETLIST)

Node Name	Cell I/O List
VDD	Circ2(External.1), G1(SDFF.1), G2(SREG3.1), G3(EXOR2.1), G4(EXOR2.1)
VSS	Circ2(External.2), G1(SDFF.2), G2(SREG3.2), G3(EXOR2.2), G4(EXOR2.2)
OUTPUT	Circ2(External.3), G4(EXOR2.3)
CLOCK	Circ2(External.4), G1(SDFF.6), G2(SREG3.7)
LOAD	Circ2(External.5), G1(SDFF.7), G2(SREG3.8)
INPUT	Circ2(External.6), G4(EXOR2.5)
N1	G1(SDFF.3), G3(SREG3.6)
NOCNN	G1(SDFF.4), G2(SREG3.5)
N2	G1(SDFF.5), G3(EXOR2.3), G4(EXOR2.4)
N3	G2(SREG3.4), G3(EXOR2.4)
N4	G2(SREG3.3), G3(EXOR2.5)

CELL I/O LIST (known to the program, not derived)

SDFF	VDD, VSS, Q, Q, D, CLK, SET
SREG3	VDD, VSS, Q3, Q2, Q1, IN, CLK, RESET
EXOR2	VDD, VSS, OUT, INA, INB

Figure 18 Example Circuit #2 for Evaluation

ANALYSIS OF EXAMPLE CIRCUIT #2

LOCATION ID		EVALUATION		CHECKS REMAINING		COMMENTS
Node	Cell I/O	Rule No.	Results	Checks Left	Incomplete	
N1	G1	Q3	OK	04 08	"	Check for shorts.
"	"	04	OK		"	" minimum pulse width.
"	"	08	Not supplied.			" output protocol.
"	"	P3	G2(IN)	I1 I2 I4 to I6	"	Choose next cell I/O point.
"	G2	IN	ALL OK		"	All input checks are OK.
"	"	P3	None left.		"	Choose next cell I/O point.
"	"	P2	Pop stack		"	" next node.
N3	G2	Q2	ALL OK		N1,G1(Q) 01 02 03 04 08 N2,G3(OUT) 01 02 03 04 08	All output checks are OK.
** NOTE: FEEDBACK WILL ONLY BE DETECTED ON ONE NODE IN THE LOOP. ** BECAUSE THE TIMING ANCESTRY WILL NOT YET EXIST FOR THE OTHER NODES.						
"	"	P3	G3(INA)	I1 I2	"	Choose next cell I/O point.
"	G3	INA	ALL OK		"	All input checks are OK.
"	"	P3	None left.		"	Choose next cell I/O point.
"	"	P2	Pop stack		"	" next node.
N2	G3	OUT	ALL OK		"	All output checks are OK.
"	"	P3	G1(D)	I1 I2 I4 to I6	N1,G1(Q) 02 03 04 08	Choose next cell I/O point.
"	G1	D	ALL OK		"	All input checks are OK.
"	G4	INA	P3 G4(INA)	I1 I2	"	Choose next cell I/O point.
"	"	ALL OK			"	All input checks are OK.
"	"	P3	None left.		"	Choose next cell I/O point.
"	"	P2	Pop stack		"	" next node.
N1	G1	Q	Checks already done. Verify that timing protocol assumption			checks out after looping.
"	"	P2	PROTOCOL CHECKS OK			
"	"	P1	No more nodes			Choose next node.
"	"	P3	INPUT			" new entry node.
INPUT	"	G4	G4(INB)	I1 I2		" starting cell I/O point.
"	"	ALL OK				All input checks OK.
"	"	P3	None left.			Choose next cell I/O point.
"	"	P2	OUTPUT			" next node.
OUT	"	P3	G4(OUT)	01 02 03 04 08		Check fan-out.
"	G4	OUT	01 OK	02 03 04 08		" next cell I/O point.
"	"	02 OK		03 04 08		" for feedback.
"	"	03 OK		04 08		" minimum pulse width.
"	"	04 OK				" output protocol.
"	"	08	Protocol violation!			
** THE OUTPUT PROTOCOL DOES NOT MATCH THE SUPPLIED SPECIFICATION **						
"	"	P3	None left.			Choose next cell I/O point.
"	"	P2	No more nodes			" next node.
Check #C3 for Initialization Lock-up could be run at this point. It is optional.						
"	"	-	END OF			
"	"	-	EVALUATION			

As can be seen by the two examples of this chapter, much of the power of the evaluator comes from the ability of the user to specify timing protocols including the relationships between different signals. The requirements for this, and a method for timing protocol specification, are described in the next chapter.

5.0 TIMING REPRESENTATIONS

To evaluate a circuit, it is necessary that a method of describing timing be defined for three purposes: to specify the input and expected output signals; to represent the derived timing of the internal circuit nodes; and to represent the actual output timing to the user. The requirements of the first two differ significantly from evaluator to simulator. In order that the reader may better understand the nature and extent of these differences, a short explanation of the requirements of each is presented.

5.1 A TIMING DESCRIPTION LANGUAGE

Representing timing to, and in, a simulator requires that one be able to identify where "events" occur. A pattern of

ones and zeroes must be defined for each input, and, depending on the simulator, rise and fall times may be specified, or assumed to be ideal. From a computing perspective, the simulator only cares about transitions from one logic state to another, because these define "events." The only relationships that exist are by virtue of the fact that events happen to occur on the same input node. The concepts of "bit stream," and "timing protocol" do not exist for a simulator.

As has been stated previously, the process of evaluation requires the "timing protocol" to be represented. The timing protocol is much like what one would expect to see on a component data sheet. It includes the format of the timing (where the bits are), the default value of the signal, the basic characteristics of the timing (period, rise time, fall time etc.), and the relationships that exist between the signal and other signals. Without this information, most of the timing dependent evaluation cannot be done. In addition to specifying the input and expected output signals in this way, the evaluator must be able to operate on the data to calculate the resulting timing protocol on every node in the circuit.

A method of describing the timing protocol for both requirements noted above, has been developed. It provides a flexible method of specifying input and output signals, and can

also be used to represent timing within the circuit. It should be noted that timing can be specified for any node, whether internal to the circuit or on the periphery. However, the specification is only a forcing function on inputs to the circuit. For all other nodes, the specification will be compared to the actual timing calculated by the evaluator to determine whether there is a protocol violation. This method of representing timing is called A Timing Description Language (ATDL). It provides a much more compact means of defining timing than is available using strings of events, and therefore will require much less storage for the equivalent amount of information.

5.1.1 ATDL Syntax

The specification of generalized timing requires that an arbitrary sequence of logic events be described unambiguously. In addition to this, it is frequently necessary to describe a condition which would normally be considered to be analog, i.e. that cannot be described by an idealized logic waveform. A system of specifying such data, both synchronous and asynchronous, is described in the following section. Examples of its use are presented after the syntax description.

The basic syntax for specifying generalized timing data in ATDL follows:

ATDL SYNTAX

```
Signal_name Node_type Signal_type
      TIMING: TP=_ TR=_ TF=_ <PW1=_ ... PWN=_> <+>
      PROTOCOL: Default <<Relationship Pulsetrain> <+>
               <.....>....>
```

This format permits the specification of any logic sequence with any number of relationships to any other signal, including asynchronous signals. For purposes of clarity, the syntax is shown broken into two sections (timing, and protocol), but may be implemented in many ways. The parameter definitions follow.

SIGNAL IDENTIFICATION

- **Signal_name** - is the name of the signal being specified. The attributes of the signal that follow can be assigned to more than one signal of a bus as illustrated. (e.g. XYZ(0:7) are bits 0 to 7 of bus "XYZ," and BUS(5) is bit

5 of bus "BUS"). Note: there is no distinction between a node name and a signal name.

- **Node_type** - has the value of "I," "O," "I/O," or "INT" to indicate whether the signal represents an input (I), an output (O), a bidirectional pin (I/O), or an internal node (INT).
- **Signal_type** - specifies the type of the signal as being one of the following: CLOCK, PULSE, LEVEL, or SPECIAL. The SPECIAL designation refers to a signal which cannot be completely specified using the above method, such as an analog signal. Use of this designator permits expansion of the types for future use. It may also allow the user to identify something unique about a signal.

SIGNAL TIMING

- **TP** - is the period of the signal. (Not applicable with the "LEVEL" option.) A minimum and/or maximum specification can be entered by appending MAX or MIN to the number (e.g. TP=200NSMAX TP=180NSMIN). If a MAX or MIN only is specified, or if no MIN/MAX suffix is present, then the number specified will be used as both maximum and minimum. This parameter, whether an entire bit

stream is specified or just one pulse, indicates how often the signal will repeat. Alternatively, the period can be expressed in terms of the period of another signal. For example, an input signal might occur every seven cycles of "CLOCK." This could be represented by $TP=7 \times \text{CLOCK}$. When this format is used, the relationship must be further defined in the protocol section of the syntax. By permitting this type of specification, the actual timing can be more accurately represented. This becomes particularly important when signals are "modulated" to determine their effect on the circuit.

Note: for a PULSE signal, with a sequence defined such that the sequence time exceeds the specified period, TP, the TP value is what dictates the period, and a flag should be raised indicating the discrepancy.

- TR - is the rise time of the signal. MAX and MIN suffixes can also be used with these numbers. If no suffix is provided, MAX is assumed, and in the case of either no suffix, or a MAX number only being supplied, a minimum value of zero is assumed.
- TF - is the fall time of the signal. The same options and conditions described for the rise time apply for this specification.

- "+" - is a line continuation character.
- PW - is the pulse width of the signal. The pulse width refers to the width of one "bit" of a signal, whereas the period (TP) refers to how often the entire signal will repeat. In the case of a CLOCK signal, this parameter indicates the clock pulse width; if normally low, it would indicate the high time, and vice versa. The options and conditions for the use of this parameter are the same as those of the TP (period) parameter, with one exception: the PW specification for the PULSE signal is redundant when a Relationship specification is used. As in the case of TP, it is possible to define PW in terms of the period of another signal (e.g. clock periods, $PW=3*CLOCK5$).

It is also possible to completely define a pulse by using the relationship field. PW must be defined either via a relationship field, or by assigning a value to PW.

By specifying a series of PW numbers (e.g. $PW1=50ns$, $PW2=52ns$, $PW3=49ns$) the pulse width of sequential data bits can be modulated. If this feature is used on a "CLOCK" signal, it will represent jitter in the duty

cycle of the clock (phase modulation). Phase jitter can also be bounded by simply using the minimum and maximum options on the pulse width. Similarly, frequency modulation can be represented by using these options of the period specification. If the PW string is used on a "PULSE" type signal, the number of PW numbers must correspond with the number of bits indicated in the relationship field. In such a case, the pulse width numbers correspond to the pulse width of each bit in the stream.

SIGNAL PROTOCOL

- Default - indicates the normal state of the signal, either normally high (NH), normally low (NL), normally cannot be predicted (NX), or normally high impedance (NZ). In the case of a CLOCK signal, this indicates an initial condition which allows the interpreter to assess the phase relationship between clocks.
- Relationship - is used to specify that Signal_name should remain in the current state until this condition is fulfilled. It is primarily intended for use with synchronous clocks to indicate how many cycles, and which edge, pass before the next state change occurs in the

pulse train. More than one Relationship segment can be specified before the Pulsetrain is defined (i.e. multiple relationships which must all be satisfied before a state change). The syntax for this part of the specification is as follows: Number*Relative_name R/F/Z TD=.

- Number - indicates the number of Relative_name edges to be counted before executing the next state change. If no Number is supplied, '1' is assumed. In the case of a CLOCK signal, the Number is redundant, since state changes occur on every cycle by definition.
- * - delimits "Number."
- Relative_name - is the name of the signal which is to be awaited before changing the state of the signal (Signal_name).
- R/F/Z - indicates that Signal_name is initiated by the rising (R) or falling (F) edge of the CLOCK or PULSE signal, or on the transition to a high impedance state (Z) from either a logic "0" or "1." R and F can be expanded to Rising and Falling, or abbreviations of these, to make the expressions more readable.

- TD - specifies the delay of the Signal_name state change calculated from the specified edge (R/F) of Relative_name. If multiple Relationships are defined, TD need not be specified every time, just the last time.

By using the Relationship, it is possible to indicate the exact position of timing edges without specifying the pulse width. This is important because it provides a means of maintaining the precise relationship between an edge and the signal that causes it, instead of a statement of what should happen. Such relationship information may be key to evaluating a particular piece of circuitry, and therefore must be available to the program. When a Relationship specification occurs, the PW becomes redundant.

- Pulsetrain - is a pulse sequence (only meaningful with the PULSE type) which follows the initial edge of the signal. This sequence is expressed in terms of bits as either a 1, 0, X (unknown, or don't care), or Z (a high impedance). Each of these bits will have the characteristic risetime, falltime, and pulse width specified in the main signal definition. There is no practical maximum length of the Pulsetrain, but at least one entry must be present in all PULSE specifications. Note 1: In

the case of a CLOCK signal, a Relationship entry indicates the timing relationship for state changes, and no Pulsetrain is needed. Note 2: If no Pulsetrain is specified for a PULSE signal, a single pulse is assumed. After the first Relationship condition is satisfied, a state change will occur, and after the next, the signal will be returned to its Default state until the sequence is to repeat.

Note that a single node (signal) may have both an input and an output specification. This would be required if describing a bidirectional line.

The balance of this chapter provides examples of how this syntax can be used to describe real timing situations. Five examples are given, and they are shown in order of increasing difficulty. A short summary of the syntax short forms is given in Appendix A.

Timing Example 1 (Simple Clock Inputs)

ATDL Representation

```

CLOCK1 I CLOCK
      TIMING: TP=1US TR=10NS TF=10NS PW=490NS
      PROTOCOL: NL
CLOCK2 I CLOCK
      TIMING: TP=2*CLOCK1 TR=10NS TF=10NS PW=1*CLOCK1
      PROTOCOL: NL CLOCK1 Rising TD=2NS
CLOCK3 I CLOCK
      TIMING: TP=2*CLOCK2 TR=10NS TF=10NS PW=1*CLOCK2
      PROTOCOL: NH CLOCK2 Falling TD=0

```

Figure 19. Timing Example #1

In this example, three typical clock signals are shown. The default (NL or NH, normally low or high), indicates the initial condition of the signal. This information is used to determine initial conditions when necessary, and to determine the relative clock phasing. For example, CLOCK3 is indicated as NH (normally high); on the first falling edge of CLOCK2 it will go from a high to a low state, with no delay.

Interpreting the ATDL representation, the first signal/node, CLOCK1, is defined as an input, I, and of type CLOCK. The type designation tells the interpreter how to decipher the detail that follows. The CLOCK1 signal is further defined

as having a period of one microsecond with a 50/50 duty cycle (10 nanosecond rise and fall times with 490 nanosecond high and low times). The other two signals are defined in terms of CLOCK1. CLOCK2, for example, is defined as changing state two nanoseconds ($TD=2NS$) after the rising (R) edge of CLOCK1. Similarly, CLOCK3 changes state coincident in time with the falling (F) edge of CLOCK2. This timing represents the first two bits of a synchronous counter driven by CLOCK1.

Timing Example 2 (Clock Synchronous Pulses)

ATDL Representation

```

CLOCK1 I CLOCK
      TIMING: TP=1US TR=10NS TF=10NS PW=490NS
      PROTOCOL: NL
INPUT1 I PULSE
      TIMING: TP=7*CLOCK1 TR=10NS TF=10NS
      PROTOCOL: NH CLOCK1 F TD=2NS
INPUT2 I PULSE
      TIMING: TP=7*CLOCK1 TR=10NS TF=10NS
      PROTOCOL: NL INPUT1 F CLOCK1 R TD=2NS 1 +
                1*CLOCK1 F TD=2NS 0 +
                3*CLOCK1 F TD=2NS 1 +
                1*CLOCK1 R TD=2NS 0

```

Alternate INPUT1 definition

```

INPUT1 I PULSE
      TIMING: TP=7US TR=10NS TF=10NS PW=990NS
      PROTOCOL: NH CLOCK1 F TD=2NS

```

Figure 20. Timing Example #2

In this example, the same clock as used in the previous example is shown, along with two PULSE type signals. INPUT1 is normally high (NH), and occurs every seven clock cycles. It is related to CLOCK1 in the following way: go to logic '0' two nanoseconds after the falling edge of CLOCK1 (CLOCK1 F TD=2NS), wait for one falling edge of CLOCK1 and TD (2NS)

later return to the default value ('1') -- implied. Although INPUT2 appears to be simply at twice the frequency of INPUT1, it is not. According to the definition, INPUT2 also has a period of seven clock cycles, is normally low (NL) and is related to INPUT1. The relationship section tells the interpreter what to do in order to accurately reproduce the timing. In this case, it states that INPUT2 goes high two nanoseconds after the first rising CLOCK1 edge following the falling edge of INPUT1 (INPUT1 F CLOCK1 R TD=2NS 1). It is then to wait until the next falling edge of CLOCK1, and go low after two nanoseconds (1*CLOCK1 F TD=2NS 0). Following this, a second pulse is defined which is out of clock phase with the first pulse. The specification tells the interpreter what other signals the current one is related to, how long to wait after a particular edge, and what state to then go to. In the event that the last relationship specification did not return the signal to its default state, it would be returned there automatically after one more iteration of the last relationship.

Two methods of defining INPUT1 are also shown. The latter is a valid way of defining it, but loses the fact that the rising pulse edge is also related to the clock. With this specification, the evaluator would always consider the pulse to be 990 nanoseconds wide, even if the clock were narrowed, which does not fully represent reality.

Timing Example 3 (Synchronous Bidirectional Signal)

ATDL Representation

```

CLOCK I CLOCK
      TIMING: TP=1US TR=10NS TF=10NS PW=490NS
      PROTOCOL: NL
XLINE I PULSE
      TIMING: TP=1MS TR=10NS TF=10NS
      PROTOCOL: NH CLOCK R TD=2NS 0XXXXXXXXXX
XLINE O PULSE
      TIMING: TP=1MS TR=10NS TF=10NS
      PROTOCOL: NH XLINE F 12*CLOCK R TD=2NS Z  +
                  1*CLOCK F TD=2NS XXXXXXXXXXX

```

Figure 21. Timing Example #3

This timing example, shows how a bidirectional line can be defined. Two specification lines are used for one signal (XLINE): one for the input segment, and one for the output. Note that the input spec (the first) is related to the rising edge of the CLOCK signal, and the output spec is related to the falling edge. As in the previous example, the XLINE signal will return to its default condition one cycle of the last relationship after the last pulse in the string is completed (i.e. 2 nanoseconds after the next falling edge of CLOCK following the one that created the last pulse).

Timing Example 4 (Asynchronous Data Stream) ATDL Representation

```

CLOCK I CLOCK
      TIMING: TP=1US TR=10NS TF=10NS PW=490NS
      PROTOCOL: NL
DATA I PULSE
      TIMING: TP=1000US TR=10NS TF=10NS PW=16US
      PROTOCOL: NH 0XXXXXXXXX

```

Figure 22. Timing Example #4

This diagram, showing the receive portion of a UART signal, is intended to illustrate a much higher frequency clock than the data rate. The data is not synchronized to the clock in any way, although the a derivative of the clock can be used to recover the data. The data is not shown as related to the clock, therefore, they are asynchronous to one another. Since the ATDL syntax explicitly allows for the definition of relationships, signals are made asynchronous by the absence of a relationship specification.

Timing Example 5 (A 16 bit Micro Bus)

ADD = Address, RES = Bus REServed for data,
 DVA = Data valid, - = High Impedance (Z-state)

ATDL Representation

```

CLOCK1      I CLOCK
             TIMING: TP=125NS TR=15NS TF=15NS PW=47NS
             PROTOCOL: NH
ALE          O PULSE
             TIMING: TP=5*CLOCK1 TR=15NS TF=15NS
             PROTOCOL: NL CLOCK1 F TD=5NS 1 +
                   1*CLOCK1 R TD=10NS 0
READ        O PULSE
             TIMING: TP=20*CLOCK1 TR=15NS TF=15NS
             PROTOCOL: NH ALE F CLOCK1 R TD=10NS 0 +
                   3*CLOCK1 F TD=5NS 1
WRITE       O PULSE
             TIMING: TP=20*CLOCK1 TR=15NS TF=15NS
             PROTOCOL: NH READ R ALE F CLOCK1 R TD=10NS 0 +
                   2*CLOCK1 R TD=10NS 1
BUS(0:15)   O PULSE
             TIMING: TP=20*CLOCK1 TR=15NS TF=15NS
             PROTOCOL: NZ ALE R TD=20NS X +
                   ALE F TD=20NS Z +
                   READ F CLOCK1 F TD=10NS X +
                   CLOCK1 F TD=30NS XZ +
                   READ R ALE R TD=20NS X +
                   WRITE F TD=0 XXXZ
  
```

Figure 23. Timing Example #5

The difficulty in representing this timing is that two largely independent activities can occur on the same bus: a read, and/or a write. In practice, it is not physically possible for these two things to overlap. If it were, the timing specification would be different. This timing shows a read and a write occurring at regular intervals on the bus (an approximation to reality) and has included a relationship between the two to prevent them from occurring at the same time.

Representing timing on a shared bus is one of the most difficult cases.

5.1.2 Using the ATDL Syntax

The previous examples demonstrate how the ATDL syntax can be used to represent timing for I/O specification. What has not yet been shown is how the same technique can be used for internal node timing as well. There is no difference between the way ATDL is applied to internal nodes and external nodes. As an example, a circuit is shown in Figure 24 on page 133 with its inputs and expected output specified with ATDL. Based on these inputs, the node timing would be derived by the evaluator. The results of this are shown below

the specification. This is also an ATDL representation. It is not suggested that the form shown here is optimal for storage in the computer; it is presented in this format to make it more readable.

In order for this description method to be of any real use, it must be possible for the evaluator to compare its calculated output timing with the expected output provided by the user. To do this, a more convenient form of the information is needed. For illustration purposes only, both sets of timing are converted back into a graphical format for comparison. Note that the glitch on the actual output will be flagged by a protocol check of the actual against the specified output. In practice, the program would have to convert both descriptions into a one and zero format to do a binary compare. This need only be done for two signals at a time, however. Whether this format, or some other, is chosen, the principle is that the two protocols to be compared are brought down to a common form, and compared there. Since it is possible to represent the same timing in more than one way, comparing the ATDL forms is not sufficient.

SCHEMATIC

USER SPECIFICATION

CLOCK I CLOCK
TIMING: TP=1US TR=10NS TF=10NS PW=490NS
PROTOCOL: NL
INPUT I PULSE
TIMING: TP=12*CLOCK TR=10NS TF=10NS
PROTOCOL: NH CLOCK F TD=2NS
OUT O PULSE
TIMING: TP=12US TR=10NS TF=10NS
PROTOCOL: NL INPUT R CLOCK R TD=5NS 1 +
CLOCK F TD=5NS 0

EVALUATOR DERIVED

N1 INT PULSE
TIMING: TP=12*CLOCK TR=5NS TF=5NS
PROTOCOL: NH INPUT F CLOCK R TD=2NS
N2 INT CLOCK
TIMING: TP=CLOCK TR=5NS TF=5NS PW=490NS
PROTOCOL: NH
N3 INT PULSE
TIMING: TP=N1 TR=5NS TF=5NS
PROTOCOL: NH N1 F CLOCK R TD=2NS
OUT O PULSE
TIMING: TP=N3 TR=5NS TF=5NS
PROTOCOL: NL N3 F TD=2NS 1 +
CLOCK F TD=2NS 0 +
CLOCK R TD=2NS 1 +
N3 R TD=2NS 0

GRAPHICAL REPRESENTATION OF THE TIMING

Figure 24 Example Circuit #1 with ATDL Timing

Note that the ATDL format does not represent the glitch as a special signal. In capturing the glitch in ATDL, the signals that cause it, and their precise relationship to one another are also captured. ATDL does not support a method of representing the glitch as a special case. It is up to the evaluator to decide whether the event, the glitch, represented in the protocol is a problem or not.

6.0 DISCUSSION AND CONCLUSIONS

The material in this thesis has focussed on some of the practical problems associated with implementing an automated evaluation system. A significant question remains: is it practical to implement such a system? There are several aspects to this question which will be discussed in the following paragraphs.

6.1 PROGRAM CONSIDERATIONS

The evaluation program differs significantly from simulation programs in four areas: its need for an ATDL interpreter, the processing load on the computer, the scheduling of ac-

tivity within the program, and the output processing requirements.

The main question surrounding an interpreter is -- can the ATDL syntax unambiguously represent all timing? Although it is difficult to prove this, the five examples of the previous chapter cover timing examples from the very simple, to fairly complex timing, and demonstrate that the signal can be represented. The syntax already breaks the signal down into timing and protocol, which reduces the signal to a fairly elemental form. Therefore, it is unlikely that the syntax itself is inherently ambiguous. ATDL does, however, provide a mechanism for describing very general timing, including the relationships between signals. Given this flexibility, it is certainly possible to misrepresent timing. As a result, an ATDL interpreter should have the ability to recognize nonsensical definitions.

The area of processing load in evaluation versus simulation is a contentious one. This is related to the concept of processing timing protocols versus ones and zeroes processing. Consider the make-up of the timing protocol. The signal is broken down into a timing section, and a protocol section. The information in the timing section must also be known and processed in simulation. The protocol section contains two basic elements: the timing relationships, not

available in simulation, and the data pattern. Data patterns are simply general arrangements of ones and zeroes, with the added freedom of describing a bit as a don't-care. The processing load, therefore, must be similar to that of simulation. The only real area of difference between the evaluator and the simulator lies in the processing of the timing relationships. The main load is associated with keeping lists of relationships for the signals on each node, and determining how they are propagated through logic circuits. The relationships, however, follow the logic transfer function in just the same way as the timing signals do. Since the timing protocol is a general representation of the signal, much more circuit analysis can be done on it than on individual ones and zeroes used in simulation. Therefore, although the processing load for individual bits is greater for evaluation, the number of vectors required to analyze a circuit is much smaller; the overall processing requirements of an evaluator should be much less than for the simulations required to obtain the equivalent information.

Another area of concern in developing an evaluator is its ability to operate on asynchronous timing. This, in fact, is a capability which is built into simulators by virtue of the fact that they are event driven. The "scheduler" part of the simulation program inherently has the ability to keep track of asynchronous events. Therefore, it is possible to con-

ceive of an evaluator which uses a modified event driven simulation engine, to handle the protocols. This approach to implementing an evaluator provides a way to take advantage of the extensive work done in the development of simulators, thus minimizing the incremental effort in developing the evaluation capability.

The final area is that of the output processing. As mentioned in the last chapter, it is necessary to compare the derived timing protocols with the expected outputs. To do this, the program must convert the timing protocol to a time-base such that the minimum features of both can be compared. In principle, the compare can be a simple binary compare. The challenge is in determining the minimum feature size of the calculated versus the expected output. Since the timing protocol is much like an equation, the minimum feature size for each can be calculated. This may be somewhat time-consuming, but need only be done once for each signal to be compared.

6.2 CONCLUSIONS

This thesis has developed a case for an automated logic circuit evaluation system. Toward its development, this thesis has made the following contributions.

1. First, it has provided a study of the types of errors that are commonly made in VLSI logic circuit design. This was based on a review of VLSI design errors captured in a corporate database -- BNR CHIPBUGS -- which covers over 70 designs. The results of this study showed that approximately 85% of the errors made were detectable. These errors were made even though state-of-the-art simulation tools were applied. A further study of design error coverage by simulation and timing verification shows that many of the errors made are either not addressed by these tools, or their detection is quite unlikely.

This is not to say that the problems cannot be detected by a simulation program. Rather, a problem may not be detected: through carelessness on the part of the designer (not recognizing the symptoms flagged by the simulator); because the detection of the problem lies outside of the circuit model accuracy; or, more commonly, because the problem was subtle and was never cov-

ered by any of the simulation test cases. Since evaluation is intended to look for problems, as opposed to confirming correct circuit operation, it can further be concluded that evaluation is complementary to simulation, not a replacement for it.

2. A preliminary set of evaluation rules has been developed that will detect the symptoms of many of these errors. The rules are grouped into input rules, output rules, and rules for cells, and tend to be technology independent. Included in this work is the development of a novel detection criterion ($T_{critical}$). This criterion is based on conservative design rules, and in application is used as a benchmark. Circuits which fail this test, are worthy of further checking.
3. A simple algorithm for finding evaluation paths through logic circuits has been defined. Contrary to most timing verification systems, which start their analysis at circuit outputs, this method starts at the circuit's inputs, and works forward, building information as it goes. This action mimics the process that an expert goes through in manually evaluating a circuit.
4. It has been shown that the timing information available in a circuit simulation is inadequate to do an evalu-

ation. The required information (transition timing information and the timing protocol) has been established, and a method for its description has been developed, ATDL. ATDL can be used to describe an arbitrary timing signal, capturing any relationships that it has with any other signal. This extends to being able to describe asynchronous signals which cannot be operated on in event driven simulators.

5. It is intended that ATDL be applied, not only as an I/O specification method, but that it be operated on by the evaluator to propagate the timing protocols from the inputs to the outputs. As such, it can also be used to represent the timing internal to the circuit.

6.3 RECOMMENDATIONS

This work has focussed on solving some of the key problems in developing a logic circuit evaluator. Additional work in this area is needed to bring an evaluator into being. The key areas of need are listed below.

1. One of the key components for this evaluator is an ATDL interpreter. ATDL represents timing in a much more gen-

eral way than is used in event driven simulation. Therefore, it is necessary that an interpreter be designed to demonstrate that ATDL can represent timing unambiguously.

2. Further to this, the evaluator itself operates quite differently than a simulator in that it must propagate whole protocols through the circuit, as opposed to individual events. Additional work on the evaluator structure, and optimal implementation, is needed.
3. The rule set provided is just a preliminary one. These rules should be expanded to capture as many of the special and exception cases as possible to reduce the number of flags generated. Rules to reduce the number of feedback loops reported, and to eliminate cross-coupled gates from the asynchronous flags are just two candidates in this area for further work.
4. It is also possible to expand the capabilities of such an evaluator to consider circuit hierarchy. By doing so, the evaluator could have knowledge of circuit structures that is not available to it through the netlist. With this information, it will be possible to further develop the rule set to include cases which require knowledge of

circuit structures (e.g. phase-locked loops, and asynchronous-to-synchronous interfaces).

APPENDIX A - ATDL SHORT FORM SUMMARY

The following is an abbreviated form of the syntax definition for more convenient reference.

Signal_name - the name of the signal. The signal name and node name are the same.

Node_type - I, O, I/O, INT = input, output, bidirectional, and internal node.

Signal_type - CLOCK, PULSE, LEVEL or SPECIAL

TP - period of signal, in time or in terms of a number of cycles of another signal.

TR - rise time. (Can be spec'd MIN and MAX.)

TF - fall time. (Can be spec'd MIN and MAX.)

PW - pulse width. There can be more than one for a pulse stream.

Default - NH/NL/NX/NZ = the default state of the signal. Normally (N) high (H), low (L), unknown (X) or high impedance (Z).

Relative_name - a signal name that the specified signal is related.

- R/F/Z** - indicates the edge of RELATIVENAME that is referenced: R=rising, F=falling, Z=transition to high impedance from either state.
- TD** - delay time before the signal changes state.
- Pulsetrain** - only for PULSE signals. A sequence of pulses each having the timing characteristics specified in the preceding part of the syntax.
- +** - a continuation character.

GLOSSARY OF TERMS

ASIC - Application Specific Integrated Circuit

ATDL - A Timing Description Language

BNR - Bell-Northern Research

CAD - Computer Aided Design

CAE - Computer Aided Engineering

CHIPBUGS - A database for capturing all chip errors: the symptom, the problem, the solution, and the source of the problem.

CMOS - Complementary Metal Oxide Semiconductor

ERC - Electrical Rules Checker

ESD - Electro-Static Discharge

FANOUT LIST - a listing of all nodes in the circuit and all circuit elements that are connected to each.

Glitch - a momentary, generally undesired, change in a signal

I/O - Input/Output

MSI - Medium Scale Integration

NETLIST - a listing of the circuit "network" containing the circuit elements, and their connections.

PLA - Programmable Logic Array

Protocol - Timing "protocol" is the format of data in a given signal, including a specification of the relationships between one signal and any other.

TTL - Transistor Transistor Logic

SSI - Small Scale Integration

UART - Universal Asynchronous Receiver Transmitter

VLSI - Very Large Scale Integration

REFERENCES

[BapS86] Bapat, S., Venkatesh, G., "Reasoning About Digital Systems Using Temporal Logic," Proceedings of the 23rd Design Automation Conference, IEEE, Las Vegas, 1986, pp. 215-219.

[BenL82] Bening, L.C., Lane, T.A., Alexander, C.R., Smith, J.E., "Developments in Logic Network Path Delay Analysis," Proceedings of the 19th Design Automation Conference, IEEE, Las Vegas, 1982, pp. 605-615.

[BraD86] Brand, D., Iyengar, V.S., "Timing Analysis using Functional Relationships," IEEE International Conference on Computer-Aided Design: ICCAD-86, Digest of Technical Papers, IEEE, Santa Clara, 1986, pp. 126-129.

[ChaE85] Chan, E., "Development of a Timing Analysis Program for Multiple Clocked Network," Proceedings of the 22nd Design Automation Conference, IEEE, Las Vegas, 1985, pp. 816-819.

[CurT86] Curtis, T.D.W., "VLSI Design Quality," Proceedings of the 1986 Canadian Conference on Very Large Scale Integration, AES Data; Canadian Marconi; Bell Northern Research; Concordia University; et al, Montreal, 1986, pp. 27-35.

[DagM86] Dagenais, M.R., Rumin, N.C., "Circuit-Level Timing Analysis and Design Verification of High-Performance MOS Computer Circuits," Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers: ICCD-86, Port Chester, NY, 1986, pp. 356-359.

[DagM87] Dagenais, M.R., Rumin, N.C., "Timing Analysis and Verification of Digital MOS Circuits," Proceedings of VLSI and Computers. First International Conference on Computer Technology, Systems and Applications: COMPEURO 87, Hamburg, Germany, 1987, pp. 242-245.

[FleW80] Fletcher, W. I., An Engineering Approach to Digital Design, Englewood Cliffs, N.J.:Prentice-Hall Inc., 1980. (ISBN 0-13-277699-5)

[FriA86] Friedman, A.D., Fundamentals of Logic Design and Switching Theory, Rockville, Md.: Computer Science Press Inc., 1986. (ISBN 0-88175-098-0)

[GhoS86] Ghosh, S., "A Rule-Based Approach to Unifying Functional and Fault Simulation and Timing Verification," Proceedings of the 23rd Design Automation Conference, IEEE, Las Vegas, 1986, pp. 677-682.

[GleM86] Glesner, M., Schuck, J., Steck, R.B., "SCAT - A New Statistical Timing Verifier in a Silicon Compiler System," Proceedings of the 23rd Design Automation Conference, IEEE, Las Vegas, 1986, pp. 220-226.

[HeiS87] Heinkele, S., "A Fast Timing Analysis Tool for VLSI," Proceedings of VLSI and Computers. First International Conference on Computer Technology, Systems and Applications: COMPEURO 87, Hamburg, Germany, 1987, pp. 193-195.

[HitR82] Hitchcock Sr., R.B., "Timing Verification and the Timing Analysis Program," Proceedings of the 19th Design Automation Conference, IEEE, Las Vegas, 1982, pp. 594-604.

[HwaS86] Hwang, S.H., Kim, Y.H., Newton, A.R., "An Accurate Delay Modeling Technique for Switch-Level Timing Verifica-

tion," Proceedings of the 23rd Design Automation Conference, IEEE, Las Vegas, 1986, pp. 227-233.

[JouN83] Jouppi, N.P., "Timing Analysis for nMOS VLSI," Proceedings of the 20th Design Automation Conference, IEEE, Miami Beach, 1983, pp. 411-418.

[KelV84] Kelly, V.E., "-- The Critter System -- Automated Critiquing of Digital Circuit Designs," Proceedings of the 21st Design Automation Conference, IEEE, Albuquerque, 1984, pp. 419-425.

[LeiS82] Leinwand, S.M., "Logical Correctness by Construction," Proceedings of the 19th Design Automation Conference, IEEE, Las Vegas, 1982, pp. 825-831.

[MaH87] Ma, H.T., Devadas S., Sangiovanni-Vincentelli, A., Wei, R., "Logic Verification Algorithms and their Parallel Implementation," Proceedings of the 24th Design Automation Conference, IEEE, Miami Beach, 1987, pp. 283-290.

[MarJ84] Mar, J., Wei, Y., "Performance Verification of Circuits," Proceedings of the 21st Design Automation Conference, IEEE, Albuquerque, 1984, pp. 419-425.

[MarF82] Maruyama, F., Uehara, T., Kawato, N., Saito, T., "A Verification Technique for Hardware Designs," Proceedings of the 19th Design Automation Conference, IEEE, Las Vegas, 1982, pp. 832-841.

[MeiG87] Mein, G.F., "Evaluation of CHIPBUGS Database," (BNR internal report), November 1987.

[MicA86] Miczo, A., Digital Logic Testing and Simulation, New York: Harper & Row, Publishers, Inc., 1986. (ISBN 0-06-044444-4)

[MokV85] Mokkarala, V.R., Fan, A., Apte, R., "A Unified Approach to Simulation and Timing Verification at the Functional Level," Proceedings of the 22nd Design Automation Conference, IEEE, Las Vegas, 1985, pp. 757-761.

[MonM82] Monachino, M., "Design Verification System for Large-Scale LSI Designs," Proceedings of the 19th Design Automation Conference, IEEE, Las Vegas, 1982, pp. 83-90.

[MurM85] Muraoka, M., Iida, H., Kikuchi-hara, H., Murakami, M., Hirakawa, K., "ACTAS: An Accurate Timing Analysis System for VLSI," Proceedings of the 22nd Design Automation Conference, IEEE, Las Vegas, 1985, pp. 152-158.

[NgP81] Ng, P., Glauert, W., Kirk, R., "A Timing Verification System Based on Extracted MOS/VLSI Circuit Parameters," Proceedings of the 18th Design Automation Conference, IEEE, Nashville, 1981, pp. 288-292.

[RimC83] Rimkus, C.J., Wayne, M.R., Cheng, D.D., Magistro, F.J., "Structured Design Verification: Function and Timing," Proceedings of the 20th Design Automation Conference, IEEE, Miami Beach, 1983, pp. 246-252.

[SteL84] Steinberg, L.I., Mitchell, T.M., "A Knowledge Based Approach to VLSI CAD The Redesign System," Proceedings of the 21st Design Automation Conference, IEEE, Albuquerque, 1984, pp. 412-418.

[TamE83] Tamura, E., Kimihiro, O., Nakano, T., "Path Delay Analysis for Hierarchical Building Block Layout System," Proceedings of the 20th Design Automation Conference, IEEE, Miami Beach, 1983, pp. 403-410.

[UlrE83] Ulrich, E., "A Design Verification Methodology Based on Concurrent Simulation and Clock Suppression," Proceedings of the 20th Design Automation Conference, IEEE, Miami Beach, 1983, pp. 709-712.

[Weid87] Weise, D., "Functional Verification of MOS Circuits," Proceedings of the 24th Design Automation Conference, IEEE, Miami Beach, 1987, pp. 265-270.

END

17/09.90

FIN