

OFFI CE INTERNATIONAL DE
DOCUMENTATION ET **LIB**RAIRIE

S. A. Capital 1 071 000 F
SIRET 562 122 689 00010
APE 6443

48, rue Gay-Lussac - 75240 PARIS CEDEX 05 FRANCE

☎ : (1) 43.29.21.32
C. C. P. 5780-51 X PARIS

Notre REF: SY30405/D8225
Our

Votre REF:
Your

DATE September 13, 1989

UNIVERSITY OF WATERLOO

DEPARTMENT OF COMPUTER SCIS
WATERLOO ONTARIO N2 L 3G1

CANADA

Veillez **ADRESSER** la commande suivante à : } **OFFILIB** AGREMENT DOUANE No I 115
Please **SHIP** the following order **TO** : } 48, Rue Gay-Lussac
75240 PARIS CEDEX 05 - FRANCE

1 **SEGER** - CS 88-22 : Models and algorithms for race analysis in asynchronous circuits

*sent
Oct. 2/89*

FACTURATION en 3 ex. avec votre meilleure **REMISE** à : } **OFFILIB** 48 rue Gay Lussac - 75240 Paris Cedex 05 France
Triplicate **BILL** with discount **TO** : } **MERCI** — **THANKS**

University of Waterloo
Department of Computer Science
Waterloo, Ontario N2L 3G1

March 28, 1989.

Professor D. Probst,
Concordia University,
Computer Science Dept.,
1455 Maisonneuve West,
Montreal, Quebec,
H3G 1M8

INVOICE

REPORT(S) ORDERED

CS-88-22

TOTAL COST

\$8.00

Would you please make your cheque or international bank draft payable to the Computer Science Department, University of Waterloo and forward to my attention.

Thanking you in advance.

Yours truly,

Susan DeAngelis

Susan DeAngelis (Mrs.)
Research Report Secretary
Computer Science Dept.

/sd

Encl.

PHONE CALL

Date 28 Mar 89 Time 8⁵⁰

To Joe DeAngelis

WHILE YOU WERE OUT

M. Prof. D. Probst

of CS Dept, Concordia Univ.

Phone 1455 Maisonneuve W.

Telephoned ☒ Please call ☒

Called to see you ☐ Will call again ☐

Wants to see you ☐ Returned your call ☐

MESSAGE Montreal, Quebec

Send H3G #2 Send invoice
1st class (514) 733-4921

Please call collect if necessary
to let him know if you
have CS-88-22 available.

Operator He needs it asap!

031020

today

Printing Requisition / Graphic Services

20994

1. Please complete unshaded areas on form as applicable.
2. Distribute copies as follows: White and Yellow to Graphic Services. Retain Pink Copies for your records.
3. On completion of order the Yellow copy will be returned with the printed material.
4. Please direct enquiries, quoting requisition number and account number, to extension 3451.

TITLE OR DESCRIPTION

CS-88-22

DATE REQUISITIONED

Sept. 21/88

DATE REQUIRED

ASAP

ACCOUNT NO.

1126631841

REQUISITIONER - PRINT

PHONE

2172

SIGNING AUTHORITY

S. DeAngelis / J. Brzozowski

MAILING INFO -

NAME

S. DEANGELIS

DEPT.

C.S.

BLDG. & ROOM NO.

DC 2314

DELIVER

PICK-UP

Copyright: I hereby agree to assume all responsibility and liability for any infringement of copyrights and/or patent rights which may arise from the processing of, and reproduction of, any of the materials herein requested. I further agree to indemnify and hold blameless the University of Waterloo from any liability which may arise from said processing or reproducing. I also acknowledge that materials processed as a result of this requisition are for educational use only.

NUMBER OF PAGES 210 NUMBER OF COPIES 25

TYPE OF PAPER STOCK

☒ BOND ☐ NCR ☐ PT. ☒ COVER ☐ BRISTOL ☒ SUPPLIED ☐

PAPER SIZE

☒ 8 1/2 x 11 ☐ 8 1/2 x 14 ☐ 11 x 17 ☐

PAPER COLOUR

☒ WHITE ☐ ☐ BLACK ☐

PRINTING

☐ 1 SIDE PGS. ☒ 2 SIDES PGS. FROM TO

BINDING/FINISHING

☒ COLLATING ☐ STAPLING ☐ HOLE PUNCHED ☒ PLASTIC RING

FOLDING/
PADDING

CUTTING
SIZE

Special Instructions

Math fronts + backs enclosed.
Please bind.
Thank you.

COPY CENTRE

OPER. NO. BLDG. NO. MACH. NO.

DESIGN & PASTE-UP

OPER. NO. TIME LABOUR CODE
D 0 1
D 0 1
D 0 1

TYPESETTING

QUANTITY

P A P 0 0 0 0 0 T 0 1
P A P 0 0 0 0 0 T 0 1
P A P 0 0 0 0 0 T 0 1

PROOF

P R F
P R F
P R F

NEGATIVES

QUANTITY

OPER. NO.

TIME

LABOUR CODE

F L M C 0 1
F L M C 0 1
F L M C 0 1
F L M C 0 1
F L M C 0 1

PMT

P M T C 0 1
P M T C 0 1
P M T C 0 1

PLATES

P L T P 0 1
P L T P 0 1
P L T P 0 1

STOCK

0 0 1
0 0 1
0 0 1
0 0 1

BINDERY

R N G B 0 1
R N G B 0 1
R N G B 0 1
M I S 0 0 0 0 0 B 0 1

OUTSIDE SERVICES

\$ COST

Models and Algorithms
for
Race Analysis
in
Asynchronous Circuits

Carl-Johan Henry Seger
Department of Computer Science

Research Report CS-88-22
May 1988

**Models and Algorithms
for
Race Analysis
in
Asynchronous Circuits**

by

Carl-Johan Henry Seger

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, 1988

© Carl-Johan Seger 1988

Models and Algorithms for Race Analysis in Asynchronous Circuits

ABSTRACT

This thesis addresses the problem of analyzing asynchronous sequential circuits. Our first contribution is a unified mathematical framework which enables us to derive a theory applicable to gate circuits, as well as to the more modern MOS circuits. We then study the behavior of asynchronous circuits using the abstract framework together with several different delay and race models.

The first race model developed is the “extended multiple winner” (XMW) model in which each logic component has an arbitrary finite delay. For this model we prove that the set of state variables used to analyze a circuit can be reduced to a minimal set of feedback variables, without any loss of state transition and hazard information. (This contrasts sharply with previously known models.) Secondly, we prove that ternary simulation (which is very efficient) yields the same results as the XMW analysis (which is intractable).

The XMW model is closely related to delay-independent circuits, i.e. to circuits that operate correctly, no matter what the component delays are. Using the XMW theory, it is shown that the class of delay-independent circuits is quite small, and that many common sequential behaviors cannot be realized delay-independently. The main reason for this is the unrealistic assumption that component delays can be arbitrary.

To overcome the limitations of the XMW model, we derive two race models that are more realistic. The “almost-equal-delay” model makes the somewhat idealized assumption that all component delays are approximately equal. The “extended bounded-delay” model assumes that each delay is bounded by a lower and an upper limit. These intuitive notions are formalized, and practical algorithms are developed for race analyses using these models.

Finally, we derive a number of complexity results for the race analysis problem using different delay models.

Acknowledgements

First and foremost I would like to express my sincere thanks to my supervisor, mentor, and (I hope I may say) friend, Dr. John Brzozowski. In fact, I cannot give John a higher rating as supervisor. I hope I will be able to use this experience of how supervision *should* be done the day I will supervise my own students.

The readers of my thesis, Dr. Randy Bryant, Dr. Farhad Mavaddat, Dr. Derick Wood, and Dr. Dan Younger should also be thanked for their prompt and careful reading of the thesis.

I would like to say a special thanks to Sheila Trainor for making this last year so enjoyable. Also, I would like to thank all the people who have made my stay in Waterloo a rewarding experience, due to their friendship, in particular Benny Wong, Michel Devine, Dave Neudoerffer, Mark Leitch, Ian and Ann Davis, and numerous others.

Finally, I would like to acknowledge the financial support from the Department of Computer Science, the province of Ontario through an Ontario Graduate Scholarship, the Institute for Computer Research through a scholarship, and the National Research Council of Canada under grant A0871.

To my parents

Table of Contents

Chapter 1. Introduction	1
1.1. Motivation	2
1.2. Analysis Problem for Gate Circuits	3
1.3. MOS Circuits	7
1.4. Previous Work	9
1.5. Outline of Thesis	12
Chapter 2. A Unified Framework	17
2.1. Graph Model	17
2.2. Application to Gate Circuits	20
2.3. Switch-Level Circuits	23
2.4. Application to MOS Circuits	29
Chapter 3. The Extended Multiple Winner Race Model	31
3.1. XMW Race Model	32
3.2. Ternary Simulation	37
3.3. Reduced Networks	42
3.4. Output Hazards	50
3.5. Summary	53
Chapter 4. Delay-Independent Networks	55
4.1. Introduction	55
4.2. Delay-Independence	57
4.3. Behavioral Model	58
4.4. Composite Network Function	62
4.5. Fundamental Properties of Delay-Independent Circuits	65
4.6. Alternative Notions of Delay-Independence	71
4.7. Conclusions	73
Chapter 5. The Almost Equal Delay Race Model	75
5.1. Binary Almost-Equal-Delay Model	76
5.2. Stepwise AED Model	79
5.3. Race Units	85
5.4. TAED Algorithm	88
5.5. Correspondence between AED Model and TAED Algorithm	92
5.6. Discussion	97
Chapter 6. The Extended Bounded Delay Race Model	99
6.1. Introduction	99

6.2. Extended Inertial Delay Model	101
6.3. Bounded Delay Race Model	107
6.4. Ternary Bounded Delay Algorithm	120
6.5. Correspondence between XBD Model and TBD Algorithm	126
Chapter 7. The Complexity of Race Analysis	135
7.1. Stable-State Reachability Problem	135
7.2. Limited Reachability Problem	155
Chapter 8. Conclusions and Future Research	163
8.1. Analysis Problem	163
8.2. Verification Problem	164
8.3. Design Problem	166
Appendix A. Switch-Level Models	167
A.1. Static Excitation Functions	167
A.2. Dynamic Excitation Functions	177
References	183

List of Figures

1.1.	Circuit C_1 .	4
1.2.	Unit delay analysis of circuit C_1 .	5
1.3.	Race analysis of C_1 according to the GMW model.	5
1.4.	Circuit C_2 .	6
1.5.	MOS transistor symbols: (a) N-transistor; (b) P-transistor.	7
1.6.	CMOS NOR gate.	7
1.7.	CMOS circuit realizing the function AB' .	8
1.8.	Huffman model for asynchronous gate circuits.	9
2.1.	Ternary OR, AND, and complement.	18
2.2.	Network N_1 .	19
2.3.	Gate circuit G_2 .	21
2.4.	Network N_2 .	21
2.5.	Gate circuit G_3 .	22
2.6.	(a) CMOS circuit C_4 ; (b) corresponding S-graph S_4 .	25
2.7.	Networks N_5 and N_6 .	30
3.1.	Network N_1 .	32
3.2.	XMW analysis of network N_1 .	32
3.3.	XMW analysis of network N_2 .	35
3.4.	Ternary simulation of N_1 : (a) Algorithm A; (b) Algorithm B.	41
3.5.	Ternary simulation of N_2 : (a) Algorithm A; (b) Algorithm B.	41
3.6.	Removal of vertex q 's dependency on vertex p .	43
3.7.	Network \dot{N}_1 .	47
3.8.	Analyses of \dot{N}_1 : (a) XMW; (b) ternary simulation.	47
3.9.	Analyses of \dot{N}_2 : (a) XMW; (b) ternary simulation.	48
3.10.	Network N_3 .	48
3.11.	Network \ddot{N}_3 .	49
4.1.	Gate circuit G_1 .	56
4.2.	Gate circuit G_2 .	56
4.3.	Gate circuit G_3 .	56
4.4.	Gate circuit G_4 .	60
4.5.	Network N_4 .	60
4.6.	Network N_4' .	61
4.7.	FDAM M_1 corresponding to network N_4' .	61
4.8.	(a) Original network; (b) combinational network; (c) composite network.	62
4.9.	FDAM's for a network with only one input.	66

4.10. Gate network G_5 .	70
5.1. Circuit C_1 .	76
5.2. Illustrating the “AED” idea: (a) possible transition; (b) timing diagram.	76
5.3. Network N_2 .	78
5.4. Race analysis of N_2 according to the AED method.	78
5.5. Relation R derived from ρ of Fig. 5.4.	81
5.6. AED analysis of network N_3 .	82
5.7. Stepwise AED analysis of N_2 .	86
5.8. Deterministic finite state machine corresponding to Fig. 5.7.	86
5.9. (a) Circuit C_4 ; (b) circuit C_5 .	89
5.10. Analysis of N_2 : (a) TAED; (b) stepwise AED model.	91
5.11. Network N_6 .	96
5.12. Stepwise AED analysis of N_6 .	96
5.13. (a) TAED analysis of N_6 ; (b) <i>l.u.b.</i> of the stepwise AED analysis.	96
6.1. Gate circuit G_1 .	100
6.2. Delay element.	102
6.3. Typical waveforms for FBPD model ($\Delta = 1$).	102
6.4. Typical waveforms for FBID model ($\Delta = 1$).	103
6.5. Two possible waveforms according to BID model ($1 \leq \Delta(t) < 2$).	104
6.6. A possible waveform according to XID model ($1 \leq \Delta(t) < 2$).	106
6.7. Two possible waveforms according to XID model ($1 \leq \Delta(t) < 2$).	106
6.8. Network N_2 .	109
6.9. Two possible XBD race sequences.	110
6.10. TBD analysis of N_2 .	123
6.11. Gate circuit G_3 .	123
6.12. Network N_3 corresponding to G_3 .	123
6.13. TBD analysis of N_3 .	123
7.1. A D flip-flop with completion and reset signals.	138
7.2. Construction for a variable x .	139
7.3. Construction for $\neg E_1(y)$.	139
7.4. Construction for $E_1(y) \vee E_2(y)$.	140
7.5. Construction for $\exists x[E_1(x, y)]$.	140
7.6. Complete construction.	140
7.7. Critical race generating circuit.	147
7.8. GMW analysis of the critical race generating circuit.	147
7.9. Circuit for $E = x_i$.	149
7.10. Circuit for $\neg E_1(x)$.	149
7.11. Circuit for $E_1(x) \wedge E_2(x)$.	149

7.12. Circuit for $E_1(x) \vee E_2(x)$	149
7.13. Complete circuit.	152
7.14. Complexity of the SSR problem for different race models.	154
7.15. Construction of \tilde{N}	158
7.16. Complete circuit.	159
7.17. Complexity of the LR problem for different race models.	161
A.1. (a) CMOS circuit C_1 ; (b) corresponding S-graph S_1	168
A.2. CMOS circuit S_2	173
A.3. CMOS circuit S_3	175
A.4. CMOS circuit S_4	177

List of Propositions

Proposition 2.1.	25
Proposition 3.1.	32
Proposition 3.2.	35
Proposition 3.3.	38
Proposition 3.4.	38
Proposition 3.5.	45
Proposition 3.6.	46
Proposition 3.7.	52
Proposition 4.1.	63
Proposition 4.2.	64
Proposition 5.1.	83
Proposition 5.2.	83
Proposition 5.3.	84
Proposition 6.1.	112
Proposition 6.2.	112
Proposition 6.3.	113
Proposition 6.4.	114

List of Lemmas

Lemma 3.1.	43
Lemma 3.2.	45
Lemma 4.1.	64
Lemma 4.2.	67
Lemma 4.3.	68
Lemma 4.4.	69
Lemma 5.1.	92
Lemma 6.1.	116
Lemma 6.2.	118
Lemma 6.3.	126
Lemma 6.4.	130

List of Theorems

Theorem 3.1. 39

Theorem 3.2. 40

Theorem 3.3. 46

Theorem 3.4. 51

Theorem 3.5. 53

Theorem 3.6. 54

Theorem 4.1. 65

Theorem 4.2. 66

Theorem 4.3. 66

Theorem 5.1. 84

Theorem 5.2. 92

Theorem 6.1. 126

Theorem 7.1. 136

Theorem 7.2. 143

Theorem 7.3. 144

Theorem 7.4. 144

Theorem 7.5. 146

Theorem 7.6. 146

Theorem 7.7. 147

Theorem 7.8. 155

Theorem 7.9. 156

Theorem 7.10. 157

Theorem 7.11. 160

Chapter I

Introduction

Switching theory makes use of mathematical models and techniques to handle problems associated with the design of digital circuits. The theory originated in the late 1930s, and Shannon [52] is usually considered to be its father. Digital circuits can be divided into *combinational* circuits and *sequential* circuits. The output of a combinational circuit is completely determined by the current input (possibly after a short delay), whereas the output of a sequential circuit depends also on previous input signals. Combinational circuits are well understood and will not be discussed any further in this thesis.

The class of sequential circuits is often divided into *synchronous* circuits and *asynchronous* circuits. In the former, the inputs to the circuit are allowed to change only when a global clock signal is inactive, and the circuit is essentially disabled. The clock signals are also used to “mask” the delays in wires and gates. Together, these restrictions make the analysis and design of synchronous circuits straightforward, and standard methods can be applied.

Circuits that are designed to operate without a synchronizing clock are called asynchronous[†]. The theory of asynchronous circuits is much less developed than the theory of synchronous circuits; this thesis is a study of certain aspects of asynchronous circuits.

[†] This definition is not very precise, but suffices for the present.

In this chapter, we first give some motivation for studying asynchronous circuits. We then describe the general problems that will be studied, and give a brief overview of previous work. Finally, the contents of the remaining chapters of the thesis are briefly outlined.

1.1. Motivation[†]

There are several reasons for studying asynchronous circuits. The first is the simple fact that asynchronous operation cannot be avoided. One example of such a situation is interfacing external signals with a digital system. Such signals are usually unrelated to the internal system clock; hence one has to use some kind of asynchronous circuit that reacts whenever these signals arrive. Note that the function of this asynchronous circuit may be simply to delay the input signal until the system clock signal arrives.

The second reason for studying asynchronous circuits is the fact that every synchronous circuit should be viewed as asynchronous if a more accurate model is needed. In synchronous circuits, the common assumption is that all signals are stable when the global clock signal is “active”. Furthermore, the clock signal is assumed to reach all parts of the system at the same time. These assumptions are not always valid and are often violated in large VLSI chips. This is due to the fact that the delays in the wires on a chip can be significant, even if the length of the wire is only a fraction of the size of a typical chip. Hence, one needs to study the behavior of the circuit under the assumption that the clock signal is just another input signal that may be delayed.

The third reason is speed. Since the basic philosophy of asynchronous design is to let each unit run independently, i.e. at its own maximum speed, it should be possible to build very fast asynchronous systems. For example, many circuits have a delay that strongly depends on the data being processed. In a synchronous design, the global clock frequency must be low enough to handle the worst-case delay. Because of this, the circuit will be idle quite often. On the other hand, in an

[†] This section is based largely on my M.Math thesis [49]; the reader is referred there for further details.

asynchronous design the circuit can send a completion signal when it has finished and thus be faster on the average. However, the issue is not that simple. Although asynchronous circuits are intended to work independently, some means of synchronization must be provided when data must be transferred between different units. This is usually solved by some kind of hand-shaking protocol; unfortunately, this adds to the delay and the complexity of the circuit. It is not clear whether the theoretically possible speed gain promised by asynchronous design can be achieved in practice, and this problem needs to be studied further.

The fourth reason is the lack of an adequate theory of asynchronous circuits. Today, asynchronous circuits are rarely used; the main reason for this is probably the fact that they are not well understood. In this thesis, we will show that the previously available theories are not realistic, and we will develop a theory that can explain the behavior of asynchronous circuits more accurately.

Finally, as argued above, the problem area is underdeveloped and contains many challenging and interesting problems for a theoretician. This, in itself, could serve as a motivation. However, the problems studied are also of significant practical importance. In particular, the difficulty and high cost of designing VLSI circuits imply that “ad hoc” methods for designing and verifying circuits are no longer adequate, and that more precise and formal methods are required.

1.2. Analysis Problem for Gate Circuits

There are three major problems that need to be addressed when studying switching circuits. The first is to *analyze* a circuit, i.e. to determine its behavior. The second is to *verify* that a given circuit behaves according to some given specification. The third is to *design* a circuit that realizes some given specification. The problem of analyzing a circuit is the fundamental problem that must be solved first, because the other two rely on it. Hence, this thesis will focus on the analysis problem. However, it is hoped that the insight this will give us will improve our understanding of the other two problems.

The problem of analyzing an asynchronous circuit is now introduced by means of some examples, and some terminology is defined. Consider the gate circuit C_1 of Fig. 1.1.[†] The circuit consists of an inverter, a two-input AND gate, and a two-input OR gate. The Boolean functions COMPLEMENT, AND, and OR will be denoted by $'$, concatenation, and $+$, respectively. Thus the behavior of this circuit is governed by the following equations:

$$Y_1 = x', \quad Y_2 = xy_1, \quad Y_3 = y_2 + y_3,$$

where, for each gate, y_i denotes the present output of gate i , and Y_i , called the *excitation* of gate i , gives the value of the Boolean function computed by gate i . When $y_i = Y_i$ the gate has no tendency to change, and we say that it is *stable*. If $y_i \neq Y_i$, then the gate is *unstable* and the output y_i tries to change to Y_i . However, this change does not take place instantaneously since there is always some delay in the gate. Furthermore, the change does not always happen because it is possible that an earlier change in some other gate may cause Y_i to become equal to y_i . This corresponds to the fact that the delay associated with gate i is inertial, in the sense that short periods of instability are tolerated without any change. Now suppose the circuit of Fig. 1.1 is started with $x = 0$ and $y = (y_1, y_2, y_3) = (1, 0, 0)$, which is a stable state. What happens when x is changed to 1? Such questions constitute the basic problem in analyzing asynchronous circuits and will be the main problem discussed in this thesis.

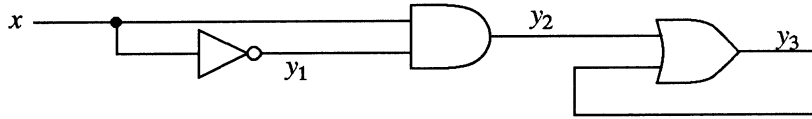


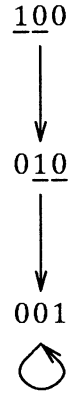
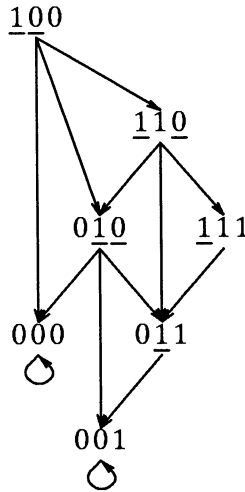
Figure 1.1. Circuit C_1 .

[†] Note that many examples in this thesis are chosen for their simplicity and not necessarily their usefulness.

Any state in which more than one gate is unstable at the same time is called a *race*. The outcome of a race depends on the delays in the circuit. If the delays were known exactly, it would be straightforward to compute the behavior of a circuit after an input change. However, the delays of two components of the same type are normally slightly different, and even the delay of a single component is not always constant. For example, the delay can change due to aging or temperature changes. In order to analyze the behavior of a circuit, so called *race models* have been developed. In a race model a class of circuits rather than a single circuit is analyzed. This class of circuits corresponds to all circuits with the same topology, but with different delays. Hence in a race model it is quite possible to have several successor states from any given starting state. It is important though to remember that the physical circuit can be in at most one state at any given time, and can have only one possible successor state (depending on the current delays). Hence, the race models are normally somewhat pessimistic, trying to take into account all possible delay distributions.

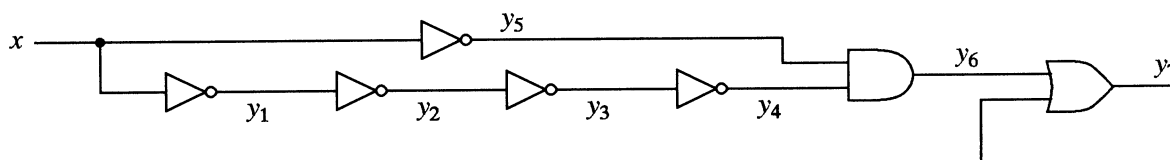
There are several theories which predict the outcome of a race. Typically the predicted outcome depends very much on the model used. Commercial simulators like SILOS [53] or MOSSIM [6] usually use the “unit delay” (UD) model, in which all gates are assumed to have exactly equal delays. In the state $x=1, y=100$ (commas and parentheses omitted from $(1,0,0)$ for simplicity) gates 1 and 2 are unstable, and will both change. Consequently, the next state is 010. Now gates 2 and 3 are unstable, and state 001 is reached. This state is stable. In summary, the unit delay model predicts that the final outcome of this transition is the stable state 001; see Fig. 1.2, where unstable gates are indicated by underlining.

In contrast to this, the General Multiple Winner (GMW) model [17] permits the possibility of unequal delays. In fact, the model assumes that delays are unknown, arbitrary, but finite; furthermore, they can also vary in time. The GMW model assumes that any nonempty subset of the set of unstable gates can change in any race. In Fig. 1.3 we show the GMW analysis of circuit C_1 . If gate 1 is faster than gate 2 in state 100, then state 000 may be reached. This is also a stable state, and represents a likely outcome. This shows that the UD model is inaccurate. In fact, the only justification for the use of the UD model appears to be its simplicity.

Figure 1.2. Unit delay analysis of circuit C_1 .Figure 1.3. Race analysis of C_1 according to the GMW model.

On the other hand, the GMW model is often too “pessimistic” as we show below.

Consider the circuit of Fig. 1.4 starting in the stable state $x=0$, $y=1010100$, and let x change to 1. It is reasonable to assume that y_5 will change to 0 before y_4 changes to 1, and that the only likely final outcome is the stable state 0101000. However, the GMW model will allow the possibility that y_4 changes before y_5 and that the state 0101001 is also reachable. In other words, the GMW model allows

Figure 1.4. Circuit C_2 .

the delay of inverter 5 to be larger than the sum of the delays of inverters 1 to 4 — a very pessimistic prediction.

The simple examples above illustrate the difficulty of the analysis problem. The study of realistic race models and practical algorithms for performing the analysis according to these models are the major topics of this thesis.

1.3. MOS Circuits

In the race analysis given in the previous section, we considered gate circuits. Gate-based models do not always apply to MOS circuits which are widely used at present. We will focus our discussion on CMOS circuits, as CMOS technology is emerging as the dominant technology for VLSI design. For a more thorough treatment of CMOS circuits, the reader is referred to [59].

A CMOS circuit consists of transistors of two types: n-channel enhancement mode transistors and p-channel enhancement mode transistors (N-transistors and P-transistors for short). An N-transistor, whose symbol is shown in Fig. 1.5(a), works roughly like a switch between terminals a and b which is closed if the voltage on the “gate” terminal is high, and open if it is low. A P-transistor, shown in Fig. 1.5(b), works dually, i.e. it is closed when the voltage on the gate terminal is low and open when it is high.

Consider the circuit of Fig. 1.6. If $A = 1$ or $B = 1$ (or both), then Z is connected to ground, i.e. the output voltage is low. The only time Z is connected to the power supply, i.e. the output voltage is high, is when both A and B are 0. Hence, the circuit is a NOR gate, i.e. $Z = (A + B)'$. This circuit is very easy to analyze. However, consider the circuit of Fig. 1.7; here it is far from clear what function is realized. With some work one can show that, if the inputs A and B are binary, the output is



Figure 1.5. MOS transistor symbols: (a) N-transistor; (b) P-transistor.

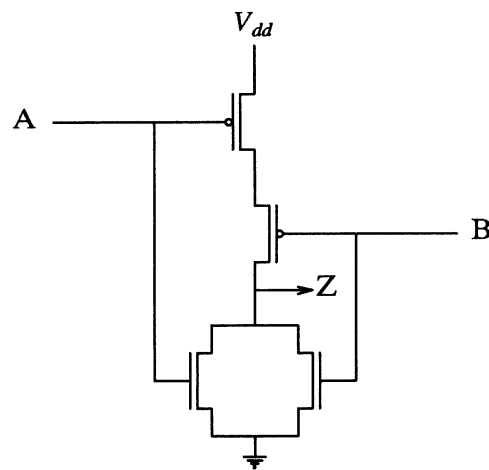
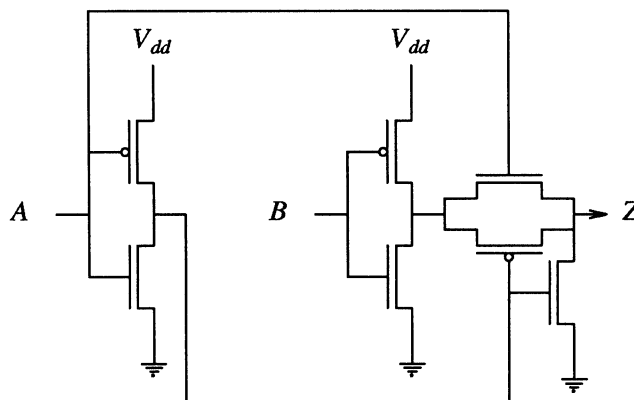


Figure 1.6. CMOS NOR gate.

$Z = AB'$. These types of complicated circuits are quite common in MOS VLSI design. Furthermore, MOS technology allows the designer to use several other “tricks”. For example, pre-charging of wires is often used: A wire is first charged to a high voltage level and then isolated. The wire has capacitance associated with it; therefore it will “remember” the high voltage for quite some time, unless it becomes connected to ground. In order to explain these very subtle phenomena properly, the so called *switch-level* models [6, 19, 30] have been developed. In such models it appears necessary to include a third value \times in order to capture output voltages that are neither 0 nor 1. In switch-level models it is possible to define “excitation functions” analogous to those in gate circuits. Once the excitation functions have been obtained, race analysis must be performed, as described in the previous section. We will return to these problems in Chapter II.

Figure 1.7. CMOS circuit realizing the function AB' .

1.4. Previous Work[†]

The theory of asynchronous circuits had its beginning in the 1950's with the work of Huffman [32], and Muller and Bartky [40, 45]. Huffman used what we will call the *feedback-delay* model where a set of feedback variables represents the state of a circuit. These variables correspond to a set of wires with the property that cutting them would break all of the loops in the circuit. With this set of state variables, Huffman used a binary race model to analyze state transitions. Muller and Bartky used what we will call the *gate-delay* model, where the outputs of all the gates constitute the state variables, and a binary model is used to analyze races. Huffman's race analysis is rather informal, whereas Muller's is formally defined. Both models use what has been later called the general multiple winner (GMW) model, in the sense that the delays are arbitrary and any nonempty subset of the set of unstable gates is allowed to change. However, in the Huffman model only the feedback wires are assumed to have delays: i.e. the gates are assumed to be ideal and delay-free. The Huffman model is often depicted as in Fig. 1.8. Usually, the minimum number of feedback variables is much smaller than the number of gates. Consequently, the feedback variable model has been, and continues to be, quite widely used.

[†] The reader is referred to [49, chapter II] for a more detailed treatment.

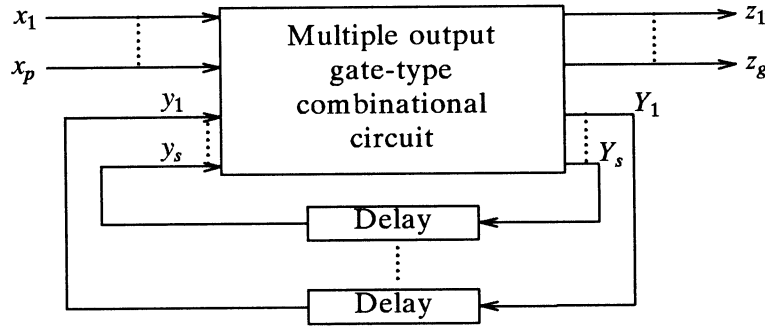


Figure 1.8. Huffman model for asynchronous gate circuits.

While the feedback-delay model gives the correct stable states, the state transitions predicted by this model are not always correct. These observations were made quite early [32, 39], and various types of hazards were then used in an attempt to explain these discrepancies. For a detailed discussion of this approach the reader is referred to the book by Unger [57] and to Chapter 2 in [49]. Roughly speaking, one first obtains a flow table using the feedback variables. The rows of the flow table correspond to those states that are stable for some input; the columns correspond to the input n -tuples. The entry in row i , column j is the stable state that the circuit eventually reaches if it is started in state i and the input j is applied and held constant; otherwise, the table entry is undefined. The undefined condition arises if more than one stable state can be reached (a “critical” race), or if the circuit enters an oscillation. The stable entries (i.e. entries where the next state is the same as the present state) are correct, but one has to perform a series of corrections to the unstable entries if certain hazards are present — not a very attractive theory. An even stronger reason for not using such a theory is Langdon’s example [34], which demonstrates that different sets of feedback variables may lead to different flow tables, and these differences cannot be accounted for by any known hazards. Thus the feedback variable approach is not entirely correct.

In view of these difficulties with the feedback variable approach, a return to Muller’s gate-delay model, using gate outputs as state variables, seemed necessary. This approach was advocated, for example, by Langdon [34], but still with binary race models. Precise mathematical formulations of such race models were developed

by Brzozowski and Yoeli [15-17]. In particular, they formalized the GMW race model. We will return to binary race models shortly.

In the mid-1960's, ternary models were introduced for the analysis of races and hazards in asynchronous gate circuits [26, 62]. In particular, Eichelberger proposed a ternary simulation of a circuit using a third value, \times , denoting an intermediate or unknown signal. His method can be used to predict state transitions and detect static hazards (a gate is said to have a static hazard for an input transition if its output has the same value before and after the transition but may momentarily change during the transition). Since the algorithm is quite efficient, it has been widely used [5, 33]. Unfortunately, some discrepancies were noted between the results predicted by ternary simulation and those predicted by the binary analysis [3, 17]. Again it seemed that the theories did not quite fit. A detailed discussion of these problems was presented by Brzozowski and Yoeli [17], who also conjectured that the theories would fit if one used not only gate outputs but also wire signals as state variables — the *gate- and wire-delay* model. This conjecture was finally proved by Brzozowski and Seger [12]. Thus a fit has been found between ternary simulation of a circuit using the gate-delay model and the binary GMW analysis of the gate- and wire-delay model. The proof of this result is quite involved.

Another approach to detect races and hazards, that has been advocated by some authors [1, 27], is the use of multiple-valued logic. Here, “multiple” means substantially more than three. The basic idea is to introduce specific values for 0 changing to 1, 1 changing to 0, static 1-hazards, static 0-hazards, etc. It is straightforward to derive the extended gate functions for simple AND, OR and inverter circuits in this multiple valued logic. Unfortunately, it is not clear whether this extra complexity really gives more information than a ternary model. Furthermore, it is not clear how to apply these ideas to switch-level models. For a more detailed discussion of these approaches, the reader is referred to [49, chapter 2].

In the last decade, digital circuit technology has undergone tremendous changes and MOS circuits have become widespread as a result of the VLSI revolution. In some ways these circuits resemble relay contact circuits more than gate circuits, and it has been recognized that theories based on gates are not adequate for some aspects

of MOS circuits. To remedy this, switch-level models like Bryant's [6] have been developed. Bryant [5] adapted Eichelberger's ternary simulation algorithm to switch-level MOS models, but justified the use of this technique only by examples. Lengauer and Näher [35] defined a sort of race analysis model for MOS circuits which uses the three values 0, 1, and \times , and proved that this race model corresponds exactly to ternary simulation, thus providing a mathematical justification for its use. At this point it appeared that the theory of MOS circuits was diverging from that of gate circuits, and that the latter would cease to have much significance. However, this thesis shows that this is not the case. In fact, we will present a theory of asynchronous circuits that handles both gate models and MOS switch-level models.

1.5. Outline of Thesis

In Chapter II we describe a very general framework for studying asynchronous circuits; this framework is used throughout the remainder of the thesis. A circuit is modeled as a directed labeled graph. There is a ternary (three-valued) excitation function associated with each vertex; thus each vertex is assumed to have a delay. The values used are the usual 0 and 1, but, in addition to these, a third value \times is introduced to denote an intermediate or undefined value. A vertex with indegree 0 is an input vertex with the excitation function equal to the value of the input variable. Vertices with indegree ≥ 1 are called function vertices. For gate circuits, the excitation function for a function vertex is the ternary extension of the Boolean function realized by the corresponding gate. For switch-level circuits, we describe several different approaches for computing the excitation functions; these represent different design philosophies for CMOS and NMOS circuits. These switch-level models properly handle pre-charged logic, charge sharing, and other very subtle properties of modern CMOS and NMOS designs. The main advantages of this framework are mathematical precision, simplicity, and generality which allow us to develop a theory that is applicable to gate circuits as well as to MOS switch-level circuits.

In Chapter III the behavior of a circuit is studied when the delays in each vertex of the network are arbitrary, but finite. A new race model, called the extended multiple winner (XMW) model, is defined. This race model can be used to analyze the behavior of a circuit under such a delay assumption. A novel feature of the XMW

model is that, if a vertex currently has the value 0 and its excitation function has the value 1, the vertex can change not only to 1, but also to \times . This captures the fact that changes can be relatively slow, and that a gate or an internal node in a MOS circuit can take on an undefined value for a non-negligible period of time during a transition from 0 to 1 or from 1 to 0.

The XMW model is computationally intractable, since it may require exponential time in the worst case. However, we show that the ternary simulation algorithm, suggested by Eichelberger [26], can be used to obtain essentially the same information in time polynomial in the number of vertices. In [13, 49] a similar result was proved that related ternary simulation to a binary race analysis of a gate circuit in which both the gates and the wires can have arbitrary, but finite, delays. The proof of this result was quite involved. In contrast to this, the proof relating ternary simulation and XMW analysis is simpler and much more natural.

As mentioned earlier, a common method of analyzing asynchronous circuits is to use a feedback-delay model. Unfortunately, in classical binary race models this is not always correct. In fact, there exist examples in which analyses using different sets of feedback lines yield completely different results. (Note that this is true even if all known types of hazards are taken into account.) In contrast to this, the use of feedback variables can be fully justified in the XMW model, as is shown in the last part of Chapter III. This solves a long-standing open problem.

The XMW model captures many of the ideas in so called *speed-independent* [45] or *delay-independent* circuit design. The underlying idea of delay-independence is to design circuits in such a way that they operate correctly, no matter what the internal delays are. With the arrival of VLSI circuits, this design philosophy has been advocated by many authors [23, 25, 37, 43]. One of the reasons for this is that the problem of distributing a high-frequency clock signal over a large VLSI chip is quite difficult to solve. However, using the theory developed for the XMW model together with the main result of [13, 49], it is shown that the class of circuits that can be realized using delay-independent design is very small. For example, in Chapter IV of the thesis, it is shown that it is not possible to construct a “divide-by-two” circuit in delay-independent fashion: i.e. it is not possible to design a delay-

independent circuit with one input and one output such that the output changes with half the frequency of the input. Recently, Ebergen [25] showed that a very large class of circuits (including divide-by-two circuits) could be realized in a delay-independent design if a small set of basic “building blocks” could be designed delay-independently. Unfortunately, our characterization of the class of delay-independent circuits shows that such building blocks do not exist.

The reason that there do not exist delay-independent designs for many functions is the “pessimism” of the assumption of delay-independence. In particular, it is not very realistic to assume that delays can be arbitrarily large. This points out the need for more realistic race models. Two such models are the topics of Chapters V and VI respectively. In the first model, called the almost-equal-delay (AED) model, the delay $\Delta(t)$ of a vertex is assumed to be in the interval $\Delta \pm \epsilon$, where ϵ is much smaller than Δ . The AED model (partly because of historical reasons, partly because of the underlying assumptions) is a binary model, and thus is tailored primarily towards gate circuits. Such a model is appropriate when a circuit is built of gates which are very similar. The AED model, as described in this thesis and in [51], is an extension of the almost-equal-delay model, originally suggested by Brzozowski and Yoeli [16]. As with most race models, using the model directly to analyze the behavior of a circuit is computationally intractable. However, it is shown in Chapter V that there also exists a practical simulation algorithm, called the TAED algorithm, that is closely related to the AED model. In fact, the TAED algorithm is as efficient as the commonly used “unit delay” model that most commercial simulators use.

The AED model is somewhat “optimistic”, not capturing timing problems that are quite likely to occur. This is perhaps not too surprising since the AED model is more closely related to the unit delay model than to the XMW model. It may not be very realistic to assume that all delays are roughly the same; for example, an inverter is almost certainly much faster than a complex gate.

In Chapter VI we first develop a realistic delay model, called the extended inertial delay (XID) model. In this model the sizes of the delays are bounded by lower and upper limits. For example, an inverter in the circuit can have its delay bounded

by $5\text{ns} \leq \Delta(t) < 7\text{ns}$, whereas the delay in a NAND gate may be bounded by $6\text{ns} \leq \Delta(t) < 10\text{ns}$. In the remaining part of the chapter, we develop a race model, called the extended bounded delay (XBD) race model, that can be used to predict the behavior of the circuit. In fact, we prove that the XBD race model captures the behavior exactly, if one assumes that each component consists of an ideal (delay-free) device connected in series with an extended inertial delay.

The XBD race model is continuous, and thus computationally intractable. However, there is also an efficient algorithm, called the ternary bounded delay (TBD) algorithm; we show that the results of this algorithm summarize the behavior of a circuit according to the XBD race model exactly.

In Chapter VII, we study the general complexity of the race analysis problem. For example, the stable state reachability (SSR) problem is quite a natural question: If a circuit is started in some stable state and some input values are changed, will it eventually end up in a unique new stable state? For the XMW model and the GMW model (with both gate and wire delays), this problem is solvable in polynomial time. However, for most other race models, the problem is shown to be intractable. In practice the situation is much better though, since if a circuit has not reached a stable state within some relatively short period of time, one may assume that it contains some design flaw that should be corrected.

Finally, in Chapter VIII we summarize the main contributions of the thesis and discuss different areas of future research. In particular, the problem of verifying the correctness of an asynchronous circuit is discussed.

Chapter II

A Unified Framework

In this chapter we introduce a general framework for studying asynchronous circuits[†]. This framework is useful for describing gate circuits as well as MOS switch-level circuits. In the first section, the framework is defined formally. Some concepts that will be used throughout the remainder of the thesis are also defined. In the second half of the chapter, we show how this framework can be used to model gate circuits and switch-level circuits.

2.1. Graph Model

A rather general concept of a *network* is introduced in this section. We will use the convention that x , y , and z denote vectors of state variables, whereas x_i , y_i , and z_i denote single components of the corresponding vectors. Similarly, a , b , c , and d denote particular constant vectors of state variable values, and a_i , b_i , c_i , and d_i denote their components.

Let $B = \{0, 1\}$ be the set of the two usual binary values, and let $T = \{0, 1, \times\}$. The symbol \times will be used to denote an unknown or intermediate value.

A *network* N is a finite directed labeled graph $N = \langle V, E, x, y, Y \rangle$, where

[†] Part of this work will appear in [14].

$V = \{1, \dots, m\}$ is a set of vertices[†],

$E \subseteq V \times V$ is a set of edges,

$x = (x_1, \dots, x_n)$, $n \leq m$, is a vector of *input variables* taking values from \mathcal{T} ,

$y = (y_1, \dots, y_m)$ is a vector of *vertex variables* taking values from \mathcal{T} ,

$Y = (Y_1, \dots, Y_m)$ is a vector of ternary *excitation functions*.

Vertices $1, \dots, n$ are all of indegree 0, and are called *input vertices*. Vertices $n+1, \dots, m$ are *function vertices* and are all of indegree ≥ 1 . The excitation function of an input vertex j is the function $Y_j: \mathcal{T} \rightarrow \mathcal{T}$ defined simply as $Y_j = x_j$. For a function vertex j the excitation function is a ternary function $Y_j: \mathcal{T}^{d_j} \rightarrow \mathcal{T}$, where d_j denotes the indegree of vertex j . An edge $(i, j) \in E$ shows that Y_j is a function of y_i . Thus, for a function vertex, Y_j depends only on some subset of $\{y_1, \dots, y_m\}$. The ordered pair $\langle x, y \rangle$, $x \in \mathcal{T}^n$, $y \in \mathcal{T}^m$, is called the *total state* of N . For notational convenience, we will treat an excitation function Y_j as a function of the total state of N , i.e. we will write $Y_j(x, y)$ [‡]. The vertex variable y_j is interpreted as the present state of a vertex, whereas the excitation function $Y_j(x, y)$ computes the value to which the vertex is trying to change, when the present input is x and the present state is y .

In the examples throughout the thesis we use the three ternary functions defined in Fig. 2.1. In fact, these functions are natural ternary extensions of their Boolean counterparts: OR, AND, and complement. More will be said about this later.

$+$	0	\times	1
0	0	\times	1
\times	\times	\times	1
1	1	1	1

\cdot	0	\times	1
0	0	0	0
\times	0	\times	\times
1	0	\times	1

a	0	\times	1
a'	1	\times	0

Figure 2.1. Ternary OR, AND, and complement.

[†] To avoid possible confusion we use the term *vertex* (not *node*) in the general network graph, reserving the term *node* to MOS circuits and related graphs.

[‡] Strictly speaking we should write $Y_j(\langle x, y \rangle)$, but the angle brackets are omitted to improve readability.

To illustrate the definition of a network consider Fig. 2.2, where the excitation functions are:

$$Y_1(x, y) = x_1 \quad Y_2(x, y) = x_2 \quad Y_3(x, y) = (y_1 + y_4)' \quad Y_4(x, y) = (y_2 + y_3)'.$$

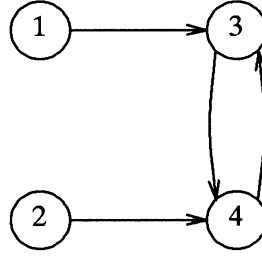


Figure 2.2. Network N_1 .

Define the partial order \sqsubseteq on \mathcal{T} as follows: $t_j \sqsubseteq t_j$ for all $t_j \in \mathcal{T}$, $0 \sqsubseteq \times$, and $1 \sqsubseteq \times$. The partial order is extended to \mathcal{T}^r , $r \geq 1$, in the obvious way: $s \sqsubseteq t$ iff $s_j \sqsubseteq t_j$ for $1 \leq j \leq r$. Also, the partial order is extended to ordered pairs in the natural way: $\langle a, b \rangle \sqsubseteq \langle c, d \rangle$ iff $a \sqsubseteq c$ and $b \sqsubseteq d$. For $s, t \in \mathcal{T}^r$, $r \geq 1$, we write $s \sqsubset t$ when $s \sqsubseteq t$ and $s \neq t$. The least upper bound, denoted *l.u.b.*, on the partial order \sqsubseteq is defined as usual.

Intuitively, \times is used to denote an “unknown” or “intermediate” value. Thus $s \sqsubset t$ indicates that s has less “uncertainty” (more binary values) than t .

The following fundamental assumption is made about the excitation function of any network N :

$$\langle a, b \rangle \sqsubseteq \langle c, d \rangle \text{ implies } Y(a, b) \sqsubseteq Y(c, d).$$

This is a monotonicity property of the excitation function that is consistent with our use of the value \times . Basically, if the total state is more uncertain, the excitation cannot become less uncertain.

If $y_i = Y_i(x, y)$ then vertex i is *stable*; otherwise it is *unstable*. A given total state $\langle x, y \rangle$ is *stable* if each vertex is stable. A network will remain in a stable total state indefinitely, unless the input changes, in which case the state becomes unstable. If there are two or more vertices that are unstable in a total state, we say that there is a *race*. In general, there may be several possible successor states for a given unstable

state. This set of possible successor states depends on the race model used, as elaborated in Chapters III, V and VI.

The following definitions will be used throughout the thesis. For any $a \in \mathcal{T}^n$ and $b \in \mathcal{T}^m$, define $U(a, b)$ to be the set of unstable vertices in b , i.e.

$$U(a, b) = \{j : 1 \leq j \leq m, \text{ and } b_j \neq Y_j(a, b)\}.$$

Similarly, let $B(b)$ denote the set of vertices that have binary values in $\langle a, b \rangle$, i.e.

$$B(b) = \{j : 1 \leq j \leq m, \text{ and } b_j \in B\}.$$

Finally, let $BE(a, b)$ denote the set of vertices that have binary excitations in the total state $\langle a, b \rangle$, i.e.

$$BE(a, b) = \{j : 1 \leq j \leq m, \text{ and } Y_j(a, b) \in B\}.$$

2.2. Application to Gate Circuits

It is shown in this section how gate circuits can be analyzed using the framework established in the previous section. The correspondence between a gate circuit and the graph model is very natural, as described below.

Given a gate circuit with n external inputs, the graph N is formed in the following way: There is an input vertex for each input variable to the gate circuit and a function vertex for every gate. There is an edge between vertex i and vertex j iff the gate j has at least one of its inputs connected to the output of gate i if $i > n$, or to input x_i if $i \leq n$. Note that there may be more than one wire from the output of gate i to gate j in a circuit, whereas in N we have at most one edge from i to j . This will affect the function associated with each vertex. Given any Boolean function $f : B^r \rightarrow B$ the *ternary extension* \mathbf{f} of f is defined as follows. For any $\mathbf{z} \in \mathcal{T}^r$,

$$\mathbf{f}(\mathbf{z}) = l.u.b. \{f(z) : z \in B^r, z \sqsubseteq \mathbf{z}\}.$$

The ternary extensions of OR, AND and complement functions were shown in Fig. 2.1.

Two steps have to be carried out to compute the ternary function associated with the function vertex i . First, the ternary extension of the original Boolean gate function must be computed. Second, the input variables are identified with the output vertex feeding them.

To illustrate the above procedure, consider the gate circuit G_2 of Fig. 2.3. Using the procedure outlined above one can easily derive the abstract network N_2 of Fig. 2.4, with the excitation functions:

$$\begin{array}{lllll} Y_1(x, y) = x_1 & Y_3(x, y) = y_1 y_9 & Y_5(x, y) = (y_3 + y_6)' & Y_7(x, y) = y_2 y_6 & Y_9(x, y) = (y_7 + y_{10})' \\ Y_2(x, y) = x_2 & Y_4(x, y) = y_1 y_{10} & Y_6(x, y) = (y_4 + y_5)' & Y_8(x, y) = y_2 y_5 & Y_{10}(x, y) = (y_8 + y_9)' \end{array}$$

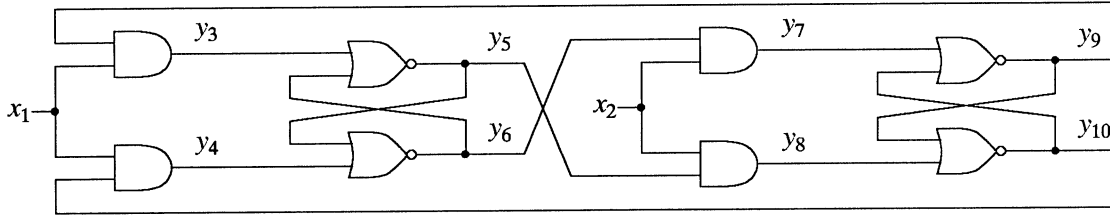


Figure 2.3. Gate circuit G_2 .

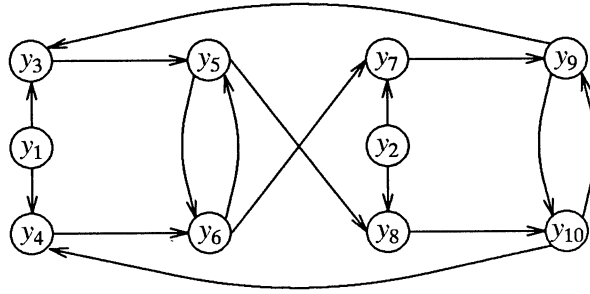


Figure 2.4. Network N_2 .

A more complicated situation for deriving the excitation function is demonstrated in the following example. Consider the gate circuit G_3 of Fig. 2.5 consisting of a 2-input XOR (exclusive OR) gate, with both inputs fed by a single inverter. The ternary extension of the XOR gate is $c'd + cd'$, where c and d denote the two inputs. Identifying the inputs and simplifying according to the ternary algebra gives the excitation functions:

$$Y_1(x, y) = x_1 \quad Y_2(x, y) = y_1' \quad Y_3(x, y) = y_2' y_2$$

Note that the last excitation function is *not* identical to 0.

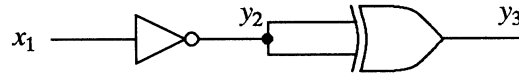


Figure 2.5. Gate circuit G_3 .

In classical race models, the basic assumption is that gates can only be in the states 0 and 1, and that transitions from 0 to 1 or from 1 to 0 are instantaneous. However, it is reasonable to assume that a transition takes a nonnegligible amount of time, and may go through an intermediate voltage. Some gates with inputs connected to a vertex with an intermediate voltage may interpret this voltage as a 1, whereas others may interpret it as 0. The \times value and the monotonicity requirement of Y together capture this uncertainty.

In the model above we associated vertices (and therefore delays) with the gates only. However, if the delays in the wires need to be taken into account, one simply adds “delay” vertices in the abstract network. A delay vertex is a vertex with indegree 1, outdegree 1, and with the identity function as its excitation function.

2.3. Switch-Level Circuits

Switch-level models are frequently used for simulating MOS VLSI circuits. In switch-level models, each transistor is viewed as a switch that can be turned on or off by the signal on the gate of the transistor. The first formal switch-level model was introduced by Bryant [6] for general MOS circuits. Specializations of such models to CMOS circuits were described in [19]. In this section we show how the models of [6] and [19] can be adapted to our framework. The notation is modeled after [19].

An *S-graph* (S for switch) is a finite, undirected, labeled graph with:

- 1) *Supply nodes* shown as black dots and labeled by 0 or 1.
- 2) *Internal nodes* shown as white dots.
- 3) *Key nodes*, which are special internal nodes, each of which is labeled by a different *key letter* Q_1, \dots, Q_k .
- 4) *Input letters* A_1, \dots, A_n .
- 5) *Edges*, each of which is labeled with a symbol of the form Q^N or Q^P , where Q is either an input letter or a key letter. Each key letter appears as some edge label.

The edges represent the transistors in the circuit, whereas the nodes represent the connection points. A superscript P on an edge label indicates that the transistor is P-type; a superscript N denotes an N-type transistor.

A (ternary) *input-key state* is an assignment of 0, 1, or \times to each input letter and to each key letter. Similarly, a *total state* of an S-graph is an assignment of 0, 1, or \times to each input letter and to each internal node. A *path* in the graph is a sequence of nodes in which consecutive nodes are connected by at least one transistor and no internal node in the sequence is a supply node. (The reason for this last condition will be explained later.) Note that we allow the first or the last node in a path to be a supply node. A path in the graph consisting of only P -transistors is called a *P-path*. An *N-path* is defined similarly. A path is said to be *definite* if the values of the gates of all the P-transistors in the path are 0, and the values of the gates of all

N -transistors in the path are 1. A path is said to be *indefinite* if it is not definite, and the values of the gates of all the P-transistors in the path are either 0 or \times , and the values of the gates of all the N -transistors in the path are either 1 or \times .

Following [6], we define a *channel-connected subgraph* to be a subgraph \dot{S} of the S -graph such that there is at least one path between any two internal nodes in \dot{S} . Except for the supply nodes, the channel-connected subgraphs are disjoint.

A property of CMOS circuits that many switch-level models fail to capture is the fact that a P-transistor conducts a 1-signal well, but a 0-signal rather poorly. Similarly, an N-transistor conducts a 0 well, and a 1 poorly. For this reason, we must distinguish between P- and N-paths[†]. We do this by introducing three different *path functions*. However, before describing these functions, we make the following notational assumption. Assume that the nodes in the S -graph are numbered as follows: The ground node (the supply node labeled 0) is numbered 0, the power supply node (the supply node labeled 1) is numbered 1, and the remaining internal nodes are numbered 2, 3, For a channel-connected subgraph \dot{S} and any two nodes i and j in \dot{S} the three path functions are defined as follows: First, if $i = j$, let $p_{ii} = n_{ii} = t_{ii} = 1$. Otherwise, let

$$p_{ij} = \begin{cases} 1 & \text{if there is a definite P-path from } i \text{ to } j \\ \times & \text{if there is no definite P-path but there is an indefinite P-path from } i \text{ to } j \\ 0 & \text{if there is no definite or indefinite P-path from } i \text{ to } j \end{cases}$$

$$n_{ij} = \begin{cases} 1 & \text{if there is a definite N-path from } i \text{ to } j \\ \times & \text{if there is no definite N-path but there is an indefinite N-path from } i \text{ to } j \\ 0 & \text{if there is no definite or indefinite N-path from } i \text{ to } j \end{cases}$$

and

[†] We say that an N-path (P-path) to 0 (1) is “good”, whereas a P-path (N-path) or a mixed P- and N-path to 0 (1) is “bad”.

$$t_{ij} = \begin{cases} 1 & \text{if there is a definite path from } i \text{ to } j \\ \times & \text{if there is no definite path but there is an indefinite path from } i \text{ to } j \\ 0 & \text{if there is no definite or indefinite path from } i \text{ to } j. \end{cases}$$

(Note that certain types of paths, e.g. “self-dependent” paths [19], may be disregarded to model the circuit behavior more accurately.)

The path functions can be computed in a number of different ways. One possibility is to enumerate all possible paths and then identify their types. However, a more efficient method is to use a “signal flow graph” approach, similar to the procedure for finding a regular expression from a finite automaton [11]. This approach can be viewed as solving a set of linear equations with a somewhat modified version of Gaussian elimination. For more details, the reader is referred to [8, 9] where this approach is analyzed for different graphs and is applied to switch-level simulation.

Consider the CMOS circuit C_4 of Fig. 2.6(a). The corresponding S-graph is shown in Fig. 2.6(b). (The input and key node naming convention has been ignored in order to simplify the notation.) Note that C_4 consists of two channel-connected subgraphs. For the internal node a , it is easy to verify that, $p_{a1} = t_{a1} = A'$, $n_{a1} = 0$, $n_{a0} = t_{a0} = A$, and $p_{a0} = 0$. Using the ternary OR, AND and complement defined in Section 2.1, we find the path functions for node c :

$$\begin{aligned} p_{c1} &= Q'B' & n_{c1} &= 0 & t_{c1} &= p_{c1} + AB' \\ p_{c0} &= 0 & n_{c0} &= Q + AB & t_{c0} &= n_{c0} + Q'B. \end{aligned}$$

In general, t_{i1} must always be of the form $p_{i1} + f_{i1}$ for some function f_{i1} . The same holds for t_{i0} .

Let Y_i denote the node excitation function of the internal node i . There are several ways to use the ternary path functions to compute this function. However, before we describe these models, we prove the following proposition for use later.

Proposition 2.1 If $a \in \mathcal{T}$ and $b \in \mathcal{T}$ then

$$a + (a+b)' \times = a + b' \times.$$

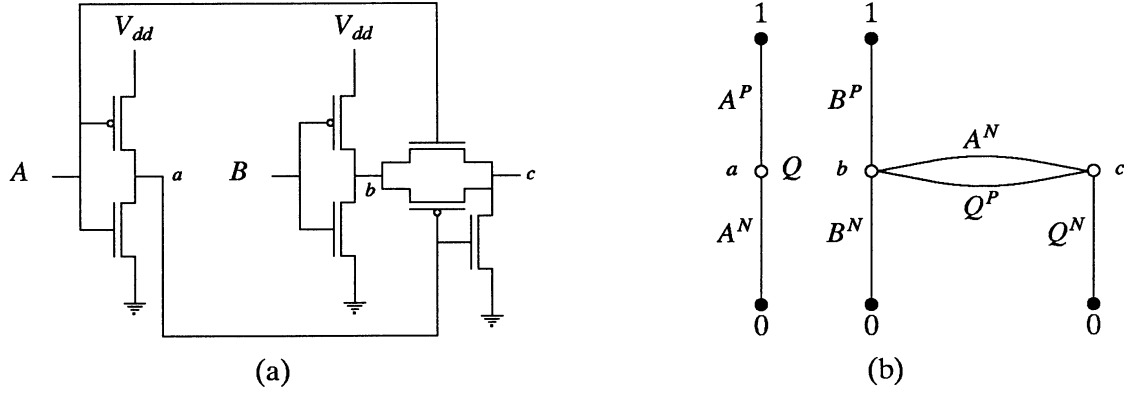


Figure 2.6. (a) CMOS circuit C_4 ; (b) corresponding S-graph S_4 .

Proof: There are three cases. If $a=1$ both the LHS and the RHS are 1, since $1+0=1+1=1+\times=1$. Secondly, if $a=\times$, both the LHS and the RHS are \times , since the second terms in the LHS and the RHS can only contribute an \times or a 0, and $\times+\times=\times+0=\times$. Finally, if $a=0$ then the LHS is equal to $0+(0+b)'\times=b'\times$, which is equal to the RHS. ■

There are two distinct approaches to defining the excitation function for a node: we can either use a *static* approach or a *dynamic* approach. In the static approach the channel-connected subgraph is viewed as a “super gate”, with inputs and outputs. The inputs are the values of the gates of the transistors of the subgraph, whereas the internal nodes are the outputs. The excitation function for a node gives the value that the node eventually would take if the input signals were frozen at their current values. (The excitation function in these models is often called the steady-state response.) This approach implies that potential races inside the channel-connected network are completely ignored. On the other hand, in a dynamic approach the excitation function of a node depends almost entirely on the values of its direct neighbor nodes and on the transistors connecting the node with its direct neighbors. Hence, in a dynamic approach, races inside the channel-connected network *are* taken into account.

Virtually all switch-level models suggested in the literature, e.g. [6, 30, 46, 56], have used the static approach. Sundblad and Svensson [55] are the only authors that have considered a dynamic approach, as far as we are aware. In this section we will limit our discussion to some typical excitation models using the static approach. However, in Appendix A, we describe several families of excitation functions derived using both static and dynamic approaches.

The excitation models defined below use the same basic concept: the node excitation function Y_i will always be of the form $o_i + (o_i + z_i)' \times$. Intuitively, o_i denotes the conditions under which Y_i should be 1, and z_i gives the conditions under which Y_i should be 0. In view of Proposition 2.1, we will immediately simplify these expressions to the form $o_i + z_i' \times$. The first two models are:

$$\text{Model 1: } Y_i = p_{i1}t'_{i0} + (n_{i0}t'_{i1})' \times$$

$$\text{Model 2: } Y_i = t_{i1}t'_{i0} + (t_{i0}t'_{i1})' \times$$

Model 1, introduced in [19], yields a binary output iff there is a definite P(N)-path to 1(0) and no path whatsoever to 0(1). Furthermore, whenever there is a “fight”, i.e. both a path to 1 and a path to 0, whenever there is only a bad path to 0 or 1, or whenever the node is isolated, i.e. there is no path to 1 and no path to 0, the node excitation is \times . This is a very restrictive model and captures stringent design rules for combinational static CMOS circuits. The second model is more traditional and corresponds to a special case of the model in [6]. Here there is no distinction at all between P- and N-transistors, so we have $Y = 1$ (0) iff there is some definite path to 1 (0) but no path to 0 (1).

Consider again the CMOS circuit C_4 of Fig. 2.6. Using the path functions derived earlier for node a , it is easy to verify that both models yield the node excitation function $Y_a = A'$. A more complicated example is node c . Model 1 and 2 yield the following node excitation functions:

$$\begin{aligned}
\text{Model 1: } Y_c &= p_{c1}t'_{c0} + (n_{c0}t'_{c1})' \times = \\
&= Q'B'(Q + AB + Q'B)' + ((Q + AB)(Q'B' + AB'))' \times = \\
&= \dots = Q'B' + (Q'A' + AB') \times
\end{aligned}$$

$$\begin{aligned}
\text{Model 2: } Y_c &= t_{c1}t'_{c0} + (t_{c0}t'_{c1})' \times = \\
&= (Q'B' + AB')(Q + AB + BQ')' + ((Q + AB + BQ')(Q'B' + AB'))' \times = \\
&= \dots = Q'B' + (A'QQ' + AB') \times.
\end{aligned}$$

Note that a substantial amount of reduction has been performed above in order to get the final expressions. However, all simplifications consist of sequences of simple steps, and can be carried out by a program. Also, even if the cost of doing the simplification is substantial,[†] this is a preprocessing step which is done only once. The efficiency gained in the simulation phase should justify the preprocessing. Fortunately, many channel-connected subgraphs are quite small, making a brute force approach feasible. Furthermore, it is certainly not necessary to find the minimum-sized formulae.

The above node excitation functions fail to capture the fact that there is a certain amount of capacitance in MOS circuits. In particular, the key nodes can have some capacitance associated with them; hence there is a certain amount of “memory” in each key node. The case when there is only one key node in every channel-connected subgraph can be handled in a straightforward way. The basic assumption is that a 1 (0) stored on a key node can only determine the excitation of that node, if the node is completely isolated from 0 (1). One can verify that the following models do take this into account.

$$\text{Model } 1^M: Y_i = (p_{i1} + y_i)t'_{i0} + ((n_{i0} + y_i')t'_{i1})' \times$$

$$\text{Model } 2^M: Y_i = (t_{i1} + y_i)t'_{i0} + ((t_{i0} + y_i')t'_{i1})' \times$$

For example, using model 1^M , we get Y to be 1 if there is a good path to 1 and no path to 0, or if the previous value was 1 and there is no path to 0. A dual situation holds for $Y = 0$.

[†] The simplification problem is well known to be NP-hard.

Consider again the CMOS circuit C_4 of Fig. 2.6. Using model 1^M and assuming that node c is a key node, we get the node excitation functions for nodes a and c :

$$\begin{aligned}
 Y_a &= (p_{a1} + y_a)t'_{a0} + ((n_{a0} + y'_a)t'_{a1})' \times = \\
 &= (A' + y_a)A' + ((A + y'_a)(A'))' \times = \\
 &= \dots = A' \\
 Y_c &= (p_{c1} + y_c)t'_{c0} + ((n_{c0} + y'_c)t'_{c1})' \times = \\
 &= (Q'B' + y_c)(Q + AB + Q'B)' + (((Q + AB) + y'_c)(Q'B' + AB'))' \times = \\
 &= \dots = Q'B' + (Q'A'y_c + AB') \times
 \end{aligned}$$

In the above discussion only the key nodes were assumed to have memory. Furthermore, it was assumed that all key nodes have the same “size”. A common technique in MOS circuits is the use of pre-charged lines. In such circuits, certain nodes are designed with a substantially higher capacitance. This can be modeled as if these nodes had “greater size” than normal nodes. Furthermore, transistors may have different “strengths” (conductances). We will not derive node excitation functions for such cases here, but the interested reader is referred to Appendix A and to Bryant’s work [8, 9]. Here it suffices to say that, using the MOSSIM model [6] and the procedure described in [8, 9], one can derive ternary functions for the node excitations. Using this approach, not only CMOS, but also NMOS circuits can be handled.

2.4. Application to MOS Circuits

It is shown in this section how all the different switch-level models described in the previous sections can be handled within the framework established earlier. There is one input vertex for every input letter in the S-graph. Furthermore, there is one function vertex for each internal node in the S-graph. The excitation function associated with a function vertex is simply one of the node excitation functions described in Section 2.3. (It is easy to verify that all node excitation functions of Section 2.3 satisfy the monotonicity property of Section 2.1.) There is an edge between vertex i and vertex j if the excitation function of vertex j depends on the input letter A_i if $i \leq n$, or on the key node y_i if $i > n$.

For example, consider the CMOS circuit C_4 of Fig. 2.6. The corresponding network N_5 , when the node excitation model 1 is used, is shown in Fig. 2.7. The excitation functions are:

$$Y_1 = A, \quad Y_2 = B, \quad Y_3 = y_1', \quad Y_4 = y_2'(y_3' + y_1'y_3 + \times), \quad Y_5 = y_2'y_3' + (y_1'y_3' + y_1y_2')\times$$

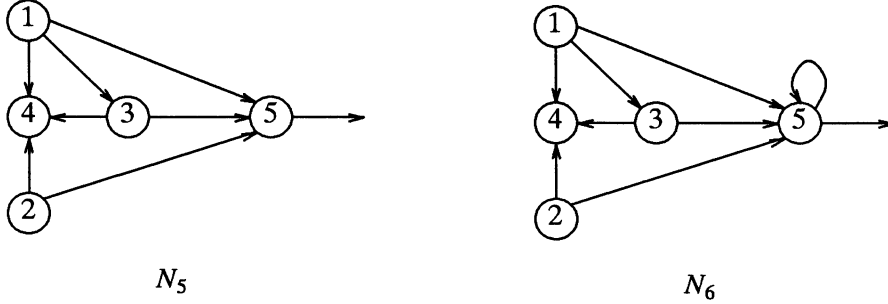


Figure 2.7. Networks N_5 and N_6 .

On the other hand, if the node excitation function 1^M is used, we get the network N_6 of Fig. 2.7(b), with excitation functions:

$$Y_1 = A, \quad Y_2 = B, \quad Y_3 = y_1', \quad Y_4 = y_2'(y_3' + y_1'y_3 + \times), \quad Y_5 = y_2'y_3' + (y_1'y_3'y_5 + y_1y_2')\times$$

In the model above, only the input letters and the internal nodes have state variables associated with them. There are several natural ways to modify this model. First, we can associate a state variable with each internal node and with each transistor. The excitation function for a transistor is trivial: for an N-transistor i , controlled by node y_j , the excitation is $Y_i = y_j$; similarly, for a P-transistor i , controlled by y_j , $Y_i = y_j'$. On the other hand, for many types of circuits and excitation models, it is sufficient to associate state variables with the input letters and the key nodes — sometimes a substantial saving.

In summary, MOS switch-level models fit very nicely into the general framework introduced in Section 2.1.

Chapter III

The Extended Multiple Winner Race Model

In this chapter we study the behavior of a network when the relative sizes of the delays in the network can be arbitrarily large, though finite[†]. For such a delay model we define a race model, called the extended multiple winner (XMW) model, that can be used to analyze the behavior of the network when the network is started in some stable total state and the input changes. The XMW model is computationally intractable since it may involve an amount of work exponential in the number of state variables. However, in this chapter we show that the result of ternary simulation, as suggested by Eichelberger in 1965, exactly summarizes the XMW analysis. Hence, the behavior of a network under the above delay assumption can be computed very efficiently. Finally, we show that in the XMW model it is possible to reduce the number of state variables to the input vertices and a feedback vertex set and the analysis is still correct. This is a result that sharply contrasts with the classical race models in which analyses based on feedback variables are not always correct.

[†] Part of this work will appear in [14].

3.1. XMW Race Model

There are three basic ideas behind the XMW model. First, it is assumed that the input remains fixed after each change to give the network a chance to “stabilize”. This corresponds to the fundamental-mode operation assumption of [39]. Second, the past history is completely ignored in the sense that all unstable vertices have the same chance of “winning” a race no matter when they entered the race. Third, any unstable vertex with a binary present value may take on the intermediate value \times .

More formally, the XMW relation R_a on the set \mathcal{T}^m defines the set of successors for any total state $\langle a, b \rangle$, $a \in \mathcal{T}^n$, $b \in \mathcal{T}^m$, as follows. If b is stable, i.e. if $U(a, b) = \emptyset$, then the only possible successor is b , i.e. $b R_a b$. Otherwise, let $b R_a \bar{b}$, for any \bar{b} such that:

- 1) $\bar{b} \neq b$, and
- 2) $\bar{b}_i \in \{b_i, Y_i(a, b), l.u.b. \{b_i, Y_i(a, b)\}\} \quad 1 \leq i \leq m$.

No other pairs are related by R_a .

To illustrate the above definition, consider the network N_1 of Fig. 3.1 with excitation functions given by:

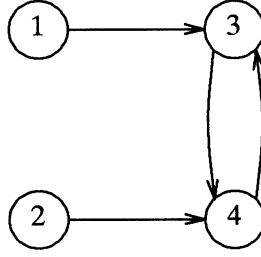
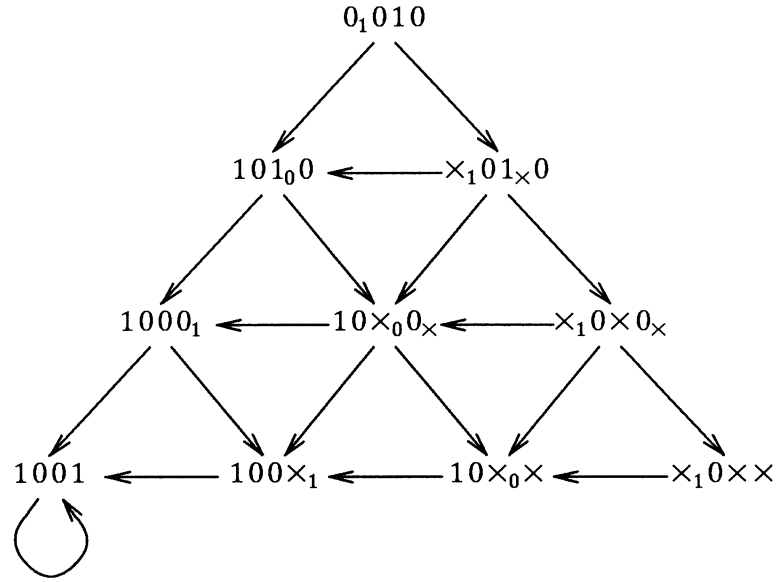
$$Y_1(x, y) = x_1 \quad Y_2(x, y) = x_2 \quad Y_3(x, y) = (y_1 + y_4)' \quad Y_4(x, y) = (y_2 + y_3)'$$

Assume that the network is started in the stable state $\langle \hat{a}, b \rangle = \langle 00, 0010 \rangle$ and that the input changes to $a = 10$. In Fig. 3.2 we show the XMW analysis for this transition. (Only those states that are reachable from 0010 are shown.) Unstable states are shown subscripted; the subscript denotes the value of the excitation function for that vertex. For example, $0_1 010$ indicates that vertices 2, 3 and 4 are stable and that vertex 1 is unstable with an excitation of 1.

The following proposition about the XMW relation is used later and can be easily verified.

Proposition 3.1 If $b, c \in \mathcal{T}^m$ and $b R_a c$, then $c \sqsubseteq l.u.b. \{b, Y(a, b)\}$.

For any input vector $a \in \mathcal{T}^n$ and any state $b \in \mathcal{T}^m$, define the set $\text{cycl}(R_a, b)$ to be the set of total states of N that appear in cycles reachable from b in the relation R_a .


 Figure 3.1. Network N_1 .

 Figure 3.2. XMW analysis of network N_1 .

Note that each stable state reachable from b is in $\text{cycl}(R_a, b)$. Formally,

$$\text{cycl}(R_a, b) = \{c \in T^m : b R_a^* c \text{ and } c R_a^+ c\},$$

where R_a^+ is the transitive closure of R_a , and R_a^* is the reflexive and transitive closure of R_a .

The concept of a transient cycle is introduced in order to capture the fact that delays cannot be infinite. This concept is very similar to the definition of a transient cycle in the GMW model [17] (which will also be considered later), except for one important difference. The basic idea is to call a cycle transient if there is some vertex that is unstable in all the states of the cycle and has the same value in all these states. However, this definition is slightly too restrictive as the following example shows. Suppose we have a cycle like $0_1 \times_0 \rightarrow 1_0 \times_1 \rightarrow 0_1 \times_0$ etc. Note that the vertex with the value \times is unstable in all the states and has the same value in all the states. However, since the excitation of that vertex oscillates between 0 and 1 it is reasonable to assume that such a cycle can persist indefinitely. Because of this, the definition of a transient cycle in the XMW model is somewhat more complicated than the corresponding idea in the GMW model.

A cycle is called *transient* if there exists a vertex v which is unstable in all of the states in the cycle, has the same value in all these states, and either that value is binary or the excitation of v is the same in all these states. If a cycle is not transient, it is called *nontransient*. Let

$$out(R_a, b) = \{c \in cycl(R_a, b) : c \text{ appears in a nontransient cycle} \}$$

The set $out(R_a, b)$ is the *outcome* of the XMW analysis of the behavior of N when started in total state b , in the sense that it consists of all the states N can be in, under nontransient conditions.

To illustrate the concepts above consider Fig. 3.2. There is only one cyclic state, namely 1001. Since this state is stable, the cycle is nontransient. Thus $out(R_{10}, 0010) = \{1001\}$. A more complicated example is provided by the network N_2 specified by the following excitation functions:

$$Y_1 = x_1 \quad Y_2 = (y_1 + y_3)' \quad Y_3 = (y_1 + y_2)'.$$

Let $\langle \hat{a}, b \rangle = \langle 1, 100 \rangle$ be the initial stable state and let the new input be $a = 0$. The states reachable from 100 are shown in Fig. 3.3. There are the following cycles:

$$[000, 011], [0 \times 0, 01 \times], [00 \times, 0 \times 1], [010], [0 \times \times], [001]$$

None of these cycles are transient, and hence the outcome is

$$out(R_0, 100) = \{000, 001, 010, 011, 0\times 0, 00\times, 01\times, 0\times 1, 0\times\times\}.$$

To illustrate the concept of a transient cycle, consider the network with the excitation functions:

$$Y_1 = x_1 \quad Y_2 = y_1' \quad Y_3 = (y_1 y_2 y_3)'.$$

One easily verifies that the cycle $[111, 110]$ is transient, when the network is started in the stable state $x_1 = 0, y = 011$ and x_1 is changed to 1.

The reader should note that $out(R_a, b) \neq \emptyset$. This follows from the fact that the sequence of states obtained by changing all unstable vertices to their excitations in every state of the sequence must eventually lead to a cycle. This cycle of states must be nontransient by definition, and thus all such states appear in $out(R_a, b)$.

The following property of the XMW relation will be used later. Assume the binary input vectors \hat{a} and a differ in at least two components, and that \tilde{a} is a binary input vector “between” \hat{a} and a (i.e. that \tilde{a}_j is either equal to \hat{a}_j or a_j for $j = 1, \dots, n$, but $\tilde{a} \neq \hat{a}$ and $\tilde{a} \neq a$). If a network N is started in the stable total state $\langle \hat{a}, b \rangle$, and the input is first changed to \tilde{a} and then later to a , the proposition states that the outcome after this second change is contained in the outcome obtained when the input is changed immediately from \hat{a} to a . More formally:

Proposition 3.2 Let $\hat{a}, \tilde{a}, a \in B^n$ be three input vectors such that $\tilde{a} \neq \hat{a}$, $\tilde{a} \neq a$, and $\tilde{a} \sqsubseteq l.u.b. \{\hat{a}, a\}$. Furthermore, assume that $\langle \hat{a}, b \rangle$ is a stable total state of N . Then

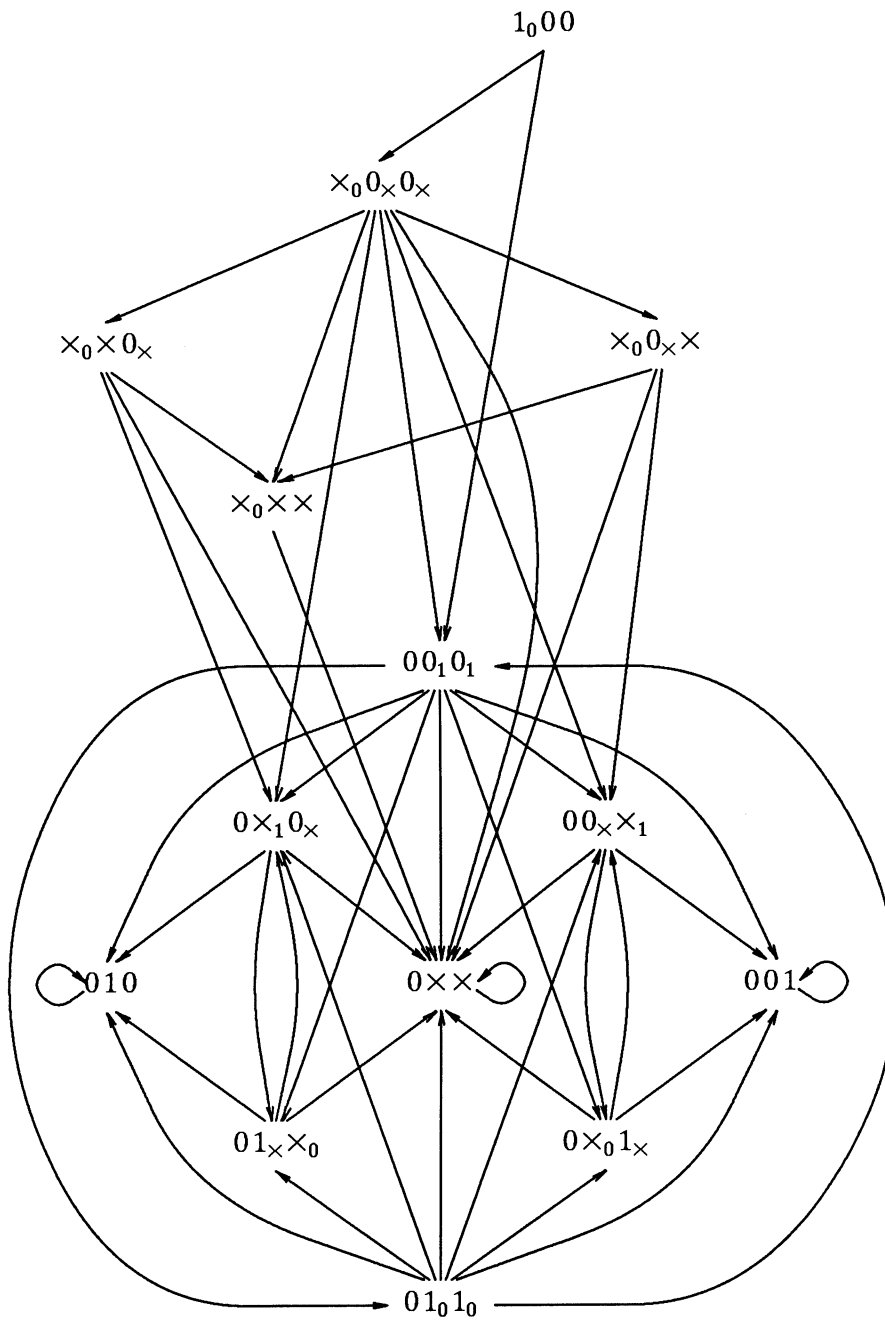
$$\{d : d \in out(R_a, c), \text{ where } c \in out(R_{\tilde{a}}, b)\} \subseteq out(R_a, b).$$

Proof: We prove the following stronger version of the claim:

$$\{d : d \in out(R_a, c), \text{ where } b R_{\tilde{a}}^* c\} \subseteq out(R_a, b).$$

Proposition 3.2 then follows directly from the fact that $out(R_{\tilde{a}}, b) \subseteq \{c : b R_{\tilde{a}}^* c\}$.

The proof consists of two steps. First we show that if $b R_{\tilde{a}}^* c$, then $b R_a^* c$. Second we show that if $b R_a^* c$, then $out(R_a, c) \subseteq out(R_a, b)$. Together, these give that

Figure 3.3. XMW analysis of network N_2 .

$$\{d : d \in \text{out}(R_a, c), \text{ where } b R_a^* c\} \subseteq \{d : d \in \text{out}(R_a, c), \text{ where } b R_a^* c\} \subseteq \text{out}(R_a, b)$$

establishing the claim.

The first property follows from the following observations. First, since $\langle \hat{a}, b \rangle$ is a stable total state of N , we can conclude that $(b_1, \dots, b_n) = \hat{a}$. Furthermore, by the definition of \tilde{a} , \tilde{a}_j is either equal to \hat{a}_j or a_j . Hence, it follows that $U(\tilde{a}, b) \subset U(a, b)$ and that $Y_j(\tilde{a}, b) = Y_j(a, b)$ for all $j \in U(\tilde{a}, b)$. Since only unstable vertices can change according to the XMW relation, it is easy to prove by induction on k that: if $b R_a^k d$ then $b R_a^k d$, $U(\tilde{a}, d) \subset U(a, d)$ and that $Y_j(\tilde{a}, d) = Y_j(a, d)$ for all $j \in U(\tilde{a}, d)$. We leave the details of the proof to the reader.

The second property, i.e. that if $b R_a^* c$ then $\text{out}(R_a, c) \subseteq \text{out}(R_a, b)$, follows immediately from the definition of out . ■

The XMW model permits us to predict the outcome after any input change under very general assumptions about delays in a network. In fact each vertex may have an arbitrary finite inertial delay. The model, though conceptually simple and relatively natural, is computationally intractable; in the worst case, the graph of the relation R_a may have $O(3^m)$ vertices. Fortunately, there exists an efficient algorithm computing essentially the same information, as described in the next section.

3.2. Ternary Simulation

A ternary simulation of binary networks has been proposed by Eichelberger [26]. Algorithms A and B described below are an adaptation of his work. Let N be a network, $\hat{a} \in \mathcal{T}^n$ be an input vector, and $b \in \mathcal{T}^m$ be such that $\langle \hat{a}, b \rangle$ is a stable total state. Furthermore, let $a \in \mathcal{T}^n$ be a new input vector and $\mathbf{a} = l.u.b.\{\hat{a}, a\}$. Algorithm A is defined by:

Algorithm A

```

 $h := 0;$ 
 $\mathbf{y}^0 := b;$ 
repeat
   $h := h + 1;$ 
  for  $i = 1$  to  $m$ 
     $\mathbf{y}_i^h = Y_i(\mathbf{a}, \mathbf{y}^{h-1});$ 
until  $\mathbf{y}^h = \mathbf{y}^{h-1};$ 

```

In the following we will use A (B) to denote the name of the algorithm, and A (B) to denote the length of the sequence of states produced by Algorithm A (B).

Proposition 3.3 Algorithm A produces a finite sequence $\mathbf{y}^0, \mathbf{y}^1, \dots, \mathbf{y}^A$ of states, where $A \leq m$, and $\mathbf{y}^{h+1} \sqsubset \mathbf{y}^h$ for $0 \leq h < A$.

Proof: We first prove by induction on h that $\mathbf{y}^{h+1} \sqsupseteq \mathbf{y}^h$, for $h \geq 0$. The basis, $h = 0$, follows because (i) $b = Y(\hat{a}, b)$, by the stability requirement, (ii) $\langle \mathbf{a}, b \rangle \sqsupseteq \langle \hat{a}, b \rangle$, by the definition of *l.u.b.*, and (iii) $\mathbf{y}^1 = Y(\mathbf{a}, \mathbf{y}^0) = Y(\mathbf{a}, b) \sqsupseteq Y(\hat{a}, b) = b = \mathbf{y}^0$, by the monotonicity of Y . Assuming that $\mathbf{y}^{h+1} \sqsupseteq \mathbf{y}^h$, it follows by the monotonicity of Y that $\mathbf{y}^{h+2} = Y(\mathbf{a}, \mathbf{y}^{h+1}) \sqsupseteq Y(\mathbf{a}, \mathbf{y}^h) = \mathbf{y}^{h+1}$, and the induction goes through. In each step either $\mathbf{y}^{h+1} \sqsubset \mathbf{y}^h$ or the algorithm terminates. At least one new vertex becomes \times if $\mathbf{y}^{h+1} \sqsubset \mathbf{y}^h$; therefore $A \leq m$. ■

Algorithm B is defined next:

Algorithm B

```

 $h := 0;$ 
 $\mathbf{z}^0 := \mathbf{y}^A;$ 
repeat
   $h := h + 1;$ 
  for  $i = 1$  to  $m$ 
     $\mathbf{z}_i^h = Y_i(\mathbf{a}, \mathbf{z}^{h-1});$ 
until  $\mathbf{z}^h = \mathbf{z}^{h-1};$ 

```

Proposition 3.4 Algorithm B produces a finite sequence $\mathbf{z}^0, \mathbf{z}^1, \dots, \mathbf{z}^B$ of states, where $B \leq m$, and $\mathbf{z}^{h+1} \sqsubset \mathbf{z}^h$ for $0 \leq h < B$.

Proof: The proof is dual to the proof of Proposition 3.3, with \sqsubseteq replaced by \sqsupseteq , etc. ■

The following results are an adaptation of the work of Lengauer and Naher [35]. They used a somewhat different model, but the main ideas are the same. The notation R_a^h denotes the composition of h copies of R_a for $h \geq 1$, and R_a^0 is the identity relation.

Theorem 3.1 The result \mathbf{y}^A of Algorithm A is the least upper bound of all the states reachable from the initial state b in the XMW transition relation, i.e.

$$\mathbf{y}^A = l.u.b.\{c \in \mathcal{T}^m : b R_a^* c\}.$$

Proof: We claim that $\mathbf{y}^h R_a \mathbf{y}^{h+1}$ for $0 \leq h < A$. Note that there is a vertex variable y_i for each input variable x_i , $i = 1, \dots, n$. In Algorithm A, if $y_i^0 \neq y_i^1$, for some $1 \leq i \leq n$, then $\hat{a}_i \neq a_i$. Thus, in the XMW model, vertex i is unstable and can change to \times . It therefore follows that $\mathbf{y}^0 R_a \mathbf{y}^1$. For $h > 0$, the first n components of the state can be held fixed at the value \mathbf{a} . Then, for $n+1 \leq j \leq m$

$$\mathbf{y}_j^{h+1} = Y_j(\mathbf{a}, \mathbf{y}^h) = Y_j(a, \mathbf{y}^h).$$

Thus $\mathbf{y}^h R_a \mathbf{y}^{h+1}$ as claimed.

Since $\mathbf{y}^0 = b$, we have that $b R_a^* \mathbf{y}^A$ and so $\mathbf{y}^A \sqsubseteq l.u.b.\{c \in \mathcal{T}^m : b R_a^* c\}$. To prove the converse, i.e. that $\mathbf{y}^A \sqsupseteq l.u.b.\{c \in \mathcal{T}^m : b R_a^* c\}$, we show the following claim by induction on h : $\mathbf{y}^A \sqsupseteq l.u.b.\{c \in \mathcal{T}^m : b R_a^h c\}$ for all $h \geq 0$.

Basis:

$h = 0$. Trivially true since $\mathbf{y}^A \sqsupseteq \mathbf{y}^0 = b$.

Induction hypothesis:

Assume that $\mathbf{y}^A \sqsupseteq l.u.b.\{c \in \mathcal{T}^m : b R_a^h c\}$ for some $h \geq 0$.

Induction step:

Assume $c \in \mathcal{T}^m$, and $b R_a^{h+1} c$. There must exist a state $d \in \mathcal{T}^m$ such that $b R_a^h d$ and $d R_a c$. By the induction hypothesis $\mathbf{y}^A \sqsupseteq d$. Furthermore, $d R_a c$ implies $c \sqsubseteq l.u.b.\{d, Y(a, d)\}$, by Proposition 3.1. Since Y is monotone and $\mathbf{y}^A = Y(\mathbf{a}, \mathbf{y}^A)$, it follows that

$$c \sqsubseteq l.u.b. \{d, Y(a, d)\} \sqsubseteq l.u.b. \{d, Y(\mathbf{a}, d)\} \sqsubseteq l.u.b. \{\mathbf{y}^A, Y(\mathbf{a}, \mathbf{y}^A)\} = \mathbf{y}^A.$$

Thus the induction goes through and the claim follows. \blacksquare

As above, let $\mathbf{z}^0, \mathbf{z}^1, \dots, \mathbf{z}^B$ denote the intermediate values produced by Algorithm B.

Theorem 3.2 The result \mathbf{z}^B of Algorithm B is the least upper bound of all the non-transient cyclic states reachable from b in the XMW transition relation, i.e.

$$\mathbf{z}^B = l.u.b. \text{ out}(R_a, b).$$

Proof: Since $\mathbf{y}^0 = b$, $\mathbf{y}^0 R_a^* \mathbf{y}^A$, $\mathbf{z}^0 = \mathbf{y}^A$, and $\mathbf{z}^0 R_a^* \mathbf{z}^B$, we have that $b R_a^* \mathbf{z}^B$. Also $\langle a, \mathbf{z}^B \rangle$ is a stable total state; hence $\mathbf{z}^B \in \text{out}(R_a, b)$ and $\mathbf{z}^B \sqsubseteq l.u.b. \text{ out}(R_a, b)$. To prove that $\mathbf{z}^B \sqsupseteq l.u.b. \text{ out}(R_a, b)$ we show that $\mathbf{z}^h \sqsupseteq l.u.b. \text{ out}(R_a, b)$ for $h \geq 0$, by induction on h .

Basis:

$h = 0$. Since $\mathbf{z}^0 = \mathbf{y}^A$, $\mathbf{y}^A = l.u.b. \{c \in \mathcal{T}^m : b R_a^* c\}$ (by Theorem 3.1), and $\text{out}(R_a, b) \subseteq \{c \in \mathcal{T}^m : b R_a^* c\}$, the result follows immediately.

Induction hypothesis:

Assume $\mathbf{z}^h \sqsupseteq l.u.b. \text{ out}(R_a, b)$ for some $h \geq 0$.

Induction step:

Let c be an arbitrary state in $\text{out}(R_a, b)$. By the induction hypothesis $\mathbf{z}^h \sqsupseteq c$ and, by the monotonicity of Y , $Y(a, \mathbf{z}^h) \sqsupseteq Y(a, c)$. Furthermore, since $\mathbf{z}^{h+1} = Y(a, \mathbf{z}^h)$, we have

$$\mathbf{z}^{h+1} \sqsupseteq Y(a, c). \tag{i}$$

Now consider any binary value in \mathbf{z}^{h+1} , say $\mathbf{z}_j^{h+1} = \alpha \in \mathcal{B}$. By (i) it follows that $Y_j(a, c) = \alpha$. Since c is arbitrary, it follows that

$$Y_j(a, c) = \alpha \text{ for all } c \in \text{out}(R_a, b),$$

i.e. the excitation of vertex j is α in all the states in $\text{out}(R_a, b)$. We claim that this implies that $c_j = \alpha$ for all $c \in \text{out}(R_a, b)$. To show this, suppose there exists a $\hat{c} \in \text{out}(R_a, b)$, such that $\hat{c}_j = \beta \neq \alpha$. Consider any nontransient cycle in $\text{out}(R_a, b)$

containing \hat{c} . Since the excitation is the same in all states in $out(R_a, b)$, it follows that vertex j cannot have the value β in all the states in the cycle (otherwise the cycle would be transient). Suppose it changes to γ in the state \tilde{c} in the cycle. If $\gamma = \alpha$, vertex j would be stable from here on, and could never change back to β . Hence, $\gamma \neq \alpha$. Altogether, both β and γ can only have values in $\{\alpha', \times\}$. Since the excitation is α in all states in $out(R_a, b)$, the only possible transition is from $\beta = \alpha'$ to $\gamma = \times$. However, there cannot be any transition between a state in which vertex j is \times and a state in which it is α' , since such a transition would violate the definition of R_a . Hence such a cycle in $out(R_a, b)$ containing \hat{c} does not exist, and we can conclude that $c_j = \alpha$ for all $c \in out(R_a, b)$. In summary, we have shown that for any binary value α in \mathbf{z}^{h+1} , the corresponding vertex will have the same value α in all the states in $out(R_a, b)$. Therefore the induction step goes through and the claim follows. ■

Note that the graph of R_a may contain cycles with the following property: The value of vertex j is $\alpha \in \mathcal{B}$ in all states of the cycle, but the excitation has the value α' in some states of the cycle and \times in the remaining states. Such cycles are transient according to the definition in Section 3.1. However, it is easy to verify that Theorem 3.2 still holds if the definition of transient is changed in such a way that these cycles belong to $out(R_a, b)$.

The results of this section will now be illustrated by examples. The ternary simulations corresponding to the XMW analyses of Figs. 3.2 and 3.3 are shown in Figs. 3.4 and 3.5 respectively. Note that, in both cases, \mathbf{y}^A is equal to the *l.u.b.* of all reachable states, and \mathbf{z}^B is equal to the *l.u.b.* of the outcome of the XMW analysis.

$y^0 = 0010$	$z^0 = \times 0 \times \times$
$y^1 = \times 010$	$z^1 = 10 \times \times$
$y^2 = \times 0 \times 0$	$z^2 = 100 \times$
$y^3 = \times 0 \times \times = y^A$	$z^3 = 1001 = z^B$
(a)	(b)

Figure 3.4. Ternary simulation of N_1 : (a) Algorithm A; (b) Algorithm B.

$y^0 = 100$	$z^0 = \times \times \times$
$y^1 = \times 00$	$z^1 = 0 \times \times = z^B$
$y^2 = \times \times \times = y^A$	
(a)	(b)

Figure 3.5. Ternary simulation of N_2 : (a) Algorithm A; (b) Algorithm B.

3.3. Reduced Networks

It is proved in this section that the XMW and ternary analyses can be applied to a much smaller network and still give the same amount of race information. It is first shown how to transform a network so that the dependence of a vertex on another vertex can be removed in certain cases.

Let $N = \langle V, E, x, y, Y \rangle$ be any network. Assume that $(p, q) \in E$, $p > n$, $p \neq q$, and that $(p, p) \notin E$, i.e. that q is a function vertex whose excitation function depends on the value of another vertex p , where p is a function vertex and does not have a self-loop. Now, let \bar{N} be the network: $\bar{N} = (V, \bar{E}, x, y, \bar{Y})$, where $\bar{E} = E \cup \{(i, q) : (i, p) \in E\} - \{(p, q)\}$. Also, $\bar{Y}_j(x, y) = Y_j(x, y)$ for all $j \neq q$ and $\bar{Y}_q(x, y) = Y_q(x, (y_1, \dots, y_{p-1}, Y_p(x, y), y_{p+1}, \dots, y_m))$. The transformation is performed to remove the dependence of vertex q on the value of vertex p . Note that only edges from function vertices can be removed.

A typical transformation to remove vertex q 's dependence on the value of vertex p is shown in Fig. 3.6, i.e. we want to remove the *-marked edge. Assume that in N we have $Y_p = (y_a y_b y_q)'$ and $Y_q = y_b y_c y_p$. Therefore vertex q depends on vertex p

(hence the edge from p to q). Moreover, vertex p does not have a self-loop and is also a function vertex. In this case we get $\bar{Y}_q = y_b y_c (y_a y_b y_q)'$, which can be simplified to $\bar{Y}_q = y_a' y_b y_c + y_b y_b' y_c + y_b y_c y_q'$. Note that, since the composition is performed for ternary functions, the term $y_b y_b' y_c$ in \bar{Y}_q cannot be removed. Note also that the vertex q gets a self-loop by this transformation.

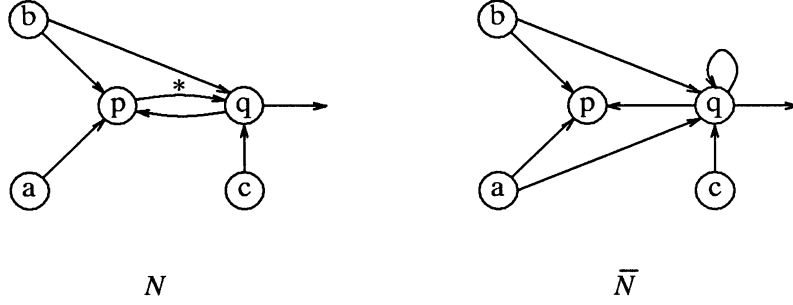


Figure 3.6. Removal of vertex q 's dependency on vertex p .

We now prove, by a series of lemmas, that the ternary simulation yields identical results for N and \bar{N} . Assume N and \bar{N} are started in the stable total state $\langle \hat{a}, b \rangle$ and that the input changes to a . Let $\mathbf{a} = l.u.b.\{\hat{a}, a\}$. Now, let $\bar{y}^0, \bar{y}^1, \dots, \bar{y}^{\bar{A}}$ and y^0, y^1, \dots, y^A be the results of Algorithm A for \bar{N} and N respectively.

Lemma 3.1 The result y^A of Algorithm A for network N is equal to the result $\bar{y}^{\bar{A}}$ of Algorithm A for network \bar{N} .

Proof:

First, if the excitation of vertex p never differs from the state of vertex p , the lemma holds trivially. Otherwise, assume that the excitation of vertex p differs from the state of vertex p for the first time at step r , $r \geq 0$. From the definition and the monotonicity of Algorithm A, we can conclude that

$$Y_p(\mathbf{a}, y^i) = \begin{cases} b_p & \text{if } i < r \\ \times & \text{if } i \geq r, \end{cases} \quad (i)$$

and that

$$\mathbf{y}_p^i = \begin{cases} b_p & \text{if } i < r+1 \\ \times & \text{if } i \geq r+1. \end{cases} \quad (\text{ii})$$

Clearly $\mathbf{y}^i = \bar{\mathbf{y}}^i$ for $0 \leq i \leq r$. This together with (i) and the monotonicity of Algorithm A implies that

$$\bar{Y}_p(\mathbf{a}, \bar{\mathbf{y}}^i) = Y_p(\mathbf{a}, \bar{\mathbf{y}}^i) = \begin{cases} b_p & \text{if } i < r \\ \times & \text{if } i \geq r. \end{cases} \quad (\text{iii})$$

From (ii), and (iii) it follows that

$$\mathbf{y}_p^i \sqsubseteq Y_p(\mathbf{a}, \bar{\mathbf{y}}^i) \sqsubseteq \mathbf{y}_p^{i+1} \quad \text{for } i \geq 0. \quad (\text{iv})$$

We now prove by induction on h that $\mathbf{y}^h \sqsubseteq \bar{\mathbf{y}}^h \sqsubseteq \mathbf{y}^{h+1}$, for all $h \geq 0$. By the monotonicity of Algorithm A, the claim in the lemma then follows immediately.

Basis:

$h = 0$. Since N and \bar{N} are started in the same state, $\mathbf{y}^0 = \bar{\mathbf{y}}^0$. By the monotonicity of Algorithm A, it follows that $\mathbf{y}^0 \sqsubseteq \mathbf{y}^1$, and the claim is true for the basis.

Induction hypothesis:

Assume that $\mathbf{y}^h \sqsubseteq \bar{\mathbf{y}}^h \sqsubseteq \mathbf{y}^{h+1}$, for some $h \geq 0$.

Induction step:

By the monotonicity of Y , the definition of Algorithm A, and the induction hypothesis we have for $i \neq q$:

$$\mathbf{y}_i^{h+1} = Y_i(\mathbf{a}, \mathbf{y}^h) \sqsubseteq Y_i(\mathbf{a}, \bar{\mathbf{y}}^h) = \bar{Y}_i(\mathbf{a}, \bar{\mathbf{y}}^h) = \bar{\mathbf{y}}_i^{h+1} \sqsubseteq Y_i(\mathbf{a}, \mathbf{y}^{h+1}) = \mathbf{y}_i^{h+2}.$$

Finally, since $\mathbf{y}_p^h \sqsubseteq Y_p(\mathbf{a}, \bar{\mathbf{y}}^h) \sqsubseteq \mathbf{y}_p^{h+1}$, by (iv), it follows that

$$\begin{aligned} \mathbf{y}_q^{h+1} &= Y_q(\mathbf{a}, (\mathbf{y}_1^h, \dots, \mathbf{y}_{p-1}^h, \mathbf{y}_p^h, \mathbf{y}_{p+1}^h, \dots, \mathbf{y}_m^h)) \\ &\sqsubseteq Y_q(\mathbf{a}, (\bar{\mathbf{y}}_1^h, \dots, \bar{\mathbf{y}}_{p-1}^h, Y_p(\mathbf{a}, \bar{\mathbf{y}}^h), \bar{\mathbf{y}}_{p+1}^h, \dots, \bar{\mathbf{y}}_m^h)) = \bar{\mathbf{y}}_q^{h+1} \\ &\sqsubseteq Y_q(\mathbf{a}, (\mathbf{y}_1^{h+1}, \dots, \mathbf{y}_{p-1}^{h+1}, \mathbf{y}_p^{h+1}, \mathbf{y}_{p+1}^{h+1}, \dots, \mathbf{y}_m^{h+1})) = \mathbf{y}_q^{h+2}. \end{aligned}$$

Hence, $\mathbf{y}^{h+1} \sqsubseteq \bar{\mathbf{y}}^{h+1} \sqsubseteq \mathbf{y}^{h+2}$ and the induction step goes through. ■

In a similar way one can verify the following “dual” version of Lemma 3.1.

Lemma 3.2 The result z^B of Algorithm B for network N is equal to the result \bar{z}^B of Algorithm B for network \bar{N} .

Note that the above procedure can be repeated. A network \dot{N} obtained by carrying out the reduction procedure some number of times and then removing all vertices with outdegree 0 will be called a *reduced* network. Furthermore, a network obtained by carrying out the reduction procedure as far as possible will be called a *completely reduced* network. Note that, in general, the completely reduced network is not unique. Note also that in a completely reduced network all function vertices have self-loops. For the remaining part of this section the following definition will be needed: A set of vertices F is called a *feedback vertex set* for a directed graph G iff removing the vertices in F and all their incident edges gives an acyclic graph. The following proposition is easily verified.

Proposition 3.5 If \dot{N} is a reduced version of N , then the function vertices of \dot{N} constitute a feedback vertex set of the original network N .

Proof: The reduction process can be viewed as a two step process. First, as many edges as possible are removed by using the above reduction procedure. Second, when no more edges can be removed, all vertices with outdegree 0 are removed, yielding the network \dot{N} . It is sufficient to show that, for any cycle in N , at least one function vertex of the cycle will remain in \dot{N} . This follows trivially from the fact that removing an edge in a cycle using the reduction procedure will only shorten the cycle. Eventually, one of the vertices of the cycle will have a self-loop and no further reduction can be made. Since a vertex with a self-loop has outdegree ≥ 1 it will not be removed. Hence, it follows that at least one function vertex of every cycle of N remains in \dot{N} , thus forming a feedback vertex set. ■

The converse, i.e. that, for any feedback vertex set F of N , one can find a reduced network \dot{N} such that the set of function vertices is identical to F is treated in the following proposition.

Proposition 3.6 Given any feedback vertex set F for a network N , there exists a reduced network \dot{N} such that the set of function vertices of \dot{N} is equal to F .

Proof: The proof constructs the reduced network “around” the given vertices in F . The idea is to remove edges leading *into* the vertices in F . More formally, we define a sequence of reduced networks \bar{N}^h recursively as follows:

Basis:

$$\bar{N}^0 = N.$$

Induction step:

Given \bar{N}^h , such that F is a proper subset of the function vertices of \bar{N}^h , find an edge satisfying the following conditions: i) $(i, j) \in \bar{E}^h$, ii) $i \notin F$, and iii) $j \in F$. Remove edge (i, j) using the procedure described above. If vertex i now has outdegree 0, remove it too. Let \bar{N}^{h+1} be the network so obtained.

The crucial observation for this algorithm is that the set of vertices in \bar{N}^h that have self-loops is a subset of F . This follows from the fact that all vertices of N that have self-loops must be in F (otherwise F would not be a feedback vertex set) and that the application of the reduction procedure can only cause a vertex in F to get a self-loop. From this it is easy to see that we will eventually get some \bar{N}^k such that the set of function vertices in \bar{N}^k is equal to F . ■

We are now ready to state and prove the main result of this section. For simplicity, a reduced version of N will be called a feedback vertex model of N . The theorem states essentially that an XMW analysis of a feedback vertex model of N is sufficient, i.e. that it is not necessary to include all the vertices of N in a race analysis. This is a result that contrasts radically with the “classical” binary race models, where a feedback variable analysis (even if augmented with a hazard analysis) is not always correct (for example see Langdon [34]).

Theorem 3.3 The l.u.b. of the outcome of an XMW analysis of a feedback vertex model \dot{N} of a network N is consistent with the l.u.b. of the outcome of an XMW analysis of N , in the sense that both analyses produce the same values for the feedback vertices.

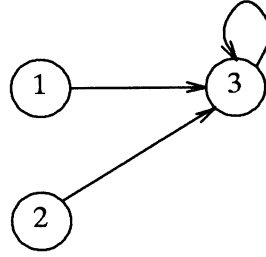
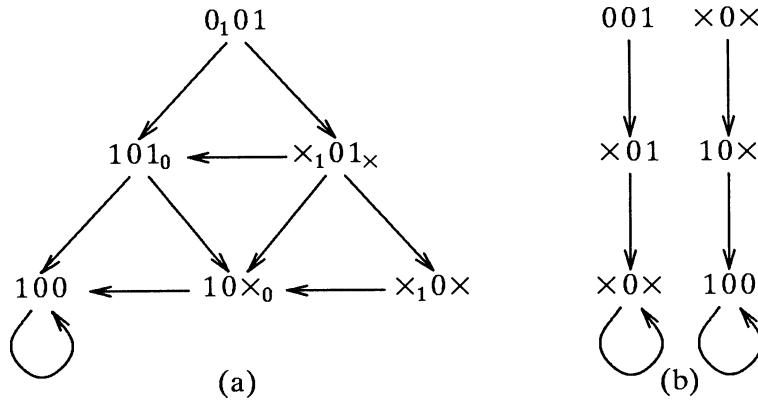
Proof: By Theorem 3.2 it follows that ternary simulation can be used instead of the XMW analysis, since the result of ternary simulation corresponds exactly to the l.u.b. of the outcome of the XMW analysis. Furthermore, Lemmas 3.1 and 3.2 together show that ternary simulation yields the same result for N and \bar{N} , when only one edge has been removed. It now follows, by induction on the number of times the reduction process is carried out, that ternary analysis yields identical results for N and \dot{N} , where \dot{N} is the reduced network. (Clearly, the removal of vertices with out-degree 0 does not change the result of ternary simulation.) Finally, by Proposition 3.6, we know that, given a feedback vertex set, we can construct a reduced network with exactly that set as the set of function vertices. ■

In order to analyze a network as fast as possible it would appear desirable to have the smallest number of vertices in the reduced network. It is well known [28] that the problem of finding a minimal feedback vertex set is NP-complete; hence the best we can hope for (assuming $P \neq NP$) is an approximation algorithm. However, it is not necessarily optimal to find a minimal feedback vertex set and reduce the network down to this set. It is not difficult to find examples in which the cost of ternary simulation of a reduced network is *higher* than the cost of ternary simulation of the original network. The opposite is also possible. For example, the new excitation functions can be sometimes drastically simplified, leading to a reduced network that is substantially more efficient to simulate than the original network. (Modeling a combinational static CMOS circuit in a switch-level model is a good example of the latter case.) Hence, the questions of whether to reduce a network or not, and how much reduction should be done, must unfortunately be decided according to some heuristic. (The answer will also depend heavily on implementation details.)

The ideas above will now be illustrated by examples. Consider network N_1 of Fig. 3.1. A possible feedback vertex set is $\{3\}$. Using the approach described above we get the reduced network \dot{N}_1 of Fig. 3.7 with excitation functions:

$$Y_1 = x_1 \quad Y_2 = x_2 \quad Y_3 = (y_1 + (y_2 + y_3)')'$$

Figure 3.8 shows both the XMW analysis and the ternary simulation for the same transition as was analyzed in Fig. 3.2, i.e. with $\langle \hat{a}, b \rangle = \langle 00, 001 \rangle$ and new input $a = 10$.

Figure 3.7. Network \dot{N}_1 .Figure 3.8. Analyses of \dot{N}_1 : (a) XMW; (b) ternary simulation.

Next consider network N_2 of Section 3.1. The set $\{2\}$ constitutes a feedback vertex set of the network N_2 . Reducing N_2 yields network \dot{N}_2 with excitation functions:

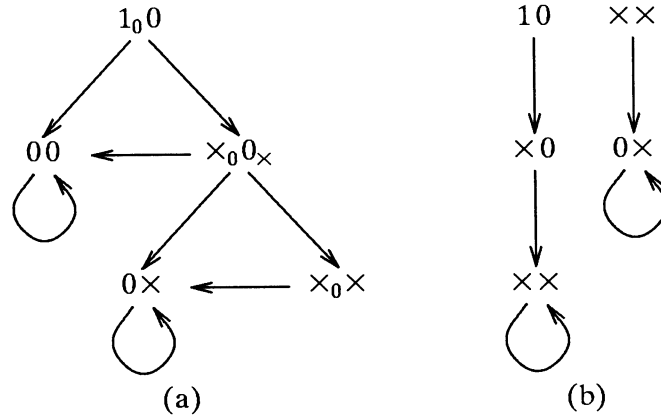
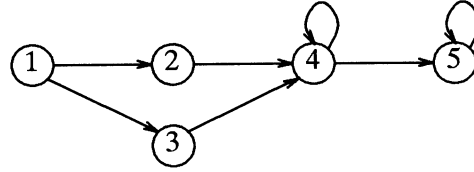
$$\dot{Y}_1 = x_1 \quad \dot{Y}_2 = (y_1 + (y_1 + y_2)')'.$$

Figure 3.9 shows the XMW analysis and the ternary simulation for the same transition as was analyzed in Fig. 3.3, i.e. with $\langle \hat{a}, b \rangle = \langle 1, 10 \rangle$ and new input $a = 0$.

The next example shows that a further reduction, below a minimal feedback vertex set, may be possible in some cases. Consider network N_3 of Fig. 3.10 with excitation functions:

$$Y_1 = x_1 \quad Y_2 = y_1 \quad Y_3 = y_1 \quad Y_4 = y_2 y_3 + y_2 y_4 + y_3 y_4 \quad Y_5 = y_4 + y_5$$

If the edge $(2, 4)$ is removed, we get \bar{N}_3 with excitation functions:

Figure 3.9. Analyses of \dot{N}_2 : (a) XMW; (b) ternary simulation.Figure 3.10. Network N_3 .

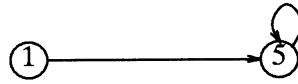
$$\bar{Y}_1 = x_1 \quad \bar{Y}_2 = y_1 \quad \bar{Y}_3 = y_1 \quad \bar{Y}_4 = y_1 y_3 + y_1 y_4 + y_3 y_4 \quad \bar{Y}_5 = y_4 + y_5$$

Furthermore, if the edge (3, 4) is also removed, we get \dot{N}_3 with excitation functions:

$$\dot{Y}_1 = x_1 \quad \dot{Y}_2 = y_1 \quad \dot{Y}_3 = y_1 \quad \dot{Y}_4 = y_1 y_1 + y_1 y_4 + y_1 y_4 \quad \dot{Y}_5 = y_4 + y_5$$

It is easy to see that we cannot remove any other edges, and that vertices 4 and 5 constitute a minimal feedback vertex set. However, \dot{Y}_4 can be simplified to $\dot{Y}_4 = y_1$. Here the edge (4, 4), indicating that the excitation of vertex 4 depends on the value of vertex 4, can simply be erased. Note that vertex 4 *did* depend on its own value in the original network. After the edge (4, 4) has been erased, the edge (4, 5) can also be removed. If we then remove the vertices with outdegree 0, we obtain network \ddot{N}_3 , shown in Fig. 3.11, with excitation functions:

$$\ddot{Y}_1 = x_1 \quad \ddot{Y}_5 = y_1 + y_5$$

Figure 3.11. Network \dot{N}_3 .

Reductions below a minimal feedback vertex set, such as the one described above, may be possible because some of the excitation functions may become degenerate. On the other hand, the feedback vertex set reflects only the structure of the graph. It can be easily verified that the reduction which uses functional degeneracies to remove self-loops is also valid, in the sense of Theorem 3.3. Unfortunately, the problem of determining whether a ternary function depends on a particular variable is NP-complete in general. (This can be verified by transformation from Boolean satisfiability).

3.4. Output Hazards

In the previous sections we were concerned only with the nontransient behavior of a given network. We concentrated on the detection of critical races and oscillations, but did not consider hazards explicitly. In fact, an XMW analysis does take into account implicitly all possible hazards associated with the vertex variables, when determining the nontransient outcome of a transition. Thus, if one is only interested in the nontransient behavior of a network, it is sufficient to find the outcome using the XMW analysis, or preferably the equivalent ternary simulation, and no further analysis is required. However, the network being analyzed is often a part of a larger system. In such cases, some subset of the vertices may be “visible” to the rest of the system. We will call such vertices *output vertices*. When output vertices are present, there is a new problem. Consider, for example, an output vertex that has the value 0 initially, and also in all the states of $out(R_a, b)$. It is quite possible that the vertex has the value 1 or \times in some of the transient states during the transition. Such short pulses must be detected, since they may cause unwanted state changes in the rest of the system controlled by this output vertex. In this section we formally define the notion of output hazards, and also give methods for detecting them. We also show that the same information may be obtained from a reduced network.

Assume that a network N is started in the stable total state $\langle \hat{a}, b \rangle$ and that the input changes to a . We say that there is a *static 1-hazard* on an output vertex i for the transition $\hat{a} \rightarrow a$ iff $b_i = 1$, $c_i = 1$ for every state $c \in \text{out}(R_a, b)$, and there exists a state d such that $b R_a^* d$ and $d_i \neq 1$. A *static 0-hazard* is defined similarly.

The following theorem shows how the results of ternary simulation can be used to detect static output hazards [26, 62].

Theorem 3.4 Assume network N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a . Let \mathbf{y}^A be the result of Algorithm A and \mathbf{z}^B be the result of Algorithm B. Then output vertex i has a static 1(0)-hazard iff $b_i = \mathbf{z}_i^B = 1(0)$ and $\mathbf{y}_i^A = \times$.

Proof: Suppose there is a static output hazard on output vertex i . Without loss of generality, assume it is a static 1-hazard. Then $b_i = 1$, and $c_i = 1$ for all states c in $\text{out}(R_a, b)$. Also, there must exist a state d , reachable from b , such that $d_i \neq 1$. Since $\mathbf{z}^B = l.u.b.(\text{out}(R_a, b))$, by Theorem 3.2, it follows that $\mathbf{z}_i^B = 1$. Furthermore, since $b R_a^* d$ and $d_i \neq 1$, we must have $l.u.b.\{e_i : b R_a^* e\} = \times$. However $\mathbf{y}^A = l.u.b.\{e : b R_a^* e\}$, by Theorem 3.1; hence $\mathbf{y}_i^A = \times$, and the claim follows.

Conversely, assume, without loss of generality, that $\mathbf{z}_i^B = b_i = 1$ and that $\mathbf{y}_i^A = \times$. By Theorem 3.2 it follows that $c_i = 1$ for any state c in $\text{out}(R_a, b)$. By Theorem 3.1, there must exist a state d reachable from b , such that $d_i \neq 1$; otherwise \mathbf{y}_i^A would be 1. Hence the claim goes through and the theorem holds. ■

In connection with static hazards, one may ask whether all timing problems can be detected by an XMW analysis of a network. In particular, can wire delays, i.e. delays associated with the edges, create new timing problems? One can account for wire delays in our model by simply inserting a “delay vertex” in each edge; the excitation function of such a vertex is the identity function. It is easily verified that the network with wire delays yields the same results. First, by Theorem 3.4, ternary simulation can be used instead of an XMW analysis for the detection of static hazards. Furthermore, by Lemmas 3.1 and 3.2 in Section 3.3, the dependence on a function vertex without a self-loop can be removed without affecting the ternary simulation. In particular, the dependence on a delay vertex can be so removed. By

induction on the number of wire delays in the network, it follows that a network without wire delays has the same static output hazards as a network with any number of wire delays added.

As shown in Section 3.3, it is sometimes advantageous to reduce the original network to a feedback vertex network. However, the reduction described in Section 3.3 does not take care of output vertices. There are two solutions to this problem. The first alternative is to change the reduction procedure in such a way that output vertices are never removed. Unfortunately, this may lead to a substantially larger network. The other alternative is to perform the analysis on a feedback vertex model, and then recreate the output values and transitions using the results obtained from the feedback vertex model. We are going to focus on this second alternative. Note however that the first alternative can be viewed as a special case of the second approach (the reduction is simply not carried out as far as possible and the output functions are trivial).

In the following discussion, we will assume that \dot{N} is a reduced version of some network N . Without loss of generality assume that vertices $n+1$ to $n+o$ are output vertices in N . The idea is to add a set of *output functions*, O_1, \dots, O_o , to the feedback vertex network. In general, function O_i is a function from $\tau^{\dot{m}}$ to τ , and is computed as follows:

```

 $O_i = y_i;$ 
while  $O_i$  depends on any  $y_k$  such that  $k \notin \dot{V}$  do
    replace every occurrence of  $y_k$  in  $O_i$  with  $Y_k$ 

```

Since \dot{V} is a feedback vertex set, the procedure is guaranteed to halt with an output function that depends only on input vertices and feedback vertices. (As usual, the final function O_i can be simplified using the laws of the ternary algebra.) The following result, stating that the values of the feedback variables uniquely determine the output values, is easily verified.

Proposition 3.7 Assume $\langle a, b \rangle$ is a stable total state in N . Let $\langle a, \dot{b} \rangle$ be the corresponding stable total state in \dot{N} . Then the value of output vertex i , i.e. b_{n+i} , is equal to $O_i(a, \dot{b})$.

Proof: This follows from the fact that $\langle a, b \rangle$ is a stable total state, i.e. $b_j = Y_j(a, b)$, and from the construction of O_i . ■

Proposition 3.7 together with Theorem 3.4 gives us the following method of analyzing a network N . First, reduce the network as much as desirable to \dot{N} . Also, compute the output functions. Use ternary simulation to get the two state vectors \dot{y}^A and \dot{z}^B . Evaluate the output functions for \dot{b} , \dot{y}^A and \dot{z}^B . The results are interpreted as in Theorems 3.2 and 3.4. Hence, both the outcomes of the transitions and potential static output hazards are correctly computed.

3.5. Summary

It is interesting to compare the results of the previous sections with earlier results concerning gate circuits. The classical transition model, used when no assumptions are made about the delays in the circuit, is the General Multiple Winner (GMW) model [17]. The GMW transition relation is very similar to the XMW relation, except that all the states are assumed to be binary. Otherwise, the two models are identical, i.e. any nonempty subset of the unstable gates can change to their excitation values.

Let \tilde{N} denote the network obtained from a network N by adding a delay element in every wire. We have the following result for gate circuits.

Theorem 3.5 The following analysis techniques are all equivalent for gate circuits from the point of view of nontransient state behavior and static output hazards:

1. GMW analysis of \tilde{N} .
2. XMW analysis of (a) \tilde{N} , (b) N , and (c) \dot{N} , any feedback vertex model of N .
3. Ternary simulation of (a) \tilde{N} , (b) N , and (c) \dot{N} .

Proof: The equivalence of 3(a), 3(b), and 1 for computing the nontransient behavior was shown in [12, Theorem 1]. Furthermore, by arguments similar to those in Theorem 3.4, it is easy to show that this equivalence also holds for detecting static hazards. (This was not shown in [12], but all the necessary results are there.) The

equivalence between 3b and 3c follows from Theorem 3.3. Also, by Theorems 3.1, 3.2 and 3.4, we immediately have the equivalence of 2a and 3a, 2b and 3b, and 2c and 3c. ■

For switch-level models we can also summarize the results of this chapter. Let \tilde{N} denote the network obtained when both the internal nodes as well as the transistors are assumed to have delays. Similarly, let \dot{N} denote a reduced version of N . The following results hold for switch-level circuits:

Theorem 3.6 The following analysis techniques are all equivalent for switch-level circuits, using any one of the node excitation models described in Chapter 2, Section 3, from the point of view of nontransient state behavior and static output hazards:

1. XMW analysis of: (a) \tilde{N} , (b) N , and (c) \dot{N} .
2. Ternary simulation of: (a) \tilde{N} , (b) N , and (c) \dot{N} .

Proof: The equivalence of XMW analysis and ternary simulation of the same network follows from Theorems 3.1, 3.2 and 3.4. Hence, we will only discuss the equivalence of the different networks for ternary simulation. Since no function vertex representing a transistor in \tilde{N} can have a self loop, Lemmas 3.1 and 3.2 apply. Hence, without affecting the ternary simulation, we can remove all the transistor function vertices, yielding network N . This, together with Theorem 3.4, demonstrates the equivalence of the ternary simulations of N and \tilde{N} . Using the same arguments, it follows that ternary simulations of N and \dot{N} are equivalent. ■

Chapter IV

Delay-Independent Networks

The XMW model allows us to study the behavior of a circuit when the delays are completely unknown. This leads naturally to the concept of delay-independence. The basic idea of delay-independence is to design circuits in such a way that they operate correctly, no matter what the internal delays are. In this chapter we formalize this intuitive idea and, using the theory developed in Chapter III, derive some very stringent conditions for a behavior to be realizable by a delay-independent design. In fact, we show that the class of delay-independent circuits is quite small.

4.1. Introduction

The ideas to be discussed will be introduced by means of some examples. To make the examples manageable, we will use gate circuits and will analyze the transitions according to the GMW race model. However, in the theory we develop later, we will be using the framework defined in Chapter II and the XMW model of Chapter III.

Consider the gate circuit G_1 of Fig. 4.1, started in the stable total state $x=0$, $y=011$. It is easy to verify that, if the input changes to $x=1$, the circuit will eventually end up in the new stable state $y=101$. Since this state is the only nontransient state reachable no matter what the gate delays are, we say that the transition is delay-independent. On the other hand, consider the gate circuit G_2 of Fig. 4.2. If this circuit is started in the stable state $x=0$, $y=100$ and the input changes to 1, the circuit can end up in either the state 000 or 001 depending on the relative sizes of

the delays in gates 1 and 2. In fact, gates 1 and 2 both become unstable after the input changes, i.e. these two gates are involved in a race. If gate 1 changes first, the outcome is 000; if gate 2 is faster, state 001 can also be reached. Since the non-transient outcome of this transition depends on the internal delays in the circuit, the transition is not delay-independent.

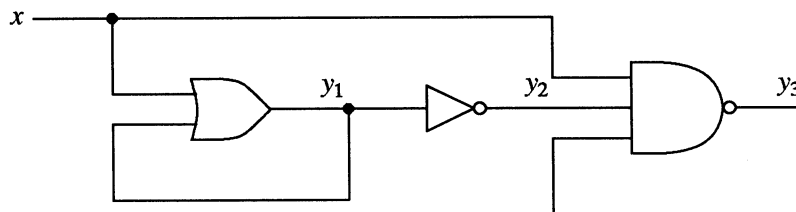


Figure 4.1. Gate circuit G_1 .

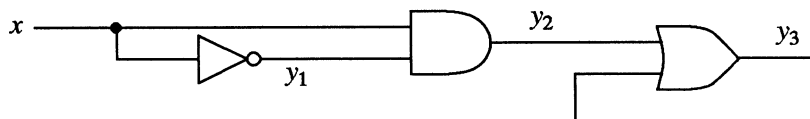
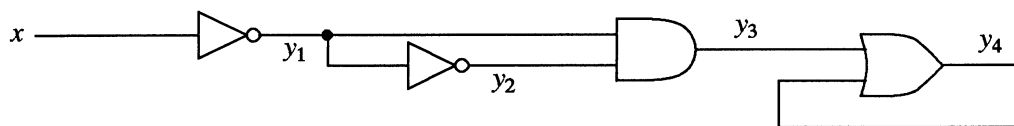


Figure 4.2. Gate circuit G_2 .

It is important to note that the classification of transitions according to delay-independence is very sensitive to the race model chosen and the delay assumptions made. Consider, for example, the gate circuit G_3 of Fig. 4.3. Assume the circuit is started in the stable state $x=0$, $y=1000$ and that the input changes to $x=1$. It is easy to verify that this transition is delay-independent if only the gates have delays associated with them. However, if the wire between the first inverter (gate 1) and the AND gate (gate 3) also has a delay, the transition is no longer delay-independent, since the circuit can then end up in either the state $y=0100$ or the state $y=0101$. The latter state can be reached when the wire delay between gates 1 and 3 exceeds the delay of gate 2.

The first study of delay-independent circuits was done by Muller and Bartky in the late 1950's [40, 45]. They assumed that only the gates have delays associated with them, that all transitions are instantaneous, and that the circuits do not have

Figure 4.3. Gate circuit G_3 .

any inputs. Using these assumptions, they developed a theory for such delay-independent circuits. For more details, the reader is referred to [40, Chapter 10] which contains a thorough treatment of this theory. One of the main differences between Muller and Bartky's work and ours is the fact that they developed a theory for circuits that do not have any input signals, whereas we are interested in the response of a circuit to a sequence of input changes.

One of the problems with delay-independent design is the difficulty of verifying that a given circuit really is delay-independent. For many delay models, the problem of determining the behavior of a circuit is NP-hard [35, 46] or even PSPACE-complete[†] and thus intractable for anything but trivial circuits. Fortunately, there do exist reasonably realistic race and delay models in which the outcome of a transition can be determined in polynomial time. The XMW model together with ternary simulation is such a model. In the next section we formally define the notion of delay-independence based on this race model.

4.2. Delay-Independence

The XMW model, as described in Chapter III, permits us to predict the outcome of any input change under very general assumptions about delays in a network. In fact, each vertex may have an arbitrary finite inertial delay. It is natural to define the concept of delay-independence in terms of a network's behavior according to an XMW analysis.

[†] For more details, see Chapter VII.

A network N , started in a binary stable total state $\langle \hat{a}, b \rangle$ and with a new binary input vector a , is said to be *delay-independent with respect to $\langle a, b \rangle$* if $out(R_a, b)$ contains a single binary state.

The reader should note that, by the definition of *out*, the single state in $out(R_a, b)$ must be a stable state. Note also that the term delay-independence is defined with respect to a certain starting configuration. We will later extend this idea to cover the general behavior of a network.

The above definition differs from Muller and Bartky's original definition of delay-independence [40] in three ways. First, the underlying race model is somewhat different, in that the XMW model includes an intermediate value \times , whereas their model is a classical binary race model. Second, our definition excludes transitions that can cause the network to enter a nontransient oscillation. We will return to these differences in Section 4.6. Finally, our network model is different in that we include inputs to the circuit, whereas Muller and Bartky's model does not. In fact, the characterization of the behavior of a delay-independent circuit in response to a sequence of input changes is the main problem studied in this chapter.

The concept of delay-independence, as defined above, is not very convenient, since it is defined in terms of an XMW analysis. However, by Theorem 3.2 it is easy to verify that the following definition of delay-independence is equivalent to that above. A network N started in the binary stable total state $\langle \hat{a}, b \rangle$ and with a new binary input vector a , is said to be delay-independent with respect to $\langle a, b \rangle$ if the result of Algorithm B for this change is binary.

4.3. Behavioral Model

In this section we combine the different concepts introduced in the earlier sections to define an abstract finite-state machine that describes the delay-independent behavior of a given network. Note that we assume that the network is operated according to the fundamental-mode assumption, i.e. the network is given sufficient time to “settle down” after every input change before the input changes again.

The *fundamental-mode, delay-independent, asynchronous machine* (FDAM), M , corresponding to a network N , consists of the following:

- 1) A finite set Q of internal states. Each state $q \in Q$ corresponds to a binary stable state of N .
- 2) A finite set Σ of input symbols. Each input symbol $\sigma \in \Sigma$ is a nonempty subset of $\{1, \dots, n\}$ representing the set of input variables being complemented.
- 3) A finite nonempty set $Z \subseteq \{1, \dots, m\}$ of output vertices.
- 4) A mapping τ (called the *transition map* of M) of a subset D of $Q \times \Sigma$ to Q .
- 5) An initial stable state $q^0 \in Q$.

Furthermore, the transition map must satisfy the following condition: The network N is delay-independent with respect to the state p for the input change σ and this transition takes N to the state q iff $\tau(p, \sigma) = q$. The reader can easily verify that this condition implies that M is deterministic.

We denote M by the ordered quintuple $M = (Q, \Sigma, Z, \tau, q^0)$. Note that M may be incompletely specified, i.e. $D \subset Q \times \Sigma$. This means that if, for example, $(p, \sigma) \notin D$, then N is not delay-independent with respect to the state p for the input change σ . Hence, a network may be delay-independent only with respect to certain transitions. The reader should also note that there cannot exist any state $p \in Q$ and input symbol $\sigma \in \Sigma$ such that $\tau(p, \sigma) = p$, i.e. there cannot be any self loops in a diagram of M . The reason for this is that each state $p \in Q$ represents a stable state of N , and the state of the network includes the values of the input vertices. Since an input symbol σ represents some input variables being complemented, it follows that if $\tau(p, \sigma) = q$, then $p \neq q$.

The definition of M is somewhat redundant, since the input symbol σ that takes the machine from state p to state q can be deduced from the values of the first n components of p and q . However, the redundancy simplifies the notation and will be retained.

In many cases it is appropriate to impose one further restriction on M . As described in Chapter III, Section 4, it is often important to ensure that the transitions of a network are free of output hazards. A FDAM M is said to be *hazard-free* if for all $p, q \in Q$ and $\sigma \in \Sigma$ such that $\tau(p, \sigma) = q$, there is no hazard on any output vertex during the transition from p to q .

To illustrate the above definitions, and also to give an example of a nontrivial delay-independent network, we present the gate circuit G_4 of Fig. 4.4. Following [14], and associating one vertex with each gate, we derive the abstract network N_4 of Fig. 4.5, with the excitation functions:

$$\begin{array}{lllll} Y_1 = x_1 & Y_3 = y_1 y_9 & Y_5 = (y_3 + y_6)' & Y_7 = y_2 y_6 & Y_9 = (y_7 + y_{10})' \\ Y_2 = x_2 & Y_4 = y_1 y_{10} & Y_6 = (y_4 + y_5)' & Y_8 = y_2 y_5 & Y_{10} = (y_8 + y_9)' \end{array}$$

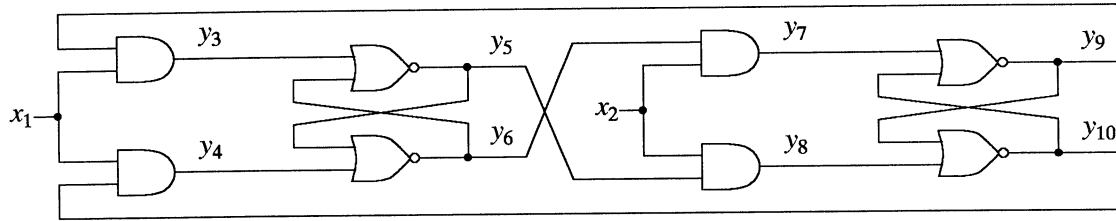


Figure 4.4. Gate circuit G_4 .

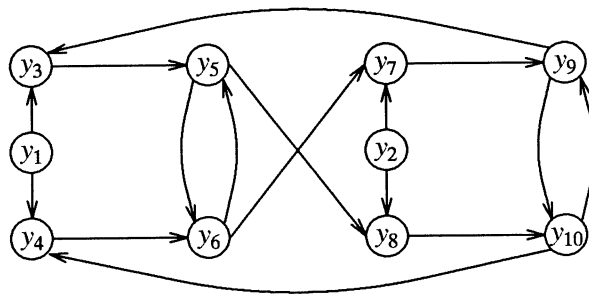


Figure 4.5. Network N_4 .

In order to simplify the analysis, the reduction procedure described in Chapter III is carried out using the set $\{5, 9\}$ as the feedback vertex set. This yields the reduced network N_4^r of Fig. 4.6 with excitation functions:

$$\begin{aligned} Y_1 &= x_1 & Y_5 &= (y_1 y_9 + (y_1 (y_2 y_5 + y_9)' + y_5)')' \\ Y_2 &= x_2 & Y_9 &= (y_2 (y_1 (y_2 y_5 + y_9)' + y_5)' + (y_2 y_5 + y_9)')' \end{aligned}$$

(Remember that it was shown in Chapter III that XMW analyses of N_4 and N_4^r are equivalent. We will return to this in Section 4.6.)

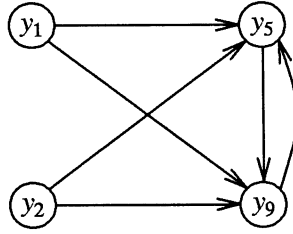
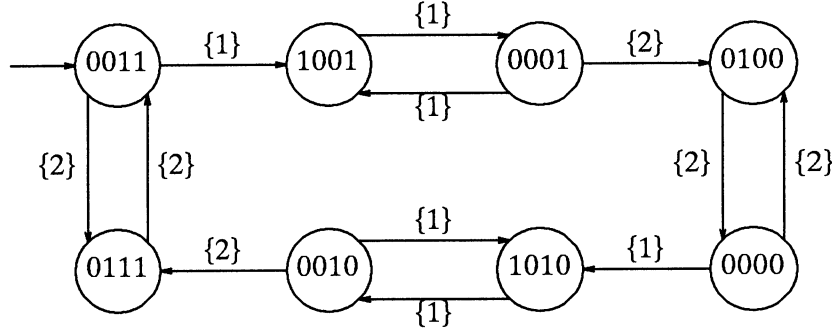


Figure 4.6. Network N_4^r .

Assume that N_4^r is started in the stable total state $\langle 00, 0011 \rangle$, i.e. assume that $q^0 = 0011$. The FDAM M_1 corresponding to the network N_4^r is shown in Fig. 4.7. (To simplify the picture, it is assumed that all vertices of N_4^r are output vertices.) Note that no transition caused by a multiple-input change is delay-independent for this network. It is interesting to note that for any state p in M for which an input symbol σ is allowed, any odd number of σ 's will take the machine to the same state. For example, $\tau(0011, \{1\}) = 1001$, $\tau(1001, \{1\}) = 0001$ and $\tau(0001, \{1\}) = 1001$. In Section 4.5 we will show that this is no coincidence but a fundamental property of delay-independent networks.

Figure 4.7. FDAM M_1 corresponding to network N_4^I .

4.4. Composite Network Function

In this section, we show how to derive a combinational network from a sequential network in such a way that the two networks yield identical results for ternary simulation. This transformation will serve as one of our basic tools to derive the behavior of an arbitrary delay-independent network in Section 4.5. The intuitive idea can be illustrated as follows. Suppose we are given a network N with m vertices as in Fig. 4.8(a). Assume the state $\langle \hat{a}, b \rangle$ is a stable total state of N and that the input changes to a . Normally, using the ternary simulation algorithm described in Chapter III, Section 2, we start with the state b , and then iteratively compute “next-states”, y^1, y^2, \dots, y^A . The next-states are computed as follows:

$$y^i = Y(\mathbf{a}, y^{i-1}),$$

where \mathbf{a} is the *l.u.b.* $\{\hat{a}, a\}$. This process is repeated until a stable state y^A is reached. We know that $A \leq m$ by the monotonicity of Algorithm A. Consider now the following alternative approach. Suppose that all edges leaving the m vertices are broken. We then get a combinational network of the form shown in Fig. 4.8(b). By construction, this combinational network is such that $y_j^{out} = Y_j(x, y^{in})$ for $1 \leq j \leq m$. Now, by connecting m copies of this combinational network in series we get a combinational network of the form shown in Fig. 4.8(c). It is easy to see that, if $y^{in} = b$ and $x = a$, we get $y^{out} = y^A$. The reader can easily verify that the same idea can be applied for Algorithm B as well. These concepts will be made more precise below.

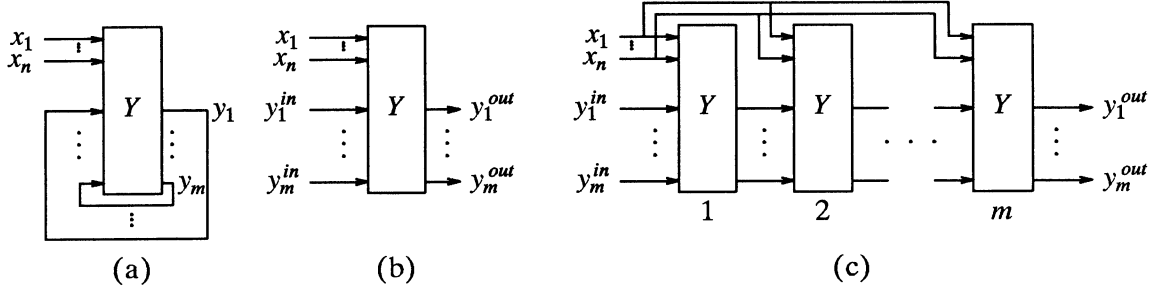


Figure 4.8. (a) Original network; (b) combinational network; (c) composite network.

Given a network N , define its *composition function* $F : \mathcal{T}^{n+m} \rightarrow \mathcal{T}^m$ as $F(x, y) = Y^{(m)}(x, y)$, where $Y^{(h)}$ is defined recursively as follows:

$$Y^{(h)}(x, y) = \begin{cases} Y(x, Y^{(h-1)}(x, y)) & \text{if } h \geq 1 \\ y & \text{if } h = 0 \end{cases}$$

The following properties of F will be used later.

Proposition 4.1 $F(x, y)$ is monotonic, i.e. $\langle \tilde{a}, \tilde{b} \rangle \sqsubseteq \langle a, b \rangle$ implies that $F(\tilde{a}, \tilde{b}) \sqsubseteq F(a, b)$.

Proof: Assume that $\langle \tilde{a}, \tilde{b} \rangle \sqsubseteq \langle a, b \rangle$. We prove by induction on h that $Y^{(h)}(\tilde{a}, \tilde{b}) \sqsubseteq Y^{(h)}(a, b)$ for $h \geq 0$. From this it follows immediately that $F(\tilde{a}, \tilde{b}) \sqsubseteq F(a, b)$, and hence that F is monotonic.

Basis:

$h = 0$. Trivially true.

Induction hypothesis:

Assume that $Y^{(h)}(\tilde{a}, \tilde{b}) \sqsubseteq Y^{(h)}(a, b)$, for some $h \geq 0$.

Induction step:

By the definition of $Y^{(h+1)}$ it follows that $Y^{(h+1)}(\tilde{a}, \tilde{b}) = Y(\tilde{a}, Y^{(h)}(\tilde{a}, \tilde{b}))$, and that $Y^{(h+1)}(a, b) = Y(a, Y^{(h)}(a, b))$. Since $Y^{(h)}(\tilde{a}, \tilde{b}) \sqsubseteq Y^{(h)}(a, b)$, by the induction hypothesis, and Y is assumed to be monotonic, we can conclude that $Y(\tilde{a}, Y^{(h)}(\tilde{a}, \tilde{b})) \sqsubseteq Y(a, Y^{(h)}(a, b))$. Hence the induction step goes through and the proposition follows. ■

Proposition 4.2 If $\langle a, b \rangle$ is a stable total state of N , then $F(a, b) = b$.

Proof: We prove by induction on h that $Y^{(h)}(a, b) = b$ for $h \geq 0$. It then follows immediately that $F(a, b) = b$.

Basis:

$h = 0$. Trivially true.

Induction hypothesis:

Assume that $Y^{(h)}(a, b) = b$, for some $h \geq 0$.

Induction step:

By the definition of $Y^{(h+1)}$ it follows that $Y^{(h+1)}(a, b) = Y(a, Y^{(h)}(a, b))$. Since $Y^{(h)}(a, b) = b$, by the induction hypothesis, and $\langle a, b \rangle$ is assumed to be a stable total state, i.e. $Y(a, b) = b$, we can conclude that $Y^{(h+1)}(a, b) = b$ and the induction step goes through. ■

Lemma 4.1 Assume a network N is started in the stable total state $\langle \hat{a}, b \rangle$ and that the input is changed to a . Let \mathbf{y}^A and \mathbf{z}^B be the results of Algorithms A and B for this input change respectively. Furthermore, let $\mathbf{a} = l.u.b.\{\hat{a}, a\}$ and let $F(x, y)$ be the composition function of N . We then have the following properties of F :

- (i) $F(\hat{a}, b) = b$ (stability)
- (ii) $F(\mathbf{a}, b) = \mathbf{y}^A$ (result of Alg. A)
- (iii) $F(\mathbf{a}, \mathbf{y}^A) = \mathbf{y}^A$ (stability)
- (iv) $F(a, \mathbf{y}^A) = \mathbf{z}^B$ (result of Alg. B)
- (v) $F(a, \mathbf{z}^B) = \mathbf{z}^B$ (stability)

Proof: Properties (i), (iii), and (v) follow immediately from Proposition 4.2 and the definition of Algorithms A and B. For property (ii), let $\mathbf{y}^0, \dots, \mathbf{y}^A$ be the sequence of states produced by Algorithm A. Note that $\mathbf{y}^0 = b$ and that $A \leq m$. We now show, by induction on h , that $Y^{(h)}(\mathbf{a}, b) = \mathbf{y}^h$ for $0 \leq h \leq A$.

Basis:

$h = 0$. Since $Y^{(0)}(x, b) = b$, and $\mathbf{y}^0 = b$, the result follows immediately.

Induction hypothesis:

Assume $Y^{(h)}(\mathbf{a}, b) = \mathbf{y}^h$ for some $h \geq 0$.

Induction step:

By the definition of $Y^{(h+1)}$ it follows that $Y^{(h+1)}(\mathbf{a}, b) = Y(\mathbf{a}, Y^{(h)}(\mathbf{a}, b))$. Similarly, by the definition of Algorithm A, it follows that $\mathbf{y}^{h+1} = Y(\mathbf{a}, \mathbf{y}^h)$. Since $Y^{(h)}(\mathbf{a}, b) = \mathbf{y}^h$, by the induction hypothesis, we can conclude that

$$Y^{(h+1)}(\mathbf{a}, b) = Y(\mathbf{a}, Y^{(h)}(\mathbf{a}, b)) = Y(\mathbf{a}, \mathbf{y}^h) = \mathbf{y}^{h+1}.$$

Hence the induction step goes through.

From the above we can conclude in particular that $Y^{(A)}(\mathbf{a}, b) = \mathbf{y}^A$. If $A = m$, then $F(\mathbf{a}, b) = Y^{(m)}(\mathbf{a}, b) = Y^{(A)}(\mathbf{a}, b) = \mathbf{y}^A$ and property (ii) is proved. Otherwise, i.e. if $A < m$, it is easy to verify that $Y^{(h+1)}(\mathbf{a}, b) = Y^{(h)}(\mathbf{a}, b) = Y^{(A)}(\mathbf{a}, b) = \mathbf{y}^A$ for $h \geq A$. This follows simply from the fact that $\langle \mathbf{a}, \mathbf{y}^A \rangle$ is a stable total state, i.e. that $\mathbf{y}^A = Y(\mathbf{a}, \mathbf{y}^A)$. Altogether this establishes property (ii). Using similar arguments, it is easy to verify property (iv). ■

The above lemma is crucial for the proofs in Section 4.5, since it allows us to use the composite network function F instead of the network N directly. Hence, we can establish the results using a combinational, rather than sequential, network — a substantial simplification.

4.5. Fundamental Properties of Delay-Independent Circuits

In this section we derive some general properties that are common to all delay-independent networks. The following three theorems summarize the main results of this chapter.

Theorem 4.1 Let N be any network. Let $M = (Q, \Sigma, Z, \tau, q^0)$ be the fundamental-mode, delay-independent, asynchronous machine corresponding to N . If there exist states p, q and $r \in Q$, and an input symbol $\sigma \in \Sigma$, such that $\tau(p, \sigma) = q$ and $\tau(q, \sigma) = r$, then $\tau(r, \sigma) = q$.

In other words, any odd number of changes of the same set of inputs must leave the network in the same state. An interesting special case occurs when the network has only one input. From the above theorem it follows that any FDAM, for a network with only one input, can have at most 3 states. Since the value of the input vertex is part of the state of the network, any such FDAM must have at least 2

states. Hence, a one-input network can only have an FDAM with 2 or 3 states. In fact, it is easy to see that the only possible machines are the ones shown in Fig. 4.9.



Figure 4.9. FDAM's for a network with only one input.

From this we can conclude, for example, that there does not exist a delay-independent mod-2 counter (a mod-2 counter is a circuit with one input, one output, and whose output changes with half the frequency of its input signal). In fact, there does not exist a delay-independent mod- k counter for any $k > 1$.

The next theorem deals with multiple-input changes, i.e. when more than one input vertex changes at the same time.

Theorem 4.2 Let N be any network. Let $M = (Q, \Sigma, Z, \tau, q^0)$ be the fundamental-mode, delay-independent, asynchronous machine corresponding to N . If there exist states p, q and $r \in Q$ and input symbols σ and σ_1 , such that σ_1 is a proper subset of σ , $\tau(p, \sigma) = r$, and $\tau(p, \sigma_1) = q$, then $\tau(q, \sigma - \sigma_1) = r$.

The theorem states, roughly, that if a multiple-input change is delay-independent, then the network must end up in the same state if this multiple-input change is made step by step.

Theorem 4.1 above dealt with the total state of a delay-independent network. Our last theorem of this section gives conditions on the values of specific output vertices. Since we deal here with outputs, we restrict our attention to hazard-free transitions.

Theorem 4.3 Let N be any network. Let $M = (Q, \Sigma, Z, \tau, q^0)$ be the hazard-free, fundamental-mode, delay-independent, asynchronous machine corresponding to N . Assume vertex j is an output vertex. If there exist states p, q and $r \in Q$, and an input symbol $\sigma \in \Sigma$, such that $\tau(p, \sigma) = q$, $\tau(q, \sigma) = r$ and $p_j = q_j = \alpha \in \mathcal{B}$, then $r_j = \alpha$.

From the above and from Theorem 4.1 we can draw the following conclusion. If an output does not change value for some input change σ , then it will not change for any sequence of σ 's.

We prove the above theorems with the aid of a series of lemmas. In fact, we prove a somewhat stronger result in that we do not restrict our attention to only binary stable states of N . The following assumptions will be used for Lemmas 4.2, 4.3 and 4.4. Let N be any network operated according to the fundamental-mode assumption, i.e. the network is given sufficient time to “settle down” after every input change before the input changes again. Let F denote the composite network function of N as defined in Section 4.4. Furthermore, assume that the input sequence is given by $a^0, a^1, a^2, a^3, \dots = \hat{a}, a, \hat{a}, a, \dots$, i.e. that the input is cycled between the binary input vectors \hat{a} and a . Assume that $\langle \hat{a}, b^0 \rangle$ is a stable total state of N . Let $b^{i,i+1}$ denote the result of Algorithm A for the transition from the stable total state $\langle a^i, b^i \rangle$ when the input changes to a^{i+1} . Similarly, let b^{i+1} denote the result of Algorithm B for the same transition. The following lemma is the key lemma to all subsequent results. It states that if, at some point, a vertex with a binary value does not react to an input change, it will never react thereafter.

Lemma 4.2 If there exists a $k \geq 1$ such that $b_j^{k-1} = b_j^{k-1,k} = b_j^k = \alpha \in B$, then $b_j^{i-1,i} = b_j^i = \alpha$ for all $i \geq k$.

Proof: We prove this by induction on i .

Basis:

$i = k$. Trivially true by the assumptions in the lemma.

Induction hypothesis:

Assume $b_j^{i-1,i} = b_j^i = \alpha$ for some $i \geq k$.

Induction step

First note that $\text{l.u.b.}\{a^{i-1}, a^i\} = \text{l.u.b.}\{a^i, a^{i+1}\} = \text{l.u.b.}\{\hat{a}, a\} = \mathbf{a}$. By the monotonicity of Algorithm B (Proposition 3.4), it follows that $b^{i-1,i} \supseteq b^i$ and hence, by the monotonicity of the composite network function (Proposition 4.1), that $F_j(\mathbf{a}, b^{i-1,i}) \supseteq F_j(\mathbf{a}, b^i)$. However, by Lemma 4.1 (iii), $F(\mathbf{a}, b^{i-1,i}) = b^{i-1,i}$ and, in particular, $F_j(\mathbf{a}, b^{i-1,i}) = b_j^{i-1,i}$ which is equal to α by the induction hypothesis.

Hence, $\alpha = F_j(\mathbf{a}, b^{i-1,i}) \sqsupseteq F_j(\mathbf{a}, b^i)$ and thus $F_j(\mathbf{a}, b^i) = \alpha$. Furthermore, by Lemma 4.1 property (ii), it follows that $b_j^{i,i+1} = F_j(\mathbf{a}, b^i)$ and hence $b_j^{i,i+1} = \alpha$. In other words, the value of vertex j after Algorithm A for the input change a^i to a^{i+1} will be α . Finally, by the monotonicity of Algorithm B (Proposition 3.4) it follows immediately that $b^{i,i+1} \sqsupseteq b^{i+1}$ and therefore that $b_j^{i+1} = \alpha$. Hence the induction step goes through and the lemma follows. ■

From Lemma 4.2 we get the following corollary.

Corollary 1 (Monotonicity for change sequences) For all $k \geq 1$, $b^{k-1,k} \sqsupseteq b^{k,k+1}$.

Proof: It suffices to show that whenever $b_j^{k-1,k}$ is binary, then $b_j^{k,k+1}$ has the same value. Suppose $b_j^{k-1,k} = \alpha \in B$. From the monotonicity of Algorithm A (Proposition 3.3) and the monotonicity of Algorithm B (Proposition 3.4), it follows that $b_j^{k-1} = b_j^k = \alpha$. Hence, $b_j^{k-1} = b_j^{k-1,k} = b_j^k = \alpha \in B$ and Lemma 4.2 applies. Thus $b_j^{i-1,i} = b_j^i = \alpha$ for all $i \geq k$ and, in particular, $b_j^{k,k+1} = \alpha$. ■

The following two lemmas give conditions on the values of a vertex after an odd and an even number of input changes respectively. The first lemma states that, if a vertex has a binary value after one input change, then it will have the same value after any odd number of input changes. The second lemma is similar, but for an even number of changes.

Lemma 4.3 If $b_j^1 = \alpha \in B$, then $b_j^{2i-1} = \alpha$ for all $i \geq 1$.

Proof: By induction on i .

Basis:

$i = 1$. Trivially true by the assumption in the lemma.

Induction Hypothesis:

Assume $b_j^{2i-1} = \alpha$ for some $i \geq 1$.

Induction Step:

Since $i \geq 1$ and thus $2i-2 \geq 0$, the state $b^{2i-2,2i-1}$ is well defined. By Lemma 4.2, property (iv), it follows that $b^{2i-1} = F(a^{2i-1}, b^{2i-2,2i-1})$ and, in particular, that $b_j^{2i-1} = F_j(a^{2i-1}, b^{2i-2,2i-1})$. By the same arguments, $b_j^{2i+1} = F_j(a^{2i+1}, b^{2i,2i+1})$.

However, by Corollary 1 it follows that $b^{2i-2,2i-1} \sqsubseteq b^{2i-1,2i} \sqsubseteq b^{2i,2i+1}$. Also, by assumption, $a^{2i-1} = a^{2i+1} = a$ and thus $\langle a^{2i-1}, b^{2i-2,2i-1} \rangle \sqsubseteq \langle a^{2i+1}, b^{2i,2i+1} \rangle$. This, together with the monotonicity of F (Proposition 4.1), shows that $F(a^{2i-1}, b^{2i-2,2i-1}) \sqsubseteq F(a^{2i+1}, b^{2i,2i+1})$. Thus, $b_j^{2i-1} = F_j(a^{2i-1}, b^{2i-2,2i-1}) \sqsubseteq F_j(a^{2i+1}, b^{2i,2i+1}) = b_j^{2i+1}$, and since $b_j^{2i-1} = \alpha$, by the induction hypothesis, it follows that $b_j^{2i+1} = \alpha$ and the induction step goes through. ■

Lemma 4.4 If $b_j^2 = \alpha \in B$, then $b_j^{2i} = \alpha$ for all $i \geq 1$.

Proof: By arguments similar to those in the proof of Lemma 4.3. ■

We are now in a position to prove Theorem 4.1.

Proof of Theorem 4.1: There are two cases to consider. First, if $r = p$, the result follows trivially from the fact that M is a deterministic machine. Otherwise, when $r \neq p$ we prove the result by contradiction. Assume $\tau(r, \sigma) = s$ for some $s \neq q$. Then the state s must differ from the state q in at least one component, say $q_j \neq s_j$. However, by the definition of M it follows that all the states p , q , r , and s are binary stable states of N . Since $\tau(p, \sigma) = q$ and $q_j = \alpha \in B$, Lemma 4.3 applies, showing that s_j must be equal to α , contradicting the assumption that $q_j \neq s_j$. Hence the result follows. ■

Theorem 4.2 follows from a more fundamental property of the XMW race model.

Proof of Theorem 4.2: Let $\hat{a} = (p_1, \dots, p_n)$, $\tilde{a} = (q_1, \dots, q_n)$, and $a = (r_1, \dots, r_n)$. It is easy to verify that $\tilde{a} \neq \hat{a}$, $\tilde{a} \neq a$, and that $\tilde{a} \sqsubseteq l.u.b.\{\hat{a}, a\}$. Since p is assumed to be a stable state of N , it follows that $\langle \hat{a}, p \rangle$ must be a stable total state of N , and hence, by Proposition 3.2, it follows that:

$$\{d : d \in out(R_a, c), \text{ and } c \in out(R_{\tilde{a}}, p)\} \subseteq out(R_a, p).$$

Since $\tau(p, \sigma) = r$ it follows that $out(R_a, p) = \{r\}$, and similarly, since $\tau(p, \sigma_1) = q$, that $out(R_{\tilde{a}}, p) = \{q\}$. Altogether this gives that $out(R_a, q) \subseteq out(R_a, p)$. However, since $out(R_a, q) \neq \emptyset$ and $out(R_a, p) = \{r\}$ it follows that $out(R_a, q) = \{r\}$ and thus that $\tau(q, \sigma - \sigma_1) = r$ establishing the theorem. ■

Finally we prove Theorem 4.3.

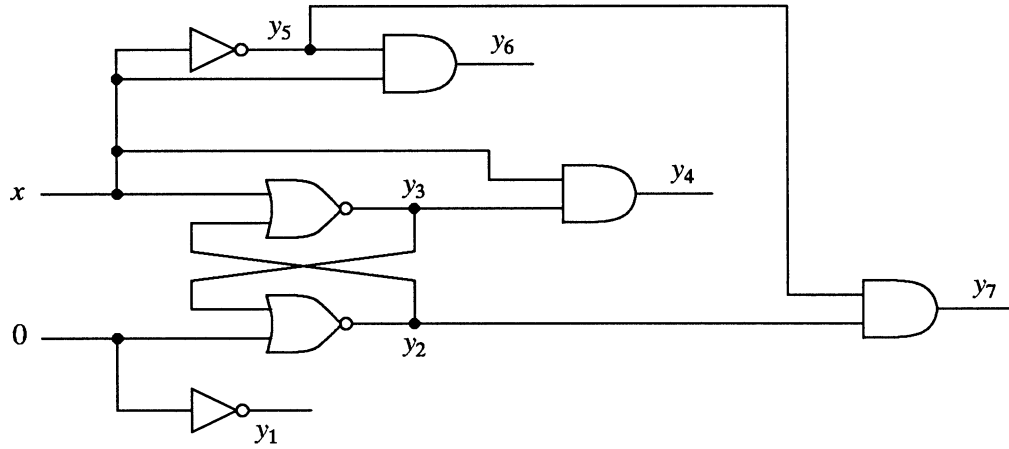
Proof of Theorem 4.3: There are two cases to consider. First, if $r = p$, the result follows trivially. Otherwise, if $p \neq r$, consider the output vertex j . Since all transitions are assumed to be hazard-free, the transition from p to q in particular must be hazard-free. However, since $p_j = q_j = \alpha \in \mathcal{B}$, Theorem 3.3 applies, showing that the result of Algorithm A for this transition must be equal to α . Using the same notation as in Lemmas 4.2, 4.3 and 4.4 above, we can conclude that $p_j = b_j^0 = b_j^{0,1} = b_j^1 = q_j = \alpha$. Hence Lemma 4.2 applies, establishing that $b_j^i = \alpha$ for all $i \geq 1$, and in particular that $r_j = \alpha$. ■

Using the above results, it is easy to verify that the following six types of vertex behavior are the only ones possible for a vertex in a delay-independent network when the input alternates between the two binary input vectors \hat{a} and a :

- 1) The vertex never reacts.
- 2) The vertex changes value on the first input change and keeps this value from then on.
- 3) The vertex changes value for every input change.
- 4) The vertex keeps the same value, although there may be a short pulse during every input change.
- 5) The vertex keeps the same value, although there may be a short pulse during the first input change.
- 6) The vertex keeps the same value for the first input change, except that there may be a short pulse during this transition. For the remaining changes, the vertex changes value for every input change.

Note that only behaviors 1-3 are normally acceptable for an output vertex.

The gate circuit G_5 of Fig. 4.10 contains gates of all the above types if it is started in the stable total state $x = 0$, $y = (y_1, \dots, y_7) = 1010100$, and the input oscillates between 1 and 0. In particular, gate 1 is of type 1, gates 2 and 3 are of type 2, gate 4 is of type 5, gate 5 is of type 3, gate 6 is of type 4, and finally gate 7 is of type 6.

Figure 4.10. Gate network G_5 .

4.6. Alternative Notions of Delay-Independence

In the previous section we derived some general properties of delay-independent networks. However, we used a rather restricted definition of delay-independence. In this section, we extend these results to different definitions of delay-independence.

Ternary simulation was the main tool used to prove the results in the previous section. It is easy to verify that all the results of Section 4.5 (except possibly Theorem 4.2) carry over to any race and network model that can be shown to correspond to ternary simulation. In Chapter III two summarizing theorems were proved for gate circuits and switch-level circuits respectively. These two theorems allow us to give a number of different definitions of delay-independence, that are all equivalent in the sense that the results of Section 4.5 hold for each of them. For example, if we assume that only the feedback lines of a gate circuit can have delays and that transitions can go through an intermediate value \times (i.e. using the XMW model) we know, by Theorem 3.5, that Theorems 4.1 and 4.3 are still valid. Hence, even in such a restricted model, where only the feedback lines have delays and the rest of the circuit is built of ideal (delay-free) components, one cannot construct a delay-independent mod-2 counter.

One of our basic assumptions in the definition of delay-independence was that the networks are operated in fundamental mode [39], i.e. input changes occur so seldom that the networks have time to settle down after each change before the input changes again. This assumption seems to contradict the basic idea of delay-independence, since here we implicitly introduce an assumption about the sizes of the delays in the network. However, since this assumption makes it easier to design delay-independent networks, and our main results are negative, it follows that if the fundamental-mode assumption is removed, the class of delay-independent networks can only become smaller. In fact, suppose that a network behaves correctly for a change σ_1 followed by σ_2 , where $\sigma_1 \cap \sigma_2 = \emptyset$ and σ_2 may occur before the network has reached a stable state. It is easy to verify that a necessary and sufficient condition for this is that the multiple-input change $\sigma_1 \cup \sigma_2$ must be delay-independent.

Another possible change in the definition of delay-independence is to relax the condition that a network must reach a unique stable state after each input change. This can be achieved by allowing the network to enter any nontransient oscillation as a result of an input change. In such a case, one can define a nondeterministic version of the sequential fundamental-mode machine of Section 4.3. However, it is not difficult to show that, for the XMW model, the results of Section 4.5 still hold for such a model. In particular, one can show that the *l.u.b.* of all the states reachable after some sequence of input changes is equal to the result obtained by using ternary simulation for every input change. Since Lemmas 4.2-4.4 did not require that the total states of N reached after each input change be binary, the result follows immediately. Whether this result also holds for the GMW model using gate and wire delays is still an open question.

Dennis and Patil [22] studied the realizability of delay-independent circuits in 1971, but used a different notion of delay-independence. They introduced special elements, called generators and absorbers, to model inputs and outputs of a circuit. The definition of these elements implies that all communication with the outside world is performed using a handshake protocol. For a circuit consisting of gates, wires, generators, and absorbers, they essentially performed a GMW analysis. However, in their model, one is not interested in reaching stable states (in fact such states are undesirable), but rather non-transient cycles. In the work of Dennis and

Patil a circuit is said to be *live* if in each nontransient cycle every gate changes at least once. Furthermore, a circuit is said to be *persistent* if in every nontransient cycle any gate that becomes unstable will have the same excitation until it changes. Dennis and Patil [22] showed that if both the gates and wires can have arbitrary but finite delays, then only some very trivial circuit behaviors can be designed with the live and persistent properties. Unfortunately, the condition that a circuit is live and persistent is only a sufficient condition for delay-independence [22], and thus the result is not as strong as one would like. Also, their definition of delay-independence is so different from ours that it is very difficult to compare their result with our results.

Finally, it is worth mentioning that the race model used by Muller and Bartky in their original work on delay-independence was the GMW model, but one in which only the gates were assumed to have delays. It is not known whether using such a definition of delay-independence would substantially increase the size of the class of delay-independent gate networks.

4.7. Conclusions

Recently, Ebergen [25] showed that a very large class of networks (including mod-2 counters) could be realized in a delay-independent design if a small set of basic “building blocks” could be designed delay-independently. Unfortunately, the results of previous sections show that such building blocks do not exist. In fact, we showed that the class of delay-independent networks is quite small, and that many useful functions cannot be realized with a completely delay-independent design. One might argue that this implies that the concept of delay-independence must be abandoned. However, one may also interpret the results as showing that the definitions of delay-independence used in this chapter are too pessimistic — in particular, that the race models chosen are too pessimistic. It is not very surprising that one cannot design a network that is guaranteed to work when the “devil’s advocate” is allowed to insert arbitrarily many, arbitrarily large delays anywhere in the network. This points to the need of more realistic race models — models in which delays are bounded by some lower and upper bounds. One could then define a network to be delay-insensitive if its behavior is independent of the size of the delays under the

assumption that the delays are within these bounds. In the next two chapters we develop two such more realistic race models.

Chapter V

The Almost Equal Delay Race Model

In this chapter we study a race model, called the “Almost-equal-delay” (AED) model[†]. For historical reasons, but also due to the basic assumptions in the model, the race model is binary. Hence, the model is primarily tailored towards gate circuits. The AED model was originally suggested by Brzozowski and Yoeli [16]. It represents an attempt to reduce the pessimism in the GMW model by assuming that (roughly speaking) no gate delay exceeds the sum of any two gate delays. In this chapter we define a stepwise AED model and compare it with the original AED model. We show that the two models are equivalent with respect to their capability of predicting the outcome of a transition. However, the new stepwise model has the attractive property of being closely related to a time scale; hence, it is possible to obtain some timing information from the analysis.

A major difficulty with the AED model is that the number of steps involved to compute the outcome of a single “race unit” can be exponential in the number of gates. To overcome this, we describe a new ternary algorithm, called the ternary almost-equal-delay (TAED) algorithm. This is a stepwise algorithm of the same complexity as the unit-delay method, but one which takes into account possible races. In fact, we show that the results of the TAED algorithm are closely related to the results predicted by the stepwise AED model.

[†] Part of this work was presented in [50]. A more complete version will appear in [51].

5.1. Binary Almost-Equal-Delay Model

The “almost-equal-delay” (AED) model was originally defined using the “single-winner” concept for simplicity [15, 16]. In this chapter we consider the more general “multiple-winner” version of the model which is described below. This generalization is quite straightforward.

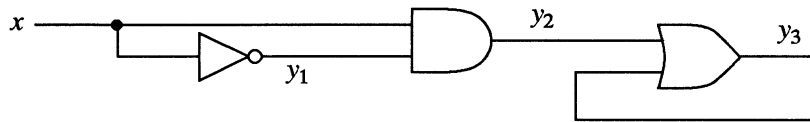


Figure 5.1. Circuit C_1 .

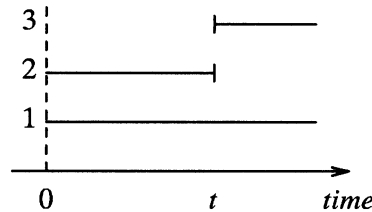
The basic idea is illustrated in Fig. 5.2. Suppose circuit C_1 of Fig. 5.1 starts in state $y = (y_1, y_2, y_3) = 100$ at time 0 with gates 1 and 2 unstable as shown in Fig. 5.2. Suppose now that gate 2 wins the race at time t and that, as a result of that change in gate 2, gate 3 becomes unstable. Under the almost-equal-delay assumption, it is unreasonable to let gate 3 win the new race between gates 1 and 3, since gate 1 has already been “waiting” for t units of time. The model will remember this fact and will only permit gate 1 to change in state 110, predicting the next state as 010. Informally, we can consider that at time 0 a “race unit” has started involving gates 1 and 2. No other gate can “enter” this race unit until all the gates in the original unit are somehow “satisfied”. A gate becomes satisfied if it either changes or becomes stable as a result of some other change. These ideas will be made precise below.

10010

↓

10101

(a)



(b)

Figure 5.2. Illustrating the “AED” idea: (a) possible transition; (b) timing diagram.

In the AED model, a certain amount of previous race history is necessary to determine the outcome of a race. We therefore define a *race state* to be an ordered pair $[y, S]$, where $y \in B^m$ denotes a state of the network, and S denotes the set of gates that are unstable in that state and are candidates to change according to the AED model, as explained below.

The AED model, as defined in this chapter, is a binary race model. Hence, we will assume that the excitation functions are the ternary extensions to some Boolean functions, i.e. we assume that $\langle a, b \rangle \in B^{n+m}$ implies that $Y(a, b) \in B^m$, and that for any $\langle a, b \rangle \in T^{n+m}$ we have $Y(a, b) = l.u.b. \{Y(\tilde{a}, \tilde{b}) : \langle \tilde{a}, \tilde{b} \rangle \in B^{n+m} \text{ and } \langle \tilde{a}, \tilde{b} \rangle \sqsubseteq \langle a, b \rangle\}$. We will return to this in Section 5.6.

Given any state $y = (y_1, \dots, y_m)$ of a network N , and any subset P of the set $\{1, \dots, m\}$, we define $y^{(P)}$ to be the vector obtained from y by complementing all the y_i such that i is in P . For example, if $y = (1, 0, 0)$ and $P = \{2, 3\}$ then $y^{(P)} = (1, 1, 1)$.

We now define a set K of race states and a binary relation ρ on K , for a network started in the stable total state $\langle \hat{a}, b \rangle$ and with the input changing to a . The set K and the relation ρ are defined inductively as follows:

Basis: $[b, U(a, b)] \in K$.

Induction Step: Given $[y, V] \in K$,

1. If $V = \emptyset$, then $[y, \emptyset] \rho [y, \emptyset]$.
2. If $V \neq \emptyset$, then for each nonempty subset P of V compute $W_P = (V - P) \cup U(a, y^{(P)})$.

(a) If $W_P = \emptyset$, then $[y^{(P)}, U(a, y^{(P)})] \in K$ and $[y, V] \rho [y^{(P)}, U(a, y^{(P)})]$.

(b) If $W_P \neq \emptyset$, then $[y^{(P)}, W_P] \in K$ and $[y, V] \rho [y^{(P)}, W_P]$.

Nothing else is in K or ρ .

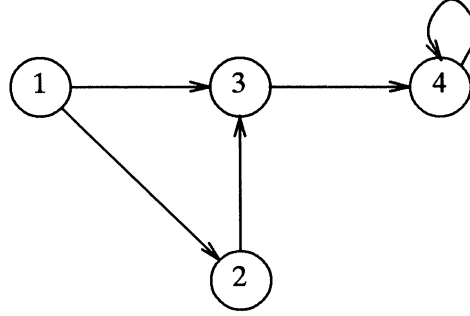
The set K defined above represents the set of all possible race states reachable from the initial race state $[b, U(a, b)]$, and the relation ρ describes the immediate successor race states. In particular, given $[y, V]$, if $V = \emptyset$ then y is a stable state, and this is indicated by stating that $[y, \emptyset]$ is the only successor of itself. Next, if $V \neq \emptyset$, it turns out by this definition that V will always represent a subset of the vertices which are unstable in y . The model now assumes, according to the multiple winner principle, that the vertices in any nonempty subset P of V may all change and so state $y^{(P)}$ may be reached. The vertices in P are considered to have completed the race they were involved in. As for the vertices in $V - P$, one of two things can happen: Either a vertex y_i remains unstable in $y^{(P)}$, i.e. $i \in W_P$, or the instability of y_i is removed when y changes to $y^{(P)}$. In the latter case, $i \notin W_P$. Now if $W_P = \emptyset$, all of the instabilities have been removed from V , one way or another. Thus we consider the previous race unit as being completed, and can now start a new one by entering the race state $[y^{(P)}, U(a, y^{(P)})]$ where each unstable vertex of $y^{(P)}$ has an equal chance of winning. If $W_P \neq \emptyset$, then some of the instabilities of V still remain unsatisfied; these are precisely all the vertices in W_P . These vertices are given preference over any new instabilities that may have been introduced by the change from y to $y^{(P)}$.

We illustrate this definition by the following example. Consider the network N_2 of Fig. 5.3, with excitation functions given by

$$Y_1 = x \quad Y_2 = y_1' \quad Y_3 = (y_1 y_2) \quad Y_4 = y_3 + y_4.$$

Assume N_2 is started in state $\hat{a} = 0$, $b = 0100$ which is stable. We now let $a = 1$ and note that $U(a, b) = \{1\}$. Thus the network starts in the race state $[0100, \{1\}]$. The graph of the relation ρ is shown in Fig. 5.4, where each edge has a label showing the vertices that change during the transition corresponding to the edge. (For now, ignore the * marking on some edges.)

The relation graph shown in Fig. 5.4 describes all the possible states that can be reached during the transition from the initial state $b = 0100$, when the input changes from 0 to 1. However, we are normally interested only in the “final” outcome of the transition. Thus we will consider only the set of cycles in the relation graph of ρ . Note that, by the definition of ρ , there cannot be any transient cycles in the graph.

Figure 5.3. Network N_2 .

(Recall that a cycle is said to be transient if there is a vertex that has the same value and is unstable in all of the states of the cycle [17].) In this example the network may end up in either one of two possible cycles, namely the stable states 1000 and 1001. Thus the initial race is critical because the final outcome depends on the relative delays in the network.

For technical reasons which will become clearer later, we mark certain edges by a * in the graph of ρ as follows. All edges of the type $([y, \emptyset], [y, \emptyset])$ are marked. Also, every edge $([y, V], [y^{(P)}, U(y^{(P)})])$ added to ρ by Rule 2(a), i.e. with $W_P = \emptyset$, is marked. No other edges are marked. (Note that if an edge is introduced because of Rule 2(a), that same edge cannot also be generated by Rule 2(b). This follows because y , V , and $y^{(P)}$ uniquely determine W_P .) For example, Fig. 5.4 shows all of the marked edges for the previous example.

5.2. Stepwise AED Model

In this section we introduce a new binary race model, very closely related to the AED model of Section 5.1. This model, which will be called the *stepwise AED* model, has the advantage of showing more clearly certain timing information. Also the stepwise model will be used later to establish a related ternary model.

To obtain more timing information from ρ we now define a new relation R derived from ρ as described below. Intuitively, one can interpret this relation R in the following way. Suppose that all the vertices have approximately the same delay,

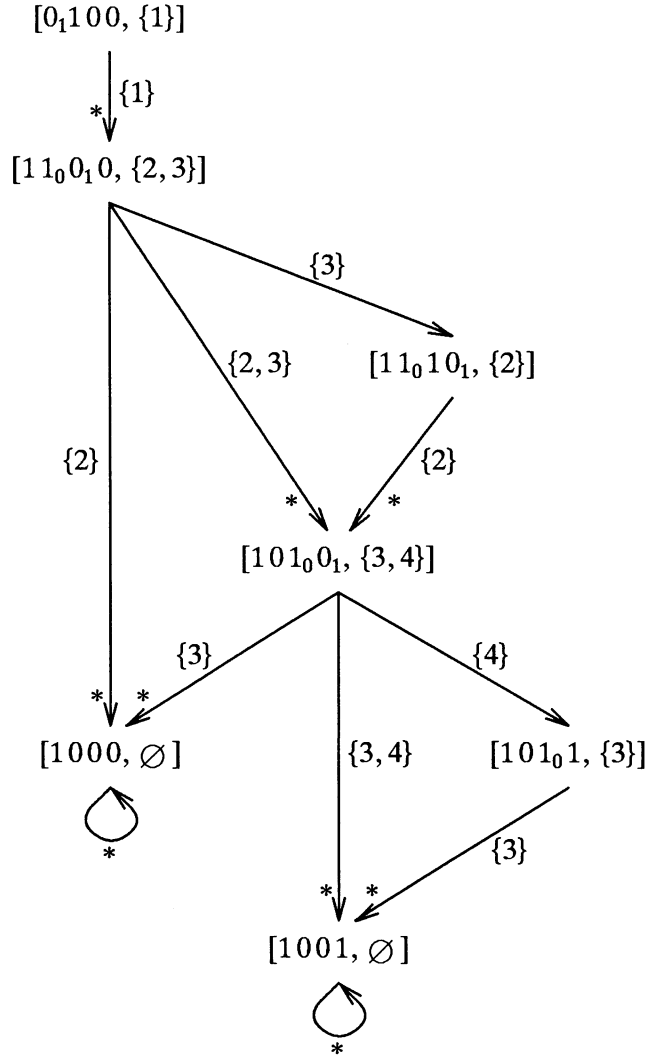


Figure 5.4. Race analysis of N_2 according to the AED method.

$\delta \pm \epsilon$. Then any race unit lasts for approximately δ units of time. Thus a transition between two distinct states related by R represents roughly δ units of time. In contrast to this, consider the relation ρ of Fig. 5.4 and the sequence $[1100, \{2,3\}]$, $[1110, \{2\}]$, $[1010, \{3,4\}]$ where consecutive race states are related by ρ . Here the first transition takes about δ units of time, whereas the second — only ϵ units of time, because vertex 2 became unstable in the first state and lagged behind vertex 3 by only a small amount of time.

Note that the above intuitive explanation is adequate as long as ϵ is much smaller than δ . Note also that the *-marking introduced in Section 5.1 has the following meaning: A transition is marked iff that transition completes a race unit, or it is a self-loop on a stable state.

Formally, the relation R is defined on a subset Q of the set K of race states as follows:

$$Q = \{[b, U(a, b)]\} \cup \{[y, V] : \text{there is a marked edge into } [y, V] \text{ in the graph of } \rho\}$$

For $[y, V], [\bar{y}, \bar{V}] \in Q$, we define $[y, V]R[\bar{y}, \bar{V}]$ iff there exists a path from $[y, V]$ to $[\bar{y}, \bar{V}]$ in the graph of ρ , such that only the last edge of the path is marked. Note, in particular, that $[y, \emptyset]R[y, \emptyset]$ for all the stable states $[y, \emptyset]$ in Q .

To illustrate this consider the example of Fig. 5.4. We find

$$Q = \{[0100, \{1\}], [1100, \{2, 3\}], [1010, \{3, 4\}], [1000, \emptyset], [1001, \emptyset]\},$$

and the reader can verify that R is as shown in Fig. 5.5.

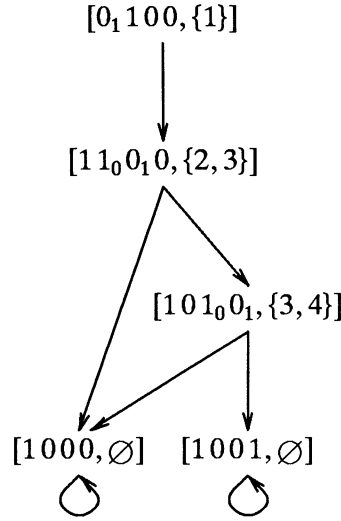
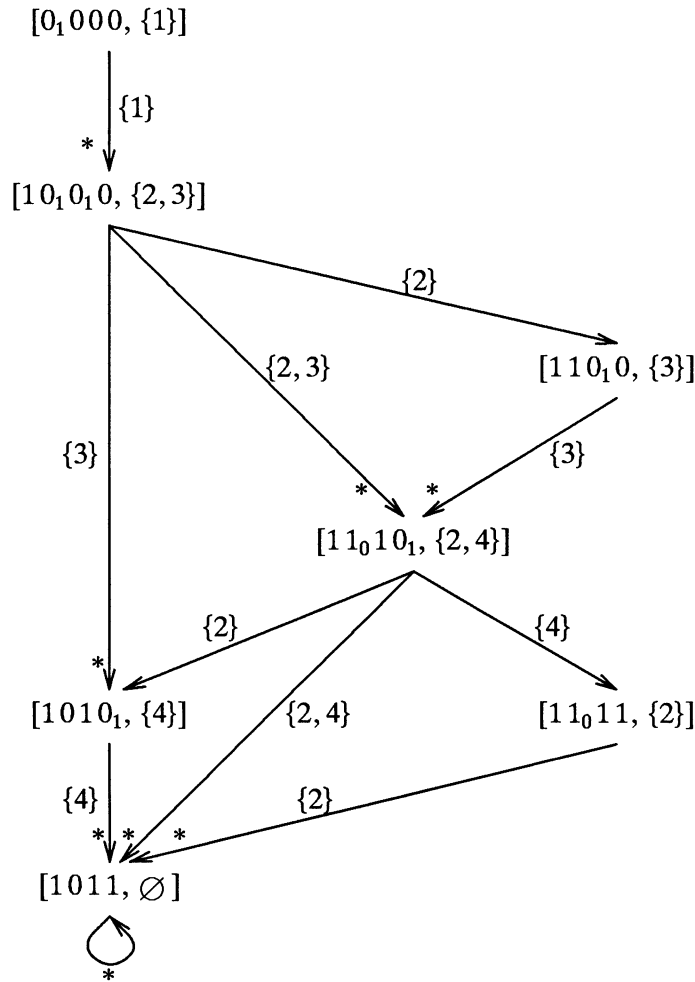


Figure 5.5. Relation R derived from ρ of Fig. 5.4.

Figure 5.6. AED analysis of network N_3 .

The reader should note that the knowledge of a race state $[y, V]$ alone does not provide sufficient timing information about race units. Consider the network N_3 described by the following equations:

$$Y_1 = x \quad Y_2 = y_1 y_3', \quad Y_3 = y_1, \quad Y_4 = y_3$$

started in the stable state $\hat{a} = 0$, $b = 0000$ and with the new input $a = 1$. (We use some strange excitation functions in this network in order to keep the example small; similar phenomena occur in more realistic larger networks.) In Fig. 5.6 we show the graph of the relation ρ for this transition. It is easy to verify that the states

$[0000, \{1\}]$, $[1000, \{2, 3\}]$, $[1010, \{4\}]$, and $[1110, \{2, 4\}]$ are all in Q , and that $[1000, \{2, 3\}]R[1010, \{4\}]$ because the change in vertex 3 completes the race unit started in $[1000, \{2, 3\}]$. When state $[1010, \{4\}]$ is reached in this way the instability of vertex 4 represents a new race unit. However, despite the fact that the states $[1110, \{2, 4\}]$ and $[1010, \{4\}]$ are both in Q and $[1110, \{2, 4\}]\rho[1010, \{4\}]$, they are not related by R . The reason for this is that the transition from $[1110, \{2, 4\}]$ to $[1010, \{4\}]$ does not represent the completion of a “race unit”, because vertex 4 is still unstable, and that condition started in state $[1110, \{2, 4\}]$. By requiring that the *last* edge in the ρ -path be marked, we make sure that the R relation holds only between states reachable by complete race units.

We will show that the R relation contains all the “useful” information of the old AED relation ρ . Assume that the network is started in the stable total state $\langle \hat{a}, b \rangle$ and that the new input is a . For convenience, let G_ρ be the directed graph of the relation ρ , and G_R be the directed graph of the relation R . Note that G_ρ and G_R depend not only on the network, but also on the initial state and the input change.

Proposition 5.1 In the graph G_R every race state $[y, V] \in Q$ is reachable from the initial race state $[b, U(a, b)]$.

Proof: First note that, by the definition of ρ , all race states in G_ρ are reachable from the initial race state $[b, U(a, b)]$. Consider $[y, V] \in Q$; either $[y, V]$ is the initial state or it has a marked edge $([\bar{y}, \bar{V}], [y, V])$ in G_ρ into it. Obviously, if it is the initial state there is nothing to prove. Otherwise, study the state $[\bar{y}, \bar{V}]$. Note that $[\bar{y}, \bar{V}]$ must be reachable from $[b, U(a, b)]$ in G_ρ , and hence there must exist a path in G_ρ from $[b, U(a, b)]$ to $[y, V]$ ending with a marked edge. In this path some other edges might also be marked, but that will only mean that we can reach $[y, V]$ in G_R by going through more than one race state. Hence the claim follows. ■

Proposition 5.2 The stable states of G_ρ are the same as the stable states of G_R .

Proof: All stable states of G_ρ are of the form $[y, \emptyset]$, with an edge $([y, \emptyset], [y, \emptyset])$. Note that this type of edge in G_ρ is always marked and hence will also exist in G_R . Conversely, every stable state of G_R is a stable state of G_ρ by construction of R . ■

For the next proposition we need a new concept. Given any cycle $[y^1, V^1], \dots, [y^r, V^r], [y^1, V^1]$ in the graph G_ρ , we call a race state $[y^i, V^i]$ *initiating in this cycle* iff the edge $([y^{i-1}, V^{i-1}], [y^i, V^i])$ is marked, where $[y^0, V^0]$ is interpreted as $[y^r, V^r]$.

Proposition 5.3 Every simple cycle C in G_ρ of length ≥ 2 must contain at least two distinct initiating states in C .

Proof: For any edge $([y^{i-1}, V^{i-1}], [y^i, V^i])$ in such a cycle, either the edge is marked (Case 2(a)), or V^i is a proper subset of V^{i-1} (Case 2(b)). Thus starting from any race state in C we must reach an initiating state in C in a finite number of steps. Starting from any initiating state $[y, V]$ in C we eventually must reach another initiating state $[\hat{y}, \hat{V}]$ in C . We argue that these two race states must be different. This is because the edge leaving $[y, V]$ involves changing at least one variable, say y_i , from V . This variable cannot change again until a new initiating state in C is reached. Thus \hat{y} and y differ at least in y_i . ■

We now wish to show that the cyclic behavior predicted by ρ is in some sense preserved in R . For any cycle C of race states in the graph G_ρ , let $\Pi(C)$ be the sequence of initiating states of C in the same order as in C . The AED model of Section 5.1 (relation ρ) and the stepwise model introduced in this section (relation R) are related by the following result. Assume that the network is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a . Compute ρ and R starting from this condition.

Theorem 5.1 For every cycle C in the graph G_ρ there exists a cycle $\Pi(C)$ in the graph G_R . Conversely, for every cycle D in G_R there exists a cycle C in G_ρ such that $D = \Pi(C)$. Furthermore, given two corresponding cycles C_ρ in G_ρ and C_R in G_R , and any variable y_i , either y_i has the same binary value in all the states of C_ρ and C_R or y_i oscillates (assumes the values 0 and 1) in both C_ρ and C_R .

Proof: Consider a cycle C_ρ in G_ρ . If the length of the cycle is 1, i.e. the state is a stable state, then the result follows immediately from Proposition 5.2. Hence assume the length of the cycle is ≥ 2 . By the definition of an initiating state in a

cycle and the definition of $\Pi(C_\rho)$, it follows that if $[y, V]$ and $[\bar{y}, \bar{V}]$ are any two consecutive race states in $\Pi(C_\rho)$, then the path in C_ρ from $[y, V]$ to $[\bar{y}, \bar{V}]$ will have only the last edge marked. Therefore we can conclude that $[y, V]R[\bar{y}, \bar{V}]$. This, together with Proposition 5.3 shows that $\Pi(C_\rho)$ is a cycle in G_R of length ≥ 2 .

Conversely, let D_R be a cycle of G_R . According to the definition of R , for any two consecutive race states $[y, V]$ and $[\bar{y}, \bar{V}]$ in D_R , there must exist a path in the graph G_ρ from $[y, V]$ to $[\bar{y}, \bar{V}]$ with only the last edge marked. Hence there must exist in G_ρ a cycle C_ρ corresponding to the cycle D_R in G_R , such that $\Pi(C_\rho) = D_R$.

The final part of Theorem 5.1 follows immediately from the observation that a vertex can change at most once between two initiating states. ■

5.3. Race Units

The definition of Q and R as given above can be simplified since it is possible to avoid using race states. This follows because, in any state $[y, V]$ in Q , the set V is uniquely determined by y ($V = U(a, y)$). Below we give a different algorithm for computing Q and R , where we do not explicitly form the graph of ρ . The reader can easily verify that the result is equivalent to the previous definition.

A sequence, $(z^0, S^0), (z^1, S^1), \dots, (z^k, S^k)$, $k \geq 0$, is called a *race unit* if

- i) $z^0 \in B^n$ is a state of N and $S^0 = U(a, z^0)$,
- ii) z^{i+1} is state z^i with all the vertices in the set Δ^i complemented, where Δ^i is any nonempty subset of S^i . Also $S^{i+1} = (S^i - \Delta^i) \cap U(a, z^{i+1})$, and
- iii) $S^k = \emptyset$.

Note that $S^0 \supset S^1 \supset \dots \supset S^k$, where \supset denotes proper containment.

We compute the set Z and also the relation σ on Z inductively as follows:

Algorithm 5.1: Let $\langle \hat{a}, b \rangle$ be a stable total state of N and let the input change to a .

Basis:

$b \in Z$.

Induction step:

For every $y \in Z$, if there exists a race unit beginning with $(y, U(a, y))$ and ending with (z, \emptyset) , then $z \in Z$ and $y\sigma z$.

Now to obtain Q and R , replace each state $y \in Z$ by $[y, U(a, y)]$ in Q , and let $[y, U(a, y)]R[z, U(a, z)]$ iff $y\sigma z$. From now on we will be working with Z and σ rather than with Q and R when referring to the stepwise AED model.

To illustrate these ideas, consider network N_2 of Fig. 5.3, as before started in the stable state $\hat{a} = 0$, $b = 0100$ and with the new input $a = 1$. Since $U(1, 0100) = \{1\}$, there is only one possible race unit starting with $(0100, \{1\})$: $(0100, \{1\}) (1100, \emptyset)$. Hence, we add the state 1100 to Z , and add the pair $(0100, 1100)$ to the relation σ . Since $U(1, 1100) = \{2, 3\}$, there are only three possible race units starting with $(1100, \{2, 3\})$:

- 1 $(1100, \{2, 3\}) (1000, \emptyset)$,
- 2 $(1100, \{2, 3\}) (1110, \{2\}) (1010, \emptyset)$, and
- 3 $(1100, \{2, 3\}) (1010, \emptyset)$.

Therefore we add the states 1000 and 1010 to Z , and add the pairs $(1100, 1000)$ and $(1100, 1010)$ to the relation σ . It is easy to show that from the state 1000 we can only go to 1000, and from state 1010 we can end up in 1000 or 1001. From 1001 we can only reach 1001. In summary, $Z = \{0100, 1100, 1000, 1010, 1001\}$, and the relation σ over Z is as illustrated by the graph of Fig. 5.7. Note the correspondence between Fig. 5.7 and Fig. 5.5.

It is now convenient to interpret Fig. 5.7 as a nondeterministic finite state machine with initial state 0100, and δ as its only input letter. After one race unit (i.e. after roughly δ units of time), we reach the state 1100 and after two race units we may nondeterministically reach the states 1000 or 1010, etc. Let $Z^0 = \{b\}$ and let Z^i be the set of states of Z reachable after i steps, i.e. $Z^i = \{z : b\sigma^i z\}$. Note that these sets are the same as the subsets constructed by the subset construction when converting the nondeterministic finite state machine to a deterministic finite state machine [31]. In Fig. 5.8 we show the deterministic finite state machine corresponding to the nondeterministic machine of Fig. 5.7.

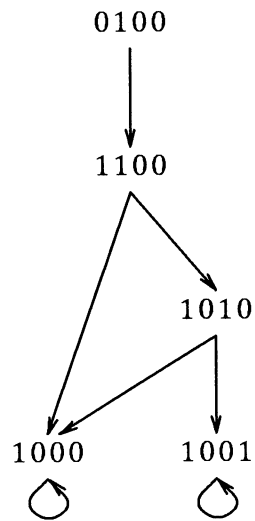


Figure 5.7. Stepwise AED analysis of N_2 .

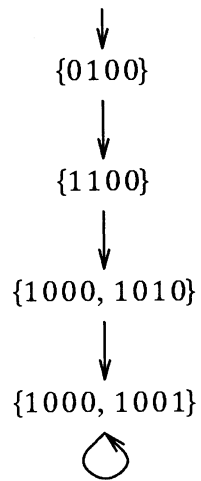


Figure 5.8. Deterministic finite state machine corresponding to Fig. 5.7.

We close this section with a discussion of some of the limitations of the stepwise AED model. One of the basic assumptions in this model is that delays are only associated with the vertices, and that delays are inertial in their nature. In many real circuits these assumptions are justified. However, there is also a danger that the model may become unrealistic under certain conditions. The model is only accurate as long as races from different race units do not get “mixed up”. In general, under the assumption that all vertices have delays Δ_i in the interval $[\delta - \epsilon, \delta + \epsilon]$, the following condition makes sure that the k^{th} race unit does not get mixed up with the $(k-1)^{\text{st}}$:

$$k(\delta - \epsilon) > (k-1)(\delta + \epsilon).$$

The condition states that the time to complete k changes in the fastest vertices (each with delay $\delta - \epsilon$) should be greater than the time required for $k-1$ slowest vertices (each with delay $\delta + \epsilon$). What can happen, if the above condition is not satisfied, is that the model may omit certain races that potentially could exist. In summary, a sufficient condition for the stepwise AED model to be accurate for at least k steps, is that

$$\frac{\epsilon}{\delta} < \frac{1}{2k-1}.$$

For example, if the uncertainty of the delays in the vertices is 10%, the stepwise AED model is accurate for at least 5 steps. However, the reader should note that the above condition is only a sufficient condition; in many cases the AED model is accurate for more than k steps.

5.4. TAED Algorithm

In this section we describe a ternary simulation method which is related to the stepwise AED model. The underlying idea behind the TAED method, formally defined below, is to find all of the unstable vertices in a state y , and then determine which of these unstable vertices *must* change in this race unit. Consider network N_4 corresponding to the circuit C_4 of Fig. 5.9(a) started in the stable state $\hat{a}=0$, $b=011$ and with the new input $a=1$. The first step of the TAED algorithm is to calculate

the *l.u.b.* of the present state, and the excitation of the network. In this intermediate state, called t in the algorithm, a vertex will have the value \times if it is unstable. In network N_4 , vertex 1 is unstable and hence $t = \times 11$. In order to determine which of the unstable vertices must change in the present race unit, the excitations of the unstable vertices (i.e. the vertices for which $t_j = \times$) are re-evaluated. However, this re-evaluation is done assuming t is the total state of the network. If the excitation of an unstable vertex j is still binary, then, independently of changes in the other unstable vertices, this vertex has to change before the present race unit can finish. On the other hand, if the “new” excitation of an unstable vertex j is \times , then that instability “depends” on some vertices that are also unstable. Hence it is possible to change these other unstable vertices first and thereby remove the instability of vertex j . In network N_4 we get $Y_1(1, \times 11) = 1$, so the new state we reach is 111. It is easy to see that vertex 1 must change in any race unit, since the excitation of the vertex depends only on the new input value. Hence in the stepwise AED model we can reach, in one race unit, the state 111. In this state both vertex 2 and vertex 3 are unstable and hence $t = 1 \times \times$. In network N_4 we get $Y_2(1, 1 \times \times) = 1' = 0$ and $Y_3(1, 1 \times \times) = (1 \times)' = \times$, so the new state we reach is $10 \times$. It is easy to see that the inverter must change in any race unit, since the excitation of the gate depends only on the new input value. On the other hand, the instability of the NAND gate depends on the inverter which is unstable; if the inverter changes first, the NAND gate becomes stable. Hence in the stepwise AED model in the second race unit we can reach the states 100 or 101. Note that in the next race unit, the NAND gate will again be “unstable” ($= \times$) but when re-evaluated, it will be “forced” to the binary value 0. This is also consistent with the stepwise AED model.

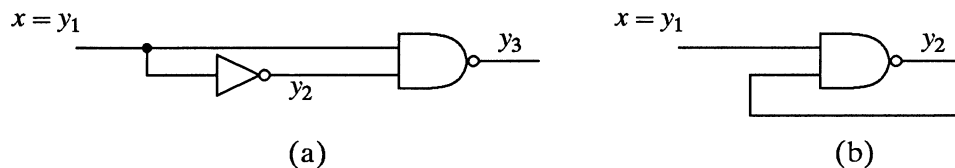


Figure 5.9. (a) Circuit C_4 ; (b) circuit C_5 .

In the description above, we simplified the re-evaluation of the excitation of the unstable vertices slightly. As the following example shows, it is not correct to use the values of t for all vertices. Consider network N_5 corresponding to circuit C_5 of Fig. 5.9(b) started in the stable state $\hat{a}=0$, $b=01$ and with the new input $a=1$. After one race unit we reach the state 11 . In the next race unit we get $t=1\times$, but if we use that value for the re-evaluation, the TAED model will predict that the state after one race unit will be $1\times$. However, the stepwise AED model predicts the state $y=10$. The reason for this discrepancy is that, when there is a self-loop, the \times that the vertex depends on, is coming from itself. Hence the only way to remove the instability of such a vertex is to change its state. The solution to this problem is simply to use the “old” value for vertex j when re-evaluating vertex j . We will use the notation $t^{(j)}$ to denote the vector obtained from t by replacing the j^{th} component by y_j . In network N_5 we get $Y(a, t^{(1)}) = Y(1, 11) = (11)' = 0$, and hence the TAED model will correspond to the stepwise AED model. We now formally define the TAED method.

We define the function *next* as follows. Let a be the new binary input, and let y be any ternary state of N . We calculate the successor state of y as shown below.

```

function next( $y \in \mathcal{T}^m$ )  $\in \mathcal{T}^m$ 
{
  for  $j = 1$  to  $m$  do           -- First calculate the intermediate state  $t$ 
     $t_j = l.u.b(y_j, Y_j(a, y))$ 
  for  $j = 1$  to  $m$  do           -- Re-evaluate the excitation
    if  $t_j = \times$  then
       $\tilde{y}_j = Y_j(a, t^{(j)})$       --  $t^{(j)} = t_1 \dots t_{j-1} y_j t_{j+1} \dots t_m$ 
    else
       $\tilde{y}_j = y_j$ 
  return( $\tilde{y}$ )
}

```

The TAED algorithm consists of repeatedly applying *next*. Note that this will either lead to a stable ternary state, or to an oscillation. In this respect, the TAED method is similar to the unit-delay method used by most commercial simulators.

To illustrate the algorithm, consider network N_2 of Fig. 5.3 started in the stable total state $\langle 0, 0100 \rangle$ and with the new input $a=1$. The first time we call *next* we get the intermediate state: $t = \times 100$, and thus the next state $y^1 = 1100$. The second time

we call *next* we get the following intermediate values:

$$\begin{aligned} t_1 &= l.u.b.\{y_1^1, Y_1(a, y^1)\} = l.u.b.\{1, a\} = l.u.b.\{1, 1\} = 1, \\ t_2 &= l.u.b.\{y_2^1, Y_2(a, y^1)\} = l.u.b.\{1, y_1'\} = l.u.b.\{1, 0\} = \times, \\ t_3 &= l.u.b.\{y_3^1, Y_3(a, y^1)\} = l.u.b.\{0, (y_1 y_2)\} = l.u.b.\{0, 1\} = \times, \text{ and} \\ t_4 &= l.u.b.\{y_4^1, Y_4(a, y^1)\} = l.u.b.\{0, (y_3 + y_4)\} = l.u.b.\{0, 0\} = 0. \end{aligned}$$

Hence, $t = 1 \times \times 0$. In the second step of the algorithm, the vertices with \times in the intermediate state are re-evaluated using the intermediate state, except for the value of the vertex itself for which we use the “old” value. For network N_2 we re-evaluate vertices 2 and 3, and we get the following values:

$$\begin{aligned} \tilde{y}_2 &= Y_2(a, t^{(1)}) = y_1' = 0, \text{ and} \\ \tilde{y}_3 &= Y_3(a, t^{(2)}) = (y_1 t_1) = (1 \times) = \times. \end{aligned}$$

Hence, we obtain the new state $\bar{y} = 10 \times 0$. In Fig. 5.10(a) we give the complete TAED analysis of network N_2 .

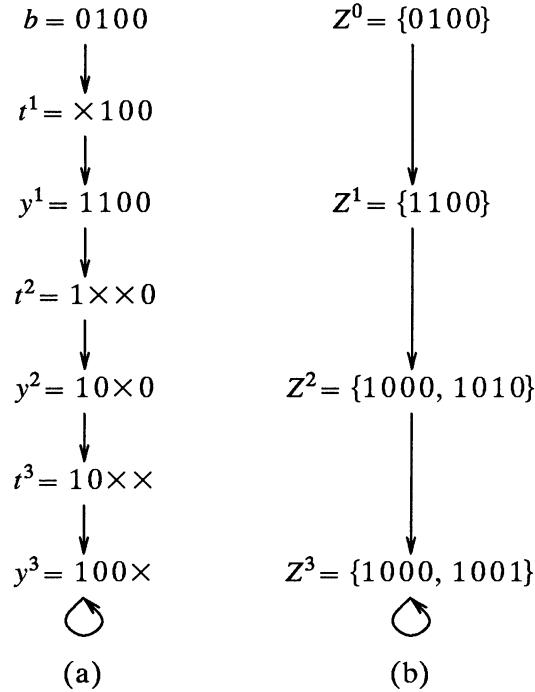


Figure 5.10. Analysis of N_2 : (a) TAED; (b) stepwise AED model.

It is interesting to note that the *l.u.b.* of the states reachable after i race units in the stepwise AED model, corresponds exactly to the outcome of the TAED method after i steps; see Figs. 5.10(a) and (b). In the next section we explore the relationship between the two models.

5.5. Correspondence between AED Model and TAED Algorithm

In this section we show a partial correspondence between the binary stepwise AED model and the results of the TAED algorithm described above. The main result is:

Theorem 5.2 Let $\langle \hat{a}, b \rangle$ be a stable total state of N , and let a denote a new input vector. Let y^i denote the results of the TAED method after $i \geq 0$ steps starting in $y^0 = b$. Then:

$$y^i \sqsupseteq l.u.b.(Z^i),$$

where Z^i is the set of states reachable in i steps in the stepwise AED model.

Before proving the theorem we will establish Lemma 5.1 below. The following observation is useful in proving the lemma. If y is the input to *next*, then $t^{(j)} \sqsupseteq y$. Hence, by the monotonicity property of the ternary extension,

$$Y_j(a, t^{(j)}) \sqsupseteq Y_j(a, y). \quad (1)$$

Lemma 5.1 Let $y \in T^m$ and let \tilde{y} be *next*(y). Then

$$w \sqsubseteq y \text{ and } w \sigma z \text{ implies } z \sqsubseteq \tilde{y}.$$

Proof: If $\tilde{y}_j = \times$, then for any such z we will trivially have $\tilde{y}_j \sqsupseteq z_j$. Hence, study the cases when $\tilde{y}_j \in \mathcal{B}$. Note that, by the definition of t , it follows that $y_j = \times$ implies $t_j = \times$. Therefore there are only four cases to consider.

	1	2	3	4
$y_j =$	α	α	α	\times
$t_j =$	α	\times	\times	\times
$\tilde{y}_j =$	α	α	α'	α

Here α stands for some binary value (0 or 1), and α' denotes the complement of α .

Case 1: $y_j = \alpha$, $t_j = \alpha$, and $\tilde{y}_j = \alpha$.

By the definition of t_j , $t_j = \alpha$ implies that $Y_j(a, y) = \alpha$. Furthermore, by the definition of ternary extension, we must also have $Y_j(a, w) = b$ for every $w \sqsubseteq y$. Since $w_j \sqsubseteq y_j = \alpha$, vertex j is stable in any such state w . Consequently, vertex j remains unchanged in any race sequence beginning in state w . Therefore, if $w \sigma z$, we must have $z_j = \alpha$. By assumption, $\tilde{y}_j = \alpha$ and $\tilde{y}_j \sqsupseteq z_j$ holds.

Case 2: $y_j = \alpha$, $t_j = \times$, and $\tilde{y}_j = \alpha$.

If $t_j = \times$ then $Y_j(a, y)$ is either \times or α' . By (1) $Y_j(a, t^{(j)}) \sqsupseteq Y_j(a, y)$. Thus \tilde{y}_j , as computed by *next*, must have the value \times or α' . But we have assumed $\tilde{y}_j = \alpha$. Hence this case is impossible.

Case 3: $y_j = \alpha$, $t_j = \times$, and $\tilde{y}_j = \alpha'$.

We first prove that in any state $w \sqsubseteq y$, vertex j is unstable. Since $t_j = \times$ we know that $\tilde{y}_j = Y_j(a, t^{(j)})$. Also, by assumption, $\tilde{y}_j = \alpha'$. Altogether we have $Y_j(a, t^{(j)}) = \alpha'$. By (1) we know that $Y_j(a, t^{(j)}) \sqsupseteq Y_j(a, y)$, so $\alpha' = Y_j(a, y)$. Furthermore, by the definition of ternary extension, it follows that $Y_j(a, w) \sqsubseteq Y_j(a, y)$ when $w \sqsubseteq y$. Hence $\alpha' = Y_j(a, w)$ for $w \sqsubseteq y$. Since $w \sqsubseteq y$, and $y_j = \alpha$, we must have $w_j = \alpha$. Together, this shows that vertex j is unstable in any state $w \sqsubseteq y$.

Secondly, we show that for any race sequence starting in state w , vertex j must change, and therefore for any z such that $w \sigma z$ we will have $z_j = \alpha'$. We prove this by contradiction. Assume there exists a race sequence $[z^0, S^0], [z^1, S^1], \dots, [z^k, S^k]$ ($k \geq 0$) with $z^0 = w$, $S^0 = U(a, w)$, and $S^k = \emptyset$, such that $z_j^k = \alpha$. Since, according to the first part above, vertex j is unstable in

any state $w \sqsubseteq y$, we can conclude that $k \geq 1$. Furthermore, since $w \sqsubseteq y$, and, by assumption, $y_j = \alpha$, we must have $z_j^0 = \alpha$. However, by the definition of a race sequence it follows that a vertex can change at most once during a race sequence; since $z_j^0 = \alpha$, and z_j^k is assumed to be α , it follows that $z_j^p = \alpha$ for $p = 0, \dots, k$. Since $t = l.u.b.(y, Y(a, y))$, it follows by the definition of ternary extension that $t \sqsupseteq l.u.b.(w, Y(a, w))$ for $w \sqsubseteq y$. Furthermore, by the definition of a race unit, it follows that for $j = 1, \dots, m$, and $p = 0, \dots, k$, we have that z_j^i is either equal to w_j or $Y_j(a, w)$. We can therefore conclude that $z^p \sqsubseteq l.u.b.(w, Y(a, w))$, and hence $t \sqsupseteq z^p$ for $p = 0, \dots, k$. This, together with the fact that $z_j^p = \alpha = y_j$ for $p = 0, \dots, k$ shows that $t^{(j)} \sqsupseteq z^p$ for $p = 0, \dots, k$. By the monotonicity of ternary extension $Y_j(a, t^{(j)}) \sqsupseteq Y_j(a, z^p)$ for $p = 0, \dots, k$. Furthermore, since $t_j = \times$ we have $\tilde{y}_j = Y_j(a, t^{(j)})$, and, since $\tilde{y}_j = \alpha'$, by assumption, it follows that $Y_j(a, t^{(j)}) = \alpha'$. Together the above two facts imply that $Y_j(a, z^p) = \alpha'$ for $p = 0, \dots, k$. However, this implies that j is in $U(a, z^p)$ for $p = 0, \dots, k$ contradicting the assumption that $S^k = \emptyset$. Therefore the assumption must have been false, and we can conclude that all states z , such that $w \sigma z$, must have $z_j = \alpha'$. Hence, $\tilde{y}_j \sqsupseteq z_j$ holds.

Case 4: $y_j = \times$, $t_j = \times$, and $\tilde{y}_j = \alpha$.

Since $t_j = \times$ it follows that $\tilde{y}_j = Y_j(a, t^{(j)})$, and since we assumed $\tilde{y}_j = \alpha$, it follows that $Y_j(a, t^{(j)}) = \alpha$. By (1) we know that $Y_j(a, t^{(j)}) \sqsupseteq Y_j(a, y)$, so we can conclude that $Y_j(a, y) = \alpha$. By the definition of ternary extension it follows that $Y_j(a, w) = \alpha$ for any $w \sqsubseteq y$. Study any state $w \in B^m$, $w \sqsubseteq y$. We have two cases:

(i) $w_j = \alpha$.

Since $w_j = \alpha$ and $Y_j(a, w) = \alpha$, we have that vertex j is stable, and, as in Case 1 above, we can conclude that for any state z such that $w \sigma z$ we have $z_j = \alpha$. Hence, $\tilde{y}_j \sqsupseteq z_j$ holds.

(ii) $w_j = \alpha'$.

Since $w_j = \alpha'$ and $Y_j(a, w) = \alpha$ we have that vertex j is unstable. Using similar arguments as in the second part of Case 3 above, we can conclude that for any race sequence from the state w , vertex j must change and thus we have that if $w \sigma z$, then $z_j = \alpha$. Hence, $\tilde{y}_j \sqsupseteq z_j$ holds.

We have shown that $\tilde{y}_j \sqsupseteq z_j$ for all possible cases, and hence that $\tilde{y} \sqsupseteq z$ for any state z , such that $w \sigma z$, where $w \sqsubseteq y$. ■

We are now in position to prove Theorem 5.2.

Proof of Theorem 5.2: We want to show that if $y_j^i \in B$, then $\text{l.u.b.}\{z_j : b \sigma^i z\} = y_j^i$. We will prove this by induction on i .

Basis:

$i = 0$. Trivially true.

Induction hypothesis:

Assume that for all $k \leq i$, we have that $y^k \geq \text{l.u.b.}\{z : b \sigma^k z\}$.

Induction step:

We need to show that for any z such that $b \sigma^{i+1} z$, we have $y^{i+1} \sqsupseteq z$. But $b \sigma^{i+1} z$ implies there exists w such that $b \sigma^i w$ and $w \sigma z$. By the induction hypothesis $w \sqsubseteq y^i$, and Lemma 5.1 applies. Thus $y^{i+1} \sqsupseteq z$. ■

The following example shows that the inequality of the theorem can not be replaced by equality, i.e. that the TAED model is sometimes overly pessimistic. Study network N_6 of Fig. 5.11, started in the stable total state $\langle 0, 0111000 \rangle$ and with the new input $a = 1$. In Fig. 5.12 we show the binary stepwise AED analysis of the race and in Fig. 5.13 we show the TAED analysis. Note that the \times for vertex 6 in state y^3 is incorrect. The reason for this discrepancy between the ternary simulation and the binary race analysis is that in the state y^2 , not all binary states $\sqsubseteq y^2$ are reachable, and in particular, the state $y = 1011100$ is not reachable from the initial state. It is this state that causes vertex 6 to be unstable in the ternary simulation, and eventually leads to the \times in y_6^3 . In general, the discrepancy occurs because of the loss of information in the ternary simulation where we only use the “average” of the states

reachable. It appears that this pessimism occurs only in pathological examples, and that for most practical circuits, the ternary and the binary AED analysis correspond exactly. However, the problem of characterizing the class of networks for which the two models agree remains open.

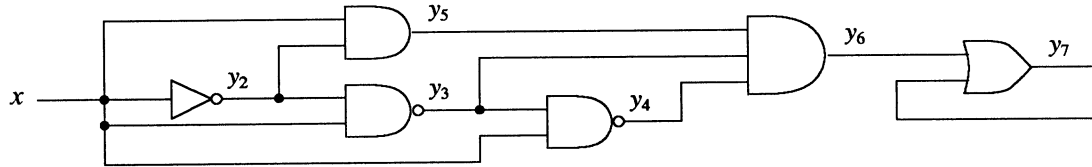


Figure 5.11. Network N_6 .

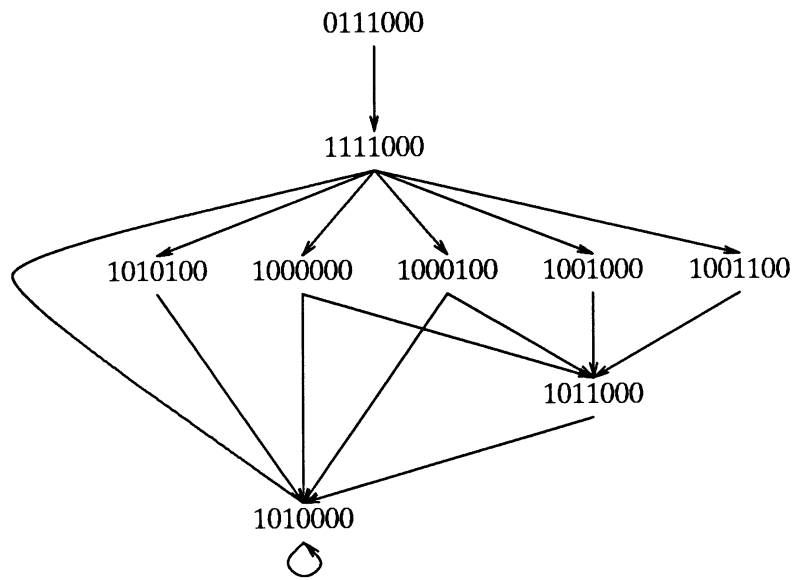


Figure 5.12. Stepwise AED analysis of N_6 .

$y^0 = 0111000$	$l.u.b.(Z^0) = 0111000$
$t^1 = \times 111000$	
$y^1 = 1111000$	$l.u.b.(Z^0) = 1111000$
$t^2 = 1\times\times\times 00$	
$y^2 = 10\times\times 00$	$l.u.b.(Z^1) = 10\times\times 00$
$t^3 = 10\times\times\times 0$	
$y^3 = 101\times 0\times 0$	$l.u.b.(Z^2) = 101\times 000$
$t^4 = 101\times 0\times\times$	
$y^4 = 101000\times$	$l.u.b.(Z^3) = 1010000$
(a)	(b)

Figure 5.13. (a) TAED analysis of N_6 ; (b) $l.u.b.$ of the stepwise AED analysis.

5.6. Discussion

In this chapter, we have presented a new ternary simulation algorithm that can easily replace the unit-delay algorithm in simulators. The algorithm is very closely related to the binary almost-equal-delay model, and hence is capable of detecting critical races under the assumption that all delays are approximately, but not exactly, equal. Computationally, the ternary algorithm is of the same order of complexity as the unit-delay method (in the worst case it involves only twice as many function evaluations).

It is possible to extend the AED model to allow changes to go through \times in the same way as was done in the XMW model. One would then obtain an extended almost-equal-delay (XAED) model. We will not derive such a model here; the formulation of it is left as an exercise for the interested reader[†]. However, it appears plausible that the results of an XAED analysis correspond even closer to the results of the TAED algorithm than the results of an AED analysis. To get a complete correspondence though, we believe the TAED algorithm has to be modified slightly.

[†] The ideas for this extension are straightforward, but the detailed formulation is rather lengthy.

The modification is simply to use the state t instead of $t^{(j)}$ in the re-evaluation phase. To see why this appears to be necessary, study network N_5 of Fig. 5.9(b). Note that the value of the output can change to \times in an XAED model, causing the gate to become stable. However, using the TAED algorithm as described in Section 4 would yield the new value 0. On the other hand, using t instead of $t^{(j)}$ in the second half of the TAED algorithm, we get the new value of the gate to be \times . We conjecture the following:

Conjecture 5.1 Let $\langle \hat{a}, b \rangle$ be a stable total state of N , and let a denote a new input vector. Let y^i denote the results of the (modified) TAED method after $i \geq 0$ steps starting in $y^0 = b$. Then $y^i = l.u.b.(Z^i)$, where Z^i is the set of states reachable in i steps in the stepwise XAED model.

If this conjecture is correct, the (modified) TAED algorithm would provide a practical and accurate method of analyzing a circuit in which the delays are all approximately equal.

Chapter VI

The Extended Bounded Delay Race Model

In this chapter we first develop a delay model, called the extended inertial delay (XID) model, that captures the idea that the delay value lies between two bounds. Furthermore, this model also incorporates the idea that transitions are not instantaneous, but can be rather slow and go through the intermediate value \times . We also define a race model that can be used to predict the behavior of a network when each component of the network is assumed to consist of an ideal (delay-free) device connected in series with such a delay. As far as we are aware, this is the first non-trivial race model that has been formally proved to correspond to some delay model. The race model we develop here is called the extended bounded delay (XBD) model and takes all possible delay values into consideration. This implies that the model is continuous and thus computationally intractable. However, we show that there also exists an efficient algorithm, called the ternary bounded delay (TBD) algorithm, that can be used to obtain essentially the same information as the XBD race model can provide.

6.1. Introduction

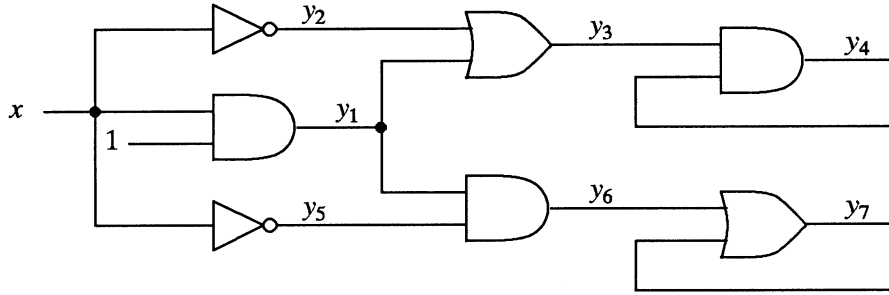
In the previous chapters we showed that the unit delay model is often overly optimistic and that the XMW model is often overly pessimistic. The AED model represent one attempt to find an intermediate model, but it may be too optimistic. What appears to be needed is a model between the AED and the XMW models, i.e. a model in which the delays cannot be arbitrary, but bounded by some lower and upper limits. For example, it is often possible to estimate the delay of a gate

reasonably accurately (e.g. $10\text{ns} \pm 20\%$). It is important to realize that (in a binary model) it is *not* sufficient to study only the “worst-case” delays. Consider, for example, the gate circuit G_1 of Fig. 6.1. The excitation functions are given by:

$$\begin{aligned} Y_1 &= 1x = x & Y_2 &= x' & Y_3 &= y_1 + y_2 & Y_4 &= y_3 y_4 \\ Y_5 &= x' & Y_6 &= y_1 y_5 & Y_7 &= y_6 + y_7 \end{aligned}$$

Assume G_1 is started in the stable state $x=0$, $y=(y_1, \dots, y_7)=0111100$ and that the input changes to $x=1$. Furthermore, assume the delay $\Delta(t)$ in a gate is bounded by $1 \leq \Delta(t) < 5$. If we only study “worst-case” delays, i.e. if we only consider delays to be close to 1 or close to 5, it can be verified that the only possible nontransient states reachable are 1011000, 1010000 and 1011001, i.e. at least one of y_4 and y_7 does not change. However, if we also allow the delays to be anywhere between the bounds, we can also reach the state 1010001, i.e. a state in which both y_4 and y_7 have changed. This state can be reached if, for example, $\Delta_1=2.5$, $\Delta_5=4.5$, and the remaining delays are equal to 1. In fact, one can verify that this state is reachable only if $2 \leq \Delta_1 < 4$. This can be shown as follows: In order for y_4 to change, y_3 must be 0 for at least one time unit. This implies that y_2 must change from 1 to 0 at least one time unit before y_1 changes from 0 to 1. However, the delay in gate 2 is at least one time unit, and thus $\Delta_1 \geq 2$ in order for y_4 to be able to change. On the other hand, in order for y_7 to change, y_6 must be 1 for at least one time unit, i.e. y_1 must change to 1 at least one time unit before y_5 changes from 1 to 0. However, the maximum delay in gate 5 is less than 5, and thus $\Delta_1 < 4$ for y_7 to change. In summary, a necessary condition for reaching a state in which both y_4 and y_7 have changed is that $2 \leq \Delta_1 < 4$. Thus a continuous model appears to be needed; we will develop such a model in this chapter.

The chapter is organized as follows. The extended inertial delay model is the topic of Section 6.2. Since the model is quite subtle, we develop it by first describing a series of simpler delay models. In Section 6.3 we define the extended bounded delay race model and show that this race model exactly captures the behavior of a network under the assumption that each component of the network consists of an ideal (delay-free) excitation function connected in series with an extended inertial

Figure 6.1. Gate circuit G_1 .

delay. In Section 6.4 we define the TBD algorithm and in Section 6.5 we prove that the results obtained from this algorithm exactly summarize all the states the network can be in according to the XBD race model. Finally, in Section 6.6 we discuss some implementation considerations.

6.2. Extended Inertial Delay Model

In this section we introduce some mathematical models of delays. The delay model used in the remaining parts of this chapter, the extended inertial delay model, is rather complicated. In order to justify it, and to give the reader some intuitive idea why certain assumptions are made, we develop this delay model step by step from a very simple model. To simplify our discussion, we will first study the behavior of a basic “delay element”. A real circuit component, like a gate, can then be modeled as an ideal (delay-free) device (gate) connected in series with such a delay element. We will first limit our discussion to binary models, but this restriction will be dropped later.

A *delay* is assumed to be a “black box” that has one input and one output (see Fig. 6.2), and whose input/output behavior is governed by a *delay model*. To simplify the definitions, assume that $in(t) = out(t) = \beta$ for $t < 0$, and that $in(t)$ has only a finite number of changes in any finite interval. One simple delay model is the *fixed binary pure delay* (FBPD) model. In the FBPD model it is assumed that $\Delta(t) = \Delta$, i.e. that the delay does not vary with time, and that $out(t) = in(t - \Delta)$, i.e. that the output is an exact replica of the input, but shifted Δ units of time. In Fig. 6.3 we

show an example of the response of a FBPD to an input signal. Unfortunately, this simple model is not very realistic since it fails to capture the fact that almost all physical delays ignore extremely short pulses, i.e. they tend to “smooth” fast varying signals.

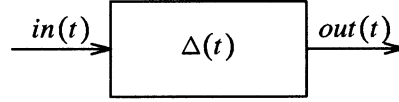


Figure 6.2. Delay element.

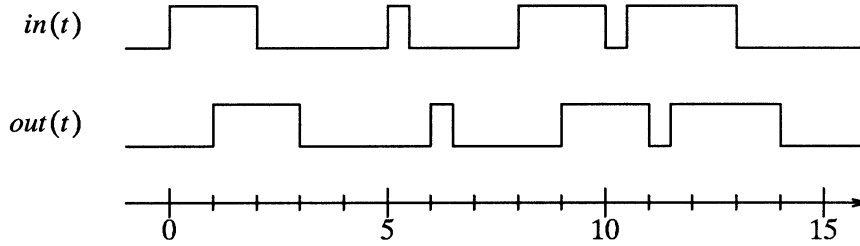


Figure 6.3. Typical waveforms for FBPD model ($\Delta = 1$).

The *fixed binary inertial delay* (FBID) model tries to capture the property mentioned above by using the following assumptions. First, the delay $\Delta(t)$ is assumed to be constant in time, i.e. $\Delta(t) = \Delta$. Second, if $out(t)$ changes from α to α' at time τ , then we must have had $in(t) = \alpha'$ for $\tau - \Delta \leq t < \tau$. Third, if $in(t)$ changes from α to α' at time τ and $out(\tau) = \alpha$, then either $in(t)$ changes back to α before $\tau + \Delta$ or $out(t)$ changes to α at time $\tau + \Delta$.

Using these assumptions directly, it is straightforward to verify that a given input/output behavior is consistent with the FBID model. However, to gain a better understanding of the model, it is useful to compute the output waveform given an arbitrary input signal waveform. Assume we are given an input waveform with a finite number of changes. Let $\tau^1, \tau^2, \dots, \tau^M$ denote the time for the first, second, ..., M^{th} change of the input signal. We will henceforth use the convention that all changes occur at times τ^i , i.e. that the input is constant in the interval $\tau^i \leq t < \tau^{i+1}$. For convenience define $\tau^{M+1} = \infty$. As above, we assume that $in(t) = out(t) = \beta$ for

$t < 0$. The value of the output of a FBID is defined inductively as follows:

Basis:

$$out(t) = \beta \text{ for } t \leq \tau^1.$$

Inductive step:

Given the value of $out(\tau^i)$ and $in(\tau^i)$ the value of out is defined for the time up to and including τ^{i+1} as follows:

- 1) If $\tau^{i+1} - \tau^i < \Delta$ then

$$out(t) = out(\tau^i) \text{ for } \tau^i < t \leq \tau^{i+1}$$

- 2) If $\tau^{i+1} - \tau^i \geq \Delta$ then

$$out(t) = \begin{cases} out(\tau^i) & \text{for } \tau^i < t < \tau^i + \Delta \\ in(\tau^i) & \text{for } \tau^i + \Delta \leq t \leq \tau^{i+1} \end{cases}$$

We claim that this definition is equivalent to the original definition of the FBID model. It is left as an exercise for the reader to verify that this is indeed the case. In Fig. 6.4 we show how a FBID would react to the same input signal as in Fig. 6.3.

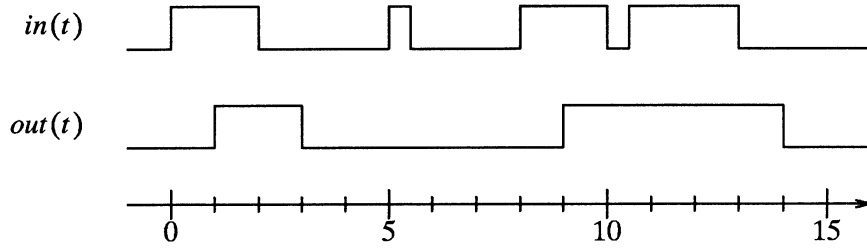


Figure 6.4. Typical waveforms for FBID model ($\Delta = 1$).

Both models described so far have one undesirable property in common — the delay is assumed to be exactly known and constant. In real circuits we have two problems. The first problem is that we normally do not know the exact size of a delay. The second problem is that delays may not be constant; for example, they can change due to temperature or aging. Furthermore, the delay of a component can depend on the previous history of the component. For example, a gate that just changed from 0 to 1, may have a shorter delay for changing back to 0 than it would

have had if it had had the value 1 for a long time. In order to model these properties we will allow delays to vary in time. This is somewhat pessimistic, but we are more concerned with avoiding false positive than false negative predictions. (A false positive prediction says that everything is fine, despite the fact that there might exist a timing problem. A false negative prediction may “cry wolf” even if there are no problems.) However, we will only allow the delay to vary within some bounds. In particular, we will henceforth assume that the delay $\Delta(t)$ is bounded by:

$$dT \leq \Delta(t) < DT$$

where d and D are positive integers and T denotes some basic time unit.

In the (variable) *binary inertial delay* (BID) model the input/output behavior must obey the following two rules:

- 1) If $out(t)$ changes from α to α' at time τ , then we must have had $in(t) = \alpha'$ for $\tau - dT \leq t < \tau$.
- 2) If $in(t) = \alpha$ for $\tau \leq t < \tau + DT$ then there must exist a time τ^c , $\tau^c < \tau + DT$, such that $out(t) = \alpha$ for $\tau^c \leq t < \tau + DT$.

In Fig. 6.5 we show two possible responses to the input waveform of Fig. 6.3 when the delay is bounded by $1 \leq \Delta(t) < 2$.

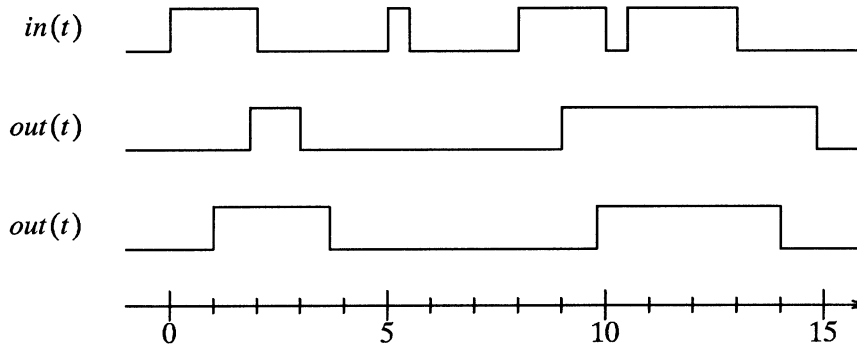


Figure 6.5. Two possible waveforms according to BID model ($1 \leq \Delta(t) < 2$).

In the above delay-models we have assumed that all signals are binary, and that changes from 0 to 1 or from 1 to 0 are instantaneous. In real circuits this is not very likely. In order to capture the fact that signals that are changing can go through an intermediate voltage level we introduce a new value \times . By using the partial order \sqsubseteq on the set $\mathcal{T} = \{0, 1, \times\}$, as defined in Chapter II, Section 2, we can model the property that a component that sees a changing value (an \times) can interpret this signal as either 0, 1, or \times . Using such an approach we define the (variable) *extended inertial delay* (XID) model as follows:

- 1) $out(t)$ can only change from a binary value to \times or from \times to a binary value. Hence, the output can never change directly from 0 to 1 or from 1 to 0.
- 2)
 - a) If $out(t)$ changes from a binary value α to \times at some time τ then we must have had $in(t) \neq \alpha$ for $\tau - dT \leq t < \tau$.
 - b) If $out(t)$ changes from \times to a binary value α at some time τ then we must have had $in(t) = \alpha$ for $\tau - dT \leq t < \tau$.
- 3)
 - a) If $in(t) = \beta \in \mathcal{T}$ for $\tau \leq t < \tau + DT$, then there must exist a τ^c , $\tau^c < \tau + DT$, such that $out(t) = \beta$ for $\tau^c \leq t < \tau + DT$.
 - b) If $in(t) \neq \alpha \in \mathcal{B}$ for $\tau \leq t < \tau + DT$, then there must exist a τ^c , $\tau^c < \tau + DT$, such that $out(t) \neq \alpha$ for $\tau^c \leq t < \tau + DT$.

Assumption 1) is made to simplify the model. One could equally well use an approach similar to the one used in the XMW model, where vertices can change both between binary values but also to \times . However, such model would make the extended bounded delay model (to be defined later) even more complex. Since the above definition of the XID model allows a vertex to stay in the \times value for an arbitrarily short period of time during a transition from 0 to 1 or from 1 to 0, we feel that this simplification can be justified.

The reason for restricting rule 3b) to binary values, is that it is quite reasonable to allow the output to be \times when the input is oscillating between 0 and 1 (assuming each period of 0(1) is strictly less than DT of course).

In Fig. 6.6 we show a possible response to the same binary input signal as in Fig. 6.3. However, the response of an XID to a signal containing \times 's is more interesting. Two possible responses to such an input signal are shown in Fig. 6.7. Note that the XID can both increase and decrease the regions of \times 's. (The latter case can occur, for example, when the delay is smaller changing from 0 to \times than when changing from \times to 1.)

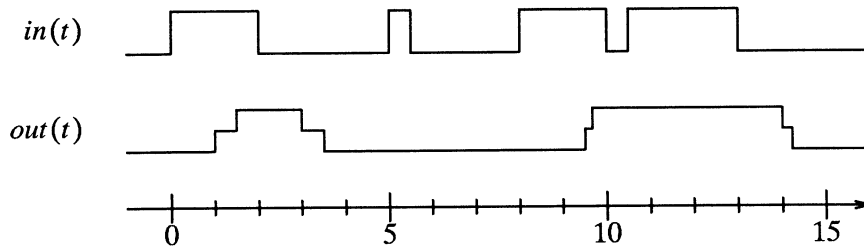


Figure 6.6. A possible waveform according to XID model ($1 \leq \Delta(t) < 2$).

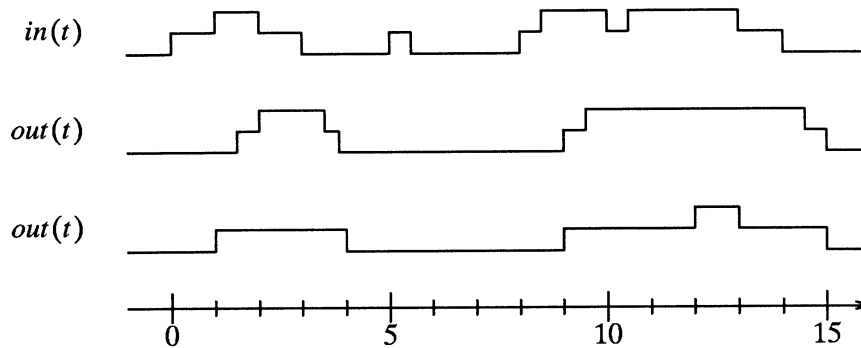


Figure 6.7. Two possible waveforms according to XID model ($1 \leq \Delta(t) < 2$).

6.3. Bounded Delay Race Model

In this section we develop a race model, called the *extended bounded delay* (XBD) race model, that will be used to analyze the behavior of a network. Later in this section we will show that the XBD race model exactly captures the behavior of a network under the assumption that each vertex consists of an ideal (delay-free) excitation function connected in series with an extended inertial delay (XID).

Assume the network N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a . Assume also that the delay $\Delta_j(t)$ of vertex j , $1 \leq j \leq m$, is bounded by $d_j T \leq \Delta_j(t) < D_j T$, for some non-negative integers d_j and D_j .

In the extended bounded delay race model a certain amount of the previous excitation history is needed. For this reason, define a *race state* to be the 4-tuple $[y, u, v, t]$ as follows: The first component, y , is the current state of the network — consequently $y \in \mathcal{T}^m$. The second component, u , is a vector of m real numbers and is used to remember how long an unstable vertex with a binary value has been unstable. The third component, v , serves a similar purpose, but is used to remember how long an unstable vertex has had a binary excitation. Finally, the last component, t , is a real number denoting the time when this state was reached. Note that, for convenience, the input is assumed to change from \hat{a} to a at time 0.

Let Q denote the set of race states that are reachable according to the extended bounded delay race model, and let R_a denote a binary relation defined on the set Q denoting the possible successor states. A XBD *race sequence* is an infinite sequence of race states, $[y^0, u^0, v^0, t^0], [y^1, u^1, v^1, t^1], \dots$, such that $[y^0, u^0, v^0, t^0] = [b, (0, \dots, 0), (0, \dots, 0), 0]$ and $[y^h, u^h, v^h, t^h] R_a [y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$, for $h \geq 0$. One can view a XBD race sequence as a sequence of “snapshots” of the network. The network starts in the state $[b, (0, \dots, 0), (0, \dots, 0), 0]$ at time 0, and hence $[b, (0, \dots, 0), (0, \dots, 0), 0] \in Q$. Given some state $[y, u, v, t]$ in Q , a possible successor state is determined as follows. If $\langle a, y \rangle$ is a stable total state, the network will remain in this state indefinitely, and thus the state has only itself as successor. Otherwise, we want to take a new snapshot of the network at time $t + \delta$, for some $\delta > 0$. However, there is no point in choosing δ too small, since a vertex with a binary value must be unstable for at least $d_j T$ units of time before it can change to \times , and an

unstable vertex must have the same binary excitation for at least $d_j T$ units of time before it can change to this excitation. These conditions imply that δ must be chosen so that there exists at least one unstable vertex j for which $u_j + \delta$ or $v_j + \delta$ is greater than or equal to $d_j T$. On the other hand, since a vertex cannot be unstable and have a binary value, or be unstable and have a binary excitation, for $D_j T$ units of time without changing, δ must be chosen so that each of $u_j + \delta$ and $v_j + \delta$ is strictly smaller than $D_j T$ for all unstable vertices j . These two conditions can be summarized as follows:

$$\min\{\min\{d_j T - u_j, d_j T - v_j\} : j \in U(a, y)\} \leq \delta < \min\{\min\{D_j T - u_j, D_j T - v_j\} : j \in U(a, y)\}.$$

Once δ has been chosen, the sets C^\times and C^B are computed. C^\times contains all the vertices that, for this δ , are candidates for changing to \times , i.e. it contains all vertices that are binary, unstable, and for which $u_j + \delta$ is greater than or equal to $d_j T$. In other words, a vertex j is in C^\times if it has had the same binary value and has been unstable ever since some time t^p and $t + \delta - t^p \geq d_j T$. The set C^B contains all the vertices that, for this δ , are candidates for changing from \times to a binary value, i.e. all the vertices that are \times , are unstable, and for which $v_j + \delta$ is greater than or equal to $d_j T$. In other words, a vertex j is in C^B if $y_j = \times$ and it has had the same binary excitation and has been unstable ever since some time t^p and $t + \delta - t^p \geq d_j T$. Finally, some nonempty subset of $C^\times \cup C^B$ is chosen, the appropriate vertex values are changed, and u and v are updated accordingly. It is important to note that, if a state in Q is unstable, it has infinitely many possible successor race states. (On the other hand, these race states may differ only in the u , v , and t values.)

Now Q and R_a are formally defined inductively as follows:

Basis: Let $[b, (0, \dots, 0), (0, \dots, 0), 0] \in Q$

Induction step: Given $q = [y, u, v, t] \in Q$:

- 1) if $U(a, y) = \emptyset$, then $q R_a q$.

2) if $U(a, y) \neq \emptyset$, then for any $\delta > 0$ such that

$$\delta_{\min} \leq \delta < \delta_{\max},$$

where

$$\delta_{\min} = \min \{ \min \{ d_j T - u_j, d_j T - v_j \} : j \in U(a, y) \}$$

and

$$\delta_{\max} = \min \{ \min \{ D_j T - u_j, D_j T - v_j \} : j \in U(a, y) \},$$

let

$$C^\times = U(a, y) \cap B(y) \cap \{j : u_j + \delta \geq d_j T\} \text{ and}$$

$$C^B = U(a, y) \cap \{j : y_j = \times\} \cap \{j : v_j + \delta \geq d_j T\}.$$

Finally, for any $P \subseteq C^\times \cup C^B$, $P \neq \emptyset$, let

$$\bar{q} = [\bar{y}, \bar{u}, \bar{v}, t + \delta] \in Q \quad \text{and} \quad q R_a \bar{q},$$

where

$$\bar{y}_j = \begin{cases} \times & \text{if } j \in P \cap C^\times \\ Y_j(a, y) & \text{if } j \in P \cap C^B \\ y_j & \text{otherwise,} \end{cases}$$

$$\bar{u}_j = \begin{cases} u_j + \delta & \text{if } j \in U(a, y) \cap B(y) \cap U(a, \bar{y}) \cap B(\bar{y}) \\ 0 & \text{otherwise,} \end{cases}$$

and

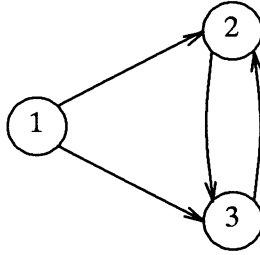
$$\bar{v}_j = \begin{cases} v_j + \delta & \text{if } j \in U(a, y) \cap BE(a, y) \cap U(a, \bar{y}) \cap BE(a, \bar{y}) \\ 0 & \text{otherwise.} \end{cases}$$

To illustrate the above definition, consider network N_2 of Fig. 6.8 with excitation functions and delay bounds given by:

$$Y_1 = x, \quad Y_2 = (y_1 + y_3)', \quad Y_3 = (y_1 + y_2)'.$$

$$1 \leq \Delta_1(t) < 3 \quad 1 \leq \Delta_2(t) < 3 \quad 1 \leq \Delta_3(t) < 3$$

Assume N_2 is started in the stable total state $\langle 1, (1, 0, 0) \rangle$ and the input changes to $x = 0$. To simplify the notation, the basic time unit, T , is set to 1. Two possible

Figure 6.8. Network N_2 .

XBD race sequences are shown in Fig. 6.9. As before, we use the notation 0_\times (0_1 , \times_0 , etc.) to denote a vertex that currently has the value 0 (0 , \times , etc.) and whose excitation is \times (1 , 0 , etc.). Furthermore, the numbers to the left of the arrows denote the limits for δ , and the numbers to the right of the arrows correspond to the δ 's chosen.

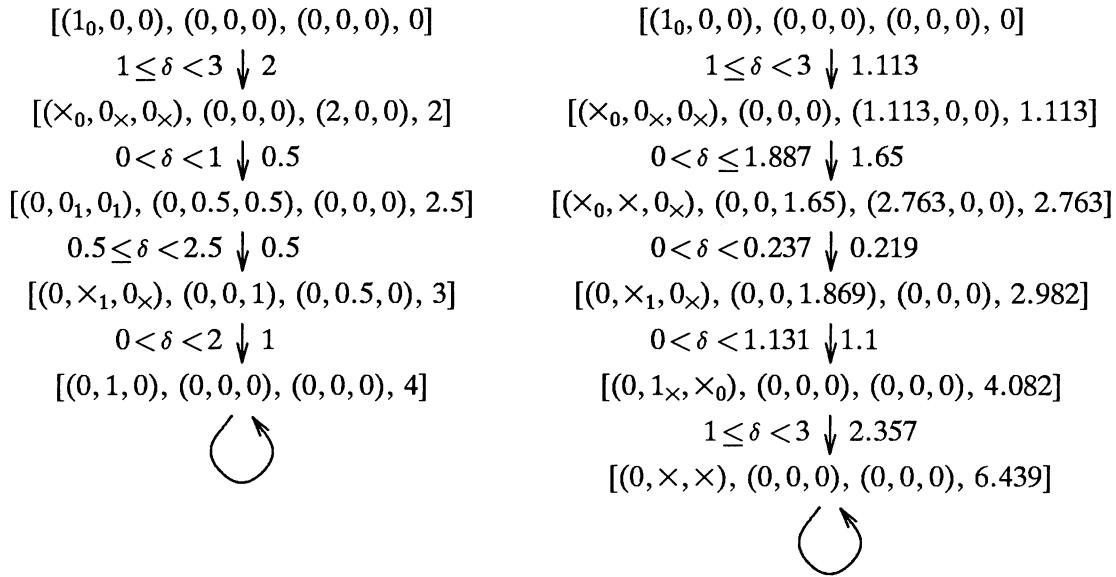


Figure 6.9. Two possible XBD race sequences.

To illustrate some steps in the computation of the race sequence, consider the first states in the left XBD race sequence of Fig. 6.9. We start in the stable state $[(1_0, 0, 0), (0, 0, 0), (0, 0, 0), 0]$, and it is easy to verify that $U(0, 100) = \{1\}$. Now, for all the unstable vertices j (in this case vertex 1 only) we compute the values of $d_jT - u_j$ and $d_jT - v_j$, and find the minimum value. In our case we get $\min_{j \in U(0, 100)} \{d_jT - u_j, d_jT - v_j\} = \min\{1 - 0, 1 - 0\} = 1$, and hence, δ must be chosen to be greater than or equal to 1. In a similar way, δ must be chosen strictly smaller than $\min_{j \in U(0, 100)} \{D_jT - u_j, D_jT - v_j\} = \min\{3 - 0, 3 - 0\} = 3$. In our example, we choose $\delta = 2$. Once δ has been chosen, we must determine the set of vertices C^\times that, for this choice of δ , are candidates for changing to \times , and the set C^B of vertices that, for this choice of δ , are candidates for changing from \times to a binary value. More specifically, the vertices in C^\times are all unstable, all have a binary value, and for all of them $u_j + \delta \geq d_jT$. C^B is defined similarly. In the present situation $C^\times = \{1\}$ and $C^B = \emptyset$. Once C^\times and C^B have been computed, we have to choose some nonempty subset of $C^\times \cup C^B$ as the set P . Here there is no choice; P must be equal to $\{1\}$. Given P and δ we now proceed as follows. The state of vertex j is changed to \times if j is in $C^\times \cap P$, and to its excitation value if j is in $C^B \cap P$. For all other vertices, the value is not changed. In our example, only y_1 is changed, and it is changed to \times . When this is done, the vectors u and v are updated as follows: If a vertex j was unstable and had a binary excitation in the previous state and the same situation holds in this new state, then u_j is incremented by δ . Otherwise, u_j is set to 0. Similarly, a vertex that was unstable and had a binary excitation in the previous state and in the new state, gets v_j incremented by δ ; all other vertices get v_j set to 0. In our example, vertex 1 is not binary in the new state and thus $\bar{u}_1 = 0$. On the other hand, vertex 1 satisfies all the conditions for getting v_1 incremented, and thus $\bar{v}_1 = \delta = 2$. The vertices 2 and 3 were not unstable in the previous race state, and thus $\bar{u}_2 = \bar{u}_3 = \bar{v}_2 = \bar{v}_3 = 0$. Hence, one possible successor state is $[(\times_0, 0_\times, 0_\times), (0, 0, 0), (2, 0, 0), 2]$.

To determine a possible successor state to the state $[(\times_0, 0_\times, 0_\times), (0, 0, 0), (2, 0, 0), 2]$, we proceed very much in the same way. First we establish that $U(0, \times 00) = \{1, 2, 3\}$. We then find that $\min_{j \in U(0, \times 00)} \{d_jT - u_j, d_jT - v_j\} =$

$\min\{1-0, 1-2\} = -1$ and $\min_{j \in U(0, \times_{00})} \{D_j T - u_j, D_j T - v_j\} = \min\{3-0, 3-2\} = 1$. However, δ must be non-negative; thus we get the condition that δ must be in the open interval $(0, 1)$. In our example, δ is chosen to be 0.5. It is easy to verify that, for this choice of δ , $C^\times = \emptyset$ and $C^B = \{1\}$. Intuitively, although vertices 2 and 3 are unstable, they have only been unstable since time 2, and since the maximum delay in vertex 1 must be less than 3, it follows that only vertex 1 will be a candidate for changing. As above, we get $P = \{1\}$ and the reader can verify that we reach the state $[(0, 0_1, 0_1), (0, 0.5, 0.5), (0, 0, 0), 2.5]$.

The following two properties of the XBD race model can be easily verified. They both deal with the minimum-sized pulse that can occur in a network according to the XBD race model. The first proposition states that the minimum time a vertex j can have a binary value is essentially $d_j T$.

Proposition 6.1 Let $\gamma = [y^0, u^0, v^0, t^0], [y^1, u^1, v^1, t^1], \dots$ be an arbitrary XBD race sequence for network N when N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a at time 0. If, for some $q \geq 1$, $y_j^{q-1} = \alpha \in B$ and $y_j^q = \times$, then $t^q \geq d_j T$ and there exists a $[y^s, u^s, v^s, t^s] \in \gamma$ such that $t^q - t^s \geq d_j T$ and $y_j^r = \alpha$ for $s \leq r \leq q-1$.

The second proposition states that the minimum time a vertex can have the value \times and then change back to its previous value is also essentially $d_j T$. Note that the XBD race model allows a vertex to have the value \times for an arbitrarily short time, provided this occurs during a transition from 0 to 1 or from 1 to 0.

Proposition 6.2 Let $\gamma = [y^0, u^0, v^0, t^0], [y^1, u^1, v^1, t^1], \dots$ be an arbitrary XBD race sequence for network N when N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a at time 0. If $y_j^{s-1} = \alpha \in B$, $y_j^s = \times$, and $y_j^q = \alpha$, for $q > s$, then $t^q - t^s \geq d_j T$.

There are two natural questions to ask when analyzing a transition caused by an input change:

- a) which state(s) can the network be in at some time t , and
- b) will the network eventually end up in a unique stable state?

The second question, the “stable state reachability” (SSR) problem is in general very difficult (see Chapter VII). Hence, we will focus on the first question, i.e. which states can the network be in at some time t . In practice, this question is also of greater importance than the SSR question, since circuits that have not reached a stable state after some reasonably short time are probably not very well designed and need to be modified. Hence, it is usually sufficient to determine whether a circuit will reach a stable state within time rT for some fixed (and relatively small) r .

Let Γ be the set of all XBD race sequences for network N when N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a at time 0. For $\gamma \in \Gamma$, the state of the network according to this race sequence is denoted $y^\gamma(t)$, and is equal to:

$$y^\gamma(t) = \begin{cases} b & \text{for } t < 0 \\ y^i & \text{for } t^i \leq t < t^{i+1} \text{ if } U(a, y^i) \neq \emptyset \text{ or for } t^i \leq t \text{ if } U(a, y^i) = \emptyset \end{cases}$$

The set of possible states the network can be in at time t is called $Reach(t)$ and is defined as:

$$Reach(t) = \{y : y = y^\gamma(t) \text{ for some } \gamma \in \Gamma\}$$

In the above example, it is easy to verify that $Reach(t) = \{(1, 0, 0)\}$ for $0 \leq t < 1$, that $Reach(1) = \{(1, 0, 0), (\times, 0, 0)\}$, and that $Reach(t) = \{(1, 0, 0), (\times, 0, 0), (0, 0, 0)\}$ for $1 < t < 2$. However, it is very laborious to compute $Reach(t)$ for $t \geq 2$. We will return to this in the next section.

In the introduction to this section we promised to show that the behavior of the network according to the XBD race model is identical to the behavior of the network when each vertex is assumed to consist of an ideal (delay-free) excitation function connected in series with an extended inertial delay. We will show this by establishing Lemmas 6.1 and 6.2. However, the following propositions will be needed before this can be done.

Proposition 6.3 Let $\langle a, b \rangle \in \mathcal{T}^{n+m}$ and $\langle a, c \rangle \in \mathcal{T}^{n+m}$ be two total states of N such that, for $1 \leq j \leq m$, $b_j \in B$ implies $c_j \supseteq b_j$. Then, for $1 \leq i \leq m$, we have $Y_i(a, b) \in B$ implies $Y_i(a, c) \supseteq Y_i(a, b)$.

Proof: By the definition of $\langle a, b \rangle$ and $\langle a, c \rangle$ there exists a lower bound of $\{\langle a, b \rangle, \langle a, c \rangle\}$ in the partial order \sqsubseteq . Let $\langle a, d \rangle$ denote such a lower bound (it may not be unique). We now argue by contradiction that the excitation cannot change from $\alpha \in B$ to α' . Assume that $Y_i(a, b) = \alpha \in B$ and that $Y_i(a, c) = \alpha'$ for some i . By the definition of a lower bound we must have $\langle a, d \rangle \sqsubseteq \langle a, b \rangle$, and thus, by the monotonicity of Y , that $Y_i(a, d) \sqsubseteq Y_i(a, b) = \alpha$. Hence, $Y_i(a, d) = \alpha$. However, by the same arguments, it follows that $\langle a, d \rangle \sqsubseteq \langle a, c \rangle$, and thus that $Y_i(a, d) \sqsubseteq Y_i(a, c) = \alpha'$, showing that $Y_i(a, d) = \alpha'$ — a contradiction. Hence, our assumption must be false and the claim follows. ■

Proposition 6.4 states essentially that the value of a vertex and the excitation of a vertex according to the XBD race model can never change directly from 0 to 1 or from 1 to 0.

Proposition 6.4 Let $\gamma = [y^0, u^0, v^0, t^0], [y^1, u^1, v^1, t^1], \dots$ be an arbitrary XBD race sequence for network N when N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a at time 0. Assume $[y^h, u^h, v^h, t^h] \in \gamma$, then

- (i) if $y_j^h \in B$, then $u_j^h \geq v_j^h$.
- (ii) if $y_j^h \in B$, then $y_j^{h+1} \supseteq y_j^h$.
- (iii) if $Y_j(a, y^h) \in B$, then $Y_j(a, y^{h+1}) \supseteq Y_j(a, y^h)$.

Proof: By induction on h .

Basis:

$h = 0$. Since $u_j^0 = v_j^0 = 0$ for $1 \leq j \leq m$, claim (i) follows trivially. Furthermore, since $\langle \hat{a}, b \rangle$ is assumed to be a stable total state, and $a \neq \hat{a}$, we must have $[y^1, u^1, v^1, t^1] \neq [y^0, u^0, v^0, t^0]$. Let C^\times and C^B be defined for $[y^0, u^0, v^0, t^0]$ and $\delta = t^1 - t^0$. If $y_j^0 = \alpha \in B$ we claim that $j \notin C^B$. Assume the contrary, i.e. assume that $j \in C^B$. Since this implies that $j \notin C^\times$ and, by assumption, $j \in B(y^0)$, it follows that we must have $v_j^0 + \delta \geq d_j T$ and $u_j^0 + \delta < d_j T$. However, this contradicts the fact that $u_j^0 = v_j^0 = 0$. Hence our assumption that $j \in C^B$ must be false. However, $j \notin C^B$ implies that $y_j^1 \in \{\alpha, \times\}$ and property (ii) holds for the basis. Finally, property (iii) follows from property (ii) and Proposition 6.3.

Induction hypothesis:

Assume the claim holds for some $h \geq 0$.

Induction step:

Consider $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$. If $U(a, y^{h+1}) = \emptyset$, properties (i)-(iii) follow trivially. Hence, assume that $U(a, y^{h+1}) \neq \emptyset$. To verify property (i) we argue by contradiction. Assume $y_j^{h+1} = \alpha \in B$ and that $u_j^{h+1} < v_j^{h+1}$. Consider the value of vertex j at step h , i.e. y_j^h . By the induction hypothesis, property (ii), y_j^h must be equal to either α or \times .

Case 1: If $y_j^h = \alpha$, then by the induction hypothesis, property (i), $u_j^h \geq v_j^h$. Since we assumed $u_j^{h+1} < v_j^{h+1}$ and u_j^{h+1} is either equal to $u_j^h + \delta$ or 0 and v_j^{h+1} is either equal to $v_j^h + \delta$ or 0, it follows that $u_j^{h+1} = 0$ and that $v_j^{h+1} = v_j^h + \delta$. However, this implies that

$$j \notin U(a, y^h) \cap B(y^h) \cap U(a, y^{h+1}) \cap B(y^{h+1})$$

and that

$$j \in U(a, y^h) \cap BE(a, y^h) \cap U(a, y^{h+1}) \cap BE(a, y^{h+1}).$$

Hence, $j \notin B(y^h) \cap B(y^{h+1})$. However, we assumed that $y_j^h = y_j^{h+1} = \alpha \in B$; thus we have a contradiction.

Case 2: $y_j^h = \times$. Since $y_j^{h+1} = \alpha$, we must have $Y_j(a, y^h) = \alpha$. By the induction hypothesis, property (iii), it therefore follows that $Y_j(a, y^{h+1}) \in \{\alpha, \times\}$. If $Y_j(a, y^{h+1}) = \alpha$, then $y_j^{h+1} = Y_j(a, y^{h+1})$ and thus $j \notin U(a, y^{h+1})$ and therefore $u_j^{h+1} = v_j^{h+1} = 0$, which contradicts our assumption that $u_j^{h+1} < v_j^{h+1}$. Hence, $Y_j(a, y^{h+1})$ must be equal to \times . However, this implies that $j \notin BE(a, y^{h+1})$ and therefore that $v_j^{h+1} = 0$. Since $u_j^{h+1} \geq 0$ this also leads to a contradiction.

Since we have a contradiction in both cases, it follows that our assumption that $u_j^{h+1} < v_j^{h+1}$ must be false and thus that property (i) holds for $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$.

If $y_j^{h+1} = \alpha \in B$ we claim that $j \notin C^B$ and thus that $y_j^{h+2} \in \{\alpha, \times\}$. Assume the contrary, i.e. that $j \in C^B$. This implies that $j \notin C^\times$ and since, by assumption, $j \in B(y^{h+1})$ we must have $u_j^{h+1} + \delta < d_j T$ and $v_j^{h+1} + \delta \geq d_j T$. However, we showed

above that $y_j^{h+1} = \alpha \in \mathcal{B}$ implies that $u_j^{h+1} \geq v_j^{h+1}$ and thus we have a contradiction. Hence, property (ii) follows.

Finally, property (iii) follows from property (ii) and Proposition 6.3. Hence, we have verified that properties (i)-(iii) all hold for $h+1$, and thus the induction step goes through and the claim follows. ■

We are now in a position to state and prove that the input/output behavior of any vertex in N according to the XBD race model is consistent with the XID model.

Lemma 6.1 Let $\gamma = [y^0, u^0, v^0, t^0], [y^1, u^1, v^1, t^1], \dots$ be an arbitrary XBD race sequence for network N , when N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a at time 0. Furthermore, let

$$x(t) = \begin{cases} \hat{a} & \text{for } t < 0 \\ a & \text{for } t \geq 0 \end{cases}$$

and

$$y(t) = \begin{cases} y^0 & \text{for } t < t^0 \\ y^i & \text{for } t^i \leq t < t^{i+1} \text{ if } U(a, y^i) \neq \emptyset \text{ or for } t^i \leq t \text{ if } U(a, y^i) = \emptyset \end{cases}$$

Then the input/output behavior of every vertex j is consistent with the XID model, i.e. for each j , $Y_j(x(t), y(t))/y_j(t)$ is an acceptable input/output waveform according to the XID model.

Proof: We need to verify the five conditions for the input/output behavior given in Section 3. Consider vertex j . First, by Proposition 6.4(ii), it follows immediately that $y_j(t)$ can never change from 0 to 1 or from 1 to 0, and condition 1) is satisfied.

If $y_j(t)$ changes from a binary value α to \times at some time τ , then we must have $y_j^h = \alpha$ and $y_j^{h+1} = \times$ for some $h \geq 0$, i.e. $\tau = t^{h+1}$. However, this implies that $u_j^h + \delta \geq d_j T$, i.e. that $y_j(t)$ has had a binary value and has been unstable in the period $t^{h+1} - d_j T \leq t < t^{h+1}$. This, together with Proposition 6.4(ii), implies that $y_j(t) = \alpha$ and that $Y_j(x(t), y(t)) \neq \alpha$ for $\tau - d_j T \leq t < \tau$ and condition 2a) is satisfied. Condition 2b) can easily be verified using similar arguments.

To verify condition 3a) assume that $Y_j(x(t), y(t)) = \beta \in \mathcal{T}$ for $\tau \leq t < \tau + D_j T$ and that τ is the smallest value ≥ 0 for which this is true. Note that $\tau = t^h$ for some $h \geq 0$. If $y_j(\tilde{\tau}) = \beta$ for some $\tilde{\tau}$ such that $\tau \leq \tilde{\tau} < \tau + D_j T$, then, since the vertex is stable, it follows immediately that $y_j(t) = \beta$ for $\tilde{\tau} \leq t < \tau + D_j T$. Hence, it suffices to show that such $\tilde{\tau}$ must exist. We argue by contradiction; assume $y_j(t) \neq \beta$ for $\tau \leq t < \tau + D_j T$. Let f be the largest integer such that $t^f < \tau + D_j T$. There are two cases to consider: If $\beta = \times$ then, by Proposition 6.4(ii), $y_j(t) = \alpha \in \mathcal{B}$ for $\tau \leq t < \tau + D_j T$. This implies that vertex j has a binary value and is unstable in the race states $h, h+1, \dots, f$, and thus that $u_j^f \geq t^f - \tau$. Since $j \in U(a, y^f)$ there must exist a successor race state $[y^{f+1}, u^{f+1}, v^{f+1}, t^{f+1}]$ different from $[y^f, u^f, v^f, t^f]$. Furthermore, $\delta = t^{f+1} - t^f$ must satisfy $\delta < D_j T - u_j^f$. However, by the definition of f , we must have $t^{f+1} \geq \tau + D_j T$ and together with the fact that $u_j^f \geq t^f - \tau$ we have $\delta = t^{f+1} - t^f \geq \tau + D_j T - t^f \geq D_j T - u_j^f$, which is a contradiction. Hence consider the alternative, i.e. that $\beta \in \mathcal{B}$. If $y_j(t) \neq \beta$ for $\tau \leq t < \tau + D_j T$, we must have $v_j^f \geq t^f - \tau$. Since $j \in U(a, y^f)$ it follows, by similar arguments as in previous case, that there is a successor state $[y^{f+1}, u^{f+1}, v^{f+1}, t^{f+1}]$ different from $[y^f, u^f, v^f, t^f]$ and this state is such that $\delta = t^{f+1} - t^f$ satisfies $\delta < D_j T - v_j^f$. However, by the definition of f , it follows that $t^{f+1} \geq \tau + D_j T$ and thus that $\delta = t^{f+1} - t^f \geq \tau + D_j T - t^f \geq D_j T - v_j^f$, which is a contradiction. Since both cases lead to a contradiction, our assumption must be false and condition 3a) holds.

Finally, to verify condition 3b) assume that $Y_j(x(t), y(t)) \in \{\times, \alpha'\}$ for $\tau \leq t < \tau + D_j T$ and that τ is the smallest value ≥ 0 for which this is true. Note that $\tau = t^h$ for some $h \geq 0$. It is easy to verify that if $y_j(\tilde{\tau}) \in \{\times, \alpha'\}$ for some $\tilde{\tau}$ such that $\tau \leq \tilde{\tau} < \tau + D_j T$, then $y_j(t) \in \{\times, \alpha'\}$ for $\tilde{\tau} \leq t < \tau + D_j T$. Hence, it is sufficient to verify that a vertex cannot keep the value α for this period. Assume the contrary, i.e. that $y_j(t) = \alpha$ for $\tau \leq t < \tau + D_j T$. Let f be the largest integer such that $t^f < \tau + D_j T$. Note that vertex j has a binary value and is unstable in the race states $h, h+1, \dots, f$, and thus that $u_j^f \geq t^f - \tau$. Since $j \in U(a, y^f)$ there must exist a successor race state $[y^{f+1}, u^{f+1}, v^{f+1}, t^{f+1}]$ different from $[y^f, u^f, v^f, t^f]$. Furthermore, $\delta = t^{f+1} - t^f$ must satisfy $\delta < D_j T - u_j^f$. However, by the definition of f , we must have $t^{f+1} \geq \tau + D_j T$ and together with the fact that $u_j^f \geq t^f - \tau$ we have $\delta = t^{f+1} - t^f \geq \tau + D_j T - t^f \geq D_j T - u_j^f$,

which contradicts the condition that $\delta < D_j T - u_j^f$. Hence our assumption must be false and condition 3b) follows. Since we have verified that all the conditions for the input/output behavior according to the XID model for an arbitrary vertex j , the claim of the lemma follows. ■

The converse result is taken care of by Lemma 6.2. Let $y(t)$ denote the state of network N at time t . Assume the input/output behavior of every vertex j is consistent with the XID model, and that the input changes from \hat{a} to a at time 0. Let $\tau^0 = 0$ and τ^1, τ^2, \dots denote the time for the first, second, etc. change of the state of the network. It is trivial to verify that $\tau^1 > 0$. Define $[y^h, u^h, v^h, t^h]$, $h \geq 0$, inductively as follows:

Basis:

$$[y^0, u^0, v^0, t^0] = [y(0), (0, \dots, 0), (0, \dots, 0), 0].$$

Inductive step:

Given $[y^h, u^h, v^h, t^h]$. If $U(a, y^h) = \emptyset$ then let $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}] = [y^h, u^h, v^h, t^h]$.

Otherwise, let

$$\begin{aligned} y^{h+1} &= y(\tau^{h+1}), \\ u_j^{h+1} &= \begin{cases} \tau^{h+1} - t^h + u_j^h & \text{if } j \in U(a, y^h) \cap B(y^h) \cap U(a, y^{h+1}) \cap B(y^{h+1}) \\ 0 & \text{otherwise,} \end{cases} \\ v_j^{h+1} &= \begin{cases} \tau^{h+1} - t^h + v_j^h & \text{if } j \in U(a, y^h) \cap BE(a, y^h) \cap U(a, y^{h+1}) \cap BE(a, y^{h+1}) \\ 0 & \text{otherwise} \end{cases} \\ t^{h+1} &= \tau^{h+1}. \end{aligned}$$

Lemma 6.2 $[y^0, u^0, v^0, t^0] R_a^h [y^h, u^h, v^h, t^h]$ for all $h \geq 0$.

Proof: By induction on h .

Basis:

$h = 0$. Trivially true.

Induction hypothesis:

Assume $[y^0, u^0, v^0, t^0] R_a^h [y^h, u^h, v^h, t^h]$ for some $h \geq 0$.

Induction step:

Consider $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$. If $U(a, y^h) = \emptyset$ the claim follows trivially. Hence, assume $U(a, y^h) \neq \emptyset$. Since, by condition 1) of the XID model, we do not have any vertex changing directly from 0 to 1 or from 1 to 0, we only need to verify the following four properties:

- i) If $y_j^h = \alpha \in B$ and $y_j^{h+1} = \times$ then $Y_j(a, y^h) \neq \alpha$ and $u_j^h + (t^{h+1} - t^h) \geq d_j T$.
- ii) If $y_j^h = \times$ and $y_j^{h+1} = \alpha \in B$ then $Y_j(a, y^h) = \alpha$ and $v_j^h + (t^{h+1} - t^h) \geq d_j T$.
- iii) $u_j^h + (t^{h+1} - t^h) < D_j T$, for all $j \in U(a, y^h)$.
- iv) $v_j^h + (t^{h+1} - t^h) < D_j T$, for all $j \in U(a, y^h)$.

If $y_j^h = \alpha \in B$ and $y_j^{h+1} = \times$, i.e. the output of delay j changes from α , a binary value, to \times at time t^{h+1} , then, by condition 2a) of the XID model, we must have had $Y_j(a, y(t)) \neq \alpha$ for $t^{h+1} - d_j T \leq t < t^{h+1}$. However, this implies that

$$u_j^h + (t^{h+1} - t^h) \geq \left(t^h - (t^{h+1} - d_j T) \right) + (t^{h+1} - t^h) = d_j T,$$

and property (i) holds.

Similarly, if $y_j^h = \times$ and $y_j^{h+1} = \alpha \in B$ then, by condition 2b) of the XID model, we must have had $Y_j(a, y(t)) = \alpha$ for $t^{h+1} - d_j T \leq t < t^{h+1}$. However, this implies that

$$v_j^h + (t^{h+1} - t^h) \geq \left(t^h - (t^{h+1} - d_j T) \right) + (t^{h+1} - t^h) = d_j T,$$

and property (ii) follows.

To verify property (iii) we argue by contradiction. Assume there exist race states $[y^h, u^h, v^h, t^h]$, $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$, and a vertex $j \in U(a, y^h)$ such that $u_j^h + (t^{h+1} - t^h) \geq D_j T$. In other words, that there exists t^s such that $y_j(t) = \alpha \in B$ and $Y_j(a, y(t)) \neq \alpha$ for $t^s \leq t < t^{h+1}$ and that $t^{h+1} - t^s \geq D_j T$. However, this contradicts condition 3b) of the XID model, since $Y_j(x(t), y(t)) \neq \alpha$ for $t^s \leq t < t^s + D_j T$ implies that y_j must change at some time $\tilde{\tau}$ such that $\tilde{\tau} < t^s + D_j T \leq t^{h+1}$. Therefore, our assumption must be false and property (iii) follows.

Finally, to verify property (iv) we argue again by contradiction. Hence,

assume there exist race states $[y^h, u^h, v^h, t^h]$, $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$, and a vertex $j \in U(a, y^h)$ such that $v_j^h + (t^{h+1} - t^h) \geq D_j T$. In other words, that there exists t^s such that $Y_j(a, y(t)) = \alpha \in B$ and $y_j(t) \neq \alpha$ for $t^s \leq t < t^{h+1}$ and that $t^{h+1} - t^s \geq D_j T$. However, this contradicts condition 3a) of the XID model, since $Y_j(x(t), y(t)) = \alpha$ for $t^s \leq t < t^s + D_j T$ implies that y_j must change to α at some time $\tilde{\tau}$ such that $\tilde{\tau} < t^s + D_j T \leq t^{h+1}$. Hence, our assumption must be false and property (iv) follows.

Since we have verified properties (i)-(iv), the induction step goes through and the lemma follows. ■

In summary, the XBD race model can be used to predict the outcome of transitions caused by some change of some input variables under the assumption that the delays are extended inertial and thus bounded by lower and upper limits. The model is natural and can also be well motivated from real circuit behavior. However, the race model is computationally intractable; in fact it is continuous. In the next section we describe an efficient algorithm that computes essentially the same information.

6.4. Ternary Bounded Delay Algorithm

In this section we define an efficient algorithm, called the *ternary bounded delay (TBD)* algorithm, for simulating a circuit. We will later show that the results obtained by applying this method correspond exactly with the outcome predicted by the XBD race model. The basic idea behind the algorithm is quite simple and can be summarized in the following two rules:

- 1) change an unstable vertex to \times as soon as allowed by the minimum delay;
- 2) change a vertex from \times to a binary value as late as possible.

These rules appear to be natural. In fact, an algorithm based on similar ideas was described by Chappell and Yau [20] already in 1971. However, their algorithm was substantially more complicated than the TBD algorithm. Furthermore, they gave only a very informal motivation for the correctness of the approach, whereas we will completely characterize the results obtained from the TBD algorithm.

In the TBD algorithm it is necessary, once again, to remember a certain amount of previous excitation history. For this reason, define a *tbd-state* to be the triple $\langle z, U, V \rangle$. The first component, z , is the current “summarized” state of the network. The second component, U , is a vector of m integer values. For a stable vertex j , $U_j = 0$, whereas for an unstable vertex with a binary value, U_j denotes the current “race unit” the vertex is in. For example, in the starting state every unstable binary vertex will have $U_j = 1$, denoting that the vertex is currently in its first unstable time slot. Similarly, V is used as U , but here the criterion is that the vertex is unstable and has a binary excitation. In the TBD algorithm the summarized state of the network is computed at times $T, 2T, 3T, \dots$. However, this is a two-step process. Given the state $\langle z^{h-1}, U^{h-1}, V^{h-1} \rangle$, an intermediate state $\langle \tilde{z}^h, \tilde{U}^h, \tilde{V}^h \rangle$ is first computed. Intuitively, this state is the summarized state the network would be in at time $hT - \epsilon$ for an arbitrarily small ϵ . To compute \tilde{z}^h , only the vertices that have to change to their binary excitation are changed. These vertices are the vertices that have $V_j^{h-1} = D_j$. Due to these changes, some other vertices may become stable, and these vertices are removed from U and V respectively.

Once the intermediate tbd-state is calculated, we compute the new “next” state, i.e. the summarized state of the network at time hT . First, to obtain z^h from \tilde{z}^h , all vertices that may change to \times are changed to \times . These vertices are the vertices that have $\tilde{U}_j^h = d_j$. All other vertices are kept the same. Secondly, the vectors U and V are updated. However, this time the update consists of incrementing U_j and V_j by 1 if vertex j satisfies the conditions for U and V respectively, and resetting them to 0 otherwise. For example, a vertex j that is unstable and has a binary value in z^h will get $U_j^h = \tilde{U}_j^h + 1$, whereas a vertex that became stable in z^h will get $U_j^h = 0$.

More formally, the TBD algorithm is defined inductively as follows:

Basis:

$$\begin{aligned}
 z^0 &= b \\
 U_j^0 &= \begin{cases} 1 & \text{if } j \in U(a, z^0) \cap B(z^0) \\ 0 & \text{otherwise} \end{cases} \\
 V_j^0 &= \begin{cases} 1 & \text{if } j \in U(a, z^0) \cap BE(a, z^0) \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

Induction Step:

Given $\langle z^{h-1}, U^{h-1}, V^{h-1} \rangle$, we first compute the intermediate tbd-state $\langle \tilde{z}^h, \tilde{U}^h, \tilde{V}^h \rangle$ as follows:

$$\begin{aligned}
 \tilde{z}_j^h &= \begin{cases} Y_j(a, z^{h-1}) & \text{if } V_j^{h-1} = D_j \\ z_j^{h-1} & \text{otherwise} \end{cases} \\
 \tilde{U}_j^h &= \begin{cases} U_j^{h-1} & \text{if } j \in U(a, \tilde{z}^h) \cap B(\tilde{z}^h) \\ 0 & \text{otherwise} \end{cases} \\
 \tilde{V}_j^h &= \begin{cases} V_j^{h-1} & \text{if } j \in U(a, \tilde{z}^h) \cap BE(a, \tilde{z}^h) \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

Once this intermediate state is known, the state $\langle z^h, U^h, V^h \rangle$ is computed as follows:

$$\begin{aligned}
 z_j^h &= \begin{cases} \times & \text{if } \tilde{U}_j^h = d_j \\ \tilde{z}_j^h & \text{otherwise} \end{cases} \\
 U_j^h &= \begin{cases} \tilde{U}_j^h + 1 & \text{if } j \in U(a, z^h) \cap B(z^h) \\ 0 & \text{otherwise} \end{cases} \\
 V_j^h &= \begin{cases} \tilde{V}_j^h + 1 & \text{if } j \in U(a, z^h) \cap BE(a, z^h) \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

Consider again network N_2 of Fig. 6.8 started in the stable state $\langle 1, (1, 0, 0) \rangle$ when the input changes to $x=0$. In Fig. 6.10 we show the results of the TBD algorithm. It is easy to verify that z^3 is stable, and hence that the algorithm can be terminated at this point.

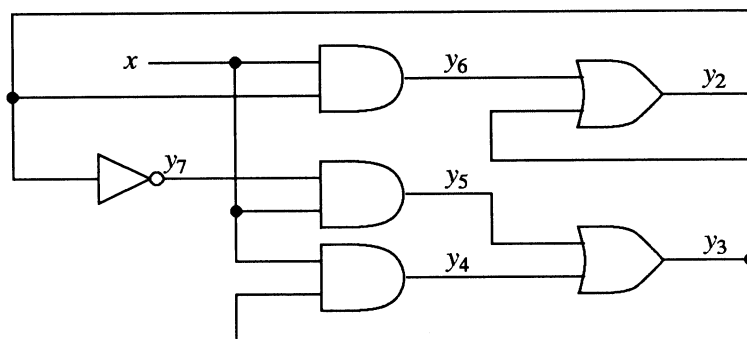
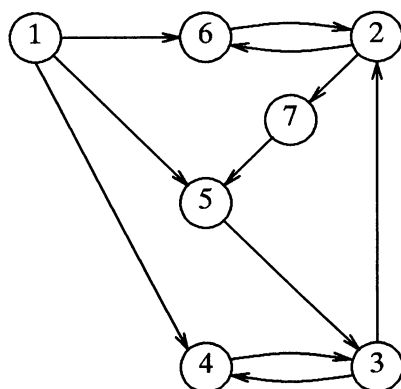
$$\begin{aligned}
 z^0 &= \langle (1_0, 0, 0), \quad (1, 0, 0), \quad (1, 0, 0) \rangle \\
 \bar{z}^1 &= \langle (1_0, 0, 0), \quad (1, 0, 0), \quad (1, 0, 0) \rangle \\
 z^1 &= \langle (\times_0, 0_\times, 0_\times), \quad (0, 1, 1), \quad (2, 0, 0) \rangle \\
 \bar{z}^2 &= \langle (\times_0, 0_\times, 0_\times), \quad (0, 1, 1), \quad (2, 0, 0) \rangle \\
 z^2 &= \langle (\times_0, \times, \times), \quad (0, 0, 0), \quad (3, 0, 0) \rangle \\
 \bar{z}^3 &= \langle (0, \times, \times), \quad (0, 0, 0), \quad (0, 0, 0) \rangle \\
 z^3 &= \langle (0, \times, \times), \quad (0, 0, 0), \quad (0, 0, 0) \rangle
 \end{aligned}$$

Figure 6.10. TBD analysis of N_2 .

Our next example is more complicated. Consider the gate circuit G_3 of Fig. 6.11. The corresponding abstract network N_3 , shown in Fig. 6.12, has the excitation functions given by:

$$\begin{aligned}
 Y_1 &= x & Y_2 &= y_3 + y_6 & Y_3 &= y_4 + y_5 & Y_4 &= y_1 y_3 \\
 & & Y_5 &= y_1 y_7 & Y_6 &= y_1 y_2 & Y_7 &= y_2'
 \end{aligned}$$

Assume that $1 \leq \Delta_i(t) < 2$, for all vertices except vertices 2 and 3, and that $2 \leq \Delta_2(t) < 3$, and $2 \leq \Delta_3(t) < 3$. If N_3 is started in the stable total state $x=0$, $y=0000001$, and the input changes to $x=1$, we get the TBD analysis shown in Fig. 6.13.

Figure 6.11. Gate circuit G_3 .Figure 6.12. Network N_3 corresponding to G_3 .

$$\begin{aligned}
z^0 &= \langle (0_1, 0, 0, 0, 0, 0, 1), & (1, 0, 0, 0, 0, 0, 0), & (1, 0, 0, 0, 0, 0, 0) \rangle \\
\bar{z}^1 &= \langle (0_1, 0, 0, 0, 0, 0, 1), & (1, 0, 0, 0, 0, 0, 0), & (1, 0, 0, 0, 0, 0, 0) \rangle \\
z^1 &= \langle (\times_1, 0, 0, 0, 0_{\times}, 0, 1), & (0, 0, 0, 0, 1, 0, 0), & (2, 0, 0, 0, 0, 0, 0) \rangle \\
\bar{z}^2 &= \langle (1, 0, 0, 0, 0_{\times}, 0, 1), & (0, 0, 0, 0, 1, 0, 0), & (0, 0, 0, 0, 0, 0, 0) \rangle \\
z^2 &= \langle (1, 0, 0_{\times}, 0, \times_1, 0, 1), & (0, 0, 1, 0, 0, 0, 0), & (0, 0, 0, 0, 1, 0, 0) \rangle \\
\bar{z}^3 &= \langle (1, 0, 0_{\times}, 0, \times_1, 0, 1), & (0, 0, 1, 0, 0, 0, 0), & (0, 0, 0, 0, 1, 0, 0) \rangle \\
z^3 &= \langle (1, 0, 0_{\times}, 0, \times_1, 0, 1), & (0, 0, 2, 0, 0, 0, 0), & (0, 0, 0, 0, 2, 0, 0) \rangle \\
\bar{z}^4 &= \langle (1, 0, 0_{\times}, 0, 1, 0, 1), & (0, 0, 2, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 0, 0) \rangle \\
z^4 &= \langle (1, 0_{\times}, \times_1, 0_{\times}, 1, 0, 1), & (0, 1, 0, 1, 0, 0, 0), & (0, 0, 1, 0, 0, 0, 0) \rangle \\
\bar{z}^5 &= \langle (1, 0_{\times}, \times_1, 0_{\times}, 1, 0, 1), & (0, 1, 0, 1, 0, 0, 0), & (0, 0, 1, 0, 0, 0, 0) \rangle \\
z^5 &= \langle (1, 0_{\times}, \times_1, \times, 1, 0, 1), & (0, 2, 0, 0, 0, 0, 0), & (0, 0, 2, 0, 0, 0, 0) \rangle \\
\bar{z}^6 &= \langle (1, 0_{\times}, \times_1, \times, 1, 0, 1), & (0, 2, 0, 0, 0, 0, 0), & (0, 0, 2, 0, 0, 0, 0) \rangle \\
z^6 &= \langle (1, \times, \times_1, \times, 1, 0_{\times}, 1_{\times}), & (0, 0, 0, 0, 0, 1, 1), & (0, 0, 3, 0, 0, 0, 0) \rangle \\
\bar{z}^7 &= \langle (1, \times, 1, \times, 1, 0_{\times}, 1_{\times}), & (0, 0, 0, 0, 0, 1, 1), & (0, 0, 0, 0, 0, 0, 0) \rangle \\
z^7 &= \langle (1, \times_1, 1, \times_1, 1_{\times}, \times, \times), & (0, 0, 0, 0, 1, 0, 0), & (0, 1, 0, 1, 0, 0, 0) \rangle \\
\bar{z}^8 &= \langle (1, \times_1, 1, \times_1, 1_{\times}, \times, \times), & (0, 0, 0, 0, 1, 0, 0), & (0, 1, 0, 1, 0, 0, 0) \rangle \\
z^8 &= \langle (1, \times_1, 1_{\times}, \times_1, \times, \times, \times), & (0, 0, 1, 0, 0, 0, 0), & (0, 2, 0, 2, 0, 0, 0) \rangle \\
\bar{z}^9 &= \langle (1, \times_1, 1_{\times}, 1, \times, \times, \times), & (0, 0, 0, 0, 0, 0, 0), & (0, 2, 0, 0, 0, 0, 0) \rangle \\
z^9 &= \langle (1, \times_1, 1, 1, \times, \times, \times), & (0, 0, 0, 0, 0, 0, 0), & (0, 3, 0, 0, 0, 0, 0) \rangle \\
\bar{z}^{10} &= \langle (1, 1, 1, 1, \times, \times, \times), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 0, 0) \rangle \\
z^{10} &= \langle (1, 1, 1, 1, \times, \times_1, \times_0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 1, 1) \rangle \\
\bar{z}^{11} &= \langle (1, 1, 1, 1, \times, \times_1, \times_0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 1, 1) \rangle \\
z^{11} &= \langle (1, 1, 1, 1, \times, \times_1, \times_0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 2, 2) \rangle \\
\bar{z}^{12} &= \langle (1, 1, 1, 1, \times, 1, 0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 0, 0) \rangle \\
z^{12} &= \langle (1, 1, 1, 1, \times_0, 1, 0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 1, 0, 0) \rangle \\
\bar{z}^{13} &= \langle (1, 1, 1, 1, \times_0, 1, 0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 1, 0, 0) \rangle \\
z^{13} &= \langle (1, 1, 1, 1, \times_0, 1, 0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 2, 0, 0) \rangle \\
\bar{z}^{14} &= \langle (1, 1, 1, 1, 0, 1, 0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 0, 0) \rangle \\
z^{14} &= \langle (1, 1, 1, 1, 0, 1, 0), & (0, 0, 0, 0, 0, 0, 0), & (0, 0, 0, 0, 0, 0, 0) \rangle
\end{aligned}$$

Figure 6.13. TBD analysis of N_3 .

6.5. Correspondence between XBD Model and TBD Algorithm

In this section we will give a characterization of the results obtained by the TBD algorithm. Assume N is started in the stable total state $\langle \hat{a}, b \rangle$ and that the input changes to a . Let z^0, z^1, z^2, \dots be the sequence of states computed by the TBD algorithm and $Reach(t)$ be the outcome according to the XBD race model. The following theorem shows that the TBD algorithm can be used to get essentially the same information as the XBD race model.

Theorem 6.1 $z^r = l.u.b. Reach(rT)$ for $r = 0, 1, \dots$.

We will prove this result with the aid of two lemmas. Lemma 3 states that z^r “covers” all possible states the network can be in at time rT according to the XBD race model. Lemma 4 shows that the sequence of states generated by the TBD algorithm in itself is a valid XBD race sequence.

Lemma 6.3 $z^r \supseteq l.u.b. Reach(rT)$ for $r = 0, 1, \dots$

Proof: We will prove something slightly stronger. Let $\gamma = [y^0, u^0, v^0, t^0], [y^1, u^1, v^1, t^1], \dots$ be an arbitrary XBD race sequence for network N , when N is started in the stable total state $\langle \hat{a}, b \rangle$ and the input changes to a at time 0. Define, for $i = 0, 1, \dots$,

$$y(t) = \begin{cases} y^0 & \text{for } t < t^0 \\ y^i & \text{for } t^i \leq t < t^{i+1} \text{ if } U(a, y^i) \neq \emptyset \text{ or for } t^i \leq t \text{ if } U(a, y^i) = \emptyset. \end{cases}$$

Also, for $r = 1, 2, \dots$, let

$$\epsilon^r = \min \left\{ \frac{T}{10}, \frac{rT - \max\{t^i : [y^i, u^i, v^i, t^i] \in \gamma \text{ and } t^i < rT\}}{2} \right\}$$

$$z(t) = \begin{cases} z^{r-1} & \text{for } (r-1)T \leq t < rT - \epsilon^r \\ \bar{z}^r & \text{for } rT - \epsilon^r \leq t < rT \end{cases}$$

Note that ϵ^r and $z(t)$ are defined in such a way that the change in $z(t)$ from z^{r-1} to \bar{z}^r is guaranteed to occur in the open interval $(r-1)T - rT$, and *after* the last change of $y(t)$ in this interval. (The constant 10 in the definition of ϵ^r is chosen somewhat arbitrarily, and the reader can easily verify that any number greater than 1 could be used.) We will show that $z(t) \supseteq y(t)$ for all $t \geq 0$. Since $z(rT) = z^r$, and γ is an arbitrary

XBD race sequence, the claim in the lemma then follows immediately.

Let $\tau^0 = 0$ and τ^h , $h \geq 1$, denote the time for *event* h , where a change in $z(t)$ and/or a change in $y(t)$ is considered an event[†]. It clearly suffices to show that $z(\tau^h) \sqsupseteq y(\tau^h)$ for all $h \geq 0$, since, by definition, neither z or y changes between two consecutive events. We will prove that $z(\tau^h) \sqsupseteq y(\tau^h)$ by induction on h .

Basis:

Since $\tau^0 = 0$ and $z(0) = y(0) = b$, the claim is trivially true.

Induction hypothesis:

Assume that $z(\tau^{\bar{h}}) \sqsupseteq y(\tau^{\bar{h}})$, $0 \leq \bar{h} \leq h$ for some $h \geq 0$.

Induction step:

Consider vertex j at event $h+1$. There are two cases to handle: either $z_j(\tau^h) \neq z_j(\tau^{h+1})$ or $z_j(\tau^h) = z_j(\tau^{h+1})$. Consider first the case when z_j is changing at step $h+1$, i.e. when $z_j(\tau^h) \neq z_j(\tau^{h+1})$. If $z_j(\tau^h) = \alpha \in B$ and $z_j(\tau^{h+1}) = \times$, it follows trivially that $z_j(\tau^{h+1}) \sqsupseteq y_j(\tau^{h+1})$. Note that this holds irrespectively of whether y_j changes or not. On the other hand, if z_j changes from \times to a binary value, we claim the following:

$$z_j(\tau^h) = \times \text{ and } z_j(\tau^{h+1}) = \alpha \in B \text{ implies } y_j(\tau^h) = y_j(\tau^{h+1}) = \alpha.$$

The only place in the TBD algorithm where the value of a vertex can change from \times to a binary value is in computing an intermediate state. Hence, we can conclude that $\tau^{h+1} = qT - \epsilon^q$ for some non-negative integer q , $q \geq D_j$. By construction, ϵ^q is such that y does not change at time $qT - \epsilon^q$, i.e. $y(\tau^{h+1}) = y(\tau^h)$. Furthermore, $z_j^{q-1} = \times$ and $\bar{z}_j^q = \alpha$ imply that $V_j^{q-1} = D_j$. This implies that $Y_j(a, z^p) = \alpha$ for $q - D_j \leq p \leq q - 1$ and that $Y_j(a, \bar{z}^{\bar{p}}) = \alpha$ for $q - D_j < \bar{p} \leq q - 1$ and thus that $Y_j(a, z(\tau^i)) = \alpha$ for all τ^i such that $(q - D_j)T \leq \tau^i < \tau^{h+1}$. By the induction hypothesis we have that $z(\tau^i) \sqsupseteq y(\tau^i)$ for all τ^i such that $(q - D_j)T \leq \tau^i < \tau^{h+1}$, and therefore, by the monotonicity of Y , that $\alpha = Y_j(a, z(\tau^i)) \sqsupseteq Y_j(a, y(\tau^i))$ for all τ^i

[†] Note that more than one vertex can change value at the same event.

such that $(q-D_j)T \leq \tau^i < \tau^{h+1}$. Hence, it follows that $Y_j(a, y(\tau^i)) = \alpha$ for all τ^i such that $(q-D_j)T \leq \tau^i < \tau^{h+1}$. We now claim that this implies that $y_j(\tau^h) = \alpha$. Suppose not, i.e. suppose that $y_j(\tau^h) \neq \alpha$. Since $Y_j(a, y(\tau^i)) = \alpha$ for all τ^i such that $(q-D_j)T \leq \tau^i \leq \tau^h$ this assumption implies that $y_j(\tau^i) \neq \alpha$ for all τ^i such that $(q-D_j)T \leq \tau^i \leq \tau^h$. Furthermore, since $y(\tau^h) = y(\tau^{h+1})$, as shown above, and thus $Y(a, y(\tau^h)) = Y(a, y(\tau^{h+1}))$, the earliest possible time for y_j to change or for the excitation of vertex j to change is at τ^{h+2} . By the definition of ϵ^q , it follows that $\tau^{h+2} \geq qT$. However, this implies that y_j is unstable with the same binary excitation, α , for at least $\tau^{h+2} - (q-D_j)T \geq D_jT$ time — a result that contradict the definition of the XBD model. Thus, our assumption that $y_j(\tau^h) \neq \alpha$ must be false. Hence, $z_j(\tau^h) = \times$ and $z_j(\tau^{h+1}) = \alpha$ implies that $y_j(\tau^{h+1}) = \alpha$. In summary, if z_j changes at time τ^{h+1} , then $z_j(\tau^{h+1}) \supseteq y_j(\tau^{h+1})$.

Now consider the case when z_j does not change at time τ^{h+1} . If y_j does not change either it follows immediately, by the induction hypothesis, that $z_j(\tau^{h+1}) \supseteq y_j(\tau^{h+1})$. Hence, consider the case when y_j changes at time τ^{h+1} . There are two sub-cases to consider: when y_j changes from \times to a binary value, and when y_j changes from a binary value to \times .

(i) $y_j(\tau^h) = \times$ and $y_j(\tau^{h+1}) = \alpha \in B$:

If $y_j(\tau^h) = \times$ it follows by the induction hypothesis that $z_j(\tau^h) = \times$. However, since, by assumption, z_j does not change at time τ^{h+1} , it follows that $z_j(\tau^{h+1}) = \times$, and thus that $z_j(\tau^{h+1}) \not\supseteq y_j(\tau^{h+1})$.

(ii) $y_j(\tau^h) = \alpha \in B$ and $y_j(\tau^{h+1}) = \times$:

Since z_j does not change at time τ^{h+1} we need to show that $z_j(\tau^h) = \times$. We prove this by contradiction. Suppose that $z_j(\tau^h) \neq \times$. By the induction hypothesis we have that $z_j(\tau^h) \supseteq y_j(\tau^h)$. Since $y_j(\tau^h) = \alpha$ it follows that $z_j(\tau^h)$ is either equal to \times or α . However, we assumed $z_j(\tau^h) \neq \times$ and thus $z_j(\tau^h) = \alpha$. Since y_j changes to \times at time τ^{h+1} it follows, by Proposition 6.1 and the definition of the XBD race model, that there must exists a $\tau^s \geq 0$ such that:

- 1) $\tau^{h+1} - \tau^s \geq d_j T$,
- 2) $y_j(\tau^i) = \alpha$ for $s \leq i \leq h$, and
- 3) $Y_j(a, y(\tau^i)) \neq \alpha$ for $s \leq i \leq h$.

Property 3, together with the induction hypothesis and the monotonicity of Y , implies that $Y_j(a, z(\tau^i)) \neq \alpha$ for $s \leq i \leq h$. This, together with the fact that $z_j(\tau^h) = \alpha$ implies that $z_j(\tau^i) = \alpha$ for $s \leq i \leq h$. Now define the non-negative integers p and q as follows:

$$pT \leq \tau^s < (p+1)T \quad \text{and} \quad qT \leq \tau^{h+1} < (q+1)T$$

Note that $(q-p)T = \tau^{h+1} - \tau^s + ((\tau^s - pT) - (\tau^{h+1} - qT))$, and therefore, by property 1, $(q-p)T \geq d_j T + ((\tau^s - pT) - (\tau^{h+1} - qT))$. However, by the definition of p and q we have $0 \leq \tau^s - pT < T$ and $0 \leq \tau^{h+1} - qT < T$, and thus $-T < (\tau^s - pT) - (\tau^{h+1} - qT) < T$. This, together with the fact that p, q , and d_j are non-negative integers, implies that $q - p \geq d_j$. Furthermore, note that $\tau^s < (p+1)T - \epsilon^{p+1}$, by the definition of ϵ^{p+1} . Hence, since z does not change between pT and $(p+1)T - \epsilon^{p+1}$, it follows that $z^p = z(\tau^s)$, and therefore $z_j^p = \alpha$ and $Y_j(a, z^p) \neq \alpha$. Altogether, we have that $z_j^r = \alpha$ and $Y_j(a, z^r) \neq \alpha$ for $p \leq r \leq q$ and $\bar{z}_j^{\bar{r}} = \alpha$ and $Y_j(a, \bar{z}^{\bar{r}}) \neq \alpha$ for $p < \bar{r} \leq q$. However, this contradicts the definition of the TBD algorithm, since we would have $\tilde{U}_j^q = d_j$ and thus $z_j^q = \times$. Hence, our assumption that $z_j(\tau^h) \neq \times$ must be false.

In summary, if z_j does not change at time τ^{h+1} then $z_j(\tau^{h+1}) \supseteq y_j(\tau^{h+1})$.

Since we have showed that for any vertex j , $z_j(\tau^{h+1}) \supseteq y_j(\tau^{h+1})$, the induction step goes through and the lemma follows. ■

In the next lemma we show the converse, by establishing that the change sequence generated by the TBD algorithm is in fact a valid XBD race sequence.

Assume once again that N is started in the stable total state $\langle \hat{a}, b \rangle$ and that the input changes to a . Let $z^0, \bar{z}^1, z^1, \dots$ be the sequence of states computed by the TBD algorithm.

Lemma 6.4 $z^r \sqsubseteq l.u.b. \text{Reach}(rT)$ for $r = 0, 1, \dots$

Proof: We prove this by constructing an XBD race sequence that corresponds to the sequence of states generated by the TBD algorithm and which is equal to z^r at time rT , for $r = 0, 1, \dots$. Let $y(t)$ be defined as follows:

$$y(t) = \begin{cases} z^{i-1} & \text{for } (i-1)T \leq t < (i - \frac{i}{i+1})T \\ z^i & \text{for } (i - \frac{i}{i+1})T \leq t < iT \end{cases}$$

where $i = 1, 2, \dots$. Let $\tau^0 = 0$ and τ^h , $h \geq 1$, denote the time for event h , where a change in $y(t)$ is considered an event. Note that τ^h is either equal to pT or equal to $(p - \frac{p}{p+1})T$ for some non-negative integer p . Now define $[y^h, u^h, v^h, t^h]$ inductively as follows:

Basis:

$$[y^0, u^0, v^0, t^0] = [y(0), (0, \dots, 0), (0, \dots, 0), 0].$$

Inductive step:

Given $[y^h, u^h, v^h, t^h]$. If $U(a, y^h) = \emptyset$ then let $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}] = [y^h, u^h, v^h, t^h]$.

Otherwise, let

$$\begin{aligned} y^{h+1} &= y(\tau^{h+1}), \\ u_j^{h+1} &= \begin{cases} \tau^{h+1} - t^h + u_j^h & \text{if } j \in U(a, y^h) \cap B(y^h) \cap U(a, y^{h+1}) \cap B(y^{h+1}) \\ 0 & \text{otherwise,} \end{cases} \\ v_j^{h+1} &= \begin{cases} \tau^{h+1} - t^h + v_j^h & \text{if } j \in U(a, y^h) \cap BE(a, y^h) \cap U(a, y^{h+1}) \cap BE(a, y^{h+1}) \\ 0 & \text{otherwise} \end{cases} \\ t^{h+1} &= \tau^{h+1}. \end{aligned}$$

We now claim that $[y^0, u^0, v^0, t^0], [y^1, u^1, v^1, t^1], \dots$ constitutes a valid XBD race sequence, i.e. that $[y^0, u^0, v^0, t^0] R_a^h [y^h, u^h, v^h, t^h]$ for any $h \geq 0$. We will prove this by induction on h .

Basis:

$h = 0$: follows trivially by the definition of R_a^0 .

Induction hypothesis:

Assume that $[y^0, u^0, v^0, t^0] R_a^h [y^h, u^h, v^h, t^h]$ for some $h \geq 0$.

Induction step:

Consider $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$. If $U(a, y^h) = \emptyset$ then the claim follows trivially. Otherwise, since it is trivial to verify that no vertex can change directly from 0 to 1 or from 1 to 0, we need to verify the following properties only:

- 1) If $y_j^h = \alpha \in B$ and $y_j^{h+1} = \times$ then $Y_j(a, y^h) \neq \alpha$ and $u_j^h + (t^{h+1} - t^h) \geq d_j T$.
- 2) If $y_j^h = \times$ and $y_j^{h+1} = \alpha \in B$ then $Y_j(a, y^h) = \alpha$ and $v_j^h + (t^{h+1} - t^h) \geq d_j T$.
- 3) $u_j^h + (t^{h+1} - t^h) < D_j T$, for all $j \in U(a, y^h)$.
- 4) $v_j^h + (t^{h+1} - t^h) < D_j T$, for all $j \in U(a, y^h)$.

If vertex j changes from $\alpha \in B$ to \times it follows, by the definition of the TBD algorithm, that $t^{h+1} = qT$ for some non-negative integer q . Hence, it follows that $\tilde{U}_j^q = d_j$, and thus that $z_j^p = \alpha$ and $Y_j(a, z^p) \neq \alpha$ for $q - d_j \leq p < q$, and that $\tilde{z}_j^{\tilde{p}} = \alpha$ and $Y_j(a, \tilde{z}^{\tilde{p}}) \neq \alpha$ for $q - d_j < \tilde{p} \leq q$. However, this implies that $y_j(t) = \alpha$ and $Y_j(a, y(t)) \neq \alpha$ for $(q - d_j)T \leq t < t^{h+1} = qT$ and therefore that

$$u_j^h + (t^{h+1} - t^h) \geq \left[t^h - (qT - d_j T) \right] + (t^{h+1} - t^h) = d_j T.$$

Hence property 1) holds.

If vertex j changes from \times to $\alpha \in B$ it follows from the definition of the TBD algorithm that $t^{h+1} = (q - \frac{q}{q+1})T$ for some non-negative integer q . Furthermore we can conclude that $V_j^{q-1} = D_j$, and thus that $z_j^p \neq \alpha$ and $Y_j(a, z^p) = \alpha$ for $q - D_j \leq p < q$, and that $\tilde{z}_j^{\tilde{p}} \neq \alpha$ and $Y_j(a, \tilde{z}^{\tilde{p}}) = \alpha$ for $q - D_j < \tilde{p} < q$. This implies that $y_j(t) \neq \alpha$ and $Y_j(a, y(t)) = \alpha$ for $(q - D_j)T \leq t < t^{h+1} = (q - \frac{q}{q+1})T$. However, since $D_j \geq d_j + 1$, we can conclude that

$$\begin{aligned}
v_j^h + (t^{h+1} - t^h) &\geq \left(t^h - (q - D_j)T \right) + (t^{h+1} - t^h) = \\
&= \left(q - \frac{q}{q+1} \right)T - (q - D_j)T = \\
&= \left(D_j - \frac{q}{q+1} \right)T \geq d_j T.
\end{aligned}$$

Hence, property 2) is satisfied.

To verify property 3) we argue by contradiction. Assume there exists a vertex $j \in U(a, y^h)$ such that $u_j^h + (t^{h+1} - t^h) \geq D_j T$. In other words, assume there exists a vertex j and time $t^s \geq 0$ such that $y_j(t) = \alpha$ and $Y_j(a, y(t)) \neq \alpha$ for $t^s \leq t < t^{h+1}$, and $t^{h+1} - t^s \geq D_j T$. Assume also, without loss of generality, that t^s is the smallest value for which this is true. Let the integer p , $p \geq 0$, be defined by

$$(p-1)T < t^s \leq pT.$$

Since, by assumption, $y_j(t) = \alpha$ and $Y_j(a, y(t)) \neq \alpha$ for $t^s \leq t < t^{h+1}$, we can immediately conclude that $U_j^p = 1$. Furthermore, since $t^{h+1} - t^s \geq D_j T$ and $D_j \geq d_j + 1$ it follows that $(p + d_j)T < t^s + T + d_j T \leq t^s + D_j T = t^{h+1} - ((t^{h+1} - t^s) - D_j T) \leq t^{h+1}$, i.e. $(p + d_j)T < t^{h+1}$. However, this implies that $\tilde{U}_j^{p+d_j} = d_j$ and therefore $z_j^{p+d_j} = \times$, i.e. that $y_j((p + d_j)T) = \times$, contradicting our assumption that $y_j(t) = \alpha$ for $t^s \leq t < t^{h+1}$. Hence, such a vertex cannot exist and property 3) holds.

To verify property 4) we argue again by contradiction. Assume there exists a vertex $j \in U(a, y^h)$ such that $v_j^h + (t^{h+1} - t^h) \geq D_j T$. In other words, assume there exists a vertex j and time $t^s \geq 0$ such that $y_j(t) \neq \alpha$ and $Y_j(a, y(t)) = \alpha$ for $t^s \leq t < t^{h+1}$, and $t^{h+1} - t^s \geq D_j T$. Assume also, without loss of generality, that t^s is the smallest value for which this is true. Let the integer p , $p \geq 0$, be defined by

$$(p-1)T < t^s \leq pT.$$

Since, by assumption, $y_j(t) \neq \alpha$ and $Y_j(a, y(t)) = \alpha$ for $t^s \leq t < t^{h+1}$, we can immediately conclude that $V_j^p = 1$ and therefore that $V_j^{p+D_j-1} = D_j$. (The latter follows from the fact that we must have $t^{h+1} > (p + D_j - 1)T$, since $t^{h+1} - t^s \geq D_j T$.) Let $\tau = (p + D_j - \frac{p+D_j}{p+D_j+1})T$. Now, by the definition of $y(t)$, it follows that t^s is

either equal to pT or $(p - \frac{p}{p+1})T$ and thus

$$\tau - t^s \leq \left(p + D_j - \frac{p + D_j}{p + D_j + 1} \right) T - \left(p - \frac{p}{p+1} \right) T = \left(D_j - \frac{D_j}{(p + D_j + 1)(p + 1)} \right) T < D_j T$$

Altogether, $t^s < \tau < t^{h+1}$. However, since, $V_j^{p+D_j-1} = D_j$, it follows, by the definition of the TBD algorithm, that $\tilde{z}_j^{p+D_j} = \alpha$, and thus that $y_j(\tau) = \alpha$. However, this contradicts our assumption that $y_j(t) \neq \alpha$ for $t^s \leq t < t^{h+1}$. Hence, our assumption must be false and property 4) follows.

Since we have shown that properties 1-4 hold for $[y^{h+1}, u^{h+1}, v^{h+1}, t^{h+1}]$, the induction step goes through and the lemma follows. ■

To illustrate the use of Theorem 6.1, consider network N_3 of Fig. 6.12, and its analysis in Fig. 6.13. Because the TBD algorithm reaches a stable binary state after 14 steps, we can conclude that the network will take at most 14 basic time units to reach a unique stable binary state. Furthermore, by Theorem 6.1, we can conclude that the network may take this much time to reach this state. (Analyzing the transition assuming minimum delays shows that this state can be reached as early as at $8 + \epsilon$ basic time units, where $\epsilon > 0$ can be arbitrarily small.)

6.6. Discussion

Although the TBD algorithm, as described in Section 5, is very efficient compared to the XBD race model, one can easily improve its performance. The formulation of the algorithm was chosen to simplify the characterization of the results it produces, rather than to provide an efficient implementation. The TBD algorithm now involves a substantial amount of redundant work. In particular, when the circuit is “waiting” for a minimum delay time to elapse, and only U_j and V_j values change, a naive implementation of the algorithm would waste much time computing excitation function values that cannot have changed. We will not present a more efficient algorithm here, but using an event-scheduling approach with two distinct queues (one for “ \times -events” and one for “ β -events”) it is quite straightforward to derive an algorithm that can be used on even very large circuits. As is usual in simulators, the running time for such a modified TBD algorithm would depend

heavily on both the circuit being simulated and on the efficiency of the code.

Chapter VII

The Complexity of Race Analysis

There are two natural questions to ask when analyzing a transition caused by an input change:

- a) will the network eventually end up in a unique (binary) stable state, and
- b) will the network be in a unique (binary) stable state at time t ?

The first question will be called the *stable-state reachability* question, whereas the second will be called the *limited reachability* question. In this chapter the inherent complexity of these problems is studied.

7.1. Stable-State Reachability Problem

Assume N is a network that is started in the stable total state $\langle \hat{a}, b \rangle$. The stable-state reachability (SSR) problem is: Will the network eventually end up in a unique stable binary state if the input changes to a at time 0 and is kept at that value?

We make the following assumption about the excitation functions:

Assumption 7.1 The excitation function Y_j , $1 \leq j \leq m$, can be computed in time $O(m^k)$ for some constant $k > 0$.

This assumption seems to be quite natural; if it does not hold, even the problem of determining if a given state is unstable is intractable. In fact, for many types of circuits it is reasonable to assume that the excitation functions can be computed in constant time. (Gate circuits with bounded fan-in provide a typical example.)

Previously, Ramachandran [46] studied the “dual” of the SSR problem — the (critical) race detection problem. The race detection problem can be formulated as follows: Given a circuit started in a stable total state, a new input vector, and a vertex j , can the circuit reach two distinct stable states in which the values on node j are different? Ramachandran showed that the race detection problem in acyclic transistor switch-level circuits is NP-complete. However, this result stems more from the rather strange switch-level model she uses than the inherent difficulty of the problem. In particular, in her switch-level model a node that gets partially discharged (charged) will either remain in the state 1(0) or change to 0(1). A more realistic assumption would also allow the node to take on the value \times . If we do allow such a transition, then the race detection problem in acyclic transistor switch-level circuits becomes solvable in polynomial time.

In the following we study the complexity of the SSR problem for different race models. We assume the reader is familiar with the standard terminology, as described by Garey and Johnson [28]. We start with perhaps the simplest model — the unit-delay model. For this race model, we have the following result:

Theorem 7.1 The SSR problem is PSPACE-complete for the unit-delay race model.

Proof: For the unit-delay model, SSR is clearly in PSPACE, since we can simulate the circuit and use a counter to count the number of state changes. If the circuit has not reached a stable state within $3^m - 1$ steps, it must have entered an oscillation, and thus will never reach a stable state. Since the simulation only requires enough space to compute the excitation functions, i.e. space polynomial in m , and the counter only needs space linear in m , it follows that the UD-SSR problem is in PSPACE.

To prove the claim, it is therefore sufficient to show that a known PSPACE-complete problem can be reduced to the UD-SSR problem. The Quantified Boolean Formula (QBF) problem turns out to be appropriate. Following [31] a QBF is defined as follows:

- 1) If x is a variable, then it is a QBF. The occurrence of x is free.

- 2) If E_1 and E_2 are QBF's, so are $\neg(E_1)$, $(E_1) \wedge (E_2)$, and $(E_1) \vee (E_2)$. An occurrence of x is free or bound, depending on whether the occurrence is free or bound in E_1 or E_2 . Redundant parentheses can be omitted.
- 3) If E is a QBF, then $\exists x(E)$ and $\forall x(E)$ are QBF's. The scopes of $\exists x$ and $\forall x$ are all the free occurrences of x in E . Free occurrences of x in E are bound in $\exists x(E)$ and $\forall x(E)$. All other occurrences of variables in $\forall x(E)$ and $\exists x(E)$ are free or bound, depending on whether they are free or bound in E .

A QBF with no free variables has a value of either *true* or *false*, denoted by 1 and 0 respectively. The value of such a QBF is obtained by replacing each subexpression of the form $\exists x(E)$ by $E^0 \vee E^1$ and each subexpression of the form $\forall x(E)$ by $E^0 \wedge E^1$, where E^0 and E^1 are E with all the occurrences of x in the scope of the quantifier replaced by 0 and 1 respectively. The *QBF problem* is to determine whether a QBF with no free variables has the value *true*. Stockmeyer [31, 54] showed in 1974 that the QBF problem is PSPACE-complete.

Given a QBF E with no free variables, we will design a circuit with one input; this circuit, if started in a stable state (to be defined) and with the input changing from 0 to 1, will reach a new stable state iff E is *true*. If E is *false*, the circuit will oscillate. The basic idea is to design a sequential circuit that evaluates E . However, we must make sure that the size of this circuit, and thus the size of the UD-SSR problem, remains polynomial in the size of the QBF problem. Hence, we cannot simply take two copies of a circuit that evaluates E_1 , use them to evaluate E_1 for $x=0$ and $x=1$, and OR the results together to compute $\exists x(E_1)$, since this could generate a circuit of size exponential in the size of E_1 . However, we can design a small control circuit that first evaluates E_1 for $x=0$, stores the result, and then evaluates E_1 for $x=1$ and combines the two results together. This is the basic idea of the construction; however, the construction we will present is more complicated and far from minimal in size. We do this in order to simplify the correctness proof.

Before we describe how to transform E into N , we present our basic building block — a D flip-flop with completion signals and an asynchronous reset input. The circuit is shown in Fig. 7.1; it is an ordinary D flip-flop with reset input, but with

Assume that E is a given QBF with no free variables, and let N denote the circuit to be constructed. We first define a subcircuit \tilde{N} recursively as follows:

If $E = x$ then \tilde{N} is the circuit shown in Fig. 7.2.

Assume the circuits \tilde{N}_1 and \tilde{N}_2 have been constructed corresponding to the QBFs $E_1(x)$ and $E_2(x)$ respectively.

- 1) If $E = \neg(E_1)$ then \tilde{N} is the circuit shown in Fig. 7.3.
- 2) If $E = (E_1) \vee (E_2)$ then \tilde{N} is the circuit shown in Fig. 7.4. Similarly, if $E = (E_1) \wedge (E_2)$ then \tilde{N} is the circuit obtained by replacing the *-marked OR gate in the circuit shown in Fig. 7.4 with an AND gate.

- 3) If $E = \exists x(E_1)$ then \tilde{N} is the circuit shown in Fig. 7.5. Similarly, if $E = \forall x(E_1)$ then \tilde{N} is the circuit obtained by replacing the *-marked OR gate in the circuit shown in Fig. 7.5 with an AND gate.

Finally, if \tilde{N} is the circuit that corresponds to E , then N is the circuit shown in Fig. 7.6.

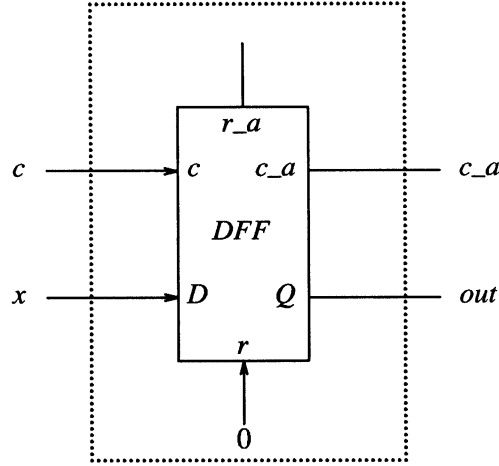


Figure 7.2. Construction for a variable x .

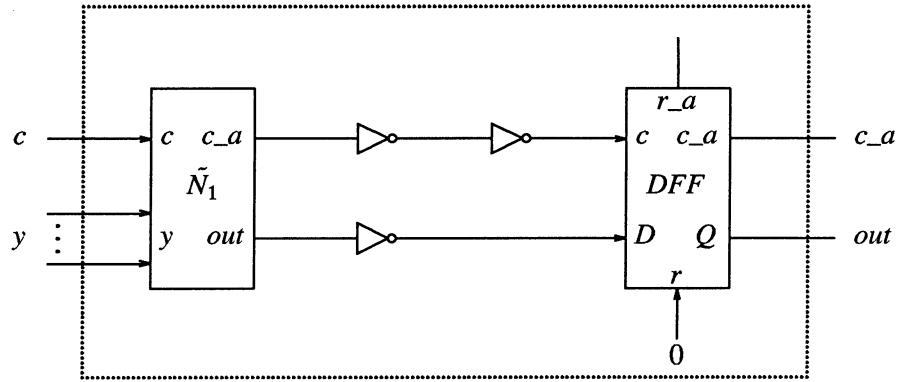
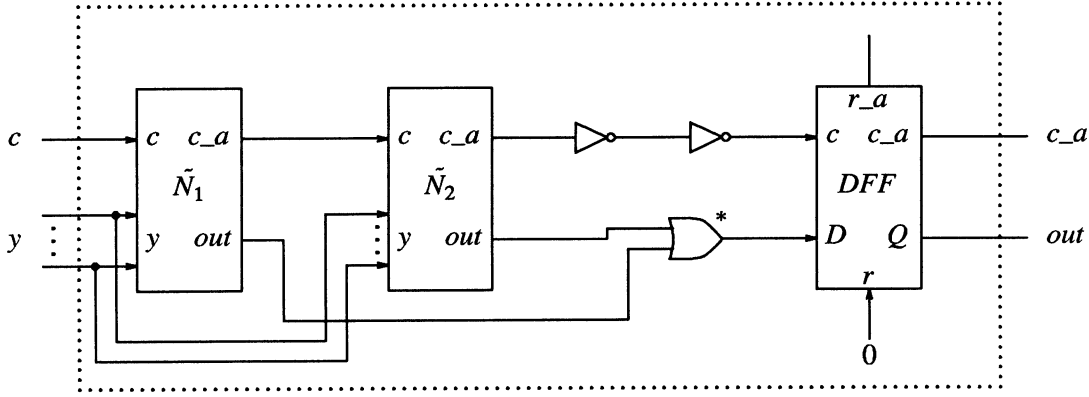
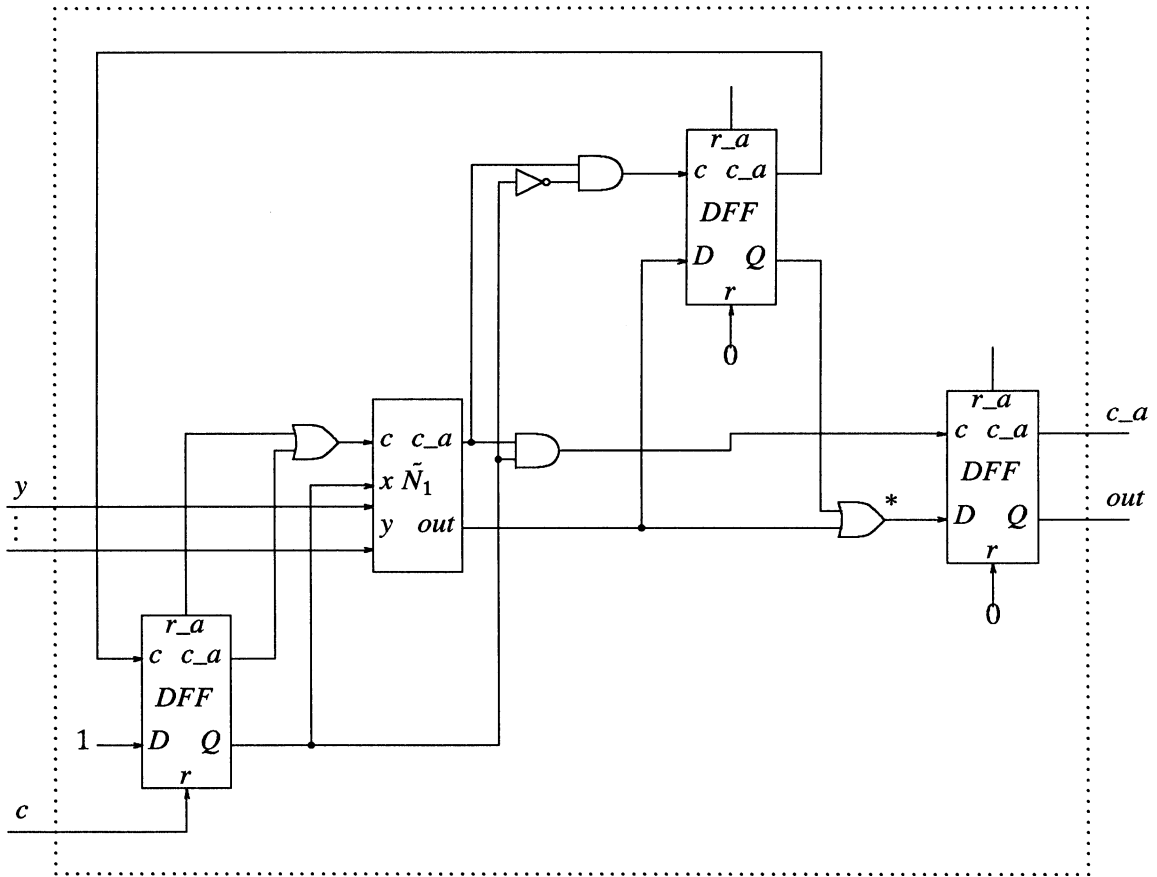


Figure 7.3. Construction for $\neg E_1(y)$.

Figure 7.4. Construction for $E_1(y) \vee E_2(y)$.

The main idea of the construction of N is to separate the “control” path from the “data” path. Furthermore, we will make sure that only one pulse propagates through the control path of the circuit, and that no signal from the data path can affect the control path. The “combinational” parts of E are rather obvious, and are shown in Figs. 7.2-7.4. For example, the circuit that corresponds to a subexpression $\neg E_1(y)$, shown in Fig. 7.3, works as follows: First, when the pulse arrives at the “start” input (c), it is immediately sent to the start input of the subcircuit that evaluates $E_1(y)$. When the completion signal arrives from this subcircuit, the pulse is delayed two units of time in order to allow the inversion of the result in the data path. Finally, this inverted result is latched in the D flip-flop. When this is done, the pulse arrives at the completion output (c_a) of the complete circuit. The other combinational circuits can be verified using similar arguments.

The circuits corresponding to the expressions $\exists x[E_1(x,y)]$ and $\forall x[E_1(x,y)]$ are more complicated. We will only discuss the circuit, shown in Fig. 7.5, that evaluates $\exists x[E_1(x,y)]$. The circuit for $\forall x[E_1(x,y)]$ works similarly. The basic idea is as follows: We want the circuit to evaluate $E_1(0,y)$, store the result, evaluate $E_1(1,y)$, OR the two results together and latch the result. This is accomplished as follows: The left-most D flip-flop serves as a “status register” keeping track of the current value of x . The top-most D flip-flop is used to remember the result of $E_1(0,y)$, and the right-most D flip-flop is used to latch the final result. When the pulse arrives at the start

Figure 7.5. Construction for $\exists x[E_1(x, y)]$.

input (c), the status register is reset. This causes the value of x to be set to 0. Once the status register has been reset, the completion signal (r_a) is fed via the left-most OR gate to the start input (c) of the subcircuit \tilde{N}_1 evaluating $E_1(x, y)$. When this result becomes available (i.e. when the pulse arrives at the c_a output), the result of $E(0, y)$ is latched in the top-most D flip-flop. This follows because when x is 0, the completion output (c_a) from \tilde{N}_1 is redirected to the store input of the top-most D flip-flop. After the value has been latched in the top-most D flip-flop, the pulse goes back to the status register. Since the D input of the status register is connected to 1, the output (Q) will change to 1, causing x to become 1. Now, when the output of the latch (Q) has obtained the value 1, the completion signal (c_a) is fed via the left-most OR gate to the c input of \tilde{N}_1 . However, this time $x = 1$, and thus \tilde{N}_1 will

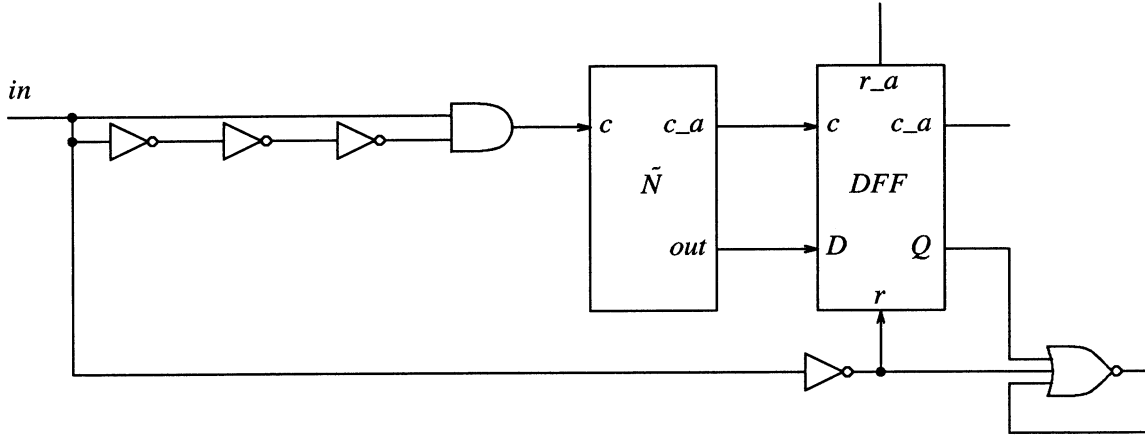


Figure 7.6. Complete construction.

evaluate $E_1(1, y)$. When the result of $E_1(1, y)$ is available, it is combined with the stored value of $E_1(0, y)$ and latched in the right-most D flip-flop. When this is done, the completion signal is used to signal to the remaining circuit that the result of $\exists x[E_1(x, y)]$ is available at the output. It is straightforward to convince oneself that the circuit of Fig. 7.5 behaves as described if the unit delay model is assumed. We leave the details to the interested reader.

The figures above have shown how to convert a QBF E with no free variables into a sequential circuit \tilde{N} . Note that for each operator in the QBF we need only a constant number of gates. In fact, it is easy to verify that for a QBF of length r , we get a circuit with at most $44r$ gates. However, we are not quite done yet. The final part of the circuit, shown in Fig. 7.6, must be put “around” \tilde{N} in order to correctly introduce the pulse into \tilde{N} and also to interpret the final result. The idea is to use a NOR gate feeding back to itself as an oscillator. The only way we can reach a stable state in the complete circuit, if the input changes from 0 to 1, is if the final latch gets the value 1. However, this can happen iff the QBF E is *true*.

Our final task is to assign the starting state to this circuit. This is simply the state in which all the flip-flops are storing the value 0, and the input in is 0. The reader can easily verify that this total state is stable.

In summary, the construction described above takes an arbitrary QBF E of size r , with no free variables, and yields a gate circuit with at most $44r + 19$ gates. If this circuit is started in the stable total state defined above, and the input changes from 0 to 1, then the circuit will eventually end up in a stable state iff E is *true*. Hence, if we can solve the UD-SSR problem in polynomial time, then we can solve all problems in PSPACE in polynomial time. This, together with the fact that the UD-SSR problem is in PSPACE, implies that it is PSPACE complete. ■

We consider the AED model next. Unfortunately, we do not know how to efficiently compute the outcome of a single race unit according to the AED model. (The corresponding TAED algorithm is sometimes overly pessimistic.) Hence, we can only state a lower bound on the complexity of the AED-SSR problem. We have the following result:

Theorem 7.2 The AED-SSR problem is PSPACE-hard.

Proof: To show this, we transform the UD-SSR problem into the AED-SSR problem. The claim then follows by Theorem 7.1. This transformation can be accomplished by adding delay elements in every wire of the circuit. Note that this can only increase the size of the circuit by $O(m^2)$, and thus the new circuit is still of size polynomial in the size of the original circuit. It is easy to convince oneself that only the wire signals can change in every odd race unit, and that only the gate signals can change in every even race unit. This implies that all the races in the circuit will be non-critical, since there are delays “isolating” the gates from each other, and gates isolating the wires from each other. Thus the outcome of every race unit will be the state in which all unstable states change to their excitation, i.e. the state the unit-delay model would predict. Hence, the sequence of states obtained by removing the odd states in the sequence of race states is the same as that which a unit-delay model would produce. Therefore, this transformed circuit will eventually reach a unique binary stable state according to the AED model iff the original circuit will eventually reach a unique binary stable state according to the unit-delay model. Thus AED-SSR is PSPACE hard. ■

In Chapter V we mentioned the possibility of redefining the AED model and creating an XAED model, which allows changes to go through the \times value. If this is done, and the conjectured correspondence between the (somewhat modified) TAED algorithm and this XAED model does indeed hold, we can then also obtain an upper bound on the (X)AED-SSR problem. We get the following result:

Theorem 7.3 If Conjecture 5.1 holds then the XAED-SSR problem is PSPACE-complete.

Proof: Since we assume that the modified TAED algorithm can be used to compute the “next-states” according to the XAED model, the XAED-SSR problem must be in PSPACE. To see this, consider the following algorithm. Use the modified TAED algorithm to simulate the behavior of the network, but also keep a counter to count how many race units have been computed. If we reach a stable state within $3^m - 1$ steps and this state is binary, it follows from the properties of the modified TAED algorithm that the answer to the XAED-SSR is YES, and that otherwise it is NO. This computation can be done in space polynomial in the size of the circuit; thus the XAED-SSR problem is in PSPACE.

To prove that the XAED-SSR problem is PSPACE complete use the same arguments as in the proof of Theorem 7.2; this is left as an exercise to the reader. ■

The next race model to consider in the “hierarchy” of race models developed in this thesis, is the extended bounded delay model. Here we need to distinguish between two cases: the general case in which the maximum delay may grow arbitrarily fast with the size of the circuit, and the more realistic case in which the maximum delay does not depend on the size of the circuit (or grows slowly with the size of the circuit.) For the first case, called the general XBD-SSR problem, we have the following result:

Theorem 7.4 If the maximum delay in the extended bounded delay model can grow arbitrarily fast with the size of the circuit, then the XBD-SSR problem is PSPACE-hard.

Proof: We will show how to transform the UD-SSR problem into the general XBD-SSR problem. In view of Theorem 7.1 the result then follows immediately. Let N be the original network, containing m vertices, for which we try to answer the UD-SSR question. Transform N to \tilde{N} by adding delay elements in every wire. This will increase the circuit size by at most $O(m^2)$ vertices. Now, let the delay in every vertex be bounded by $(D-1)T \leq \Delta_j(t) < DT$, for $D = 2 \cdot 3^m + 2$. We now argue that the answer to the XBD-SSR problem for \tilde{N} is YES iff the answer to the UD-SSR problem for N is YES. First, since all delays have the same delay bounds in \tilde{N} (and thus can be exactly the same) and there is a delay in every wire, it follows trivially that, if there is an oscillation in the UD analysis of N , then the same oscillation must exist in a XBD analysis of \tilde{N} . Hence, if the answer to the UD-SSR question for N is NO, then the answer will be NO also for the XBD-SSR question for \tilde{N} . On the other hand, if the answer to the UD-SSR question for N is YES, then the sequence of states computed by a UD analysis of this transition must be of length less than or equal to 3^m (and the last state must be stable). This implies that a UD analysis of \tilde{N} would also reach the same stable state, but that the length of the sequence would be $2 \cdot 3^m$ (every odd state corresponding to an input or function vertex change, and every even state corresponding to a wire vertex change). On the other hand, it is easy to convince oneself that a XBD analysis of \tilde{N} yields the same result as a UD analysis of \tilde{N} as long as we do not carry out the analysis for too many steps. In fact, the two correspond exactly for r steps if the time to complete $r+1$ of the fastest possible transitions is strictly greater than the time to complete r of the slowest possible transitions. In other words, the UD analysis and the XBD analysis of \tilde{N} correspond exactly for r steps as long as $(r+1)(D-1)T > rDT$, i.e. as long as $r < D-1$. However, since $D = 2 \cdot 3^m + 2$, we can conclude that a XBD analysis of \tilde{N} corresponds exactly to a UD analysis of \tilde{N} if the length of the UD sequence is at most $2 \cdot 3^m$. Altogether, we have shown that if the answer to the UD-SSR question for N is YES, then so is the answer to the XBD-SSR question for \tilde{N} . ■

The assumption that the maximum delay can grow exponentially fast with the size of the network was crucial in the proof of Theorem 7.3. However, it is a very unrealistic assumption. Hence, a more interesting question is the complexity of the XBD-SSR problem when the maximum delay is constant, or polynomial in the size

of the circuit. Unfortunately, the complexity of this restricted XBD-SSR problem is still an open problem.

We started by studying the SSR problem for the race model in which we have the most accurate delay knowledge (in fact, the unit-delay model implies that the sizes of the delays are known exactly). We will finish this study by looking at the other extreme: the case in which the delays can be arbitrary but finite, i.e. the SSR problem for the XMW model. For the XMW model, we have the following result:

Theorem 7.5 The XMW-SSR problem is solvable in polynomial time.

Proof: To determine the answer to the XMW-SSR problem, it is sufficient to study the result y^B of the ternary simulation algorithm described in Chapter III, Section 2. If this result is binary then, by Theorem 3.2, it follows that the circuit must eventually end up in this state according to an XMW analysis. On the other hand, if y^B is not binary, then, again by Theorem 3.2, we can conclude that the circuit will not end up in a unique binary state and thus may oscillate, may have a critical race or may eventually end up in a state in which at least one component has the value \times . The polynomial time result follows from the fact that the ternary simulation algorithm produces, in the worst case, a sequence of $2m$ states, each requiring at most m excitation function evaluations. This together with Assumption 7.1 gives the required result. ■

In [13] it was shown that ternary simulation also corresponds to a GMW analysis of a gate network under the assumption that both the gates and the wires can have arbitrary, but finite, delays associated with them. Using this result and the same arguments as in the proof of Theorem 7.5, we immediately get the following result:

Theorem 7.6 If both the wires and the gates have delays, the GMW-SSR problem is solvable in polynomial time.

It is interesting to compare this result with the complexity of the GMW-SSR problem when only the gates are assumed to have delays. We have the following result:

Theorem 7.7 The GMW-SSR problem is NP-hard if only the gates have delays.

Proof: To simplify the notation in this proof we use the convention that “the GMW-SSR problem” refers to the case when only the gates have delays. We will show how to transform the Boolean tautology problem to the GMW-SSR problem. The Boolean *non-tautology* problem is defined as follows: Given a Boolean expression E over the set $\{x_1, \dots, x_n\}$, using the connectives \neg , \vee , and \wedge , is E *not* a tautology, i.e. is there a truth assignment for the variables that makes E false. Cook [21] showed in 1971 that the non-tautology problem is NP-complete. This result implies that the Boolean tautology problem, defined dually, is NP-hard. Hence, to prove that the GMW-SSR problem is NP-hard, it is sufficient to show how to transform the Boolean tautology problem to the GMW-SSR problem.

Given any Boolean expression E over the set $\{x_1, \dots, x_n\}$ of variables, using the connectives \neg , \vee , and \wedge , we will show how to construct a circuit N with one input. This circuit is such that if it is started in a stable state (to be defined) and the input changes from 0 to 1, the outcome of this transition, according to a GMW analysis, consists of a single state iff E is a tautology. The basic idea is quite similar to the construction in [46]. We use a “race generating” circuit, shown in Fig. 7.7, for every input variable x_i . In Fig. 7.8 we show a GMW analysis of the race generating circuit when it is started in the stable state $y = (y_1, \dots, y_4) = 1000$, $kill = 0$ and in changes from 0 to 1. There are two important properties of the circuit: first that there are two stable states reachable (0010 and 0000), and second that when the NOR gate changes from 0 to 1, all other gates in the circuit are stable.

Assume E is a given Boolean expression. We first construct a subcircuit \tilde{N} defined recursively as follows:

Basis:

If $E = x_i$ then \tilde{N} is the (trivial) circuit shown in Fig. 7.9.

Induction step:

Assume the circuits \tilde{N}_1 and \tilde{N}_2 have been constructed corresponding to the Boolean expressions $E_1(x)$ and $E_2(x)$ respectively.

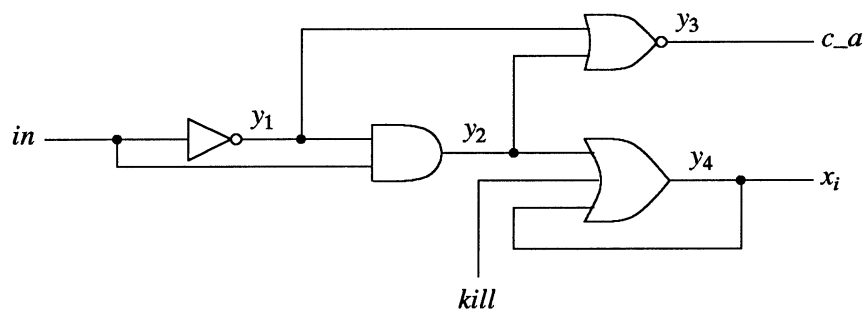


Figure 7.7. Critical race generating circuit.

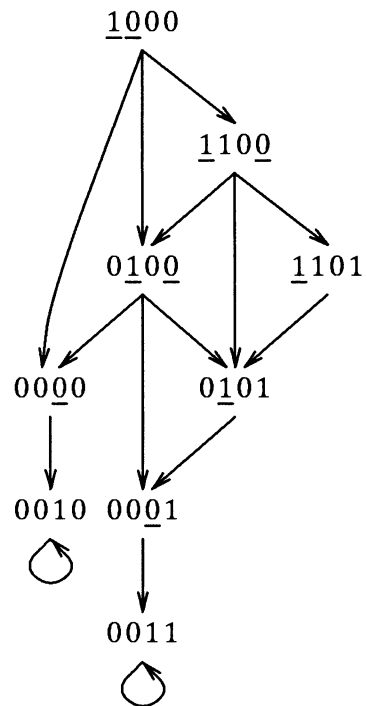
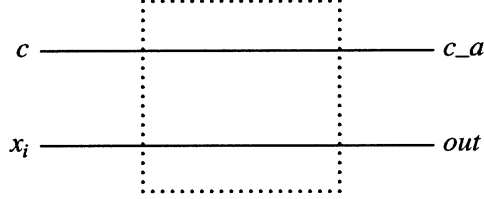
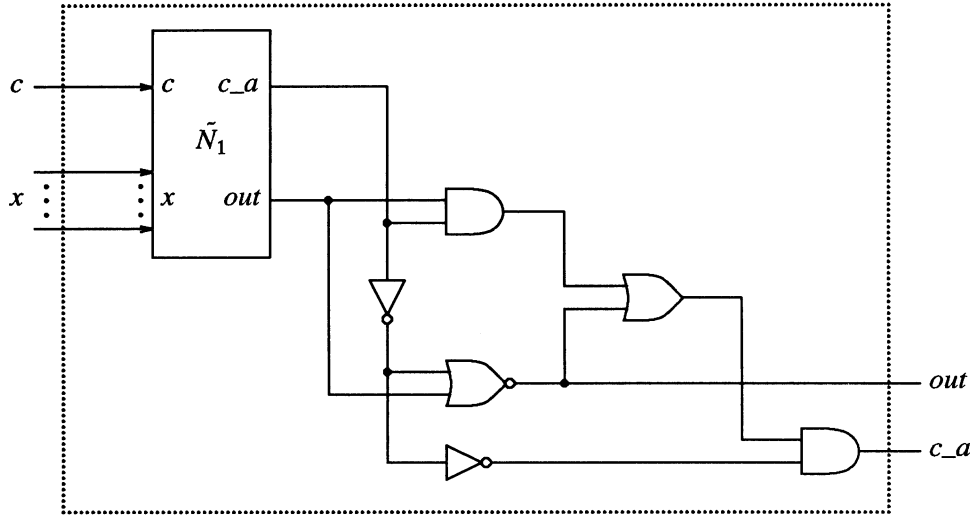
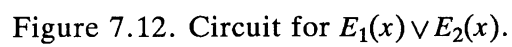
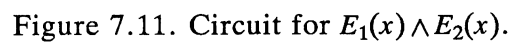


Figure 7.8. GMW analysis of the critical race generating circuit.

- 1) If $E = \neg E_1(x)$ then \tilde{N} is the circuit shown in Fig. 7.10.
- 2) If $E = E_1(x) \wedge E_2(x)$ then \tilde{N} is the circuit shown in Fig. 7.11.
- 3) If $E = E_1(x) \vee E_2(x)$ then \tilde{N} is the circuit shown in Fig. 7.12.

Figure 7.9. Circuit for $E = x_i$.Figure 7.10. Circuit for $\neg E_1(x)$.

The circuit \tilde{N} have n inputs, labeled x_1, \dots, x_n , a “compute” input c , an output out , and a “compute-acknowledge” output c_a . \tilde{N} will be started in the stable state implied by $c = 0$ and c will change to 1 at some point in time. We claim that \tilde{N} satisfies the following two properties: The output out of \tilde{N} has the value $E(x)$ when the c_a output changes from 0 to 1, and neither out nor c_a will change again if the inputs are held fixed. We prove this claim by induction.



Basis:

It is trivial to verify that the circuit of Fig. 7.9 satisfies both properties.

Induction step:

Assume the claim holds for the circuits \tilde{N}_1 and \tilde{N}_2 corresponding to the Boolean expressions $E_1(x)$ and $E_2(x)$ respectively.

- 1) If $E = \neg E_1(x)$ we need to verify the claim for the circuit shown in Fig. 7.10. Note that if the c_a output of \tilde{N}_1 is 0, then the outputs of the gates shown in Fig. 7.10 are uniquely determined. Hence, by the induction hypothesis, it is sufficient to perform a GMW analysis for the gates shown in Fig. 7.10 for different values of out from \tilde{N}_1 when the c_a signal of \tilde{N}_1 changes from 0 to 1. There are two cases to consider: when out of \tilde{N}_1 is equal to 1 and when it is equal to 0.

Suppose out of \tilde{N}_1 is 1. This implies that the NOR gate has the value 0 and that it will stay stable no matter what other gates change. However, this implies that out of \tilde{N} is 0, i.e. $\neg E_1(x)$. Also, since the only way c_a of \tilde{N} can change from 0 to 1 is when the c_a signal from \tilde{N}_1 has propagated through both inverters, through the top-most AND gate, through the OR gate, and through the last AND gate, it follows trivially that the c_a signal changes only once, and thus the claim holds.

On the other hand, consider the case when out from \tilde{N}_1 is 0. In this case the top-most AND gate will be stable (having the value 0). However, since the only way c_a of \tilde{N} can change from 0 to 1 is when the c_a signal from \tilde{N}_1 has propagated through both inverters, through the NOR gate, through the OR gate, and through the final AND gate, it follows trivially that out of \tilde{N} will have the value 1 (i.e. $\neg E(x)$) before c_a of \tilde{N} changes and that the c_a signal changes only once. Hence the claim follows.

- 2) If $E = E_1(x) \wedge E_2(x)$ we need to verify the claim for the circuit shown in Fig. 7.11. The verification of the claim follows easily from the following observations. In order for c_a of \tilde{N} to change from 0 to 1, the c_a signals from \tilde{N}_1 and \tilde{N}_2 must both have changed to 1, the signals must have propagated

through the topmost AND gate, and through both inverters. If at least one of the *out* outputs of \tilde{N}_1 and \tilde{N}_2 is 0, then the three-input AND gate must be stable having the value 0 (i.e. $E_1(x) \wedge E_2(x)$). On the other hand, if both *out* outputs are 1, then to get *c_a* of \tilde{N} to become 1 requires that the output of the three-input AND gate changes to 1 first. Hence, it is straightforward to verify the claim. We leave the details to the reader.

- 3) If $E = E_1(x) \vee E_2(x)$ we need to verify the claim for the circuit shown in Fig. 7.12. This circuit can be verified using similar arguments as in case 3). We leave the details to the interested reader.

In summary, \tilde{N} is a circuit that evaluates $E(x)$ when the compute signal c goes high. When the correct value is available on the *out* output, the *c_a* output goes high. Note that \tilde{N} works correctly no matter what the gate delays are (as long as they are finite).

Finally, in Fig. 7.13, we show how to use n race generating circuits and \tilde{N} to construct N . The basic idea is as follows. The circuit is started in the stable state determined by $in = 0$ and all OR gates with self-loops having the value 0. We have one race generating circuit for each input variable to E . When in changes from 0 to 1, each of these race generating circuits will settle down with either the value 0 or 1 on their output x_i . Since the *c_a* output of all these race generating circuits are connected via a chain of AND gates, it follows that the c input to \tilde{N} will not change to 1 until all the race generating circuits have reached a stable state. There are two cases to consider: If $E(x)$ is a tautology, then no matter what state each input variable gets, the output *out* of \tilde{N} will be 1 when *c_a* goes high on \tilde{N} . However, this implies that the last AND gate will (eventually) change to 1 causing the rightmost OR gate also (eventually) to change to 1. Once the OR gate changes to 1, it will be stable. When the output of this OR gate goes high, this will (eventually) cause the oscillation in the NAND gate to stop. Furthermore, it will set all the input variables to 1 (since the *kill* signal causes the OR gates with self-loop in the race generating circuits to change to 1). However, this will uniquely determine the values of all gates in \tilde{N} and the rightmost AND gate. Hence, if $E(x)$ is a tautology, then the circuit N will eventually end up in a unique stable state.

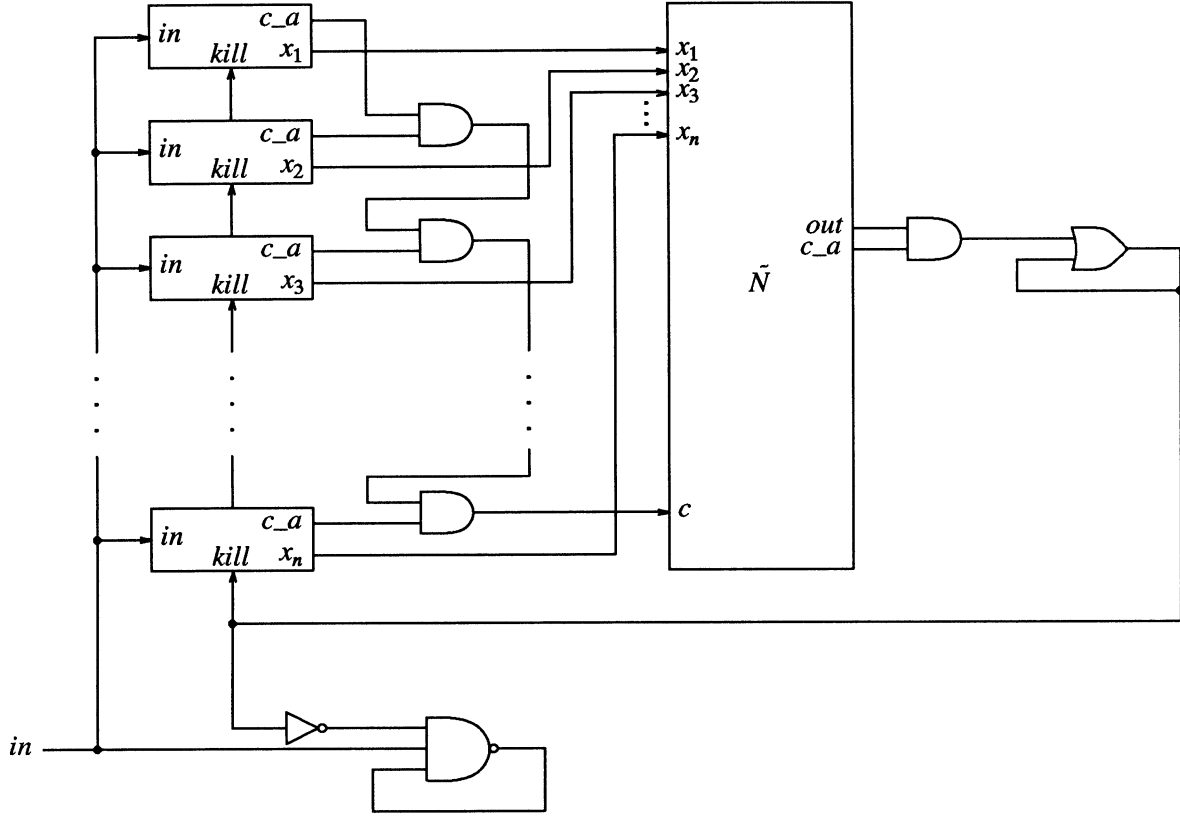


Figure 7.13. Complete circuit.

On the other hand, assume $E(x)$ is *not* a tautology. Thus for some Boolean vector $\alpha \in B^n$, $E(\alpha)$ will be false. Since each race generating circuit can reach the state 0 or 1 independently, there must exist some delay assignment such that $x_1 = \alpha_1$, $x_2 = \alpha_2, \dots, x_n = \alpha_n$. For this delay assignment, the result on *out* from \tilde{N} will be 0 when c_a changes from 0 to 1 on \tilde{N} . However, this implies that the rightmost AND gate will never change and consequently neither will the rightmost OR gate. This implies that the circuit will never reach a stable state (since the NAND gate will oscillate). In summary, if $E(x)$ is not a tautology, then the outcome of a GMW analysis of N contains more than one state.

Since the circuit N contains $O(|E|)$ gates, where $|E|$ denotes the length of the Boolean expression, E is a tautology iff the outcome of a GMW analysis of N contains a single state, and the Boolean tautology problem is NP-hard, it follows that the

GMW-SSR problem is NP-hard. ■

In Fig. 7.14 we summarize the results of this section. The following observations can be made: When the delay in each component is known exactly or almost exactly, the SSR problem is intractable (assuming $PSPACE \neq PTIME$, of course). On the other hand, when the delays can be arbitrary and changes can go through \times (i.e. when using the XMW model), the SSR problem can be solved very efficiently. It is interesting to note that for the GMW model, the difficulty of the SSR problem depends on whether the wires and the gates or only the gates have delays. In the former case the GMW-SSR problem is solvable in polynomial time, whereas in the latter case it is NP-hard. For “intermediate” race models, like the restricted XBD model, the complexity of the SSR problem is still open.

Race model	Complexity
UD	PSPACE-complete
AED	PSPACE-hard
XAED	PSPACE-complete ¹
XBD	PSPACE-hard
restricted XBD	unknown
XMW	polynomial time
GMW ²	polynomial time
GMW ³	NP-hard
1 assuming Conjecture 5.1 is true.	
2 assuming both gate and wire delays.	
3 assuming gate delays only.	

Figure 7.14. Complexity of the SSR problem for different race models.

The results of this section are rather negative, showing that the SSR problem is intractable for many realistic race and delay models. In the next section we study the more practical problem of solving the SSR problem when we also impose a condition on how long we are willing to wait before the circuit reaches a stable state.

7.2. Limited Reachability Problem

The limited reachability (LR) problem is the question whether a network will reach a unique binary stable state within time $r(m)T$, for some function $r(m)$ and some basic time unit T . Of course, if r is exponential in m , the LR problem degenerates into essentially the same problem as the SSR problem. Thus we will henceforth assume that $r(m)$ is $O(m^k)$ for some constant $k > 0$. In fact, we will sometimes restrict $r(m)$ even more.

Since the delays in the vertices can be arbitrarily large (or small) in the XMW and the GMW models, it follows that the limited reachability problem for these models is not very interesting. In fact, the answer is always NO (since the input vertices can change at any time before or after $r(m)T$).

For the UD model, the XBD model, and, if Conjecture 5.1 holds, the XAED model we have the following result:

Theorem 7.8 The limited reachability problem is solvable in polynomial time for a) the UD model, b) the XAED model, and c) the XBD model.

Proof: For the UD model, simulate the circuit for r steps. Each step requires m excitation-function evaluations, each requiring at most time polynomial in m . Since r is assumed to be bounded by some polynomial in m , claim a) follows immediately.

For the XAED model, carry out r steps of the modified TAED algorithm. Since we assumed that Conjecture 5.1 holds, the result of the TAED algorithm is the *l.u.b.* of all the states the network can be in after r complete race units. If this result is a binary stable state, the answer to the LR problem is YES, otherwise it is NO. Each step of the TAED algorithm requires at most $2m$ excitation function evaluations plus some book-keeping. Since r is assumed to be bounded by some polynomial in m , claim b) follows.

For the XBD model, carry out r steps of the TBD algorithm. By Theorem 6.1 the result of the TBD algorithm is equal to the *l.u.b.* of all the states the network can be in at time rT . If this result is a binary stable state, the answer to the LR problem is YES, otherwise it is NO. Each step of the TBD algorithm requires at most $2m$ excitation function evaluations plus some book-keeping. Since r is assumed to be bounded by some polynomial in m , claim c) follows. ■

It is interesting to compare these results with more classical binary race models. We will focus on the AED model, but it is easy to extend the results to be presented to a “binary bounded delay” model.

For the AED model the complexity of the LR problem depends on $r(m)$ and on whether the indegree of a function vertex is bounded or not. First, for indegree-bounded networks and for constant r , our first result shows that the LR problem for the AED model is solvable in time polynomial in m . However, the result is more of theoretical interest than practical, since the only known algorithm to compute the result is *exponential* in the indegree and r .

Theorem 7.9 If the maximum indegree of a function vertex is e , for some constant e , and $r(m) = r_0$ for some constant r_0 , then the LR problem for the AED model is solvable in polynomial time.

Proof: The crucial observation is that an input change can propagate through at most r_0 vertices when each vertex has a delay of approximately T units of time and we want to know the value of a vertex after r_0T time. Hence, to determine whether a vertex i will have a unique binary value and be stable at time r_0T , it is sufficient to consider the values on the vertices that are within a distance of r_0 vertices from the input of vertex i . However, we assumed a constant maximum indegree of the function vertices. This implies that there are at most $\frac{e^{r_0+1}-1}{e-1}$ vertices that can affect the value and excitation of vertex i at time r_0T . In other words, to determine vertex i 's value and excitation at time r_0T it is sufficient to perform an AED analysis on a sub-network containing a constant number of vertices (although the constant is normally quite large). Let C denote this number of vertices. Now, since the AED analysis requires at most $O(2^C)$ excitation function evaluations, C is a constant, and each

excitation function can be evaluated in time polynomial in m , this analysis can be done in time polynomial in m . Thus, determining the value and excitation of vertex i at time r_0T can be done in time polynomial in m . Using this procedure for each vertex in N immediately gives the answer to the LR problem; if each vertex reaches a unique stable state then the answer to the LR problem is YES, otherwise it is NO. We leave the details of the proof to the interested reader. ■

In the following theorems we study the complexity of the LR problem for the AED model when we use more liberal conditions on the maximum indegree of a function vertex or on the number of race units we are willing to wait. Our first result concerns the case when we allow a function vertex to have an unbounded indegree.

Theorem 7.10 If the maximum indegree of a function vertex is $\Omega(m)$ and $r(m)$ is $\Omega(1)$, the LR problem is NP-hard for the AED model.

Proof: We will transform the 3-Satisfiability problem into the LR problem. In fact, given a Boolean formula E in 3-conjunctive normal form (i.e. the formula is a product of clauses, where each clause contains three literals) we will show how to construct a network N with one input. If N is started in a stable total state (to be defined) and the input changes from 0 to 1, then an AED analysis of this transition, carried out for at least 18 race units, will indicate that N will reach a single stable state iff E is not satisfiable. In other words, the answer to the LR problem for N is YES iff E is not satisfiable. Since the size of N will be polynomial in the size of E , and the 3-Satisfiability problem is NP-complete [21], it follows that the LR problem is NP-hard.

The construction of N is very similar to the construction in the proof of Theorem 7.7. We use the same race generating circuit as was shown in Fig. 7.7. The reader can easily verify that if in changes from 0 to 1, then according to the AED model the race generating circuit can only reach the stable states 0010 or 0000, and in either case, the value of x_i is stable before the c_a signal changes from 0 to 1.

Let E be a given Boolean expression in 3-conjunctive normal form. We first construct a subcircuit \tilde{N} that evaluates the complement of E . In Fig. 7.15 we outline how such a circuit can be constructed. First the inverters compute \bar{l}_i , then the OR gates compute the values of the clauses, the multi-input AND gate computes the product of all the clauses, and finally the topmost inverter complements the result. The row of inverters at the bottom is used to delay the compute signal c so that the c_a signal cannot change from 0 to 1 until the value of $\neg E(x)$ is available on out .

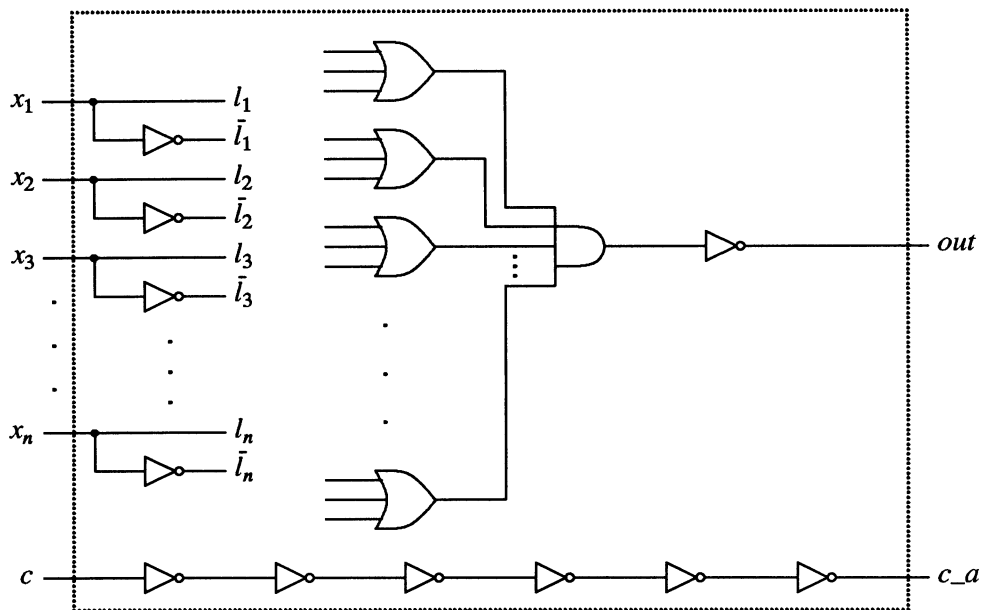


Figure 7.15. Construction of \tilde{N} .

Finally, in Fig. 7.16 we show how to use n race generating circuits and \tilde{N} to construct N . The basic idea is as follows. The circuit is started in the stable state determined by $in = 0$ and all OR gates with self-loops having the value 0. We have one race generating circuit for each input variable to E . When in changes from 0 to 1, each of these race generating circuits will settle down with either the value 0 or 1 on their output. Since the c_a outputs of all these race generating circuits are connected to an AND gate, it follows that the c input to \tilde{N} will not change to 1 until all the race generating circuits have reached a stable state. There are two cases to consider: If $E(x)$ is not satisfiable, then no matter what state each input variable gets,

the output out of \tilde{N} will be 1 when c_a goes high on \tilde{N} . However, this implies that the last AND gate will change to 1 causing the rightmost OR gate also to change to 1. Once the OR gate changes to 1, it will be stable. When the output of this OR gate goes high, this will cause the oscillation in the NAND gate to stop. Furthermore, it will set all the input variables to 1 (since the $kill$ signal causes the OR gates with self-loop in the race generating circuits to change to 1). However, this will uniquely determine the values of all gates in \tilde{N} and the rightmost AND gate. Hence, if $E(x)$ is not satisfiable, then the circuit N will end up in a unique stable state. In fact, it is easy to convince oneself that N will take at most 18 race units to reach this stable state.

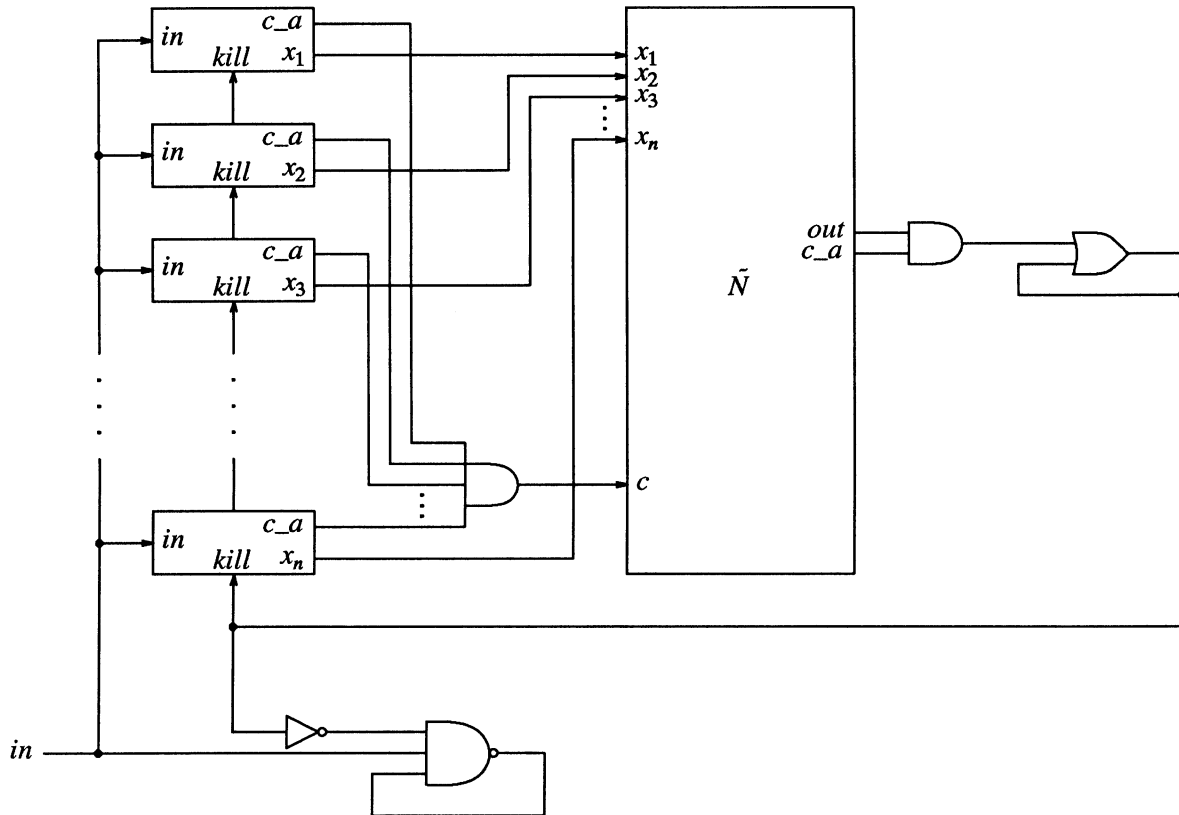


Figure 7.16. Complete circuit.

On the other hand, if $E(x)$ is satisfiable there must exist some Boolean vector $\alpha \in B^n$, such that $E(\alpha)$ is true. Since each race generating circuit can reach the state 0 or 1 independently, there must exist some delay assignment such that $x_1 = \alpha_1$, $x_2 = \alpha_2, \dots, x_n = \alpha_n$. For this delay assignment, the result on *out* from \tilde{N} will be 0 when c_a changes from 0 to 1 on \tilde{N} . However, this implies that the rightmost AND gate will never change and consequently neither will the rightmost OR gate. This implies that the circuit will never reach a stable state (since the NAND gate will oscillate). In summary, if $E(x)$ is not satisfiable, then the outcome of an AED analysis carried out for at least 18 steps contains more than one state.

Since the circuit N contains $O(|E|)$ gates, where $|E|$ denotes the length of the Boolean expression, E is not satisfiable iff the outcome of an AED analysis of N , carried out for at least 18 race units, contains a single state, and the 3-Satisfiability problem is NP-complete, it follows that the LR problem for the AED model is NP-hard. ■

The unbounded indegree assumption above is hardly very realistic. However, it allowed us to compute the AND of $O(m)$ signals in one race unit. A more realistic assumption is that the indegree is bounded by some relatively small constant. However, it is then quite reasonable to allow a circuit $\Omega(\log m)$ steps to settle down. Consider, for example, a one-output combinational circuit consisting of m vertices. Such a circuit may take $O(\log m)$ race units to settle down. If we assume bounded indegree and allow a (relatively slowly) growing $r(m)$ we get the following result:

Theorem 7.11 If the maximum indegree of a function vertex is e , for some constant e , and r is $\Omega(\log m)$, the limited reachability problem is NP-hard for the AED model.

Proof: The proof is similar to the proof of Theorem 7.10, except that the big AND gates are replaced with trees of e -inputs AND gates. Since the height of these trees are bounded by $O(\log m)$, it is easy to verify that if E is not satisfiable, then the circuit N will reach a unique stable state within $O(\log m)$ steps. On the other hand, if E is satisfiable, then N can reach an oscillation. We leave the details to the interested reader. ■

In Fig. 7.17 we summarize the results of this section. The following observations can be made. The LR problem is solvable in time polynomial in m for the UD model, the XAED model, and the XBD model. On the other hand, for binary race models, even the LR problem is almost always intractable.

Race model	Complexity
UD	polynomial time
AED ¹	polynomial time
AED ²	NP-hard
AED ³	NP-hard
XAED	polynomial time ⁴
XBD	polynomial time
GMW	trivial
XMW	trivial
1 assuming bounded indegree and $r(m) = r_0$. 2 assuming unbounded indegree and that $r(m)$ is $\Omega(1)$. 3 assuming bounded indegree and that $r(m)$ is $\Omega(\log m)$. 4 assuming Conjecture 5.1 is true.	

Figure 7.17. Complexity of the LR problem for different race models.

Chapter VIII

Conclusions and Future Research

The main results of the thesis are first summarized in this chapter. Some remaining open problems concerning the analysis of asynchronous circuits are also mentioned. Finally, some more long-term research topics that appear to be promising in view of the results of this thesis are pointed out.

8.1. Analysis Problem

The analysis problem for asynchronous circuits is difficult because the behavior depends very much on the delays in the circuit components. These delays may be unknown and may vary in time. Even in circuits with a small number of components, there is a large number of possible cases to consider.

In this thesis, we have developed models and practical algorithms for analyzing the behavior of asynchronous circuits. In particular, efficient algorithms were developed for determining the possible behavior of a circuit for several delay models. These delay models included ones in which the delays are completely unknown, bounded between upper and lower limits, or “almost equal”. Using the algorithms developed here we can now efficiently analyze the behavior of a circuit under a wide range of delay assumptions.

Although we believe that these results represent considerable progress, some problems still remain. In particular, it would be very interesting to incorporate the different race analysis algorithms described in the thesis in existing simulators, and test them on real circuits. This would both provide practical feedback on the

usefulness of the algorithms as well as show potential inaccuracies in the models.

Another question is how well the mathematical models of delays capture physical circuit behavior. It is quite probable that one can develop more accurate models, but it is not clear whether one also can develop tractable algorithms for performing the analysis.

A third open area is the complexity of the race analysis problem for different delay models. In particular, the inherent complexity of the restricted SSR question for the XBD model is open. This question is worth studying not only for its theoretical interest, but also to gain a better understanding of circuit behavior under a bounded delay assumption.

We think that, with the knowledge gained from this research, it is natural to go one step further towards the goal of being able to use asynchronous circuits efficiently. Hence, we believe that the more pressing questions now deal with the verification and design of asynchronous circuits.

8.2. Verification Problem

The use of formal mathematical methods in verifying the correctness of VLSI circuit designs is currently of great interest; see for example [2, 4, 7, 10, 23, 24, 29, 42, 58, 60]. However, very little work has been concerned with verifying asynchronous designs. (One exception to this is the work of Clarke et al. [23, 24, 42].)

Previous attempts to attack the problem of verifying asynchronous circuits have all suffered from the lack of efficient and accurate analysis methods. Typically, the methods suggested have used either the unit-delay assumption or they have used an arbitrary delay model, i.e. a model in which the delays are completely unknown. Unfortunately, the first model is very inaccurate and often fails to detect timing problems that are very likely to occur in practice. On the other hand, the arbitrary delay model is often overly pessimistic, predicting timing problems that are unlikely to occur in practice. With the results of this thesis we believe that we are now in a much better position for attacking the verification problem. In particular, the extended bounded-delay model allows us to analyze a circuit efficiently using a realistic delay model.

There are two separate issues that need to be studied in asynchronous design verification. First, it is necessary to find a suitable language for describing the desired behavior of the circuit. Second, efficient algorithms to verify that the given circuit implements the specified behavior must be developed.

The problem of specifying the desired behavior of an asynchronous circuit has received quite a lot of attention. Various approaches have been suggested: temporal logic [24, 36, 42, 44], Petri nets [47], trace theory [48], PROLOG [38], Milner's calculi [41], marked graphs [61], etc. However, it is far from clear which of these formalisms can offer the best compromise between expressive power, readability, and ease of use. In fact, it is unlikely that a single approach will suffice; it is more likely that the designer will have to choose different techniques for different applications. These tools will be more versatile, if one is able to transform one type of specification into another. This problem clearly deserves further study.

In [42] Mishra and Clarke developed a verification system based on temporal logic. Unfortunately, their system uses the unit-delay model, and thus can only detect major flaws in a design. In particular, possible errors due to race and hazard conditions cannot be detected. Despite this drawback, they were able to detect some quite subtle errors in a proposed self-timed FIFO circuit design. As a first approach, it would be interesting to replace the unit-delay algorithm in their verification system with the ternary bounded-delay algorithm. This should lead to a substantially more powerful verification system.

The formalisms mentioned above are useful mainly for describing self-timed elements, i.e. elements that use a strict "hand-shake" protocol for all communication. The main reason for this is that time is not explicitly used in the formalism, but rather events. Hence, one can make statements like "A will eventually occur", or "B will never occur", etc. However, it is not possible to describe a condition like "the acknowledgement line must go high within 100 ns after the request line goes high", etc. If we want to verify general asynchronous circuits, such constructs are clearly needed. Hence, it would be worth studying how to include such statements in the different formalisms. Temporal logic appears to be a good candidate for a model which may be extended in such a way.

8.3. Design Problem

Asynchronous circuit design today is very much based on rather ad hoc methods. In fact, it is very common that the circuits generated are incorrect when analyzed more accurately. This is clearly a very unfortunate situation.

One design philosophy that has gained a substantial amount of interest recently, is the idea of delay-independent design [23, 25, 37, 43]. In Chapter IV, we were able to derive some necessary conditions for the behavior of a completely delay-independent circuit. This partial characterization of the class of completely delay-independent circuits suggests two questions worth studying. First, can we derive both necessary and sufficient conditions for a circuit to be delay-independent, i.e. can we properly define the class of circuit behaviors that can be realized in a completely delay-independent design? Second, how can we design such delay-independent circuits?

It is interesting to note that the race model developed in the thesis for analyzing delay-independent circuits, has the very nice property that a feedback variable analysis is guaranteed to be correct. This result is promising, since a feedback analysis is substantially easier to perform. It would be interesting to investigate how this may be used to simplify the design of (realizable) delay-independent circuits.

Unfortunately, the results about delay-independent circuits also imply that many functions must be realized using delay-dependent designs, in which the nontransient behavior will depend on the relative sizes of the delays. With the development of the ternary bounded-delay simulation algorithm, at least we are now able to analyze a circuit under such a delay assumption. We believe it could be fruitful to use the ternary bounded-delay algorithm to study various proposed and used circuits. It is our hope that the insight gained from this will enable us to develop some precise and efficient design methods.

Appendix A

Switch-Level Models

In Chapter II, Section 3, we mentioned that there are two distinct approaches in defining the excitation function for a node: We can either use a static approach or a dynamic approach. In the next two sections, we will describe several different ways of defining the excitation functions in a switch-level model, first using the traditional static approach and then using a dynamic approach. These different models represent different design philosophies in CMOS and NMOS circuits.

A.1. Static Excitation Functions

In the static approach, the excitation function for a node gives the value the node would take eventually, if the input signals were frozen at their current values. We will start by deriving some very simple models and then gradually expand these basic ideas. Note that some of these models were already introduced in Chapter II, Section 3. However, we will repeat their definitions in order to put them into perspective and to be able to compare them with other models. Note also that most of the ideas in this section are derived from the work of Bryant [6, 8, 9].

We use the same notation as in Chapter II, Section 3; for example p_{ij} denotes the P-path function for nodes i and j , etc. All excitation models defined below use the same basic concept. The node excitation function Y_i will always be of the form: $o_i + (o_i + z_i)' \times$. Intuitively, o_i denotes the conditions under which Y_i should be 1, and z_i gives the conditions under which Y_i should be 0. In view of Proposition 2.1, we will immediately simplify these expressions to the form $o_i + z_i' \times$. The first four basic

models are:

$$\text{Model 1} \quad Y_i = p_{i1}t'_{i0} + (n_{i0}t'_{i1})' \times$$

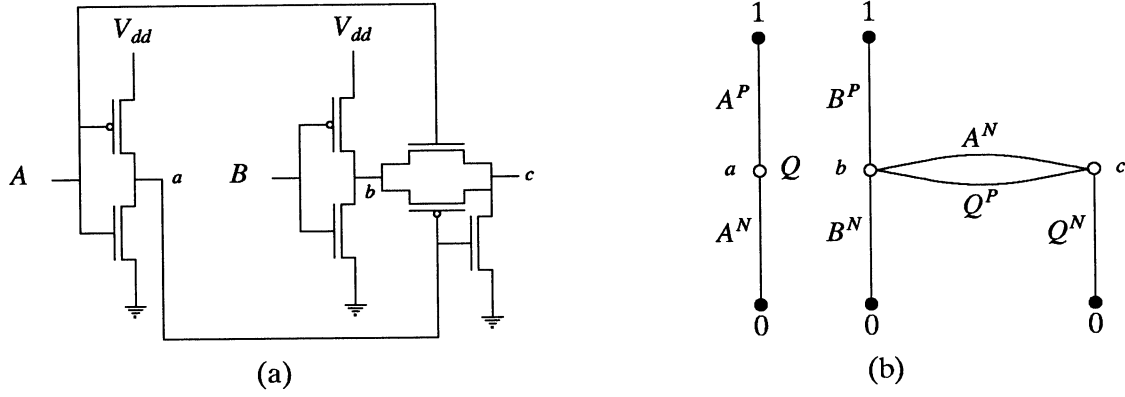
$$\text{Model 2} \quad Y_i = p_{i1}n'_{i0} + (n_{i0}p'_{i1})' \times$$

$$\text{Model 3} \quad Y_i = t_{i1}t'_{i0} + (t_{i0}t'_{i1})' \times$$

$$\text{Model 4} \quad Y_i = p_{i1}n'_{i0} + t_{i1}t'_{i0} + (n_{i0}p'_{i1} + t_{i0}t'_{i1})' \times.$$

Model 1, introduced in [18], yields a binary output iff there is a definite P(N)-path to 1(0) and no path whatsoever to 0(1). Furthermore, whenever there is a “fight” (i.e. both a path to 1 and a path to 0), whenever there is only a weak path to 0 or 1, or whenever the node is isolated (i.e. there is no path to 1 and no path to 0) the node excitation is \times . This is a very restrictive model and captures stringent design rules for combinational static CMOS circuits. The second model assumes that a P-path to 1 (a “good” path) is stronger than a P-path to 0 (a “bad” path) (and vice versa for N-paths). We get $Y_i = 1$ (0) as long as there is at least one good path to 1 (0), but no good path to 0 (1). Hence, a fight between a good path and a bad path is resolved in favor of the good path. This is a substantially more liberal rule, but seems to be necessary in order to explain certain very tricky designs [18]. The third model is more traditional and corresponds to a special case of the model in [6]. Here there is no distinction at all between P- and N-paths, and we have $Y = 1$ (0) iff there is some definite path to 1 (0) but no path to 0 (1). The final model is a mixture of models 2 and 3. Here the rules are: Good paths override bad paths; if there are no good paths, the bad paths determine the output.

Consider the CMOS circuit C_1 of Fig. A.1. Using the path functions derived earlier for node a , it is easy to verify that all models yield the node excitation function $Y_a = A'$. A more complicated example is node c . Models 1 to 4 yield the following node excitation functions:

Figure A.1. (a) CMOS circuit C_1 ; (b) corresponding S-graph S_1 .

$$\begin{aligned}
 \text{Model 1} \quad Y_c &= p_{c1}t'_{c0} + (n_{c0}t'_{c1})' \times = \\
 &= Q'B'(Q + AB + Q'B)' + ((Q + AB)(Q'B' + AB'))' \times = \\
 &= \dots = Q'B' + (Q'A' + AB') \times
 \end{aligned}$$

$$\begin{aligned}
 \text{Model 2} \quad Y_c &= p_{c1}n'_{c0} + (n_{c0}p'_{c1})' \times = \\
 &= Q'B'(Q + AB)' + ((Q + AB)(Q'B'))' \times = \\
 &= \dots = Q'B' + Q'A' \times
 \end{aligned}$$

$$\begin{aligned}
 \text{Model 3} \quad Y_c &= t_{c1}t'_{c0} + (t_{c0}t'_{c1})' \times = \\
 &= (Q'B' + AB')(Q + AB + BQ')' + ((Q + AB + BQ')(Q'B' + AB'))' \times = \\
 &= \dots = Q'B' + (A'QQ' + AB') \times.
 \end{aligned}$$

$$\begin{aligned}
 \text{Model 4} \quad Y_c &= p_{i1}n'_{i0} + t_{i1}t'_{i0} + (n_{i0}p'_{i1} + t_{i0}t'_{i1})' \times = \\
 &= Q'B'(Q + AB)' + (Q'B' + AB')(Q + AB + BQ')' + \\
 &\quad + ((Q + AB)(Q'B'))' + (Q + AB + BQ')(Q'B' + AB'))' \times = \\
 &= \dots = Q'B'.
 \end{aligned}$$

The node excitation functions above fail to capture the fact that there is a certain amount of capacitance in MOS circuits. In particular, the key nodes have a capacitance associated with them; hence there is a certain amount of “memory” in each key node. The case when there is only one key node in every channel-connected subgraph, can be handled in a very straightforward way. The basic assumption is that a 1 (0) stored on a key node can only determine the excitation of that node, if the node is completely isolated from 0 (1). One can verify that the

following models do take this into account:

$$\text{Model } 1^M \quad Y_i = (p_{i1} + y_i)t'_{i0} + ((n_{i0} + y'_i)t'_{i1})' \times$$

$$\text{Model } 2^M \quad Y_i = p_{i1}n'_{i0} + y_it'_{i0} + (n_{i0}p'_{i1} + y'_it'_{i1})' \times$$

$$\text{Model } 3^M \quad Y_i = t_{i1}t'_{i0} + y_it'_{i0} + (t_{i0}t'_{i1} + y'_it'_{i1})' \times$$

$$\text{Model } 4^M \quad Y_i = p_{i1}n'_{i0} + t_{i1}t'_{i0} + y_it'_{i0} + (n_{i0}p'_{i1} + t_{i0}t'_{i1} + y'_it'_{i1})' \times$$

For example, using model 1^M , we get Y to be 1 if there is a good path to 1 and no path to 0, or if the previous value was 1 and there is no path to 0. A dual situation holds for $Y = 0$.

Consider again the CMOS circuit C_1 of Fig. A.1. Using model 1^M and assuming node c is a key node, we get the node excitation functions for nodes a and c :

$$\begin{aligned} Y_a &= (p_{a1} + y_a)t'_{a0} + ((n_{a0} + y'_a)t'_{a1})' \times = \\ &= (A' + y_a)A' + ((A + y'_a)(A'))' \times = \\ &= \dots = A' \\ Y_c &= (p_{c1} + y_c)t'_{c0} + ((n_{c0} + y'_c)t'_{c1})' \times = \\ &= (Q'B' + y_c)(Q + AB + Q'B)' + (((Q + AB) + y'_c)(Q'B' + AB'))' \times = \\ &= \dots = Q'B' + (Q'A'y_c + AB') \times \end{aligned}$$

When there is more than one key node in each channel-connected subgraph, a more complicated node excitation function must be used. This is because the charges stored on the nodes may interact with each other, and charge can “spill over” from one node to another causing the excitation to become undefined. From here on, we will assume that charge sharing comes into effect only when the nodes are isolated from the supply nodes. This seems to be a reasonable assumption. To simplify the notation, we will write the excitation function as $Y_i = o_i + z'_i \times$ and give the definitions of the functions o and z separately. Model 1, extended to handle the multiple key node case, gives the following functions:

$$\begin{aligned}
o_i &= p_{i1}t'_{i0} + t'_{i0} \left(\bigwedge_{j \in \dot{S}} (y_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}} y_j p_{ij} \right) \right) \\
&= p_{i1}t'_{i0} + t'_{i0} y_i \bigwedge_{j \in \dot{S}} (y_j + t'_{ij}) \\
z_i &= n_{i0}t'_{i1} + t'_{i1} \left(\bigwedge_{j \in \dot{S}} (y'_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}} y'_j n_{ij} \right) \right) \\
&= n_{i0}t'_{i1} + t'_{i1} y'_i \bigwedge_{j \in \dot{S}} (y'_j + t'_{ij})
\end{aligned}$$

The basic idea is as follows. We will only discuss the o_i function, but the arguments can be trivially extended to the z_i function. First, charge sharing comes into effect only when there are no definite or indefinite paths to ground. This is captured by the condition that t_{i0} must be 0 to allow a stored charge to determine the excitation. Second, if a node i is disconnected from V_{dd} and Gnd , then all key nodes connected via some path to i must have the value 1 in order to make the excitation of i to be 1. This is captured in the second half of the formula. First, either the value of a key node j must be 1 or the node j must be disconnected from node i . Second, at least one of the nodes must be 1 and must be connected to i by a definite P-path, in order to get $o_i = 1$. (In the formulae above we assumed that y_i is a key node, and hence $t_{ii} = p_{ii} = n_{ii} = 1$, and the last equalities in the formulae follow.)

It is easy to derive similar node excitation functions for the other basic CMOS models. For example, to derive o_i in a model based on Model 2, one would simply replace the $p_{i1}t'_{i0}$ term with a $p_{i1}n'_{i0}$ term.

In the above discussion only the key nodes were assumed to have memory. Furthermore, it was assumed that all key nodes have the same “size”. A common technique in MOS circuits is the use of pre-charged lines. In such circuits, certain nodes are designed with a substantially higher capacitance. This can be modeled as if these nodes had “greater size” than normal nodes. It is quite straightforward to expand the general idea above to handle such circuits. However, in order to describe such excitation function, we need the following notation. Let \dot{S}^s , $1 \leq s \leq q$, be the set of all nodes of size s in the channel-connected subgraph \dot{S} . Assume

furthermore that the sizes are totally ordered so that a node in \dot{S}^p is substantially bigger (has a substantially higher capacitance) than a node in \dot{S}^r iff $p > r$. Model 1, extended to handle the multiple key nodes and different node sizes, gives the following functions: (We use the convention that $\bigwedge_{j \in \dot{S}^r} \dots$ is equal to 1 if $\dot{S}^r = \emptyset$ and similarly that $\bigvee_{j \in \dot{S}^r} \dots$ is equal to 0 if $\dot{S}^r = \emptyset$.)

$$o_i = p_{i1}t'_{i0} + t'_{i0} \left[\bigwedge_{j \in \dot{S}^q} (y_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}^q} y_j p_{ij} + \right. \right. \\ \left. \left. + \bigwedge_{j \in \dot{S}^{q-1}} (y_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}^{q-1}} y_j p_{ij} + \right. \right. \right. \\ \left. \left. \left. + \dots + \right. \right. \right. \\ \left. \left. \left. + \bigwedge_{j \in \dot{S}^1} (y_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}^1} y_j p_{ij} \right) \dots \right) \right) \right] \right]$$

and

$$z_i = n_{i0}t'_{i1} + t'_{i1} \left[\bigwedge_{j \in \dot{S}^q} (y'_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}^q} y'_j n_{ij} + \right. \right. \\ \left. \left. + \bigwedge_{j \in \dot{S}^{q-1}} (y'_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}^{q-1}} y'_j n_{ij} + \right. \right. \right. \\ \left. \left. \left. + \dots + \right. \right. \right. \\ \left. \left. \left. + \bigwedge_{j \in \dot{S}^1} (y'_j + t'_{ij}) \left(\bigvee_{j \in \dot{S}^1} y'_j n_{ij} \right) \dots \right) \right) \right] \right]$$

Once again, the basic idea is quite simple. First, the node must be isolated from the supply nodes before charge sharing effects need to be considered. Furthermore, in order for a node of strength r with value 1 to be relevant to the node i , either every stronger node must be isolated from i , or the stronger nodes that are

connected to i must have the value 1 (thus making the value of the node of strength r irrelevant anyway).

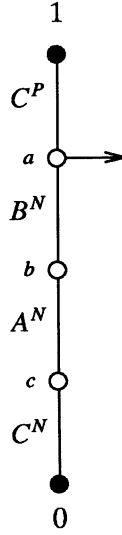


Figure A.2. CMOS circuit S_2 .

To illustrate the above idea, consider the circuit S_2 of Fig. A.2. (The circuit is a two-input NAND gate implemented in domino CMOS technology [59].) A domino CMOS circuit works as follows. There are two phases: the pre-charge phase and the evaluation phase. In the pre-charge phase the clock signal (called C in our example) is set to 0 causing the high-capacitance node a to be driven to a high voltage through transistor C^P . Note that this will happen irrespective of the values of A and B because the transistor C^N is not conducting. In the second phase, the evaluation phase, the clock signal is switched to 1. Here node a will keep its high value iff at least one of A and B has a low voltage. Since the behavior of the circuit depends on the assumption that node a has substantially higher capacitance than nodes b and c , we will assume that we have nodes of two strengths 1 and 2, and that $\dot{S}^1 = \{b, c\}$ and $\dot{S}^2 = \{a\}$. Using the model derived above, we get the following functions for the node a :

$$\begin{aligned}
o_a &= p_{a1}t'_{a0} + t'_{a0} \left[\bigwedge_{j \in \tilde{S}^2} (y_j + t'_{aj}) \left\{ \bigvee_{j \in \tilde{S}^2} y_j p_{aj} + \bigwedge_{j \in \tilde{S}^1} (y_j + t'_{aj}) \left(\bigvee_{j \in \tilde{S}^1} y_j p_{aj} \right) \right\} \right] = \\
&= C'(ABC)' + (ABC)' \left[(y_a + 0) \left\{ y_a + (y_b + B')(y_c + (AB)') \left((y_b 0 + y_c + 0) \right) \right\} \right] = \\
&= \dots = C' + (A' + B')y_a
\end{aligned}$$

and

$$\begin{aligned}
z_a &= n_{a0}t'_{a1} + t'_{a1} \left[\bigwedge_{j \in \tilde{S}^2} (y'_j + t'_{aj}) \left\{ \bigvee_{j \in \tilde{S}^2} y'_j n_{aj} + \bigwedge_{j \in \tilde{S}^1} (y'_j + t'_{aj}) \left(\bigvee_{j \in \tilde{S}^1} y'_j n_{aj} \right) \right\} \right] = \\
&= ABC(C')' + (C')' \left[y'_a \left\{ y'_a + (y'_b + B')(y'_c + (AB)') \left((y'_b B + y'_c AB) \right) \right\} \right] = \\
&= \dots = ABC + y'_a C.
\end{aligned}$$

If we combine these results, we obtain

$$Y_a = o_a + z'_a \times = C' + (A' + B')y_a + (ABC + y'_a C)' \times = C' + (A' + B')y_a.$$

It is easy to verify that the excitation function derived in this way indeed shows that the circuit behaves as intended.

It should be pointed out that the models above constitute only some of the very many possibilities. For example, in the excitation function above we required that a stored 1 on a node must drive the output node through a P-path in order to transmit a 1 properly. In some cases it may be appropriate to relax this condition and allow any kind of path. The derivation of such a model is left as an exercise for the interested reader.

So far we took care of the situation when different nodes may have substantially different capacitance, and thus can be treated as having different sizes. However, we assumed that all the transistors had the same conductance (except that the

conductance for transmitting a 1 was different from the conductance for transmitting a 0). Sometimes it is practical to use transistors with significantly different conductances. We can model this in a switch-level model by assuming that the transistors have different “strengths”. In order to describe such a model, we need to add the notion of strength to the path functions. We first treat the case when all the nodes have the same size, but the transistors may have different strengths. Let p_{ij}^s , n_{ij}^s , and t_{ij}^s denote the path functions of strength s . (As before, we assume that there are some finite number of strengths, $1, \dots, q$, and that a transistor of strength r has a substantially higher conductance than a transistor of strength p iff $r > p$.) A path is of strength s if all the transistors in the path have strength $\geq s$. The basic idea is, of course, that a signal from a strong path will override a signal from a weaker path. Model 1, extended to handle the different transistor strengths, gives the following functions:

$$o_i = (t_{i0}^q)' \left[p_{i1}^q + (t_{i0}^{q-1})' \left[p_{i1}^{q-1} + \dots + (t_{i0}^1)' \left[p_{i1}^1 \right] \dots \right] \right]$$

and

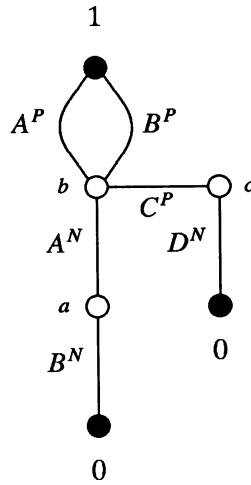
$$z_i = (t_{i1}^q)' \left[n_{i0}^q + (t_{i1}^{q-1})' \left[n_{i0}^{q-1} + \dots + (t_{i1}^1)' \left[n_{i0}^1 \right] \dots \right] \right].$$

To illustrate these ideas, consider the CMOS circuit S_3 of Fig. A.3. Assume that all the transistors except D^N have the same strength and that D^N is substantially stronger (has a much higher conductance). Assume also that all the nodes have the same size. To compute the excitation function for node c we get the following calculations:

$$o_c = (t_{c0}^2)' \left[p_{c1}^2 + (t_{c0}^1)' \left[p_{c1}^1 \right] \right] = D' \left[0 + (D + C'AB)' \left[C'(A' + B') \right] \right] = \dots = C'D'(A' + B'),$$

$$z_c = (t_{c1}^2)' \left[n_{c0}^2 + (t_{c1}^1)' \left[n_{c0}^1 \right] \right] = 0' \left[D + (C'(A' + B'))' \right] D = \dots = D,$$

and thus

Figure A.3. CMOS circuit S_3 .

$$Y_c = C'D'(A' + B') + D' \times.$$

Finally, it is possible to combine the two approaches above for circuits that contain both nodes of different sizes and transistors of different strengths. In such a situation, we will assume that charge sharing is significant only when the nodes are isolated from the supply nodes. We will not derive node excitation functions for such cases, but the interested reader can easily derive such functions. Note that two different node sizes and two different transistor strengths are sufficient for virtually all CMOS designs.

The discussion above focused entirely on CMOS circuits. However, it is quite straightforward to use the same ideas for modeling NMOS circuits. The interested reader is referred to Bryant's work [8, 9]. Here it suffices to say that, using the MOSSIM model [6] and the procedure described in [8, 9], one can derive ternary functions for the node excitations. Using this approach, not only CMOS, but also NMOS circuits can be handled.

A.2. Dynamic Excitation Functions

In this section we will outline how one can derive excitation functions that use a dynamic approach. The ideas in this section should be viewed more as examples and suggestions for future research, rather than as thoroughly researched and “finished” results.

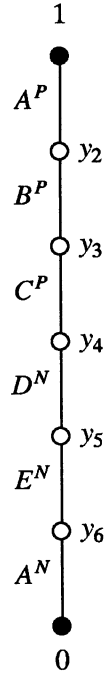
In a dynamic approach we want to derive excitation functions that models the actual circuit behavior more closely. For example, if the value on only one transistor gate in a channel-connected subgraph changes, it is reasonable to assume that the only nodes that can change in the first instance are the nodes that are connected to this transistor. For example, consider the CMOS circuit S_4 of Fig. A.4 started in the state $A = D = E = 1$, $B = C = 0$ and $y_i = 0$ for $i = 2, \dots, 6$. If now A changes to 0, a static excitation function would make all the nodes unstable, and depending on the race model chosen, one could have the following very strange change sequence: first y_6 changes to 1, then y_5 changes to 1, etc. Such change sequence is clearly impossible. On the other hand, in a dynamic model only nodes 2 and 6 can change at first. However, since the neighboring nodes to node 6 (i.e. node 5 and the *Gnd* node) all have the value 0, node 6 would not be unstable. Hence, the only successor state would be the state reached by changing y_2 to 1 (or \times , depending on the race model). Once this state has been reached, node 3 may change, and so on. In fact, the 1 signal would “ripple” down along the chain of transistors. We will formalize these intuitive ideas in this section.

As in the static approach, we will derive excitation functions of the form $Y_i = o_i + (o_i + z_i)' \times$, or simplified to $Y_i = o_i + z_i' \times$. To define the models we need the following notation. Let ν_i be the set of nodes that are connected to node i via a transistor, i.e. $\nu_i = \{j : \text{there is a transistor connecting node } i \text{ and node } j\}$.

The first model uses the following functions:

$$o_i = y_i \bigwedge_{j \in \nu_i} (y_j + t'_{ij}) + \bigwedge_{j \in \nu_i} (y_j + t'_{ij} + t'_{j0}) \left(\bigvee_{j \in \nu_i} y_j p_{ij} p_{j1} \right)$$

and

Figure A.4. CMOS circuit S_4 .

$$z_i = y'_i \bigwedge_{j \in \nu_i} (y'_j + t'_{ij}) + \bigwedge_{j \in \nu_i} (y'_j + t'_{ij} + t'_{j1}) \left(\bigvee_{j \in \nu_i} y'_j n_{ij} n_{j0} \right)$$

We will limit our discussion to o_i , but z_i can be motivated similarly. The idea is simply as follows: The first term in o_i takes care of the situation in which no change has “reached” the node. The term implies that if the present value of the node is 1 and all neighboring nodes that are connected to i also have the value 1, then the excitation is 1. Note that this term does not contain any condition on any “global” paths to 0 or 1, and thus is really the “local” part of the excitation function. If we have a situation in which some connected neighbor nodes have the value 1 and some have the value 0, it is still possible for the excitation to be 1. Here we must determine the “strength” of the different 1’s and 0’s. In particular, a 1 on a connected neighbor node that is connected via a P-path to the power supply is stronger than a 0 on another connected neighbor node if this second node is not connected to ground.

To illustrate this model consider again the CMOS circuit S_4 of Fig. A.4. Node 2 gets the excitation functions:

$$\begin{aligned} o_2 &= y_2(1+A)(y_3+B) + \left(1+A+(A'B'C'DEA)'\right) \left(y_3+B+(C'DEA)'\right) \left[\left(1A'1\right) + \left(y_3B'(A'B')\right) \right] = \\ &= \dots = A' + y_2(y_3+B) \end{aligned}$$

and

$$\begin{aligned} z_2 &= y_2'(0+A)(y_3'+B) + \left(0+A+0\right) \left(y_3'+B+(A'B')'\right) \left[\left(000\right) + \left(y_3'00\right) \right] = \\ &= \dots = y_2'A(B+y_3') \end{aligned}$$

which finally gives:

$$Y_2 = o_2 + z_2' \times = \dots = A' + y_2(y_3+B) + (y_2+B'y_3) \times$$

Hence, if A is 0, the excitation of node 2 is 1. On the other hand, if A is \times or 1, the only way the excitation of node 2 could be 1 is when the present value on node 2 is 1 and either the node is isolated (by having $B=1$) or if node 3 is also 1. This last condition captures the desired “locality of excitation”. On the other hand, the only way for the excitation to be 0 is if A is 1 and the present value is 0. Hence, it is not possible to get the excitation of 0 unless the node is started with the value 0.

It is straightforward to modify the above functions in the same way as was done in the static approach in Section A.1. For example, it may be undesirable to distinguish between P-paths and N-paths and assign the same strength to both paths. This would yield the following excitation functions:

$$o_i = y_i \bigwedge_{j \in \nu_i} (y_j + t'_{ij}) + \bigwedge_{j \in \nu_i} (y_j + t'_{ij} + t'_{j0}) \left(\bigvee_{j \in \nu_i} y_j t_{ij} t_{j1} \right)$$

and

$$z_i = y_i' \bigwedge_{j \in \nu_i} (y_j' + t'_{ij}) + \bigwedge_{j \in \nu_i} (y_j' + t'_{ij} + t'_{j1}) \left(\bigvee_{j \in \nu_i} y_j' t_{ij} t_{j0} \right)$$

The interested reader can verify that, if we use this model for the node 2 in circuit S_4 , we get the excitation function:

$$Y_2 = A' + y_2(y_3 + B) + (y_2 + A' + B'y_3)(y_3 + A' + B + C + D' + E') \times$$

Here, the excitation is 1 under the same conditions as in the first model, but there is a way of getting the excitation to become 0 even if the present value is not 0. (This can be accomplished by having $y_3 = 0$, $A = D = E = 1$ and $B = C = 0$.)

It is easy to verify that the path functions above can ignore paths that go through the node i . This follows from the fact that such paths are extended versions of paths that are already considered. For example, if we are deriving an excitation function for node 3 of circuit S_4 of Fig. A.4, it is correct to say that $t_{20} = 0$, simply because the only path from node 2 to the ground node must go through node 3.

As in the static approach, we now show how to derive excitation functions when there are nodes of different sizes. As before, assume that there are nodes of size $1, \dots, q$ and that nodes of size r are substantially larger (have substantially higher capacitance) than nodes of size p iff $r > p$. We need the following notation: Let ν_i^r , $1 \leq r \leq q$, denote the set of nodes of strength r that are connected to node i via a transistor. Also, let $\nu_i^* = \bigcup_{1 \leq r \leq q} \nu_i^r$. (We use the convention that $\bigwedge_{j \in \nu_i^r} \dots$ is equal to 1 if $\nu_i^r = \emptyset$ and similarly that $\bigvee_{j \in \nu_i^r} \dots$ is equal to 0 if $\nu_i^r = \emptyset$.) One possible excitation model is given by:

$$\begin{aligned}
o_i = & y_i \bigwedge_{j \in \nu_i^*} (y_j + t'_{ij}) + \bigwedge_{j \in \nu_i^*} (y_j + t'_{ij} + t'_{j0}) \left[\bigwedge_{j \in \nu_i^q} (y_j + t'_{ij}) \left(\bigvee_{j \in \nu_i^q} y_j p_{ij} + \right. \right. \\
& + \bigwedge_{j \in \nu_i^{q-1}} (y_j + t'_{ij}) \left(\bigvee_{j \in \nu_i^{q-1}} y_j p_{ij} + \right. \\
& + \dots + \\
& \left. \left. \left. + \bigwedge_{j \in \nu_i^1} (y_j + t'_{ij}) \left(\bigvee_{j \in \nu_i^1} y_j p_{ij} \right) \dots \right) \right] \right]
\end{aligned}$$

and

$$\begin{aligned}
z_i = & y'_i \bigwedge_{j \in \nu_i^*} (y'_j + t'_{ij}) + \bigwedge_{j \in \nu_i^*} (y'_j + t'_{ij} + t'_{j1}) \left[\bigwedge_{j \in \nu_i^q} (y'_j + t'_{ij}) \left(\bigvee_{j \in \nu_i^q} y'_j n_{ij} + \right. \right. \\
& + \bigwedge_{j \in \nu_i^{q-1}} (y'_j + t'_{ij}) \left(\bigvee_{j \in \nu_i^{q-1}} y'_j n_{ij} + \right. \\
& + \dots + \\
& \left. \left. \left. + \bigwedge_{j \in \nu_i^1} (y'_j + t'_{ij}) \left(\bigvee_{j \in \nu_i^1} y'_j n_{ij} \right) \dots \right) \right] \right]
\end{aligned}$$

As before, the model can be easily modified to ignore the difference between P-paths and N-paths. Furthermore, using a similar approach, it is quite straightforward to derive excitation functions for the case when the transistors have different strengths. However, this is outside of the scope of this thesis.

In summary, in this section we have outlined how one can derive excitation functions that are primarily local. This dynamic approach deserves more study. In particular, it would be useful to establish whether the dynamic excitation functions correspond to the static ones for certain types of circuits (for example, static CMOS cells [19]).

References

- [1] R. Andrew, "An Algorithm for Eight-Valued Simulation and Hazard Detection in Gate Networks," in *Proc. Int. Symp. on Multiple-Valued Logic*, pp. 273-280, 1986.
- [2] H. G. Barrow, "Proving the Correctness of Digital Hardware Designs," *VLSI Design*, Vol. 7, pp. 64-77, Jul. 1984.
- [3] M. A. Breuer, "A Note on Three-Valued Logic Simulation," *IEEE Transactions on Computers*, Vol. C-21, pp. 399-401, Apr. 1972.
- [4] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra, "Automatic Verification of Sequential Circuits Using Temporal Logic," *IEEE Transactions on Computers*, Vol. C-35, No. 12, pp. 1035-1044, Dec. 1986.
- [5] R. E. Bryant, "Race Detection in MOS Circuits by Ternary Simulation," in *VLSI '83*, F. Anceau and E. J. Aas, Eds., Elsevier Science Publishers B. V. (North Holland), 1983, pp. 85-95.
- [6] R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Transactions on Computers*, Vol. C-33, No. 2, pp. 160-177, Feb. 1984.
- [7] R. E. Bryant, "Symbolic Verification of MOS Circuits," in *Proc. 1985 Chapel Hill Conference on VLSI*, H. Fuchs, Ed., Computer Science Press, 1985, pp. 419-438.

- [8] R. E. Bryant, "Algorithmic Aspects of Symbolic Switch Network Analysis," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 4, pp. 618-633, July 1987.
- [9] R. E. Bryant, "Boolean Analysis of MOS Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, No. 4, pp. 634-649, July 1987.
- [10] R. E. Bryant, "A Methodology for Hardware Verification Based on Logic Simulation," Technical Report CMU-CS-87-128, Department of Computer Science, Carnegie-Mellon University, June 1987.
- [11] J. A. Brzozowski and E. J. McCluskey, Jr., "Signal Flow Graph Techniques for Sequential Circuit State Diagrams," *IEEE Transactions on Electronic Computers*, Vol. EC-12, No. 2, pp. 67-76, Apr. 1963.
- [12] J. A. Brzozowski and C-J. Seger, "Correspondence between Ternary Simulation and Binary Race Analysis in Gate Networks," in *Automata, Languages and Programming, Lecture Notes in Computer Science*, Vol. 226, L. Kott, Ed., Springer Verlag, 1986, pp. 67-78.
- [13] J. A. Brzozowski and C-J. Seger, "A Characterization of Ternary Simulation of Gate Networks," *IEEE Transactions on Computers*, Vol. C-36, No. 11, pp. 1318-1327, Nov. 1987.
- [14] J. A. Brzozowski and C-J. Seger, "A Unified Framework for Race Analysis of Asynchronous Circuits," *Journal of the Association for Computing Machinery*, (to appear). Also Technical Report CS-87-24, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
- [15] J. A. Brzozowski and M. Yoeli, "Models for Analysis of Races in Sequential Networks," in *Lecture Notes in Computer Science*, Vol. 28, A. Blikle, Ed., Springer Verlag, 1975, pp. 26-32.
- [16] J. A. Brzozowski and M. Yoeli, *Digital Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
- [17] J. A. Brzozowski and M. Yoeli, "On a Ternary Model of Gate Networks," *IEEE Transactions on Computers*, Vol. C-28, No. 3, pp. 178-183, Mar. 1979.

- [18] J. A. Brzozowski and M. Yoeli, "Combinational Static CMOS Networks," in *VLSI Algorithms and Architectures, Lecture Notes in Computer Science, Vol. 227*, F. Makedon, K. Melhorn, T. Papatheodorou and P. Spirakis, Eds., Springer Verlag, 1986, pp. 271-282.
- [19] J. A. Brzozowski and M. Yoeli, "Combinational Static CMOS Networks," *Integration, the VLSI Journal*, No. 5, pp. 103-122, 1987.
- [20] S. G. Chappell and S. S. Yau, "Simulation of Large Asynchronous Logic Circuits using an Ambiguous Gate Model," in *AFIPS Conference Proc.*, Vol. 39, pp. 651-661, 1971.
- [21] S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *Proc. 3rd Annual ACM Symp. on Theory of Computing*, pp. 151-158, 1971.
- [22] J. B. Dennis and S. S. Patil, "Speed Independent Asynchronous Circuits," in *Fourth Hawaii International Conference on System Sciences*, Honolulu, Hawaii, Jan. 1971.
- [23] D. L. Dill and E. M. Clarke, "Automatic Verification of Asynchronous Circuits Using Temporal Logic," in *Proc. 1985 Chapel Hill Conference on VLSI*, H. Fuchs, Ed., Computer Science Press, 1985, pp. 127-143.
- [24] D. L. Dill and E. M. Clarke, "Automatic Verification of Asynchronous Circuits Using Temporal Logic," in *Proc. IEE*, Vol. 133, No. 5, pp. 276-282, Sep. 1986.
- [25] J. C. Ebergen, *Translating Programs into Delay-Insensitive Circuits*, Ph.D. Thesis, Eindhoven University of Technology, 1987.
- [26] E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM Journal of Research and Development*, Vol. 9, pp. 90-99, Mar. 1965.
- [27] G. Fantauzzi, "An Algebraic Model for the Analysis of Logical Circuits," *IEEE Transactions on Computers*, Vol. C-23, No. 6, pp. 576-581, Jun. 1974.

- [28] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., New York, 1979.
- [29] M. J. C. Gordon, "Why Higher Order Logic is a Good Formalism for Specifying and Verifying Hardware," in *Formal Aspects of VLSI Design: Proc. of the 1985 Edinburgh Conference on VLSI*, G. J. Milne and P. A. Subrahmanyam, Eds., North-Holland, Amsterdam, 1986.
- [30] J. P. Hayes, "A Unified Switching Theory with Applications to VLSI Design," in *Proc. IEEE*, Vol. 70, No. 10, pp. 1140-1151, Oct. 1982.
- [31] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass., 1979.
- [32] D. A. Huffman, "The Synthesis of Sequential Switching Circuits," *J. Franklin Inst.*, Vol. 257, No. 3-4, 1954. Also in *Sequential Machines: Selected Papers*, E. F. Moore, Ed., Addison-Wesley, Reading Massachusetts, 1964, pp. 3-62.
- [33] J. S. Jephson, R. P. McQuarrie, and R. E. Vogelsberg, "A Three-Level Design Verification System," *IBM Systems Journal*, Vol. 8, No. 3, pp. 178-188, Mar. 1969.
- [34] G. G. Langdon, "Analysis of Asynchronous Circuits Under Different Delay Assumptions," *IEEE Transactions on Computers*, Vol. C-17, pp. 1131-1143, Dec. 1968.
- [35] T. Lengauer and S. Naher, "An analysis of Ternary Simulation as a Tool for Race-detection in Digital MOS Circuits," *Integration, the VLSI Journal*, No. 4, pp. 309-330, 1986.
- [36] Y. Malachi and S. S. Owicki, "Temporal Specification of Self-Timed Systems," in *VLSI Systems and Computations*, H. T. Kung, R. Sproull, and G. Steele, Eds., Computer Science Press, 1981, pp. 203-212.
- [37] A. J. Martin, "The Design of a Self-Timed Circuit for Distributed Mutual Exclusion," in *Proc. 1985 Chapel Hill Conference on VLSI*, H. Fuchs, Ed., Computer Science Press, 1985, pp. 245-260.

- [38] F. Maruyama and M. Fujita, "Hardware Verification," *Computer*, Vol. 18, No. 2, pp. 22-32, Feb. 1985.
- [39] E. J. McCluskey, Jr., "Transients in Combinational Logic Circuits," in *Redundancy Techniques for Computing Systems*, Wilcox, R. H. and W. C. Mann, Eds., Spartan Books, Washington, 1962, pp. 9-46.
- [40] R. E. Miller, *Switching Theory*, Vol. 2, Wiley, New York, 1965.
- [41] R. Milner, "Calculi for Synchrony and Asynchrony," *Theoretical Computer Science*, Vol. 25, pp. 267-310, 1983.
- [42] B. Mishra and E. M. Clarke, "Automatic and Hierarchical Verification of Asynchronous Circuits Using Temporal Logic," Technical Report CMU-CS-83-155, Department of Computer Science, Carnegie-Mellon University, Sep. 1983.
- [43] C. E. Molnar, T.-P. Fang, and F. U. Rosenberger, "Synthesis of Delay-Insensitive Modules," in *Proc. 1985 Chapel Hill Conference on VLSI*, H. Fuchs, Ed., Computer Science Press, 1985, pp. 67-86.
- [44] B. Moszkowski, "A Temporal Logic for Multilevel Reasoning about Hardware," *Computer*, Vol. 18, No. 2, pp. 10-19, Jan. 1985.
- [45] D. E. Muller and W. S. Bartky, "A Theory of Asynchronous Circuits," *Proceedings of an International Symposium on the Theory of Switching*, Vol. 29 of the *Annals of the Computation Laboratory of Harvard University*, pp. 204-243, Harvard University Press, 1959.
- [46] V. Ramachandran, "Algorithmic Aspects of MOS VLSI Switch-Level Simulation with Race Detection," *IEEE Transactions on Computers*, Vol. C-35, No. 5, pp. 462-475, May 1986.
- [47] F. J. Ramming, "Design Aids for Highly Distributed Hardware," in *Concurrent Languages in Distributed Systems*, G. L. Reijns and E. L. Dagless, Eds., Elsevier, 1985, pp. 135-149.

- [48] M. Rem, J. L. A. van de Snepscheut, and J. T. Udding, "Trace Theory and the Definition of Hierarchical Components," in *Proc. Third Caltech Conf. on VLSI*, R. E. Bryant, Ed., Computer Science Press, 1983, pp. 225-239.
- [49] C-J. Seger, *Ternary Simulation of Asynchronous Gate Networks*, Master's Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1985. Also Technical Report CS-86-19, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
- [50] C-J. Seger and J. A. Brzozowski, "An Optimistic Ternary Simulation of Gate Races," in *Canadian Conference on VLSI*, pp. 67-72, Oct. 1986. Also Technical Report CS-86-22, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
- [51] C-J. Seger and J. A. Brzozowski, "An Optimistic Ternary Simulation of Gate Races," *Theoretical Computer Science*, (to appear). Also Technical Report CS-86-22, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
- [52] C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. AIEE*, Vol. 57, pp. 713-723, 1938.
- [53] SimuTec, "SILOS: Logic Simulator User's Manual," 1985.
- [54] L. J. Stockmeyer, "The Complexity of Decision Problems in Automata Theory and Logic," Technical Report MAC TR-133, Project MAC, MIT, Cambridge, Massachusetts, 1974.
- [55] R. Sundblad and C. Svensson, "Fully Dynamic Switch-level Simulation of CMOS Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. CAD-6, pp. 282-289, Mar. 1987.
- [56] C. J. Terman, "Simulation Tools for Digital LSI Design," Technical Report MIT/LCS/TR-304, Department of Electrical Engineering and Computer Science, MIT, Cambridge, Mass., 1983.
- [57] S. T. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, New York, 1969.

- [58] D. Weise, "Functional Verification of MOS Circuits," in *ACM and IEEE Design Automation Conference*, Vol. 24, pp. 265-270, 1987.
- [59] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1985.
- [60] G. Winskel, "Models and Logic of MOS Circuits," Technical Report 96, Computer Laboratory, University of Cambridge, Oct. 1986.
- [61] M. Yoeli, "Specification and Verification of Asynchronous Circuits," Technical Report 436, Department of Computer Science, Technion — Israel Institute of Technology, Nov. 1986.
- [62] M. Yoeli and S. Rinon, "Application of Ternary Algebra to the Study of Static Hazards," *Journal of the Association for Computing Machinery*, Vol. 11, No. 1, pp. 84-97, Jan. 1964.

