Ternary Simulation of Asynchronous Gate Networks

Carl-Johan Henry Seger

CS-86-19

June, 1986

Ternary Simulation of Asynchronous Gate Networks

by

Carl-Johan Henry Seger

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, 1985

Table of Contents

Ternary Simulation of Asynchronous Gate Networks

ABSTRACT

Analyzing asynchronous sequential networks is an intricate problem. In this thesis we first give a general overview of the different methods for analysis that have been suggested in the literature. In the remaining parts of the thesis, we discuss the use of ternary, or three-valued logic, for the analysis. A ternary simulation algorithm was suggested by Eichelberger in 1965, and the method has been extensively used in simulators for gate networks, mainly because of its efficiency. However, the results obtained from the simulation have not been completely understood. The main result of this thesis is the complete characterization of the ternary simulation. In fact, we prove the Brzozowski-Yoeli conjecture (stated in 1976) about correspondence between the ternary simulation and the binary race analysis according to the General Multiple Winner model. We show that the ternary analysis of a gate network, corresponds exactly to the binary race analysis of the same network under the assumption that both wires and gates can have arbitrary, but finite delays.

As a corollary from the characterization, we prove that there cannot exist a critical race among some gates without the existence of an oscillation involving the same gates. We also suggest how to use the results from the ternary analysis for detecting possible output hazards.

Finally, we discuss some disadvantages of ternary simulation. In particular, the method can be overly pessimistic and predict timing problem in cases when this is highly unlikely. In order to reduce this pessimism, we suggest a new ternary simulation algorithm. We apply the method to a number of examples, and show that the method appears to be quite useful. We also state a conjecture about partial correspondence between this ternary algorithm, and a binary "Almost Equal Delay" model.

Acknowledgements

I consider myself very fortunate in having had Dr. John Brzozowski as my supervisor, and am very grateful for his encouragement, feedback and endless patience at all stages of my research. This includes all his help in correcting my often strange "English".

I would also like to thank Dr. Michael Yoeli for his many very useful suggestions regarding this thesis and the formulation of the proofs of the conjecture. Without his comments, the proofs would be even harder to read. The comments from the readers, Dr. K. Culik and Dr. F. Mavaddat were also very appreciated.

A very special thanks must also go to Gunnar Liden. Being the only Swedes within many miles in a country far away from home, creates a very special kind of friendship. I consider myself very lucky in having had the opportunity to learn to know Gunnar.

A very special thanks should also go to Ian and Ann Davis. They have "forced" me away from my work many times, and invited me into their home. In addition to this, all discussions with Ian have greatly helped me in gaining some understanding of Canada and Canadians in general.

Thanks should also go to Neil Calkin for his crucial remarks concerning marked graphs. The proof of Lemma 2 came out of these remarks!

I would also like to thank all the people who have made my stay in Waterloo a rewarding experience, due to their friendship, in particular Brian Smith, Daniel Cavalcanti, Sheila Trainor, Claudia Medeiros, Benny Wong, and Michel Devine.

Finally, I would like to acknowledge the financial support from Chalmers University of Technology, from the Department of Computer Science of the University of Waterloo, and from the National Sciences and Engineering Research Council of Canada under grant No. A0871.

Flyttfåglarna

I, flyktande gäster på främmande strand,
när söken I åter ert fädernesland?
När sippan sig döljer
i fädernesdalen
och bäcken besköljer
den grönskande alen,
då lyfta de vingen,
då komma de små;
väg visar dem ingen
i villande blå;
de hitta ändå.

Du, flyktande ande på främmande strand,
när söker du åter ditt fädernesland?
När palmerna mogna
i fädernevärlden,
då börjar du, trogna,
den fröjdfulla färden,
då lyfter du vingen
som fåglarna små;
väg visar dig ingen
i villande blå;
du hittar ändå.

Johan Ludvig Runeberg

List of Figures

Chapter I
Basic Motivation

1.1. Introduction

This thesis deals mainly with the analysis of asynchronous switching circuits, an area of research that was very actively studied around 1960, but was somewhat neglected in the 1970's. However, with the arrival of Very Large Scale Integration (VLSI) and with attempts to construct simulators for VLSI circuits, the area has once again become of great interest. This thesis addresses the first problem that one faces when trying to use asynchronous circuits: namely the problem of analyzing a given circuit. The second, and much harder, problem of synthesis, i.e. of constructing a realization from some specification, is only briefly mentioned. There are some "standard" ways to solve this second problem, but as will be shown in this thesis, some of the assumptions and models behind these design procedures do not hold in general, but apply only to special cases.

The fundamental difference between a synchronous and an asynchronous circuit lies in the fact that in the former all "sub-units" are kept in synchronism by a central clock, whereas in the latter each unit is allowed to work at its own pace, and only when some communication is necessary between two units are they synchronized with each other. In Section 4, we will give a more precise definition of the terms synchronous and asynchronous.

In this chapter a brief historical overview is given and some motivation for the renewed interest for asynchronous circuits is discussed. Since the development of switching theory has gone hand in hand with the development of computers, a historical overview of the development of switching theory is in many ways a brief overview of the history of computers. In Section 1.2 the early computers are described and their significance to the development of switching theory, and in particular to the development of the theory of asynchronous circuits, is pointed out. In Section 1.3 the basic problem of simulating VLSI circuits is outlined, under the assumption that only approximate information about delays is available. In Section 1.4 the merits of synchronous versus asynchronous design are discussed.

1.2. Historical background

Switching circuit theory has its roots† in the 1930's when it was developed by the telephone industry in response to the intricacy of the dial telephone system. As a spin-off from this, but also as an answer to the need for tools for calculating routine tasks, the first modern attempts to build computers were made during World War II. There are at least two candidates for the first modern computer: Bell Telephone Laboratory's Model I , and IBM's Mark I (built by IBM, but constructed by the Computation Laboratory of Harvard University). They both had one thing in common: the basic "building blocks" were relays, widely used in the telephone system and powerful enough for the new tasks of computation. As a consequence of this, both were extremely slow and also very limited in their computational power. One interesting difference between the Bell and the IBM machines, is that Mark I was basically a synchronous machine whereas Bell's Model I was asynchronous. In this context a synchronous machine is a machine in which all internal operations are synchronized with one central "clock" (which was a slowly rotating cam in the Mark I). An asynchronous machine, on the other hand, is one in which each part works independently of all the others, and only when information must be transmitted from one unit to another, is some kind of synchronization performed. In Bell's Model I this was accomplished by a "ready" signal, i.e. an extra line that went high when the current operation had completed. It is not surprising that the machine constructed by Bell Laboratories was asynchronous, as the whole telephone network is by tradition (and also by necessity) asynchronous. The telephone company knew the asynchronous design techniques well and solutions from the telephone network could therefore be taken over more or less directly. In many texts these two computers are treated as special machines and not real "computers" — a treatment that is, in many ways, unjust. In any case, the two basic approaches to the synchronization problem were already present in these "0-th" generation computers.

The next breakthrough for the development of computers came after the war when vacuum tube technology had reached a sufficient level of sophistication and reliability so that relays could be replaced by tubes. For switching theory the vacuum tube as a new logic device simplified some problems, but also created some totally new problems. Notably the problem with "glitches", i.e. very short pulses generated by "mistake", became a serious problem. In relay circuits the inertial mass of the contacts worked as a low pass filter and removed short pulses, whereas vacuum tubes were fast enough to react to these pulses and possibly behave incorrectly. One fundamental difference between relays and vacuum tubes is that relay contacts are a bidirectional elements, whereas gates constructed with vacuum tubes have a certain set of inputs and some outputs (normally one).

† For a more complete overview of the early history of computers, the reader is referred to [31].

In this "first" generation of computers, based on vacuum tubes, there were again two different schools regarding the type of synchronization that could give the fastest computer, the synchronous versus the asynchronous approach. The most famous of these computers was the ENIAC [9], developed at the Moore School of Engineering at the University of Pennsylvania in 1943. Most facts about the ENIAC are readily available from the literature, but for the purposes of this thesis one particular detail is worth mentioning. The timing in the ENIAC was provided by a master oscillator operating at 100 kHz. This oscillator drove a ring counter, from which all timing signals were derived throughout the computer. In other words, ENIAC was a typical (although complicated) example of a synchronous switching circuit.

A completely different approach was taken in the construction of the IAS computer, built at the Institute for Advanced Study in Princeton, New Jersey in 1946 [12]. One basic principle that the design team followed was stated as

> sequential control operation with each key operation initiated by the safe completion of the preceding operation

or in our terminology, an asynchronous design principle. The synchronization between different units was accomplished by a "completion" signal, generated either by a "model" circuit or simply by a delay element with a reasonably pessimistic value. Here we can already see one of the objectives for asynchronous circuits, i.e. speed, but we can also see one of the big obstacles in realizing this objective. As always (?) in computer construction, it is the objective of high speed that motivates the use of asynchronous circuits; however it is far from trivial to design an asynchronous circuit that really accomplishes this objective. In the synchronous case, the clock frequency must be adjusted to the *slowest* component, a problem that can be very serious if some component is very much slower than the remaining ones. In the asynchronous circuit, on the other hand, there must exist a certain amount of overhead in the generation of completion signals.

The next generations of computers and switching circuits (transistor circuits and bipolar MSI/LSI circuits) were able to use, for the most part, the theory developed for the vacuum tube gate circuits. Gates constructed with several inputs and one output were still the basic building blocks. Furthermore, since the circuits were easy to change, it was usually possible to "tune" them once and thus obtain close to optimal performance. This ease of design and change lead to a "trial and error" methodology that in many cases was quite sufficient. However with the introduction of custom MOS LSI and VLSI, this possibility of changing or replacing faulty units does not exist any more. Since the cost of design of these VLSI circuits is extremely high (normally many man-years) it has become crucial that a genuine understanding of the underlying theory is present in the construction phase.

Another feature of MOS technology is a return to the old concept of bidirectionality. In particular, MOS pass transistors and transmission gates (in CMOS) are bidirectional; hence there are no distinct input and output terminals for these building blocks. For this reason it has become important to review many of the results generated for relay circuits.

Today, the synchronous design approach seems to be completely dominant, and this is particularly true of VLSI circuits. There are several reasons for this but the most important ones seem to be the ease of designing synchronous circuits and the lack of efficient analysis and design procedures for asynchronous circuits. However, considerable attention has very recently been given to asynchronous self-timed systems (e.g. [10, 23, 26]), and hence the area has once again become of great interest.

1.3. Simulation of VLSI circuits

A fundamental problem with VLSI circuits is that their complexity makes it extremely hard, or perhaps even impossible, to construct a circuit without some design errors. For this reason it is highly desirable to simulate the behavior and to verify the correctness of a given design *before* the chip is produced. A considerable amount of work has already been put into this problem, but no generally accepted "universal" simulation tool has been constructed so far. There are three main types of approaches to simulation, differing in the accuracy of the model of the circuit.

1) Circuit level simulation,
2) Switch level simulation, and
3) Gate level simulation.

In transistor level simulation MOS transistors on a chip are represented by differential equations. This gives a rather accurate model for timing and electrical behavior, but unfortunately involves a substantial amount of computing. For this reason transistor level simulation is only feasible on small portions of the circuit, typically less than 100 transistors at a time. *SPICE* is probably the best known simulator of this type.

The gate level approach, on the other hand, is very efficient, but, since pass transistors and transmission gates create new types of circuits in which the notion of a gate no longer exists, this approach is not particularly practical nor accurate for anything except circuits completely based on the gate philosophy. (Such circuits, however, cannot take advantage of all the features of the MOS technology.) Several attempts have been made to add new elements, in a rather "ad hoc" way, to a gate level simulator to cope with pass transistors (e.g. [17], or [33]) but none has been very successful. In gate level simulation normally the whole circuit can be simulated and hence it might be possible to detect some very subtle communication problems .

A suitable compromise between accuracy and efficiency is provided by switch level simulation. The basic idea is to model each MOS transistor as a switch, and only have a finite (normally very small) set of strengths (basically conductivities) of the transistors. A typical simulator of this class is MOSSIM. This simulator was developed by R.E. Bryant, and appears to be the only one based on sound physical and mathematical principles [3].

Most of the above approaches have one big problem: there is no possibility of checking whether manufacturing tolerances can cause a circuit to malfunction. In other words, very few of the simulation algorithms suggested in the literature can handle the situation when only approximate information is available for the delays in the circuit. Not even in circuit level simulation, where a very exact estimate of the delay in a transistor and wire can be obtained, is there any guarantee that, for example, a $\pm$ 10% variation of some parameters could not completely change the behavior of the circuit. For gate level simulation a three-valued, or ternary, simulation algorithm was proposed by Eichelberger in 1965 [11]. The method has the great advantage of being able to handle the situation as described above, but has not been completely characterized, until 1985. In fact this characterization is the main result of this thesis†. A modified version of the algorithm suggested by Eichelberger was used by Bryant [1] in switch-level simulation, but there are still a number of questions about the results obtained.

1.4. Asynchronous circuits

In this section we present some arguments in favor of using asynchronous circuits. However, first we clarify the meaning of asynchronous and synchronous. We will use the term *asynchronous network* to refer to a general sequential network, i.e. a network upon which there are very few restrictions (if any). The term *synchronous network* is somewhat misleading, and the term *synchronous-mode network* [6] is presumably better. The reason for this is that synchronous circuits are in fact asynchronous sequential networks operating under certain restrictions. These restrictions are normally that all flip-flops are clocked by a central clock, and that all inputs to the flip-flops have to be stable when the clock is "active". In the few cases when we explicitly compare synchronous and asynchronous design, we will use a very simple definition and call all circuits that are controlled by a central clock synchronous, and the remaining circuits asynchronous. This is, of course, not a very precise definition, but it is sufficient for our purposes.

There are at least four major incentives for using asynchronous circuits instead of synchronous circuits which are generally easier to analyze and design. The first is the possibility of increased speed of operation. Since the basic philosophy of asynchronous circuits is to let each unit run independently, i.e. at

† See Chapter IV.

its own maximum speed, it should be possible to build very fast asynchronous systems. However the issue is not that simple. The major obstacle is that, in most systems, there is a relatively large amount of communication between different units, and since the synchronization of the independent units requires a substantial amount of extra communication (e.g. in the form of request-acknowledge signalling), there is a certain trade-off between the gain of speed in individual units and the loss of speed due to communication overhead. Furthermore, in many cases an asynchronous circuit is much more complicated than a synchronous circuit designed to perform the same task, and hence the speed gain obtained by using an asynchronous design might decrease, or even completely vanish.

The second motivation for asynchronous operation is the need for communication between two independent synchronous circuits. Since the two systems operate under the control of different clocks either they must be synchronized (normally by delaying one of them), or an asynchronous interface must be provided.

The third incentive is in interfacing "real world" signals with a system. For obvious reasons, signals coming from the "outside" are completely unrelated to the internal system clock, and hence either the signals have to be synchronized or one has to use asynchronous circuits that react as soon as the signals arrive. A standard way of synchronizing signals is to use a "synchronizer", which is an element used to delay the incoming signal until a clock signal arrives. There are, however, some problems related with synchronizers, most notably problems with flip-flops entering "meta-stable" states [32]. A meta-stable state is a state intermediate between 0 and 1 in which a flip-flop can remain for an unpredictable time. (The problem arises when the clock signal *and* the external signal happen to arrive exactly at the same time.) An interesting property of synchronization failure is that it cannot be removed by clever design. There are many methods to reduce the probability that such a fault would crash a system, but they all cost time and so would reduce efficiency. In [32] Seitz advocates the use of self-timed systems as a solution to this interface problem. In a self-timed system one tries to assure that all system events occur in proper sequence, but nothing ever has to occur at a particular time. This is accomplished by dividing the system into elements which are performing computational steps whose initiation is caused by signal events at their inputs, and whose completion is indicated by signal events at their outputs. In many cases it is advantageous to use asynchronous circuits as these elements. On the other hand, completely asynchronous systems also solve this problem, although they are very difficult to design.

Finally we point out the following advantage of asynchronous design. If, for example, a memory bus is completely asynchronous, the bandwidth of the bus will not be determined by some timing specification (as it would be in the synchronous design), but by the actual speed of the circuits. In particular, it becomes very easy to upgrade such a system simply by replacing a slow unit by a faster one. Note that in such a system one can use a mixture of slow and fast units, without being forced to use “wait-states”.

Chapter II
Analysis of Asynchronous Circuits

2.1. Introduction

In this chapter we give an overview of different models for asynchronous circuits that have been suggested in the literature. In Section 2.2 we describe one of the earliest models — the Huffman model. Although this model was initially developed for relay circuits, it contains some important concepts of more general applicability. In Section 2.3 we describe how this model was modified for use with gate networks. Timing problems constitute a major difficulty in the analysis and design of asynchronous circuits. These problems are discussed in Section 2.4.

The Huffman model for gate networks has the advantage of giving reasonably small tables (to be defined later) describing the behavior of the circuit. However, in Section 2.5 it is shown that the model is incorrect in some cases. A different model, due to Muller, is also described. This model, which associates one state variable to each gate in the circuit, gives a more correct picture of the behavior of the circuit, but has the serious disadvantage of being usually much more expensive computationally. In an attempt to improve the efficiency of analyzing circuits in the Muller model, Fantauzzi suggested a nine-valued algebra and a general method for analyzing certain types of asynchronous circuits. In Section 2.6 this method is outlined and some comments are given about its efficiency and usefulness.

Lastly, in Section 2.7 we describe another attempt (a more useful one) of improving the efficiency of analyzing asynchronous circuits using three-valued logic. The method, due to Eichelberger, is highly efficient and easy to implement as an automatic circuit analyzer. However the results of this ternary analysis were not completely understood. The main result of this thesis is a complete characterization of the outcome of this ternary simulation (see Chapter IV).

2.2. The Huffman model for relay circuits

The first, and still the most commonly used, model of asynchronous circuits is due to Huffman [18]. The basic formulation of the model was for relay circuits, as described below.

We start with a short description of a relay. A relay consists of two parts: the coil, and the contacts controlled by the coil. We will say that a relay is *operated* when current goes through the coil. The contacts can be of two types; *make* and *break*. A make contact is normally open (i.e. no current can go through the contact) when the relay in unoperated, but will close when the relay is operated. A break contact is opposite to a make contact, in that it is normally closed, and opens when the relay operates. In Fig. 2.1 we show the symbols used for denoting make and break contacts.

—— x ——   —— x' ——

(a)   (b)

Figure 2.1. Notation for relay contacts: (a) make; (b) break.

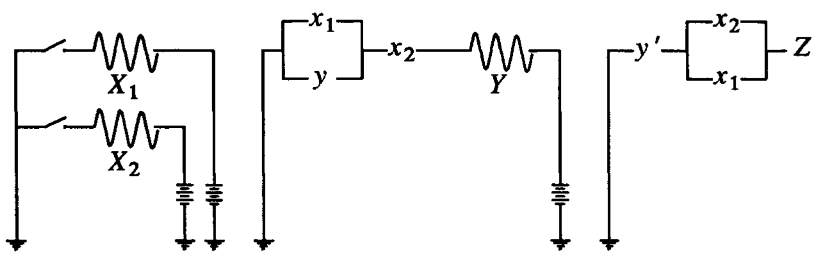


Figure 2.2. Relay network N_1.

In his model, Huffman first introduced *primary* relays $X_1, \cdots, X_n$, i.e. relays which are directly controlled by an input variable. In Fig. 2.2 the relays X_1 and X_2 are primary. All other relays were called *secondary* relays; in Fig. 2.2 relay Y is the only secondary relay. Each secondary relay Y is controlled by a contact network which depends on both the primary and the secondary relays (possibly including Y itself). Huffman labeled these secondary relays $Y_1, \cdots, Y_s$. The output terminals were labeled $Z_1, \cdots, Z_g$. Contacts associated with a given relay R were indicated by a lower-case letter. Normally open contacts were labeled r, whereas normally closed contacts were labeled r'. Since there is a certain delay between the energization of a relay and the contact operation, it is not necessarily true that $r_i = R_i$. However, in a steady state situation we must have $y_i = Y_i$ and $x_i = X_i$. The *total state* of the relay network is given by x, y, where $x = x_1, \cdots, x_n$ and $y = y_1, \cdots, y_s$.

Huffman also defined what he called the Y- and Z-matrices. The Y-matrix is the function table for the the contact network controlling the secondary relays as a function of the total state x, y. The Y-matrix is obtained by constructing a table with all possible input states as columns, and all possible y states as rows. Huffman used originally a "normal" increasing ordering of the states, but later this was changed to the ordering as in a Karnaugh map, where only one variable differs between two neighbours in the sequence. The elements in the Y-matrix are the *excitations* for the secondary relays given the corresponding input and internal state. The Z-matrix is constructed similarly, but with the output values given as functions of the x and y values. In a relay circuit the computation of Y_i, given the total state x, y, consists simply of closing all the make contacts that have the value 1, all the break contact that have the value 0, and determining whether there is a path from the relay Y_i through the contact network to ground. If there is such a path, the value of Y_i will be 1; otherwise Y_i is set to 0. If the excitation entry is the same as the present (row) value, the state is said to be *stable*, otherwise it is said to be *transient*. To illustrate these concepts study the relay circuit N_2 [18] given in Fig. 2.3, with corresponding Y- and Z-matrices in Fig. 2.4 and 2.5 respectively.

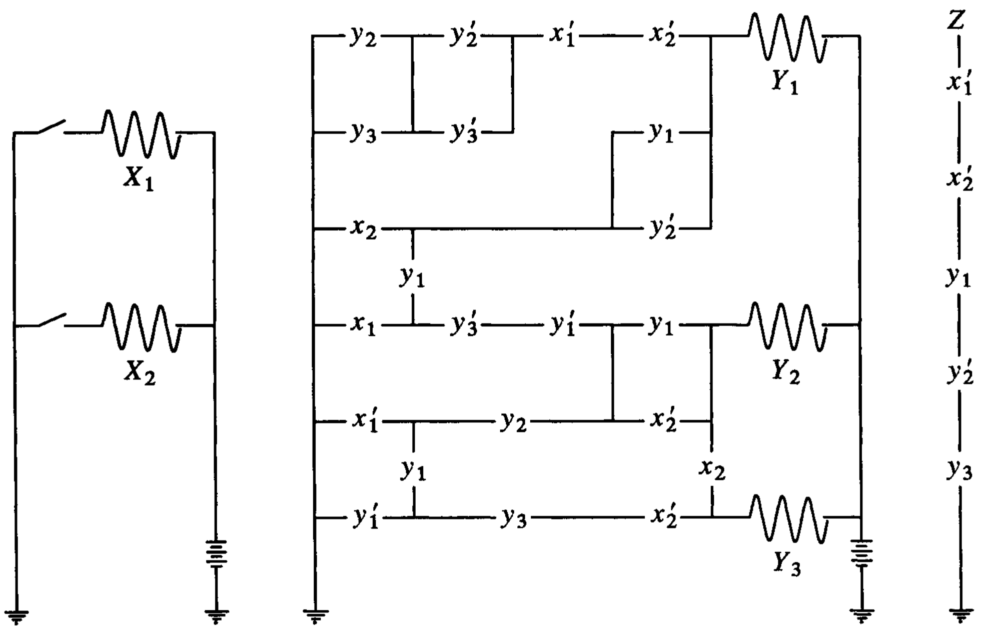


Figure 2.3. Relay switching network N_2.

$y_1y_2y_3$ \ x_1x_2	00	01	11	10
000	000	100	100	010
001	101	100	100	001
011	011	000	000	001
010	110	000	000	010
110	110	111	100	100
111	011	111	100	100
101	101	100	100	100
100	000	100	100	100

Figure 2.4. The Y-matrix corresponding to network N_2.

	x_1x_2			
$y_1y_2y_3$	00	01	11	10
000	0	0	0	0
001	0	0	0	0
011	0	0	0	0
010	0	0	0	0
110	0	0	0	0
111	0	0	0	0
101	1	0	0	0
100	0	0	0	0

Figure 2.5. The Z-matrix corresponding to network N_2.

From the Y-matrix, Huffman constructed what he called the composite transition matrix τ, which is basically the same as the Y-matrix, but with the excitation replaced by a boolean vector $b_1, \cdots, b_s$ in which $b_i = 1$ iff y_i is transient, i.e. the entry corresponding to a secondary relay is 1 iff the relay is not stable. In Fig. 2.6 we give the composite transition matrix for network N_2.

	x_1x_2			
$y_1y_2y_3$	00	01	11	10
000	000	100	100	010
001	100	101	101	000
011	000	011	011	010
010	100	010	010	000
110	000	001	010	010
111	100	000	011	011
101	000	001	001	001
100	100	000	000	000

Figure 2.6. Composite transition matrix τ for network N_2.

Using the composite transition matrix Huffman then discussed so called *race* phenomena. The idea behind a race is that when two or more relays are unstable at the same time, and therefore are "racing" to change, the following can happen. If one of the relays is faster than the other(s), the situation can occur that the faster relay changes, and that change causes one of the other unstable relays to become stable without changing. In other words, in the presence of a race several different secondary relay actions are possible depending on the magnitudes of the operate and release times for the relays involved, and also upon the past history of their excitations. The presence of more than one "1" in the composite transition matrix indicates that at least two secondary relays are simultaneously unstable, and that a race condition exists. Huffman also distinguished between two different types of races; *noncritical* and *critical*. A race is said to be noncritical if each of the alternate possibilities leads eventually to the same *ultimate* stable secondary relay state. Similarly, a race is said to be critical if there are two or more stable secondary relay states that can be possible outcomes of the race. In the remaining part of the model, only non-oscillatory circuits (i.e. circuits that do not oscillate on their own) with noncritical races (if any races at all) were considered.

For these type of circuits, the next important concept that was introduced was the *flow table* F, constructed as follows. All entries in the composite transition matrix that consist purely of zeros, i.e. all entries in which the secondary relays are stable, will have a circle in the corresponding place in the flow table. These circles are then numbered serially; the order of assignment of the numbers is unimportant. The remaining entries of F are uncircled, and each such entry tells what stable circuit condition will ultimately result if the circuit is put in a total relay state corresponding to the entry in question and if the input state remains constant. In Fig. 2.7 we show the flow table derived for the network N_2 of Fig. 2.3.

	x_1x_2			
$y_1y_2y_3$	00	01	11	10
000	①	5	7	3
001	8	5	7	②
011	④	5	7	2
010	9	5	7	③
110	⑨	10	7	6
111	4	⑩	7	6
101	⑧	5	7	6
100	1	⑤	⑦	⑥

Figure 2.7. Flow table for network N_2.

There are some very restrictive assumptions in the Huffman model that, unfortunately, are not always clearly stated. The first is that the so called *fundamental mode* operation [24] is assumed. This means that after some input(s) have changed, no further changes will occur on the input terminals until the circuit has reached a stable state. This assumption is perhaps somewhat severe, but without it (or without some very strict assumptions about the delays in the relays) it would be impossible to predict anything about the behavior of any network.

Another, more subtle, assumption is that there is a *time lag* between the excitation of the relay and the response. If this assumption is false, there could never be any unstable relays and the model is inconsistent.

The third assumption, which is more of an engineering than theoretical assumption, is that each relay has a "smoothing" effect on the input pulses. This smoothing effect is such that pulses shorter than some constant will not affect the relay, and hence the problem with "glitches" (also called hazards — these are temporary incorrect outputs) generated in the combinational network controlling a relay will not affect the relay.

The last assumption, which is not always practical, but does simplify both the analysis and the synthesis procedure, is the assumption of *single input changes*, i.e. it is assumed that only one input variable changes at a time, and that there is sufficient time between changes to allow the network to stabilize.

These assumptions, except possibly for the last one, are normally fulfilled in a relay circuit, and hence the model works quite well for these types of switching networks. However, as pointed out by Brzozowski in 1965 [4], the approach of only studying the races among the secondary relays is not always correct. There can actually exist races between primary and secondary relays, something that is completely missed in this model.

Finally, the following more intuitive model for these secondary relays given by Huffman [18, page 54] is such that we cannot resist to quote the following:

> If we were to endow the secondary relay with human attributes, we could say that each one is a *decision-making* element. After each change in stimulation from its environment, it waits a time to "decide" what the final stimulation is to be, and then reacts accordingly. The operating "hazard" occurs when this "yes-no" decision is based on observation over an insufficient length of time.

2.3. Extension of the Huffman model to gate circuits

Although the Huffman model was originally developed for relay circuits, it was very quickly modified for gate networks. However, some of the assumptions made in the model had to be very carefully examined. In particular the assumption of a lag-time between the excitation and the response had to be satisfied. The method suggested by Huffman was to add *delay* elements at each place where there should have been a secondary relay in a relay implementation. These "places" were called *secondary delay locations*. The second condition that had to be fulfilled in a gate implementation (in particular in early implementations where gates were often constructed with passive elements like resistors or diodes) was that the gain in each loop of the network had to be greater than one. If this condition were not true, the network would cease to be digital, and certainly no longer behave correctly. The general picture for an asynchronous gate network according to the Huffman model is shown in Fig. 2.8. Today, this model with delays inserted is normally called the *feedback delay* model. The reason is obvious from Fig. 2.8.

In the modern version of the Huffman model (e.g. [14, 22, 36]) the following procedure is normally carried out. First the feedback paths are identified and a subset of them are selected as state variables. This subset of the feedback paths is such that if all the feedback paths in the subset are broken, no feedback path remains in the circuit. Here we can see at once the problem with the feedback model of deciding which gate outputs are to be considered state variables. We will return to this in Section 2.5.

Another observation Huffman made was that in many cases the "extra" delays, added according to the feedback delay model, were not necessary, since the delays in the electrical circuits themselves were often sufficient. Hence the problem was to determine if and where delays were necessary to obtain correct behavior. These, and similar problems will be treated more carefully in a later section.

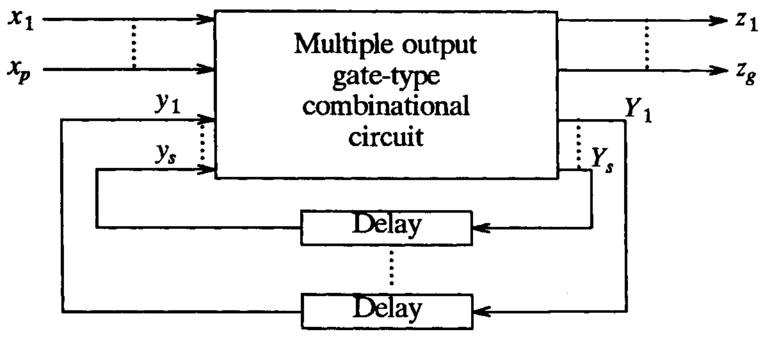


Figure 2.8. Huffman model for asynchronous gate networks.

A problem that became acute when electrical gate circuits were used, was the lack of the smoothing effect, that the inertial mass in the relays provided. As a result, even very short "glitches" could cause malfunctions in the circuit. Huffman suggested a very simple solution; add some circuits that acts as "smoothers" as a replacement of the relays. This approach has the disadvantage of slowing down the circuit, and, since one of the main incentives to change from relays to gate circuits was to gain speed, the solution is not very useful. (The same argument can also be used against "adding" delays in the feedback paths of the circuit.) From the above it is quite clear that there was a great need for solutions that did *not* slow down the circuit. For this reason the study of timing problems in asynchronous circuits received a lot of interest. In the next section we carry out a rather detailed discussion and description of the different timing problems that can occur in asynchronous circuits. The general conclusion is that the study of these problems is rather complicated, and requires great care. In a later section we also show a somewhat different model in which only races have to be considered, an approach that simplifies the analysis substantially.

2.4. Timing problems

2.4.1. Races

The general idea behind the concept of a race in a gate network is basically the same as behind the race condition in the relay version, i.e. two or more state variables are unstable at the same time and therefore the behavior of the circuit involves a certain amount of non-determinism. In the Huffman model, only the races between state variables, i.e. between two feedback paths, were considered. It will be shown later that this is not always correct.

In Fig. 2.9 we show some typical race conditions. For example, if the network is in the stable state $x = 00$, $y = 00$, and the input changes to $x = 10$ there is a critical race in the excitation table (i.e. in the Y-matrix). The reason is as follows. The "next" state is $y = 11$ according to the excitation table; this implies that *both* y_1 and y_2 are unstable and hence a race exists. If the delays for the two state variables are unknown, y might change to 01, 11, or 10. In the last case, y_2 might not change again, since the circuit is stable in $x = 10$ $y = 10$; hence the circuit can end up in state 10. On the other hand, for the first two possibilities the circuit will end up in state 11. In other words, the circuit can end up in two different stable states depending on the delays in the feedback loops. As before, such a race is called critical.

As an example of a noncritical race, study the network in the same stable total state ($x = 00$, $y = 00$) but with the input changing to $x = 01$. Independently of the delays in the feedback loops, the network will end up in the stable state $y = 11$, and hence the race is noncritical.

	x_1x_2			
y_1y_2	00	01	11	10
00	00	11	11	11
01	00	11	01	11
11	11	11	01	11
10	11	11	10	10

Figure 2.9. Race conditions in the excitation table for a network.

2.4.2. Hazards

The first problem with hazards is that there are almost as many definitions of what a hazard is, as there are authors that have discussed the issue. The most general, but also the least precise, definition could be worded like [14]:

> A circuit is said to contain a *hazard* if there exists some possible combination of values of stray delays which will produce a spurious pulse or cause the circuit to enter an incorrect stable state, for some input change.

In this context a stray delay is an unintentional delay in a wire and/or gate, something which is impossible to avoid in any real implementation of a circuit. In some texts the race is considered as a special case of a hazard, but we will adopt the more "standard" view [14, 36, 22], where races and hazards are treated separately.

As said before, there are numerous types of hazards, but one can roughly divide them into four different classes. Each of these classes can then be further divided into subclasses. The four classes are:

1) combinational hazards,
2) multiple-input-change hazards,
3) false output hazards, and
4) essential hazards.

For the first class of hazards, the combinational hazards, the assumption is that only single input changes are allowed. The combinational hazards are divided into two sub-classes: *static* and *dynamic* combinational hazards. The static hazards are further divided into static 1-hazards and static 0-hazards. In Fig. 2.10(a) and (b) we show a static 1-hazard and a static 0-hazard respectively. These are spurious pulses that can occur because of some stray delays in the circuit. In Fig. 2.11 we show an example of a circuit containing a static 0-hazard. The spurious pulse can occur when the input changes from 0 to 1.

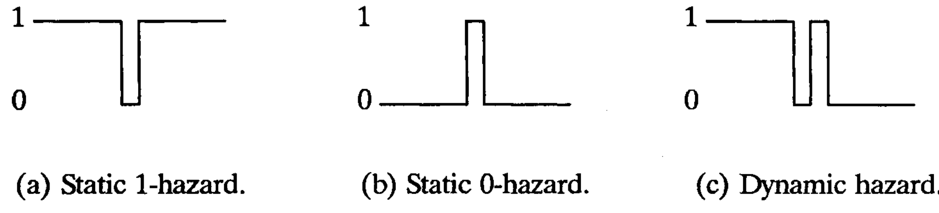


Figure 2.10. Different types of combinational hazards.

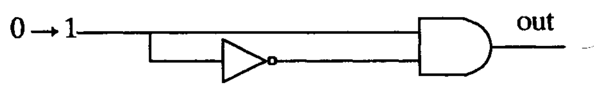


Figure 2.11. Circuit containing a static 0-hazard.

In Fig. 2.10(c) we show one of the two possible dynamic hazards. The basic idea is that a signal, instead of changing once, "oscillates" before the change takes place. In relay circuits this is a very common phenomenon, since mechanical contacts have a tendency to bounce.

McCluskey [24] described necessary and sufficient conditions for the existence of both static 1- and 0-hazards, and dynamic hazards. The notation he used was primarily tailored towards relay circuits, but these (and similar) results are available today for gate circuits in many references (e.g. [14, 36]). A major problem with these types of hazards is that it is often very time consuming to check for all possible hazards in anything but trivial circuits.

A multiple-input-change hazard occurs when more than one input is allowed to change at once. The first one to study these hazards was Eichelberger [11]. He used a slightly different notation than the one presented here, but the basic ideas are the same. For a more complete treatment of the multiple-input-change hazards, the reader is referred to [6]. One interesting characteristic of this type of hazard is that it is frequently unavoidable. Two different types are normally considered: *function* and *logic* hazards. The two types will be introduced by means of an example [14]. Study the Karnaugh map in Fig. 2.12 for a function f of the input variables $x_1, \cdots, x_4$. In Fig. 2.13 we show one possible realization of the function. If the input to the gate network realizing this function changes from $a = 0101$ (with $f(a) = 1$) to $b = 1111$ (with $f(b) = 1$), the circuit may pass through 0111 (i.e. state g) or 1101 (state e) due to stray

	x_1x_2			
x_3x_4	00	01	11	10
00	0^d	1	0	0
01	1	1^a	1^e	1^c
11	0	0^g	1^b	1
10	0	0	0	0

Figure 2.12. Karnaugh map illustrating function hazard.

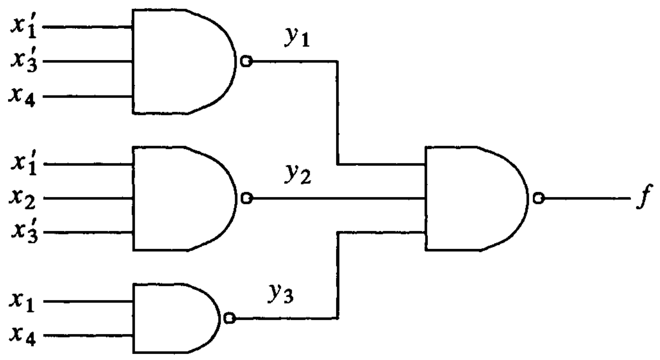


Figure 2.13. One realization containing a logic hazard.

delays. However, $f(g) = 0$; hence the output may temporarily become 0 during the transition between a and b. A hazard like the above, which is inherent in the function definition, is called a *function hazard*, and cannot be removed by normal design of the gate network. The only way to avoid having problems with these function hazards is either to restrict the input changes to only one variable at a time, or insert delays in the gate network in such a way that the transition from one state to another always follows a "safe" route.

In contrast to the above, consider a transition from a (0101) to c (1001) which may pass through 0001 or 1101. The value of the function for both transient inputs is 1. However, in the realization in Fig. 2.13, the following might happen. Suppose the 3-input NAND gates y_1 and y_2 happen to be faster than the 2-input NAND gate y_3. Then, when going from state a to state c via state e, the gates y_1 and y_2 might change from 0 to 1 before the 2-input NAND gate y_3 changes from 1 to 0. Hence, for a short time, the output NAND gate can have all its inputs high, and therefore produce a spurious 0-pulse on its output. As was pointed out by Eichelberger [11] this type of hazard can be removed simply by including an implicant covering both a and c. This type of hazard, is called a *logic hazard*, and can be always removed by proper choice of implicants realizing the function. Note that one can interpret the combinational static hazards as special cases of the logic hazards.

The name *output hazard* refers to any type of hazards that occurs on an output terminal. The importance of these hazards depends on how the outputs are going to be used. If, for instance, they are used to control an elevator, they can probably be neglected completely, but if they are used to trigger a nuclear warhead, they are (hopefully!) considered unacceptable.

The last type of hazard, the *essential* hazard, is a slightly different type, and something unique for sequential circuits. (Note that the previous types of hazards can be present in any type of switching circuit, including combinational circuits.) An essential hazard was defined by Unger [36] as:

For some initial total state and input variable x, three consecutive changes in x take the system to a state that is different from (and not equivalent to) the state reached after a single x-change.

Study for example the network N_3 in Fig. 2.14 [36, page 176], whose flow table is given in Fig. 2.15.

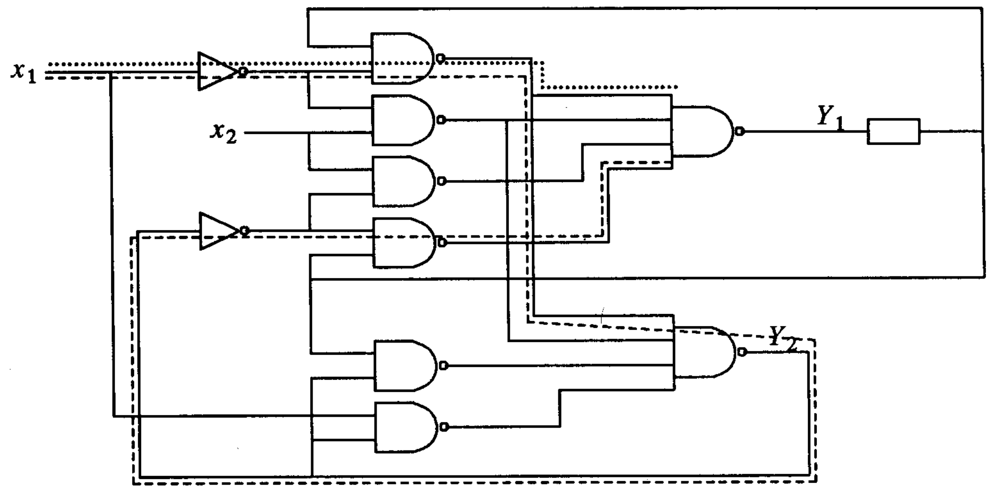


Figure 2.14. Network N_3 with an essential hazard.

	x_1x_2 00	01	11	10	y_1y_2
1	①, 0	3, 0	4, 0	①, 0	00
2	1, 0	3, 0	②, 1	②, 1	01
3	③, 0	③, 0	2, 1	2, 1	11
4	3, 0	3, 0	④, 0	④, 0	10

Figure 2.15. Flow table illustrating essential hazard.

In the following it is shown that there is an essential hazard associated with the total state $x = 10$ $y = 10$ (i.e. 4) and a change in x_1. The first time x_1 is changed (to 0) the network will end up in state 3, i.e. $y = 11$ (no race). Changing x_1 again (now to 1) will put the network in the state 2, i.e. $y = 01$ (no race). Now a third change of x_1 (to 0 again) will put the network in the state 1, i.e. $y = 00$ (no race). Hence, three changes leave the network in a different state than one change, and therefore there is an essential hazard for this particular state and changes.

Essential hazards are a property of the flow table, and hence cannot be eliminated by a clever choice of the combinational circuits. The malfunction that can happen in the presence of an essential hazard is that one portion of the circuit changes in response to an input change, and a second part of the circuit "sees' this secondary response *before* it "sees" the initial change. This second part of the circuit may therefore erroneously initiate a change, so the total result will be as if the input variable changed three times rather than once. An essential hazard may be viewed as a race condition between an input variable and a state variable.

In the example above, the "racing" paths are shown as a dotted and dashed line respectively. The following can happen. If the delay in the wire from the topmost 2-input NAND-gate to the 4-input NAND gate Y_1, is longer than the total delay through the 4-input NAND gate Y_2, the inverter, and the 2-input NAND gate, Y_1 can "see" this secondary change *before* it sees the original change. This in turn can cause Y_1 to change once, but when the "slow" signal from the wire arrives, Y_1 can change again. These changes can propagate back through the feedback lines and hence the network can end up in the "three-change" state instead of the "one-change" state.

In addition to these four basic types of hazards. several differently defined hazards have been discussed in the literature, but they are normally equivalent to one of these four types.

The Huffman model is quite adequate and correct for describing steady-state situations, but it fails to correctly model a circuit under transient conditions. One of the reasons for this failure stems from the fact that the Huffman model does not capture what happens "inside" the combinational network. The analysis of races among feedback variables is a straightforward procedure and involves only studying the flow table. However, verifying that a circuit is free from critical races does not guarantee it will function correctly. This was discovered very early, and the concepts of different types of hazards were introduced to "correct" the model. In other words, the theory of hazards represents attempts to "fix" a rather inaccurate model. However, this theory is not complete and it is hard to know which "ad hoc" methods for detecting hazards, one has to apply to be certain that a circuit will function properly. In the next section we show an example that clearly demonstrates the deficiency of the Huffman model. We also describe an approach which more accurately models the circuit.

2.5. The gate state model

As was shown in the previous section, the feedback line model gives relatively small flow tables (since there are normally few feedback lines in a circuit) and also a relatively simple race analysis, but an extremely complex hazard analysis. However, there are several problems with the feedback model; the most serious is that the result of the analysis depends on the choice of state variables, i.e. on the choice of where to "cut" the feedback lines. As an example of the above pitfall, study the network N_4 in Fig. 2.16 (due to Langdon [20]).

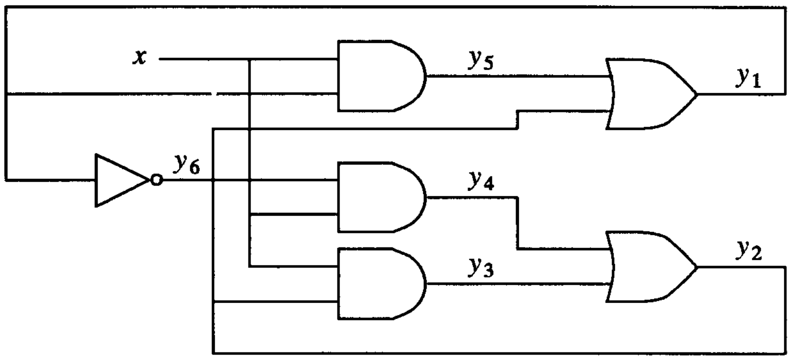


Figure 2.16. Network N_4.

(a)

y_1y_2	$x=0$	$x=1$
00	00	01
01	10	11
11	10	11
10	00	10

(b)

y_1y_3	$x=0$	$x=1$
00	00	11
01	10	11
11	10	11
10	00	10

Figure 2.17. Excitation table for N_4. (a) using y_1y_2; (b) using y_1y_3.

In Fig. 2.17(a) we show the excitation table for the circuit in the case when y_1 and y_2 are chosen as state variables, and in (b) when y_1 and y_3 are chosen. Note that both choices are correct, i.e. break all feedback paths. One can verify that in the first case there are no races, and no static or dynamic hazards. Also, since there is only one input, there cannot be any multi-input-change hazards. Furthermore, if one makes sure that the network starts in state 00 and always works in fundamental mode (i.e. waits until the network has stabilized before changing the input) it is easy to verify that there is no essential hazard. The essential hazard in $x = 1$ $y = 10$ will never affect the network since this state is not reachable. In other words, the analysis according to the Huffman model with y_1 and y_2 as state variables indicates that the network should function properly.

However, study Fig. 2.17(b). By choosing another set of state variables, y_1 and y_3, it is easy to see that, going from the total state $x = 0$, $y_1y_3 = 00$ (which corresponds to the state $y_1y_2 = 00$ in the previous version) and changing x to 1, there is a race. For this race the circuit can end up in either 11 or 10. Hence, with a different choice of state variables there is all of a sudden a critical race! (Note that this occurs even if the above initial state assumption is obeyed.) Clearly, this indicates a very serious problem with the Huffman model (or at least when the "extra" secondary delays are omitted). What is somewhat surprising is that even very recent textbooks in Digital Design, e.g. [22], still advocate the feedback model, and do not even mention these problems with the model.

This drawback with the Huffman model, was pointed out already in 1961 by Muller [28] and has been mentioned by several authors later [20, 5, 7]. As was shown by Brzozowski and Yoeli [7], even in such a simple circuit as an ordinary RS flip flop constructed with two 2-inputs NOR gates, the feedback model is inadequate and predicts different results depending on which feedback lines are chosen as state variables.

The basic approach to "correct" the model is to increase the number of state variables. However, there is the question of how many state variables are necessary, and which ones should be chosen. In general the underlying model for the network determines which state variables are necessary. For example, under the assumption that only gates have delays, it might be necessary to have one state variable associated with each gate. On the other hand if both gates and wires are assumed to have delays, it might be necessary to have one state variable for each gate *and* each wire in the circuit. The problem of determining the minimal number of state variables necessary to correctly describe a network is still an open problem.

The advantage with the Muller model is that the main timing problem is associated with races. In other words, the Muller model properly predicts transitions among states. However, one still has to analyze the circuit for output hazards. This information can be obtained from the binary race analysis by observing the transient output values. There remains the problem of efficient detection of output hazards, but this is outside the scope of the present thesis.

To illustrate the analysis using all gate outputs as state variables consider the network N_4 of Fig. 2.16. The excitation table is given in Fig. 2.18. As can be seen (with a lot of work), there are several critical races. For example, starting in the total state $x = 0$ $y = y_6y_5 \cdots y_1 = 100000$ and changing x to 1 can result in the following two sequences (among others):

	x	
$y_6y_5y_4y_3y_2y_1$	0	1
000000	100000	100000
000001	000000	010000
000010	100001	100101
000011	000001	010101
000100	100010	100010
000101	000010	010010
000110	100011	100111
000111	000011	010111
001000	100010	100010
001001	000010	010010
001010	100011	100111
001011	000011	010111
001100	100010	100010
001101	000010	010010
001110	100011	100111
001111	000011	010111
010000	100001	100001
010001	000001	010001
010010	100001	100101
010011	000001	010101
010100	100011	100011
010101	000011	010011
010110	100011	100111
010111	000011	010111
011000	100011	100011
011001	000011	010011
011010	100011	100111
011011	000011	010111
011100	100011	100011
011101	000011	010011
011110	100011	100111
011111	000011	010111

	x	
$y_6y_5y_4y_3y_2y_1$	0	1
100000	100000	101000
100001	000000	011000
100010	100001	101101
100011	000001	011101
100100	100010	101010
100101	000010	011010
100110	100011	101111
100111	000011	011111
101000	100010	101010
101001	000010	011010
101010	100011	101111
101011	000011	011111
101100	100010	101010
101101	000010	011010
101110	100011	101111
101111	000011	011111
110000	100001	101001
110001	000001	011001
110010	100001	101101
110011	000001	011101
110100	100011	101011
110101	000011	011011
110110	100011	101111
110111	000011	011111
111000	100011	101011
111001	000011	011011
111010	100011	101111
111011	000011	011111
111100	100011	101011
111101	000011	011011
111110	100011	101111
111111	000011	011111

Figure 2.18. Excitation table for network N_4 using the Muller model.

a) 100000 $\rightarrow$ 101000 $\rightarrow$ 101010 $\rightarrow$ 101111 $\rightarrow$ 011111 $\rightarrow$ 010111 (stable)

b) 100000 $\rightarrow$ 101000 $\rightarrow$ 101010 $\rightarrow$ 101011 $\rightarrow$ 011011 $\rightarrow$ 010011 $\rightarrow$ 010001 (stable).

Note that the above excitation table will contain all the possible races (under the assumption that *only* the gates have delays). The excitation table is very large and hard to read. Although it is very tedious to find out if a race is critical or noncritical this can be done mechanically by a simple program. On the other hand, complete hazard analysis in the feedback model involves a lot of "intelligent" observations. Unfortunately, when the network grows and the number of state variables (i.e. the number of gates) increases, the excitation table grows exponentially. Thus the excitation table is only feasible for very small networks.

Brzozowski and Yoeli [7] suggested in 1976 a more practical approach to the analysis. The basic idea is that in "normal" circuits there are only a few stable states even if there are potentially extremely many states. By only calculating the necessary transitions, make some observations about gates and inputs that can determine the output of a gate on their own (called *forcing* signals in their paper) and also taking advantage of symmetry (if present), the amount of work can be substantially reduced. Unfortunately, their method relies heavily on "intelligent" observations and hence is difficult to implement as a computer program. Since the work involved is substantial for a reasonably large network, computer aids are vital.

In summary, the Muller approach of using all the gates as state variables requires a rather simple race analysis, but the excitation table becomes impossible to handle very quickly. There are methods of reducing the work, but these solutions are not easily adapted for computer implementation, and are of limited interest. In the next sections we will show two additional approaches to this problem.

2.6. Race and hazard detection using multivalued logic

Fantauzzi [13] suggested in 1974 the use of a nine-valued logic for detecting hazards and races in a gate network. The idea was to have several logic values representing steady states (0 and 1), changing values (+ and -), static hazards (S1 and S0), and dynamic hazards (D+ and D-). In addition to these eight values, a ninth value (*) was introduced denoting an unknown value.

The following restrictions were imposed on the network to be analyzed: only inverters, AND-, NAND-, OR-, NOR-gates, and RS flip-flops were allowed as construction elements. Furthermore, if all the feedback paths in the RS flip-flops were broken, then no feedback path remained in the circuit (hence a lot of asynchronous circuits were not considered!).

Fantauzzi viewed each gate in the network as an operator, operating on its input values and generating an output value. He gave function tables for a number of different circuits. As an example, the function tables for an inverter, a two input AND-gate and a RS flip-flop constructed with *NAND* gates, are shown in Figs. 2.18, 2.19 and 2.20 respectively. The notation is as follows: 0 and 1 denote the usual stable binary values, + and – denote signals that are changing from 0 to 1 and 1 to 0 respectively, S0 and S1 denote possible static 0-hazards and static 1-hazards, and D+ and D- denote possible dynamic hazards — the former in a $0 \rightarrow 1$ change, the latter in a $1 \rightarrow 0$ change. The * is a special value indicating an "unknown" value.

The method will be introduced by means of an example. Study network N_5 of Fig. 2.22 [22]. If we start in the total stable state $x_1 \cdots x_4 = 0011$ and $y_1 y_2 \cdots y_7 = 1010011$, and change x_3 to 0, i.e. the new input vector is $x_1 \cdots x_4 = 0001$, the following procedure is carried out. Since x_3 is changing

input	output
0	1
+	-
S0	S1
D+	D-
*	*
1	0
0	+
S1	S0
D-	D+

Figure 2.19. Function table for inverter according to Fantauzzi.

	0	+	S0	D+	*	1	-	S1	D-
0	0	0	0	0	0	0	0	0	0
+	0	+	S0	D+	*	+	S0	D+	S0
S0	0	S0	S0	S0	S0	S0	S0	S0	S0
D+	0	D+	S0	D+	*	D+	S0	D+	S0
*	0	*	S0	*	*	*	*	*	*
1	0	+	S0	D+	*	1	-	S1	D-
-	0	S0	S0	S0	*	-	-	D-	D-
S1	0	D+	S0	D+	*	S1	D-	S1	D-
D-	0	S0	S0	S0	*	D-	D-	D-	D-

Figure 2.20. Function table for two input AND gate according to Fantauzzi.

S\R	0	+	S0	D+	*	1	-	S1	D-
0	1	-	S1	D-	*	0	+	S0	D+
+	1	*	S1	*	*	0	+	*	D+
S0	1	-	S1	D-	*	0	+	S0	D+
D+	1	*	S1	*	*	0	+	*	D+
*	1	*	S1	*	*	*	*	*	*
1	1	1	1	1	*	1/0	1/+	1/*†	1/+
-	1	-	S1	D-	*	-/0	S1/+	D-/S0	S1/D+
S1	1	*	S1	*	*	-/0	S1/+	*	S1/D+
D-	1	-	S1	D-	*	-/0	S1/+	D-/S0	S1/D+

Figure 2.21. Function table for the Q output of the operator associated to a NAND RS flip-flop with one S and one R input only. The notation a/b means a if Q was at 1 and b if Q was at 0.

† Possible undesired change. Care must be taken to ensure correct operation.

from 1 to 0, the corresponding input is assigned the value $-$. Gate output y_1 will remain 1, since x_1 and x_2 are still 0. Gate y_2 will operate on $-$ and 1; from the tables in Fig. 2.19 and 2.19 it follows that the outcome will be $+$. Similarly, y_3 will remain 1, y_4 will become $+$, and y_5 will become $+$. Finally, for the flip-flop, the results according to Fig. 2.21 will be $y_6 = *$ Hence the outcome of the analysis is that the flip-flop can end up in an unknown state.

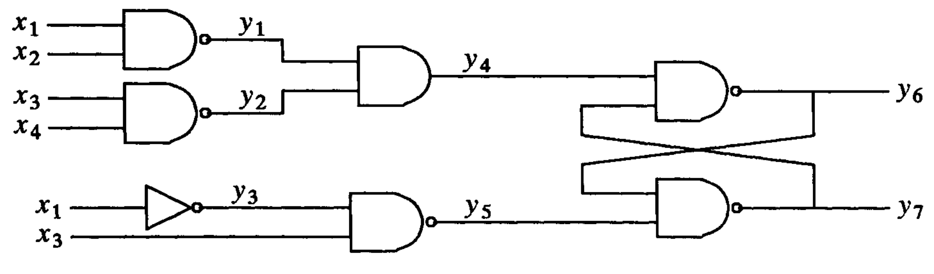


Figure 2.22. Network N_5.

As was mentioned earlier, many asynchronous circuits are not analyzable with this model because of the assumption of no feedback lines except in the RS flip-flops. This, together with the very complicated way of calculating the "excitation" of a gate (or in Fantauzzi's terminology the operator function), severely limits the usefulness of the method. In particular, extending the function tables above to other gates with more inputs, is not trivial. However, the method has the advantage of detecting hazards in very big combinational circuits very efficiently (since only one calculation is needed for each gate in the network). In summary, the method uses an interesting concept, but its applicability is limited. Since ternary simulation (to be treated in the next section) offers the same efficiency without the severe limitations, Fantauzzi's method is not attractive.

2.7. Ternary simulation techniques

Several authors have proposed the use of three-valued, or ternary, logic in the analysis of switching circuits. One of the first to suggest the use of a third value was Montgomerie [27] in 1948, as far as the author is aware. He studied relay circuits and suggested that a relay contact could be in any one of three states; operated, unoperated or "neutral" (i.e. between the operated and unoperated state). The method was primarily used in the synthesis phase, although the basic idea was quite applicable to the analysis phase as well. In relay circuits this assumption of three states was very natural; relay switching contacts (i.e. contact springs that switch between two different contacts) certainly do have an (unstable) intermediate position between the two end positions. However, the method suggested had little to do with the way ternary algebra was used later, and is more of historical interest.

In 1959 Muller [29] suggested the use of a third logic value in the analysis of switching circuits. The paper was only a very short note, and dealt mainly with the definition of AND, OR, and inverter operations with three valued operands. In particular, the application of the method to circuit analysis was not treated at all.

In 1964, Yoeli and Rinon [37] developed a formal mathematical theory for using ternary algebra for hazard detection. Like Muller [29], they used, the third value (½) to indicate an unknown, but binary value. In other words, if a state variable had the value ½, then it was assumed that it could either have the value 0 or the value 1. Yoeli and Rinon developed a ternary switching algebra in which they defined the addition, multiplication and inversion operations (i.e. OR, AND and inverter operations). The definitions were the same as Muller's [29], and are given in Fig. 2.23.

+	0	½	1
0	0	½	1
½	½	½	1
1	1	1	1

·	0	½	1
0	0	0	0
½	0	½	½
1	0	½	1

x	0	½	1
$\bar{x}$	1	½	0

Figure 2.23. Addition, multiplication and inversion defined for ternary operands.

They also defined ternary functions which were extensions of the "normal" boolean gate function. In their model, they assumed that the network function was realized using AND, OR and inverter circuits, and hence the value of the network function could be obtained by "recursively" applying the definitions above. The method of detecting hazards was basically along the same lines as the methods suggested by McCluskey [24], but was adapted to make use of the special properties of the ternary algebra.

In 1965 Eichelberger [11] suggested how to extend the ideas of Yoeli and Rinon to hazard and race detection in sequential networks. In this model it was assumed that the value ½ represented a changing or unknown value. Eichelberger used the Huffman model (i.e. feedback model), so the main objective was to study the behavior of the state variables of the feedback lines. The proposed method consisted of two algorithms applied one after the other. The first, called Procedure A was intended for determining which state variables might change, was defined as follows [11]:

Procedure A:

With the changing x variables equal to ½ and all other x and y variables as originally specified, evaluate the Y_i functions to determine if one or more have changed from 1 or 0 to ½. If one or more Y_i functions have changed from 1 or 0 to ½, change the corresponding y_i variables from 1 or 0 to ½ and repeat the process until no additional changes in the Y_i functions are determined.

The procedure is based on one very crucial observation: a state variable can only change from a binary value to a ½. The reason is that the ½ represents an unknown value, i.e. the corresponding variable can have either binary value, and the definitions of the ternary AND, OR and inverter gates are such that increased uncertainty on the inputs can only produce increased (or the same) uncertainty on the output. Hence, it follows that Procedure A always halts after a finite number of steps (actually after at most m steps, where m is the number of feedback lines). Eichelberger claimed that all state variables that might experience *some* type of change, would have the value ½ after Algorithm A.

The second part of the analysis, called Procedure B, intended to determine which state variables would eventually stabilize. The procedure was as follows:

Procedure B:

With the changing x variables equal to their new values (1 or 0), and all other x and y variables equal to their values at the end of Procedure A, evaluate all Y_i functions. If one or more of these Y_i functions changes from ½ to 1 or 0, change the corresponding y_i variable from ½ to 1 or 0 and repeat the process until no additional changes in the Y_i functions are determined.

This part of the algorithm is based on a "dual" observation: a state variable can only change from a ½ to a binary value. The reason is once again based on the definitions of the ternary AND, OR and inverter functions. In this case, decreased uncertainty on the inputs can only produce decreased (or the same) uncertainty on the outputs. The method suggested was very useful, and was quickly implemented in computer programs for analyzing switching circuits [19].

One thing to note is that Eichelberger used the Huffman model, but in the end of the paper he concluded that that was by no means necessary. Actually, when it came to describing the method of evaluating the ternary gate function, he essentially used the Muller approach. There were two problems associated with the Eichelberger approach. The first was that the formulation of the method was for the Huffman model, which is not always accurate. The second problem was that it was very unclear how to interpret the results of the method. In other words, the method was highly efficient in getting information about a circuit, but it was not clear what this information represented. (This situation is in many ways similar to the situation today when it comes to switch-level simulators for VLSI circuits!)

Brzozowski and Yoeli [8] were the first to formalize and to attempt to characterize the results obtained by the ternary algorithm. They modified the algorithm to correspond to the Muller model. Furthermore, they tried to compare the ternary analysis with a binary race analysis. However, they were only partially successful — they only managed to show that the ternary analysis "covered" the binary race analysis. In this context covers means that whenever the ternary analysis predicts a binary outcome on a gate, the binary analysis gives the same binary result. They also gave a number of examples where the binary race analysis predicted a unique binary value, whereas the ternary ended up with a ½. From these examples, they stated a conjecture about complete correspondence. The conjecture said essentially that the ternary simulation and the binary race analysis should give the same result if one assumes that both gates and wires have delays.

To conclude this section, we show the ternary analysis of a reasonably large network N_6 of Fig. 2.24 [11]. It is easy to verify that the total state $x_1x_2 = 00$, $y_1 \cdots y_8 = 11000000$ is stable. The question is what will be the result if the input changes to $x_1x_2 = 11$? In Fig. 2.25 we show the results after each "step" in the ternary analysis.

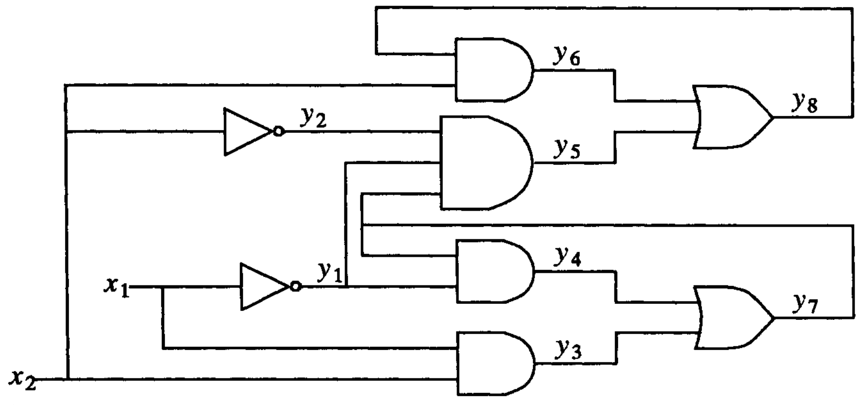


Figure 2.24. Network N_6.

Algorithm	Step	x_1	x_2	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
A:	1	½	½	1	1	0	0	0	0	0	0
A:	2	½	½	½	½	½	0	0	0	0	0
A:	3	½	½	½	½	½	0	0	0	½	0
A:	4	½	½	½	½	½	½	½	0	½	0
A:	5	½	½	½	½	½	½	½	0	½	½
A:	6	½	½	½	½	½	½	½	½	½	½
A:	7	½	½	½	½	½	½	½	½	½	½
B:	1	1	1	½	½	½	½	½	½	½	½
B:	2	1	1	0	0	1	½	½	½	½	½
B:	3	1	1	0	0	1	0	0	½	1	½
B:	4	1	1	0	0	1	0	0	½	1	½

Figure 2.25. Ternary analysis of network N_4.

In the next chapter we describe different delay models and binary methods for analyzing races. Among the described methods is the General Multiple Winner model. In Chapter IV, we will show the correspondence between the ternary simulation described above, and the binary race analysis according to the General Multiple Winner model.

Chapter III
Gate Network Model

3.1. Introduction

In this chapter we will focus on the different delay models for asynchronous circuits that have been suggested in the literature. We will also study different methods of analyzing races according to these delay models. We will use the Muller model (i.e. the gate state model) for reasons explained in Chapter II. In Section 3.2 we describe the delay models that have been proposed. We then continue in Section 3.3 by describing and criticizing different methods for analyzing races. We conclude the chapter by giving a precise mathematical description of one model of a network. This model will be used throughout the remaining parts of the thesis, and, in particular, in the characterization of the ternary simulation in Chapter IV.

3.2. Delay models

There exist several different models for taking into account different types of delays in an asynchronous circuit. The basic questions about a delay model are: locations of delays, their relative magnitudes, the types of delays, etc. Note that, in this section, our primary interest lies in stray delays, i.e. delays that are not explicitly inserted. Delays inserted by a designer will be called *delay elements*.

There are two different types of models of the delay itself. In the first model, the delay is called *pure*. The output from a pure delay is assumed to be the exact replica of the input, only delayed some time Δ. More formally, if the input to the pure delay is some function $f(t)$, (where t denotes the time), then the output is $f(t-\Delta)$. The most severe objection to this model is that no real circuit can act in such an ideal way; in particular, very short pulses will almost certainly not be transmitted through a wire or a gate without being distorted. This is because there is always a certain amount of inductance and capacitance associated with a wire, and hence the wire will work as a low pass filter, blocking very short pulses.

A very natural modification of the pure delay model, is to incorporate this filtering in the model. One model in which this has been done is the *inertial* delay model. This model assumes that a delay behaves in the following way. If an input pulse arrives and the pulse is shorter than Δ time units, then the output of the delay does not change. For all other pulses the output will be a replica of the input pulse, but delayed Δ time units. The inertial delay model is unquestionably the most commonly used in the analysis of networks, even if some of its assumptions can be argued.

Several delay assumptions have been suggested in the literature, but we will only describe a few representatives. One problem in describing the different approaches that have been taken, is that the delay assumptions and the analysis method normally go hand in hand, and few authors make any clear distinction between the two. For this reason, in many cases the naming convention is somewhat confusing and far from consistent. (The same problem will also be quite obvious in Section 3.3 where we describe different methods for analyzing races according to the different delay assumptions.)

The first, and perhaps the most general, model is the *unbounded delay* model [25]. The model was suggested by Miller in 1965, but was only briefly mentioned since it was not considered particularly realistic. As the name indicates, the basic assumption is that any delay in the circuit can have an arbitrary, and in the most general case, even infinite delay. Furthermore, no restrictions were made about the types of delays, and hence both pure and inertial delays could exist in the same circuit.

A second delay model is the *bounded delay* model, suggested by Unger [35]. In this model each stray delay is bounded above by a given value T. Normally, all stray delays are considered inertial and furthermore delay elements are permitted. These delay elements are allowed to have delays greater than T. For this delay assumption an extensive set of methods have been developed by Unger [35] and others (e.g. [16, 15]), for the synthesis of race- and hazard-free circuits. The main problem is that all of the methods suggested, require some delay elements. These delay elements will unfortunately slow down the circuit.

Yet another delay model is the so called *speed-independent* model [25], defined by Muller in 1959. In this model the basic assumption is that only gates have stray delays. Furthermore, these delays are considered inertial. Also, the delay in each gate is assumed to be arbitrary, but finite. The model does not use any delay elements. Muller [25] developed an extensive theory for the analysis and design of speed-independent asynchronous circuits.

All the previous models have one thing in common: they are all very "pessimistic", because they assume arbitrary delays. In an attempt to find a less pessimistic model Langdon [20] suggested a model he called the *bounded-ratio* delay model. The idea was to take into account the fact that in real circuits, there is always a nonzero delay in any wire or gate, but it is also almost always possible to give upper bounds on the delays. The model assumes that the maximum sum of a gate and a wire delay divided by the minimum sum of a gate and a wire delay is bounded by some constant K. One immediate consequence is that no single wire delay can exceed a chain of K wire delays. This assumption greatly simplifies the analysis and synthesis of a circuit, since only "reasonable" paths need to be considered for hazards and races.

The very last model we will describe is somewhat different. In 1975 Brzozowski and Yoeli [5] defined what they called an *Almost Equal Delay* model. Their interpretation of the delay model [6], makes four assumptions:

1. Only gates are assumed to have delays.
2. The delay of any gate is approximately Δ units of time.
3. All delays are inertial, and
4. It was assumed that $\Delta_i < \Delta_j + \Delta_k$ for any delays Δ_i, Δ_j, and Δ_k in the network.

The Almost Equal Delay model seems to rely on some very severe assumptions, but in some cases (e.g. for networks built completely with the same type of gates) the assumptions are probably valid. Unfortunately, as we will see in the next section, the method suggested for analyzing races according to this model, relies on a much stronger assumption about the delays. This will be discussed in greater detail in Section 3.3.

3.3. Methods for race analysis

In this section we will assume that the Muller model (gate state model) is used unless otherwise stated. The results do not depend on this, but by using this approach, only races have to be considered as timing problems. (Output hazards can still occur, but since the output values are part of the total state, it is possible to detect them in the race analysis.)

The basic problem of analyzing asynchronous circuits is introduced by means of an example [8]. Consider the network N_1 in Fig. 3.1. (Note that the circuit is chosen more for demonstrational purposes than for its applicability.)

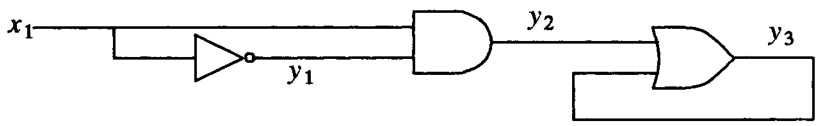


Figure 3.1. Network N_7.

The behavior of the network is governed by the following gate excitation equations:

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

where Y_i gives the value of the boolean function computed by the gate i whose output is y_i, i.e. it is the excitation of the gate. We will say that a gate j is stable if $Y_j = y_j$. It can easily be verified that the total state $x = 0$, $y = (y_1, y_2, y_3) = (1, 0, 0)$ is stable (i.e. $Y_1 = y_1$, $Y_2 = y_2$, and $Y_3 = y_3$). Suppose now that the input changes to $x = 1$; what will be the final state of the network? This question constitutes the basic problem in the analysis of asynchronous sequential networks. For obvious reasons, the answer depends largely on the underlying delay assumptions. In order to explain the different methods, the concept of an *allowed sequence*, as defined by Muller [25], will be used. The idea is to describe the behavior of a circuit by giving a sequence of total states of the circuit. In this sequence a state b can follow a state a iff b can be obtained by changing at least one unstable gate in state a, and this change is allowed according to the method studied. Since different allowed sequences can start at the same total state, we will normally draw a directed graph showing all possible successors of a state. One can view this as defining a binary relation between states "reachable" from each other. (This approach follows very closely the one taken by Muller [25].)

We will briefly describe four different methods of race analysis:

a) the maximum winner model,

b) the single winner model,

c) the general multiple winner model, and

d) the almost equal delay model.

It is important to remember that these methods are based on *mathematical* models, usually simplified (to be manageable) to such a degree that the correspondence between the real behavior of the circuit, and the predicted behavior according to the method, is sometimes questionable.

The first, and unquestionably the simplest, method is to assume that all unstable gates change simultaneously. We will call this the "Maximum Winner Model" (MWM). In the literature this method normally goes under the name "unit delay model". The former name reflects the way of computing the next state, whereas the latter refers to the underlying delay assumption. In this model only gates are assumed to have delays, all delays are *exactly* the same, and the delays are pure. However, as far as the author is aware, nobody has given any motivation for this method. The only motivation we can suggest, can be summarized in one word: simplicity. In particular, it is very easy to use this approach in a simulator program. However, the assumption about exactly equal delays is unrealistic, and hence it is not very clear how the results should be interpreted. In Fig. 3.2 we show the analysis of Network N_7 of Fig. 3.1 according to the Maximum Winner Model. Unstable gate outputs are shown underlined in the graph. Note that the Maximum Winner Model predicts that the network will end up reliably in the stable state $y = 001$.

A more sophisticated, but also more complicated, race model was suggested by Muller [30] in 1967. The model was called the "General Single Winner" model (GSW) by Brzozowski and Yoeli [5], and we will continue to use that name. As the name indicates, the model assumes that any one of the unstable gates can change, but with the extra condition that only one of them can change at the same time. In other words, two states a and b related by the GSW relation can only differ in one gate value. The model is not necessarily limited to the case when only gates have delays, but that is the most common assumption. The delay model behind the GSW model, is a slightly restricted speed-independent model. The restriction is the assumption that two delays cannot be exactly equal. In Fig. 3.3 we show the race analysis of network N_7 of Fig. 3.1 according to this model. As can be seen, the model predicts two possible stable

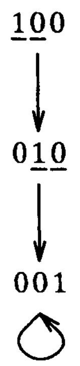


Figure 3.2. Race analysis of N_7 according to MWM.

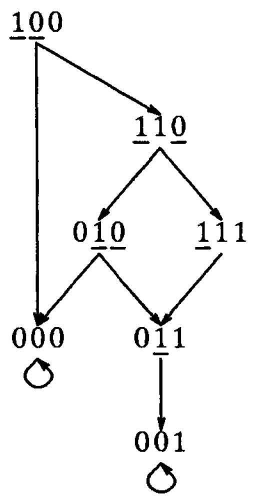


Figure 3.3. Race analysis of N_7 according to the GSW model.

outcomes, $y = 000$ and $y = 001$. This outcome seems to be reasonable, at least from an intuitive point of view. For example, if the inverter is much faster than the AND gate, it might change to 0 before the AND gate reacts. Hence, the state $y = 000$ should also be a possible outcome.

The assumption of only one "winner" in any race in the GSW model is not well justified, and the main reason seems to be to simplify the analysis. The obvious extension of the GSW model is to allow more than one gate to "win" a race. Such a model was actually defined earlier than the GSW model; Muller [25] defined this model already in 1965, but did not give it any name. For reasons of symmetry, we will use the name "General Multiple Winner" model, or GMW for short (this name was suggested by Brzozowski and Yoeli in

1975 [5]). In the GMW model, any nonempty subset of the unstable gates can change at once. This gives a rather complicated race analysis if more than two gates are unstable at some point, but this is the most general case of a race. The underlying delay model is the speed-independent model. As with GSW, it is applicable to both gate and wire delays. However, since the analysis becomes extremely large when many state variables are present, the normal assumption is that only gates have delays associated with them.

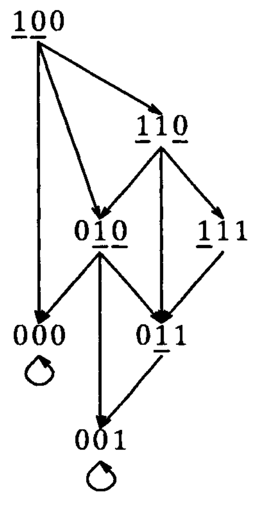


Figure 3.4. Race analysis of N_7 according to the GMW model.

In Fig. 3.4 we show the GMW analysis of Network N_7 of Fig. 3.1. In this case the result of the GMW analysis and the GSW analysis gives the same non-transient outcome, but the following example shows that that is not always the case. Study the NOR latch of Fig. 3.5. In Fig. 3.6(a) we show the GSW analysis of the latch, and in Fig. 3.6(b) we show the GMW analysis. Note that the GSW model predict that the outcome of the race will either be $y = 01$, or $y = 10$ — both stable states. However, the GMW analysis also indicates that the latch can enter an oscillation (between $y = 11$ and $y = 00$).

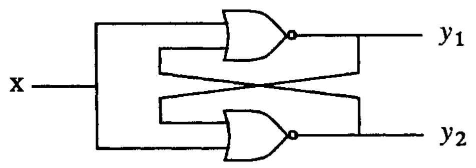


Figure 3.5. NOR latch.

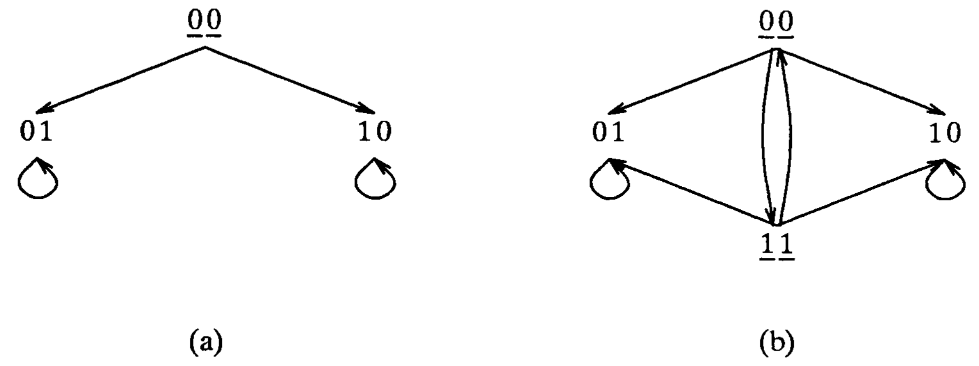

$\underline{0}\underline{0}$
01
10
(a)
$\underline{0}\underline{0}$
01
10
$\underline{1}\underline{1}$
(b)

Figure 3.6. Race analysis of NOR latch: (a) GSW analysis; (b) GMW analysis.

The last, and most likely the least well known, method described, is due to Brzozowski and Yoeli [5, 6], and is called the "Almost Equal Delay" method, or AED for short. The method is an attempt to decrease the pessimism in the GSW/GMW methods. This pessimism stems from the fact that in a GSW/GMW analysis one gate can be unstable for a very long time before changing. Consider the following case. Two gates a, and b are racing. One of the gates, say a, changes, and the other remains unstable. Now gate a that changed, might cause another gate c to become unstable. In the race that follows (between b and c) c is allowed to "win" over b. When gate c changes, it can cause a fourth gate d to become unstable, etc. What can happen is that to reach a specific state, one of the gate delays must be substantially greater than all the others (in the previous case, gate delay b). To solve this problem, the AED model simply assumes that all gates have *approximately* the same delay. How realistic this assumption is could certainly be argued, but for certain types of circuits, the assumption might be quite valid. (For example, consider the case when all gates are of the same type.) We will start by giving an informal description, mainly by means of an example. Later we will give the precise mathematical definition of the AED relation. The approach we will take follows very closely the original method as described by Brzozowski and Yoeli [6].

The basic idea behind the AED method is to make sure that all gates that are racing must all change, or become stable (because of some other unstable gate changes) *before* any "new" gate can enter the race. This will be accomplished by adding another field to the total state, indicating which gates are unstable *and* are allowed to change according to the model. We will call such extended total state a *race state*, and denote it by $<y, S>$, where y is the total state of the network, and S is the set containing all unstable gates that are allowed to change according to the model. To simplify the analysis, the assumption of a single winner is adopted. However, the method can be easily extended to multiple winners. In Fig. 3.7 we show the race analysis according to the AED method, of the network N_7 of Fig. 3.1 Note that in the race state $<\underline{1}1\underline{0}, \{1\}>$ only gate 1 is allowed to change, despite the fact that also gate 3

$$\begin{tikzcd}
<\underline{1}\underline{0}0, \{1,2\}> \arrow[dr] \arrow[ddd] & \\
& <\underline{1}1\underline{0}, \{1\}> \arrow[d] \\
& <0\underline{1}\underline{0}, \{2,3\}> \arrow[dl] \arrow[d] \\
<000, \emptyset> \arrow[loop below] & <0\underline{1}1, \{2\}> \arrow[d] \\
& 001, \emptyset> \arrow[loop below]
\end{tikzcd}$$

Figure 3.7. Race analysis of N_7 according to the AED method.

has become unstable. The reason is that gate 1 has been unstable for much longer time than gate 3, and therefore has higher "priority" to win the race. In this case the AED result was the same as the GSW result, but this is clearly not always the case. Consider for example the network N_8 of Fig. 3.8.

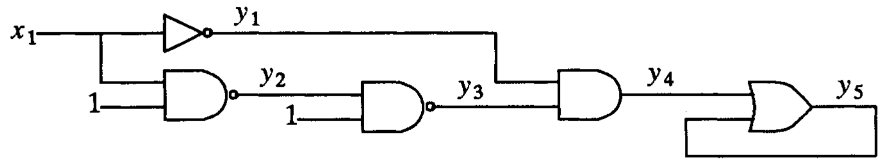


Figure 3.8. Network N_8.

In Fig. 3.9 we give the race analysis according to the GSW model and the AED model. Note that the results are not equal. The GSW model includes the case when the total delay through the two NAND gates is smaller than the delay through the inverter, whereas in the AED model that path is not allowed.

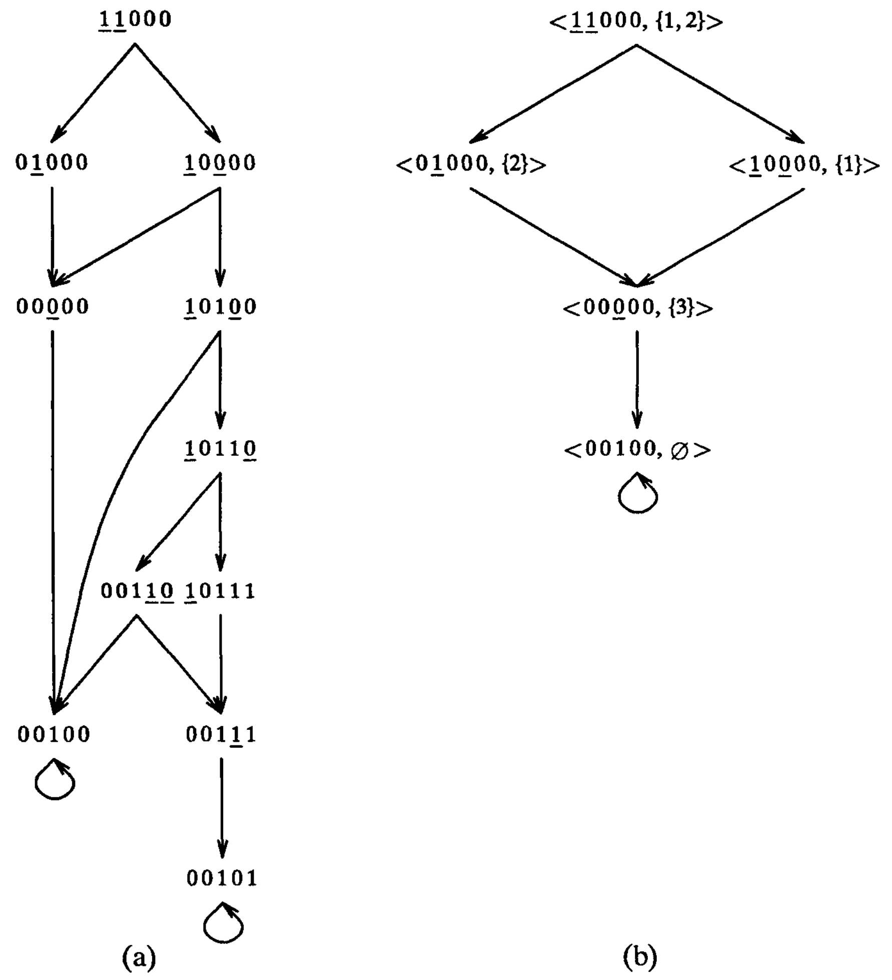


Figure 3.9. Race analysis of N_8 according to: (a) GSW; (b) AED.

In order to characterize the delay assumptions behind the AED method, we need a more precise mathematical definition of the binary AED relation. This relation will be called R for simplicity. Since a race analysis according to the AED method depends on the previous race history of the network, we will assume that the network is started in some stable total state and then some input(s) is changed. We will call this "first" state after the change *primary*. The precise definition of the AED method [6] is preferably done recursively. In the same definition, we will also define the set T of race states which are reachable from the initial state. Let $U(y)$ be the set of unstable states in the total state y.

Let $(x\,;\hat{y})$ be primary. Define T and R as follows.

```
Basis: $<\hat{y}, U(\hat{y})> \in T$.
Induction Step: Given $<y, V> \in T$.
    1. If $V = \emptyset$, then $<y, V> R <y, V>$.
    2. If $V \neq \emptyset$, for each $i \in V$, compute
            $W_i = (V - \{i\}) \cap U(y^{(i)})$
        a) If $W_i = \emptyset$, then
                $<y^{(i)}, U(y^{(i)})> \in T$
            and
                $<y, V> R <y^{(i)}, U(y^{(i)})>$.
        b) If $W_i \neq \emptyset$, then
                $<y^{(i)}, W_i)> \in T$
            and
                $<y, V> R <y^{(i)}, W_i)>$.
```

Here $y^{(i)}$ denotes the state y with gate i changed, i.e.

$$y^{(i)} = y_1, \cdots, y_{i-1}, y_i', y_{i+1}, \cdots, y_n.$$

The following observation is important. Every time case 2(a) is applied, a time of approximately Δ units has elapsed since the last application of case 2(a) (or after the primary state if this is the first time case 2(a) is applied). This follows from the fact that case 2(a) is only applied if the previous racing gates have all changed or become stable (because of other changing gates). In other words, by "marking" every edge that is added in the graph of the AED relation, because of rule 2(a), it is possible to introduce a time scale. In Fig. 3.10 we include this marking in the previous race analysis of our standard example of Fig. 3.1. From this graph it is easy to see that the network can end up in either one of two states: 000, or 001. Furthermore, the network will be unstable for at most approximately 2Δ time units. (Since the longest path has two marked edges.)

The intention of the AED method was to describe the analysis of races according to the Almost Equal Delay model (described in Section 3.2). However, there are some discrepancies between the delay model, and the results of the race analysis. The main objection against the method is that it assumes that after a race unit has finished, all the new unstable gates that start racing with each other have the same chance to win this second race. The problem is that, if δ denotes the difference between the fastest and the slowest gate in the first race unit, the difference after the second race unit can be as big as 2δ. In other words, after some k race units, the difference between the fastest and the slowest "change path", can be $k\delta$ time units. But if δ is not too small, and k reasonably large the different race units can be confused with each other. However, the method suggested does not capture this fact. Study for example Fig. 3.11. The assumption is that the faster gates have a delay approximately only 2/3 of the delay of the slower gates. Note that this is allowed in the Almost Equal Delay model. It is reasonable to assume that the gates d and e are racing, as can be seen in the figure. However, according to the method of [6], the change of gate d belongs to the second race unit whereas the change of gate e belongs to the

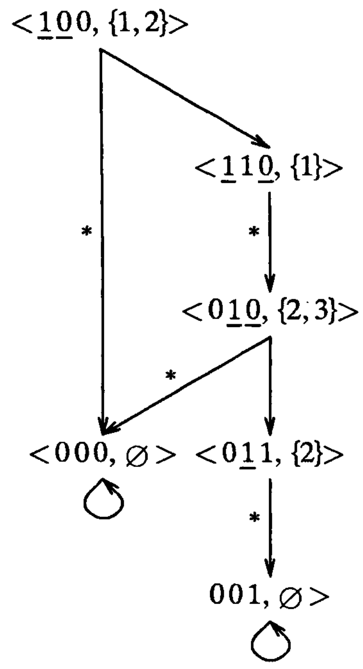


Figure 3.10. Race analysis of N_7 according to the AED method and with marking.

third race unit, and therefore the two gates are not assumed to race with each other.

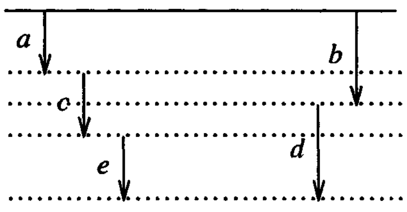


Figure 3.11. Timing diagram for a race.

There are two ways to solve this problem. Either change the method of analyzing races, or changing the underlying delay model. Only the latter will be considered here. We need only add one more assumption to the AED model to correctly describe the delay model behind the AED method. The assumption is the following. Let δ be the maximum difference between any two gate delays Δ_j and Δ_k in the network (i.e.$\delta = \Delta_{\max} - \Delta_{\min}$). Assume that the AED method is carried out for k race units; then $k\delta$ is assumed to be less than $\Delta_{\min}$. Whether this is a reasonable assumption is questionable, but one could argue that no "well-behaved" circuit should need more than some very small number of race units before it becomes stable after a change.

3.4. Gate network model

As mentioned earlier, the model of a network, and the assumptions made concerning the delays, are of vital importance to the results of an analysis. For this reason we describe a precise mathematical model of a gate network below. The delay model that will be used in the remaining part of this thesis, unless otherwise stated, is the arbitrary but finite gate and wire delay model, i.e. both the wires and the gates are assumed to have arbitrary, but finite, delays. This is an extension to the speed-independent model, since now wires can also have delays associated with them. However, we will originally model the network with only gate delays, and "add" delay elements later to represent the wire delays. The reason for this approach will be explained in Chapter IV, Section 5, but is mainly for convenience.

Assume that the network N has n inputs, described by the vector $x = x_1, \cdots, x_n$ of *input variables*. Also assume that the network has s gates. Each input and each gate will be represented by a node in a directed graph $G = (V, E)$, where V is the set of vertices or nodes and E is the set of edges. The edges will represent the connections among the input terminals and gates in the natural way. The input and gate nodes can be distinguished as follows. Nodes of indegree 0 are *input nodes*, and nodes of indegree ≥ 1 are called *gate nodes*. With each gate node we associate a boolean function g_i — the incoming edges to the gate node represent the arguments of g_i.

Sometimes it is convenient to treat all the nodes of G in the same fashion; for this reason the vector $y = y_1, \cdots, y_{n+s}$ of *node labels* or *node variables* will be used. Thus the vector y of node variables represents the *total state* (inputs and gate outputs) of the network. When it is necessary to distinguish between input and gate variables y_i is replaced by x_i for $i = 1, \cdots, n$. However, for $i = n+1, \cdots, n+s$, y_i always denotes the output of gate i.

To illustrate these ideas, consider the network N_9 of Fig. 3.12. The graph for N_9 is shown in Fig. 3.13. It has 4 nodes: one input node labeled y_1 (and also x_1) and 3 gate nodes labeled y_2, y_3, y_4. The boolean functions corresponding to these gates are g_2, g_3 and g_4.

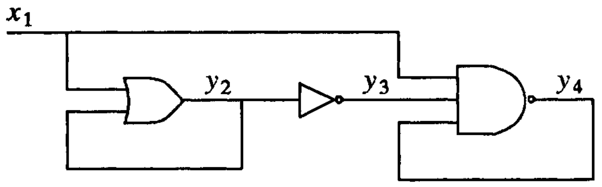


Figure 3.12. Network N_9.

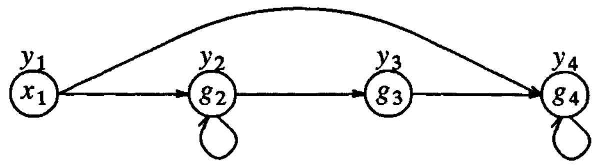


Figure 3.13. Graph of network N_9.

In summary, the following model is used. A *gate network* is a directed labeled graph
$N = <G, x, y, g>$,

where: $G = (V, E)$ is a finite directed *graph*,
$V = \{1, \cdots, n+s\}$ is the set of *vertices* of G,
$E \subseteq V \times V$ is the set of *edges* of G,
$x = x_1, \cdots, x_n$ is the vector of *input variables,*
$y = y_1, \cdots, y_{n+s}$ is the vector of *node labels* or *node variables,* and
$g = g_{n+1}, \cdots, g_{n+s}$ is a vector of boolean functions.

For an input node the variable y_i always takes the value of the external input x_i, i.e.

$$y_i = x_i \quad \text{for } i = 1, \cdots, n.$$

With each gate node j is associated the boolean function g_j. The arguments of g_j are all those node variables y_i such that $(i, j) \in E$; this represents the fact that node y_i is connected to an input of gate j. Note that, by definition, at most one such wire exists between y_i and y_j. Thus the indegree of node y_j is the number of arguments of g_j. For notational convenience we rename the arguments of g_j as follows. For each $(i, j) \in E$ the variable y_i is renamed w_{ij}. Suppose all the inputs of gate j are $w_{i_1 j}, \cdots, w_{i_{m_j} j}$; we will simply denote this vector by w_j. Thus, if the input vector for gate j is w_j, $g_j(w_j)$ is called the *excitation* of the gate and hence

$$g_j : B^{m_j} \rightarrow B,$$

where $B = \{0, 1\}$.

To illustrate these details we describe the variables and functions for the network N_9 of Fig. 3.12. The boolean functions associated with the three gates are:

$g_2(w_2) = g_2(w_{12}, w_{22}) = g_2(y_1, y_2) = g_2(x_1, y_2) = x_1 + y_2,$
$g_3(w_3) = g_3(w_{23}) = g_3(y_2) = y_2',$
$g_4(w_4) = g_4(w_{14}, w_{34}, w_{44}) = g_4(y_1, y_3, y_4) = g_4(x_1, y_3, y_4) = (x_1 y_3 y_4)'.$

It is assumed that each gate has an arbitrary but finite delay. Thus the gate output y_j may differ from its excitation $g_j(w_j)$. If $y_j = g_j(w_j)$ we say that gate j is *stable;* otherwise it is *unstable.* A network N is said to be in a *stable total* *state* iff its inputs are fixed and all its gates are stable.

In the next chapter, we will use this precise mathematical model of a gate network to characterize the ternary simulation described in Chapter II.

Chapter IV
Characterization of Ternary Simulation

4.1. Background

In this chapter we characterize the result of the ternary simulation according to the algorithm given by Eichelberger [11] and modified according to Brzozowski-Yoeli [8], as described in Chapter II. We will show that the results of ternary simulation correspond to a binary race analysis according to the General Multiple Winner model [8] for a network under the assumption that both wires and gates can have arbitrary, but finite delays.

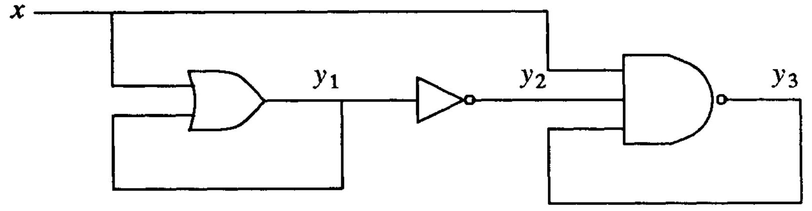


Figure 4.1. Network N_{10}.

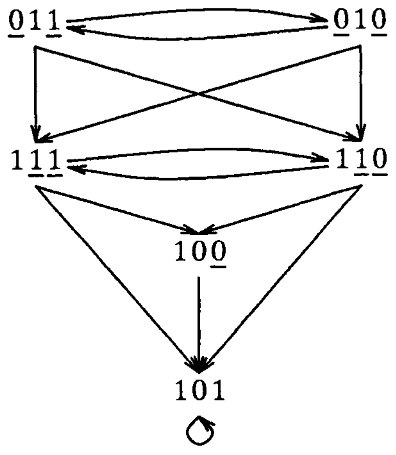


Figure 4.2. Binary analysis of N_{10}.

We introduce the problem by the example of Fig. 4.1. In Fig. 4.2 we show the binary race analysis according to the General Multiple Winner model (GMW). In contrast to this consider the following ternary approach. Starting with state $x = 0$, $y = 011$, change the input to $x = ½$, representing an unknown or changing signal. As a result of this unknown input x, gates 1 and 3 will have unknown values. In the second step, gate 2 will also become ½. This approach is summarized in Fig. 4.3(a) and corresponds to Eichelberger's Procedure A [11, 8]. In this case the figure shows that all the gates become unknown when the input is changing.

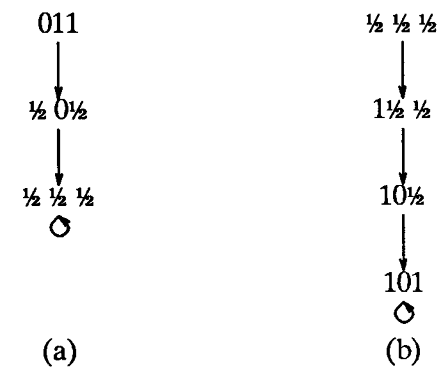


Figure 4.3. Ternary analysis of N_1: (a) Algorithm A; (b) Algorithm B.

To complete the ternary analysis we now apply the new input $x = 1$ to the ternary state ½ ½ ½ resulting from Procedure A, to see how much of the uncertainty introduced by the transient input can be removed when the final input value becomes known. First, y_1 will become 1 since the input $x = 1$ will force the output to 1, independently of the second input to the OR gate. Second, the output of the inverter will become 0 after y_1 becomes 1. Finally y_3 becomes 1 after y_2 changes to 0. This is summarized in Fig. 4.3(b). Notice that the final outcome of the ternary algorithm is 101 which is precisely the nontransient outcome of the GMW analysis.

Our second example shows what happens when the result of an input change does not lead to a unique stable state. The binary and ternary analyses of the NOR latch of Fig. 4.4 are shown in Figs. 4.5 and 4.6, respectively. The initial state is $x = 1$, $y = 00$ and the input is changed to $x = 0$. The GMW analysis of Fig. 4.5 shows three cycles: the stable states 01, and 10 and an oscillation (00, 11). The latter cycle is not transient like the ones of Fig. 4.2. However, it is "match-dependent" [8] in the sense that a network can only maintain such an oscillation if at each step the two gates have perfectly matched delays.

In the case of N_{11} the ternary model of Fig. 4.6 predicts an unknown final state. If one interprets ½ as meaning that the gate could either have the value 0 or the value 1 in any nontransient situation, then the ternary algorithm results are

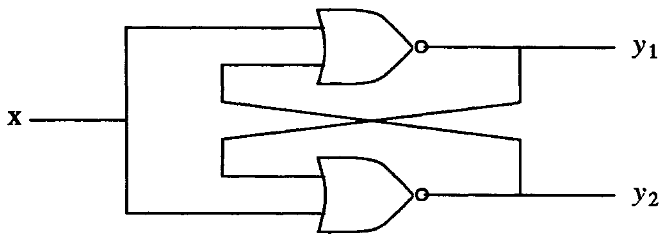


Figure 4.4. Network N_{11}.

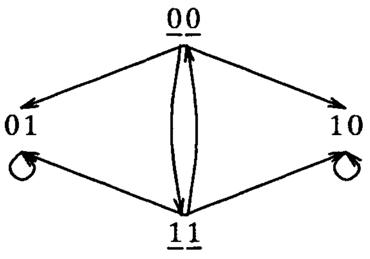


Figure 4.5. Binary analysis of N_{11}.

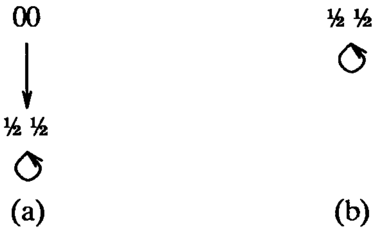


Figure 4.6. Ternary analysis of N_{11}: (a) Algorithm A; (b) Algorithm B.

correct.

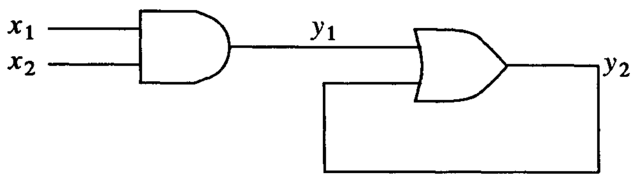


Figure 4.7. Network N_{12}.

Our final example [8] for this section shows that the ternary results do not always correspond to the binary results, if one assumes that delays are associated only with gates. If the network N_{12} of Fig. 4.7 is started with $x = 01$, $y = 00$ and the input changed to $x = 10$, the binary analysis predicts no change in the state, i.e. $y = 00$. However, the ternary algorithm yields $y = 0½$. This discrepancy can be explained by assuming that wires, as well as gates, have delays. Study for example Fig. 4.8 in which an additional state variable for the delay in the x_2 wire has been added. In Fig. 4.9 we show the binary race analysis of this "modified" network. Note that the outcome of the ternary analysis now corresponds to the binary analysis. This leads to the conjecture that the ternary and binary results correspond properly if one takes into account appropriate wire delays. It is this problem that is settled in the present chapter.

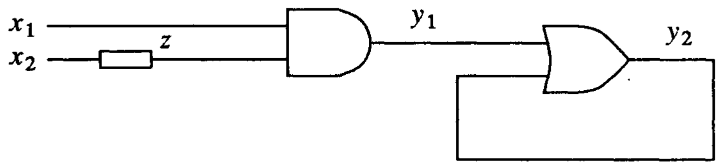


Figure 4.8. Network N_{12} with one wire delay added.

We first show that the ternary algorithm is insensitive to the addition of delays anywhere in the network. This follows basically from the monotonicity property of Algorithm A and B. As a consequence of this, we will always apply the ternary algorithm to the original network, without extra wire delays, whereas the binary race analysis will be performed on the so called "delay complete" network, in which each gate and each wire has a delay associated with it. In the remaining parts of the proof, we show how to "mimic" the ternary Algorithms A and B in this delay complete network. The fundamental observation is that the wire delays can be used to "remember" a previous node value, and hence we are free to choose between an "old" and its complemented "new" value. Another crucial observation is that if Algorithm B does not yield a binary result, i.e. there is at least one gate that remains ½ after Algorithm B has terminated, gates with a ½ must either be members of some cycles in the network, or connected to some cycles whose members are ½. This follows from the fact that the output of a gate can be ½ only if at least one of its inputs is ½. We will call these gates

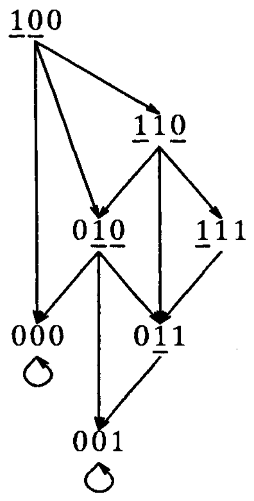


Figure 4.9. Race analysis of N_{12} with extra delay $(z\ y_1 y_2)$.

indefinite, and the cycles indefinite cycles. We then show that it is possible in the binary race analysis to reach a total state in which all gates that are binary after Algorithm B in the ternary analysis, have the same binary value, and are stable. Furthermore, this total state is such that there is at least one unstable wire delay in each of the indefinite cycles. In the last part of the proof, we show that it is possible to "move" these instabilities around in the indefinite cycles, and hence all gates that are ½ after Algorithm B can oscillate.

The chapter is structured as follows. We first formally define the binary race model and the ternary simulation algorithm in Sections 4.2 and 4.3. In Section 4.4 we state the main result of this chapter. In the remaining sections this result is proved. More specifically, in Section 4.5 we show that the ternary simulation is insensitive to the addition of delays anywhere in a wire, and hence the ternary simulation algorithm can be applied to a circuit with only gate delays. In Section 4.6 we characterize the first part of the ternary simulation algorithm. In Sections 4.7 and 4.8 we show how the last part of the ternary simulation algorithm can be "mimicked". In these sections we show also how a nontransient oscillation can be constructed involving all nodes that remain ½ after the ternary simulation algorithm has finished. Finally, Section 4.9 contains the proof of the main theorem, and also a comparison between this result and the original conjecture by Brzozowski and Yoeli. [8]. The reader should note that we consistently use the mathematical model for a gate network described in Chapter III, Section 4.

4.2. Binary race model

The race model we will use is the General Multiple Winner model [8], and it is first formally defined, using the network model described in Chapter III, Section 4.

Let $b \in B^{n+s}$ be any total state and let $a = b_1, \cdots, b_n$ be a fixed input vector. Define $U(b)$ to be the set of unstable gates in b, i.e.

$$U(b) = \{i : n+1 \leq i \leq n+s, \text{ and } b_i \neq g_i(w_i)\}.$$

The GMW relation R_a defines the set of successors for any total state b. If b is stable, i.e. if $U(b) = \emptyset$, then the only possible successor is b. If b is unstable, then every state of the form $b^{(S)}$ is a successor of b, where $b^{(S)}$ is b with some components changed as follows. Let S be any subset of the set $U(b)$ of unstable gate variables; to obtain $b^{(S)}$ complement b_i iff $i \in S$. Formally, define the GMW relation R_a, on the set B^{n+s} of total states. For $b \in B^{n+s}$,

if $U(b) = \emptyset$ then $bR_a b$;
if $U(b) \neq \emptyset$ then $bR_a b^{(S)}$ for any $S \subseteq U(b), S \neq \emptyset$.

Following [8] we define the set cycl(R_a, b) to be the set of total states of N that appear in cycles in the relation R_a and are reachable from b. For example the set of cyclic states reachable from 001 in Fig. 4.2 is {011, 010, 111, 110, 101}. Thus let

$$\text{cycl}(R_a, b) = \{c \in B^{n+s} : b\ R_a^* c \text{ and } c\ R_a^+ c\},$$

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

A cycle is called *transient* if there exists a gate i, $n+1 \leq i \leq n+s$ which is unstable in all of the states in the cycle and has the same value in all these states. For example, in Fig. 4.2, the cycle consisting of 011 and 010 is transient. Let

$$\text{trans}(R_a, b) = \{c \in \text{cycl}(R_a, b) : c \text{ appears only in transient cycle}\}$$

and

$$\text{out}(R_a, b) = \text{cycl}(R_a, b) - \text{trans}(R_a, b).$$

The set out(R_a, b) is the "outcome" of the binary 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. Note that match-dependent cycles are considered non-transient in this model.

4.3. Ternary simulation algorithm

In this section we describe the ternary simulation applied to our model in a formal way. For more details the reader should refer to [8]. Let $T = \{0,1,½\}$. The values 0 and 1 represent the usual logic levels and ½ represents an unknown value. We will use the following convention. Variables like x_i, y_i, etc. which take values from $B = \{0,1\}$ will have corresponding variables $\mathbf{x}_i, \mathbf{y}_i$, etc. taking values from T. The *partial order* $\leq$ on T is defined by

$$\mathbf{t} \leq \mathbf{t} \text{ for all } \mathbf{t} \in T,$$
$$0 \leq ½ \text{ and } 1 \leq ½.$$

The statement $\mathbf{t} \leq \mathbf{r}$ means that whenever r_i is binary then t_i has the same binary value as r_i, but $\mathbf{r}$ may contain more unknown components (i.e. components with value ½). Thus $\mathbf{r}$ has more "uncertainty" than $\mathbf{t}$.

We write $\mathbf{t} < \mathbf{r}$ if $\mathbf{t} \leq \mathbf{r}$ and $\mathbf{t} \neq \mathbf{r}$. Also, we extend the partial order $\leq$ to T^m in the usual way:

$$\mathbf{t} \leq \mathbf{r} \text{ iff } \mathbf{t}_i \leq \mathbf{r}_i \text{ for all } i = 1, \cdots, m.$$

The *μ–average* operation on nonempty subsets of the set B is defined as follows:

$$\mu\{0\} = 0, \;\; \mu\{1\} = 1, \;\; \mu\{0,1\} = ½.$$

We extend it to nonempty sets of vectors from B^m by taking the "component-by-component" μ-average. Thus, if $A \subseteq B^m$, let $A_i = \{a_i : (a_1, \cdots, a_m) \in A\}$ for $1 \leq i \leq m$ be the set of the ith components of all the vectors in A. Then define

$$\mu A = (\mu A_1, \cdots, \mu A_m)$$

For example, $\mu\{(0,0),(0,1)\} = (0,½)$. Clearly $a \leq \mu A$ for every $a \in A$.

For any boolean function $f : B^m \rightarrow B^p$ its *ternary extension* $\mathbf{f}: T^m \rightarrow T^p$ is defined by

$$\mathbf{f}(\mathbf{t}) = \mu\{f(a) : a \in B^m \text{ and } a \leq \mathbf{t}\}.$$

It follows that, for $t \in B^m$, $\mathbf{f}(t) = f(t)$, i.e. on binary vectors the ternary extension agrees with the original function. The ternary extension obeys the following monotonicity property [8]:

$$\mathbf{t} \leq \mathbf{r} \text{ implies } \mathbf{f}(\mathbf{t}) \leq \mathbf{f}(\mathbf{r}).$$

We first describe how to compute the ternary excitation **next** for any total state $\mathbf{y} \in T^{n+s}$.

```
function next($\mathbf{y} \in T^{n+s}$) $\in T^{n+s}$;
begin
    for $j$ = 1 to $n$ do
        $\mathbf{next}_j := \mathbf{y}_j$;
    for $j = n+1$ to $n+s$ do
        $\mathbf{next}_j := \mathbf{g}_j(\mathbf{w}_j)$;
end;
```

Here, $\mathbf{g}_j$ is the ternary extension of the boolean function g_j associated with gate j. Note that $\mathbf{w}_j$, as before, is a vector consisting of some components of $\mathbf{y}$. Also, it can be verified that

$$\mathbf{y} \leq \tilde{\mathbf{y}} \text{ implies } \mathbf{next}(\mathbf{y}) \leq \mathbf{next}(\tilde{\mathbf{y}}).$$

This follows from the definition of **next** and the monotonicity property of the ternary extension.

The ternary simulation consists of Algorithms A and B described below.

Let $\hat{a}, b = \hat{a}_1, \cdots, \hat{a}_n, c_1, \cdots c_s \in B^{n+s}$ be any stable total state and $a = a_1, \cdots, a_n$ be the new input vector. Let $\mathbf{u} = \mu\{\hat{a}, a\}$.

Algorithm A

```
$h$:=0;
$\mathbf{y}^0$:=$\mathbf{u}, c$;
repeat
    $h$:=$h$+1;
    $\mathbf{y}^h$:=$\mathbf{next}(\mathbf{y}^{h-1})$;
until $\mathbf{y}^h = \mathbf{y}^{h-1}$;
```

It was shown in [8] that Algorithm A always terminates, i.e. we have a sequence of p distinct ternary states, where $p \leq s+1$

$$\mathbf{y}^0, \mathbf{y}^1, \cdots, \mathbf{y}^{p-1}.$$

It was also shown in that Algorithm A can only "increase the uncertainty" [8] in the network state, i.e.

$$\mathbf{y}^h < \mathbf{y}^{h+1} \text{ for } 0 \leq h < p-1.$$

Note that $\mathbf{y}^{p-1} = \mathbf{u}, \mathbf{r}$, for some $\mathbf{r} \in T^s$.

Next, starting with state $\mathbf{y}^{p-1} = \mathbf{u}, \mathbf{r}$, we apply Algorithm B given below.

Algorithm B

```
$h$:=$p$;
$\mathbf{y}^h$:=$a, \mathbf{r}$;
repeat
    $h$:=$h$+1;
    $\mathbf{y}^h$:=$\mathbf{next}(\mathbf{y}^{h-1})$;
until $\mathbf{y}^h = \mathbf{y}^{h-1}$;
```

Algorithm B also terminates [8], i.e. we have a sequence of q distinct ternary states, where $q \leq s+1$

$$\mathbf{y}^p, \mathbf{y}^{p+1}, \ldots, \mathbf{y}^{p+q-1},$$

and now the uncertainty decreases, i.e.

$$\mathbf{y}^h > \mathbf{y}^{h+1} \text{ for } p \leq h < p+q-1.$$

Note that $\mathbf{y}^{p+q-1} = a, \mathbf{t}$ for some $\mathbf{t} \in T^s$.

4.4. Main result

The following result was proved by Brzozowski and Yoeli [8] in 1979. Let N be started in some stable state $\hat{a}, c$ and let the input vector change to a, i.e. let the new total state be $b = a, c$. Then the result $\mathbf{y}^B$ of the ternary simulation of N "covers" the nontransient states reachable from b in the GMW graph of N in the sense that:

$$\mu(\text{out}(R_a, b)) \leq \mathbf{y}^B .$$

The example of Fig. 4.7 shows that, in general, the two results are not equal. In this paper we show that the results become equal if appropriate delays are added. The main result is formally stated in this section; the following sections then contain the proof. The basic idea is to study a "delay complete" network obtained from the original network by adding an arbitrary but finite inertial delay to each wire. The new network will be called $\tilde{N}$. It contains the gates of N whose outputs are now labeled by the vector $\tilde{y}$, and the special "gates" corresponding to the added delays. The outputs of those delays will be described by the vector $\tilde{z}$. Each delay may be viewed as a gate performing the identity function. The total state of $\tilde{N}$ is now $(\tilde{y}, \tilde{z})$, but we will compare only $\tilde{y}$ with the corresponding vector y in N. In the theorem below, if $\tilde{y}, \tilde{z}$ appears in a cycle of the GMW graph of $\tilde{N}$, then we will say that $\tilde{y}$ *belongs* to that cycle. Assuming that N and $\tilde{N}$ are started in corresponding initial conditions, the main result is:

Theorem 1 The ternary result $\mathbf{y}^B$ from Algorithm B for any network N is equal to the μ-average of all the binary vectors $\tilde{y}$ which belong to nontransient cycles reachable from the initial state of $\tilde{N}$ in the GMW model. Furthermore, there exists a nontransient cycle Z in the graph of the GMW relation such that the μ-average of all the vectors $\tilde{y}$ belonging to that one cycle is equal to $\mathbf{y}^B$.

The proof proceeds as follows. In Section 4.5 we prove that the ternary simulation is insensitive to the addition of delays anywhere in the network N. In particular, the results of the ternary algorithm applied to N and $\tilde{N}$ agree in the gate variables. In Section 4.6 we characterize the result of Algorithm A and establish the first part of a sequence of states of $\tilde{N}$ that will eventually lead to the nontransient cycle Z of Theorem 1. In Section 4.7 we find the second part of that sequence, which is related to Algorithm B. In Section 4.8 we exhibit the nontransient cycle Z. Finally, in Section 4.9 we complete the proof of Theorem 1.

4.5. Ternary simulation and delays

In this section we show that the ternary simulation is insensitive to the addition of delays anywhere in the network. In the lemma below we insert one delay to N in the wire from node k to node m. The output of this delay will be designated $\tilde{z}_{km}$ and the resulting network will be $\bar{N} = <\tilde{G}, x, \tilde{y}, \tilde{z}_{km}, g>$, where the graph $\tilde{G}$ is G with an extra node added for the "gate" corresponding to the delay $\tilde{z}_{km}$, the input vector x is the same as in N, the total state of $\bar{N}$ is given by $(\tilde{y}, \tilde{z}_{km})$ and the vector g of boolean functions is the same as in N. The "gate function" of delay $\tilde{z}_{km}$ is simply y_k.

We say that Algorithm A or B is *consistent* for N and $\bar{N}$ if the final result is the same for $\tilde{y}$ and y, i.e. the output of the additional delay $\tilde{z}_{km}$ is simply ignored.

Lemma 1 The result of Algorithm A applied to any network N is consistent with the result of Algorithm A applied to the same network with one wire delay $\tilde{z}_{km}$ added.

Proof: Let $N = <G, x, y, g>$ be an arbitrary network and suppose $(k, m) \in E$. Let $\hat{y}^0 = (\hat{a}, b)$ be a stable total state of N and let $\mathbf{y}^0 = \mathbf{u}, b$, where $\mathbf{u} = \mu\{\hat{a}, a\}$, be the initial ternary state of N for Algorithm A. Let $\bar{N} = <\tilde{G}, x, \tilde{y}, \tilde{z}_{km}, g>$ be N with a delay $\tilde{z}_{km}$ inserted in the wire from node k to node m. The total state $\tilde{y}, \tilde{z}_{km} = \hat{a}, b, \hat{y}_k{}^0$ is stable in $\bar{N}$. Let $\mathbf{y}^0, \hat{y}_k{}^0$ be the initial ternary state of $\bar{N}$ for Algorithm A.

We will prove by induction on h that

$$\tilde{\mathbf{y}}^h \leq \mathbf{y}^h \leq \tilde{\mathbf{y}}^{h+1} \text{ for all } h \geq 0.$$

In view of the fact that Algorithm A converges to its final value after a finite number of steps, the lemma will then follow.

If $\mathbf{y}_k$ does not change during Algorithm A the lemma is obviously true. Otherwise, assume that $\mathbf{y}_k$ becomes ½ at step r, $r \geq 0$. Note that $r = 0$ means that y_k is an input node.

For the basis, $h = 0$, we have $\tilde{\mathbf{y}}^0 = \mathbf{y}^0$. By the monotonicity of Algorithm A, $\tilde{\mathbf{y}}^1 \geq \tilde{\mathbf{y}}^0$. Thus we have

$$\tilde{\mathbf{y}}^0 \leq \mathbf{y}^0 \leq \tilde{\mathbf{y}}^1.$$

Suppose now that $\tilde{\mathbf{y}}^h \leq \mathbf{y}^h \leq \tilde{\mathbf{y}}^{h+1}$. Consider the input variables $\mathbf{w}_{ij}$ and $\tilde{\mathbf{w}}_{ij}$ to all the gates of N. If $(i, j) \neq (k, m)$ then $\tilde{\mathbf{w}}_{ij} = \tilde{\mathbf{y}}_i$ and it is always true that $\mathbf{w}_{ij} = \mathbf{y}_i$. Hence

$$\tilde{\mathbf{w}}_{ij}{}^h \leq \mathbf{w}_{ij}^h \leq \tilde{\mathbf{w}}_{ij}{}^{h+1}$$

by the induction hypothesis for the $\mathbf{y}_i$. Also $\mathbf{w}_{km} = \mathbf{y}_k$ and $\tilde{\mathbf{w}}_{km} = \mathbf{z}_{km}$. We know that

$$\mathbf{w}_{km}^{h} = \begin{cases} \hat{y}_k^0 & \text{if } h<r \\ \frac{1}{2} & \text{if } h \geq r \end{cases}$$

and

$$\tilde{\mathbf{w}}_{km}^{h} = \begin{cases} \hat{y}_k^0 & \text{if } h<r+1 \\ \frac{1}{2} & \text{if } h \geq r+1 \end{cases}$$

Thus we also have

$$\tilde{\mathbf{w}}_{km}^{h} \leq \mathbf{w}_{km}^{h} \leq \tilde{\mathbf{w}}_{km}^{h+1}.$$

Altogether we have

$$\tilde{\mathbf{w}}_{ij}^{h} \leq \mathbf{w}_{ij}^{h} \leq \tilde{\mathbf{w}}_{ij}^{h+1} \text{ for all } (i,j) \in E.$$

It now follows by the monotonicity of $\mathbf{g}_j$ that

$$\tilde{\mathbf{y}}_j^{h+1} = \mathbf{g}_j(\tilde{\mathbf{w}}_j^h) \leq \mathbf{g}_j(\mathbf{w}_j^h) = \mathbf{y}_j^{h+1} \leq \mathbf{g}_j(\tilde{\mathbf{w}}_j^{h+1}) = \tilde{\mathbf{y}}_j^{h+2}.$$

Hence the induction step goes through and the lemma holds. □

We are now ready to prove the main theorem of this section.

Theorem 2: Informally, the ternary algorithm is not affected by adding any delays to the network being analyzed. Formally, let $N = <G, x, y, g>$ be any network and let $\bar{N} = <\tilde{G}, x, \tilde{y}, \tilde{z}, g>$ be any network obtained from N by the addition of delays in any wires of N. Then the outcomes of Algorithms A and B are consistent for N and $\bar{N}$.

Proof:

Let Lemma 1^D be Lemma 1 with Algorithm A replaced by Algorithm B, with initial state $\mathbf{y}^{p-1} = \mathbf{u},\mathbf{r}$ and new input $x = a$ for N, and appropriate initial state for $\bar{N}$ depending on the position of the added delay. One can verify that Lemma 1^D is proved by dual arguments, interchanging $\leq$ and $\geq$. We leave the details to the reader.

Altogether, Lemma 1 and Lemma 1^D show that ternary simulation yields consistent results for N and $\bar{N}$, when only one delay has been added.

It now follows by induction on the number of delays added to N that the theorem holds for any $\bar{N}$ as claimed. □

According to Theorem 2 any number of delays can be added to a network without changing the result of the ternary simulation. We now define the *delay-completion* of a network that gives a network which, in some sense, has all possible delays included. More specifically, let $N = <G, x, y, g>$ be a given network. We obtain $\tilde{N} = <\tilde{G}, x, \tilde{y}, \tilde{z}, g>$ by inserting a delay in every input line of every gate of N; $\tilde{N}$ is the delay-completion of N. Let d be the number of delays inserted. As in the case of w_j we define $\tilde{z}_j$ to be the vector $\tilde{z}_{i_1 j}, \cdots, \tilde{z}_{i_{m_j} j}$. This is the vector of input variables of gate j in network $\tilde{N}$.

To illustrate this definition we show the delay-completion of N_9 of Fig. 3.12 in Fig. 4.10.

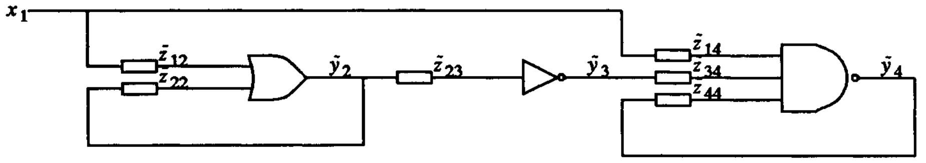


Figure 4.10. Delay-completion of N_9.

4.6. Characterization of Algorithm A

The results of this section are an adaptation of the work of Bryant [2]. In his model delays are associated only with wires and not with gates. However, the main idea for Theorem 3 is essentially the same. We assume that N and $\tilde{N}$ are started in corresponding initial conditions.

Theorem 3 The ternary result y^A from Algorithm A for any network N is equal to the μ-average of all the binary vectors $\tilde{y}$ reachable from the initial state of $\tilde{N}$ in the GMW model. Furthermore, there exists a state $\hat{y}, \hat{z}$ reachable from the initial state of $\tilde{N}$, in which each gate output that corresponds to ½ after Algorithm A in the ternary simulation of N, is the complement of its initial value, i.e. y^A is the μ-average of the initial state and $\hat{y}$.

Proof: We will prove the second part, since the first part follows immediately from the second. Clearly the second part implies that y^A is $\le \mu Y$, where Y is the set of all the binary vectors reachable from the initial state of $\tilde{N}$. On the other hand, in [8, Lemma 4] it was shown that any binary state of $\tilde{N}$ reachable from the initial state is $\le$ the result of Algorithm A for $\tilde{N}$. By Theorem 2, this result is consistent with that of Algorithm A for N. Hence $y^A \ge \mu Y$, and the first part follows.

For convenience we now define the standard initial conditions for N and $\tilde{N}$ which will always be used to study a transition from a stable total state when the input vector changes.

Standard Initial Conditions

Network N:	$<G, x, y, g>$		
	$\hat{y}^0 = \hat{a}, b$	-	given stable total state
	$\hat{a}$	-	input vector before change
	a	-	input vector after change
	$\mathbf{u} = \mu(\hat{a}, a)$	-	ternary input vector for Algorithm A
	b	-	initial state of gate nodes of N
Network $\tilde{N}$:	$<\tilde{G}, x, \tilde{y}, \tilde{z}, g>$		
	$\hat{y}^0, \tilde{z}^0$	-	stable total state corresponding to $\hat{y}^0$ of network N, where
	$\tilde{z}_{ij}^0 = \hat{y}_i^0$		for all $(i, j) \in E$
	$\tilde{y}^0, \tilde{z}^0$	-	total state of $\tilde{N}$ after input change which is also the initial state for the GMW analysis of $\tilde{N}$, where
	$\tilde{y}^0 = a, b$		
	R_a	-	GMW relation for $\tilde{N}$

The second part of the theorem can be stated more formally as: let N be any network and let $\tilde{N}$ be its delay-completion, both started in the standard initial conditions. Let $\mathbf{y}^h$, $0 \leq h \leq p-1$ be the result of Algorithm A after h steps. Then for each h there exist $\tilde{y}^{2h} \in B^{n+s}$ and $\tilde{z}^{2h} \in B^d$ such that

(i) $(\tilde{y}^0, \tilde{z}^0) R_a^{2h} (\tilde{y}^{2h}, \tilde{z}^{2h})$,

(ii) $\mathbf{y}_j^h = ½$ implies $\tilde{y}_j^{2h} = (\hat{y}_j^0)'$ for $1 \leq j \leq n+s$,

(iii) $\mathbf{y}_j^h = \hat{y}_j^0$ implies $\tilde{y}_j^{2h} = \hat{y}_j^0$ and $\tilde{z}_j^{2h} = \tilde{z}_j^0$ for $1 \leq j \leq n+s$.

We proceed by induction on h. The reader may find it useful to follow the construction in the proof of the theorem in parallel with the construction after Fig. 4.11 for network N_{13} of Fig. 4.11.

Basis, $h=0$: One verifies that $\tilde{y}^0, \tilde{z}^0$ satisfies conditions (i)-(iii).

Induction step: Assume that $\tilde{y}^{2h}, \tilde{z}^{2h}$ has been constructed and consider $\mathbf{y}_j^{h+1}$. There are two cases.

(a) $\mathbf{y}_j^{h+1} = ½$ and $\mathbf{y}_j^h = \hat{y}_j^0$. In other words $\mathbf{y}_j$ had the original binary value for the first h steps and changed to ½ in step $h+1$. There must exist at least one such j in every step h, $0 \leq h \leq p-1$. Note that $h+1 \geq 1$ implies that y_j

must be a gate output, since all the input node variables became ½ in step 0. Since $\mathbf{y}_j^{h+1} = \mathbf{g}_j(\mathbf{w}_j^h) = ½$, and $\hat{y}_j^0 = g_j(\hat{w}_j^0) \in B$ and by the definition of the ternary extension of g_j, there exists $v_j^h \in B^{m_j}$ such that $g_j(v_j^h) = (\hat{y}_j^0)'$, and $v_j^h \leq \mathbf{w}_j^h$. Note also that $\hat{w}_j^0 \leq \mathbf{w}_j^h$. If $v_{ij}^h \neq \hat{w}_{ij}^0$ (at least one such case must exist), then $\mathbf{w}_{ij}^h = ½$, i.e. $\mathbf{y}_i^h = ½$. By the induction hypothesis $\tilde{y}_i^{2h} = (\hat{y}_i^0)'$ by (ii). Also $\tilde{z}_{ij}^{2h} = \tilde{z}_{ij}^0$ because $\mathbf{y}_j^h = \hat{y}_j^0$ and (iii) applies. But $\tilde{z}_{ij}^0 = \hat{y}_i^0$, and hence the delay $\tilde{z}_{ij}$ is unstable in step $2h$, i.e.

$$\tilde{y}_i^{2h} \neq \tilde{z}_{ij}^{2h} \text{ for all } i \text{ such that } v_{ij}^h \neq \hat{w}_{ij}^0.$$

Let $\tilde{z}_{ij}^{2h+1} = \tilde{y}_i^{2h}$ if $v_{ij}^h \neq \hat{w}_{ij}^0$ and $\tilde{z}_{ij}^{2h+1} = \tilde{z}_{ij}^{2h}$ otherwise. Note that $\tilde{z}_j^{2h+1} = v_j^h$, and hence gate y_j will become unstable in step $2h+1$.

(b) If case (a) does not hold, variable $\mathbf{y}_j$ has not changed from step h to step $h+1$. Let $\tilde{z}_j^{2h+1} = \tilde{z}_j^{2h}$. Note that now gate y_j will be stable in step $2h+1$.

Now define $\tilde{y}^{2h+1} = \tilde{y}^{2h}$. It follows that

$$(\tilde{y}^{2h}, \tilde{z}^{2h})\, R_a\, (\tilde{y}^{2h+1}, \tilde{z}^{2h+1}).$$

Next define $\tilde{y}^{2h+2}, \tilde{z}^{2h+2}$ as follows:

$$\tilde{y}_j^{2h+2} = g_j(\tilde{z}_j^{2h+1}) \text{ for all } j,$$

and

$$\tilde{z}^{2h+2} = \tilde{z}^{2h+1}.$$

The reader can now verify that $\tilde{y}^{2h+2}$, $\tilde{z}^{2h+2}$ satisfies (i)-(iii). Therefore the induction step goes through and the theorem holds. □

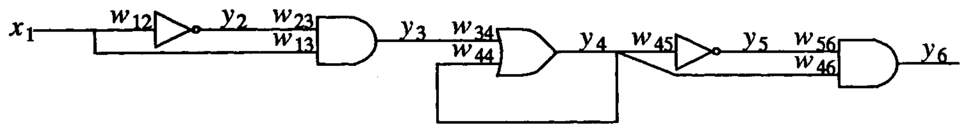


Figure 4.11. Network N_{13}.

To illustrate the construction in the proof of Theorem 3 consider network N_{13} of Fig. 4.11. Let

$$\hat{a} = 0, \quad a = 1, \quad b = 10010.$$

Then $\hat{y}^0 = 010010$ and $\mathbf{u} = ½$. Also let

$$\begin{array}{l}
\tilde{z}_2 = \tilde{z}_{12} = 0 \\
\tilde{z}_3 = \tilde{z}_{13}, \tilde{z}_{23} = 01 \\
\tilde{z}_4 = \tilde{z}_{34}, \tilde{z}_{44} = 00 \\
\tilde{z}_5 = \tilde{z}_{45} = 0 \\
\tilde{z}_6 = \tilde{z}_{46}, \tilde{z}_{56} = 01
\end{array}$$

Hence

$$\tilde{y}^0, \tilde{z}^0 = 110010,\ 0\ 01\ 00\ 0\ 01$$

The construction described in the proof of Theorem 3 is shown below. Note that all the gates of $\tilde{N}$ are stable in $\tilde{y}^6$, but there are some unstable delays. This fact will be used in a later section.

$$\begin{array}{lll}
\mathbf{y}^0 = \tfrac{1}{2}10010 & \tilde{y}^0, \tilde{z}^0 = & 110010,\ \underline{0}\ \underline{0}1\ 00\ 0\ 01 \\
 & \tilde{y}^1, \tilde{z}^1 = & 1\underline{10}010,\ 1\ 11\ 00\ 0\ 01 \\
\mathbf{y}^1 = \tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2}010 & \tilde{y}^2, \tilde{z}^2 = & 101010,\ 1\ 1\underline{1}\ \underline{0}0\ 0\ 01 \\
 & \tilde{y}^3, \tilde{z}^3 = & 101\underline{0}10,\ 1\ 1\underline{1}\ 10\ 0\ 01 \\
\mathbf{y}^2 = \tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2}10 & \tilde{y}^4, \tilde{z}^4 = & 101110,\ 1\ 1\underline{1}\ 1\underline{0}\ \underline{0}\ \underline{0}1 \\
 & \tilde{y}^5, \tilde{z}^5 = & 1011\underline{10},\ 1\ 1\underline{1}\ 1\underline{0}\ 1\ 11 \\
\mathbf{y}^3 = \tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2} & \tilde{y}^6, \tilde{z}^6 = & 101101,\ 1\ 1\underline{1}\ 1\underline{0}\ 1\ 1\underline{1}
\end{array}$$

4.7. Algorithm B - definite nodes

Let N be any network started in the standard initial conditions. Let $\mathbf{y}^B$ be the result of Algorithm B. The *indefinite nodes* or *indefinite gates* of N are those gates whose outputs are ½ in $\mathbf{y}^B$; the other nodes will be called *definite.*

Assuming that there is at least one indefinite gate j (i.e. Algorithm B does *not* yield a binary result) that gate must have at least one input w_{ij} where y_i is also an indefinite gate (possibly with $i=j$). Otherwise all inputs to gate j would be binary and its output could not be ½. Since the network N is finite we must have at least one cycle of indefinite nodes; such a cycle will be called indefinite.

Consider now $\tilde{N}$; the indefinite gates of $\tilde{N}$ are the same as those of N. Any delay between two indefinite gates will be called *indefinite.* Eventually we want to show that, if the result of Algorithm B contains at least one ½, there exists a nontransient cycle of length ≥ 2 (i.e. an oscillation) in the graph of the relation R_a for $\tilde{N}$ such that all indefinite gates "take part" in that oscillation, i.e. each gate variable will take on both values 0 and 1 in the cycle. Furthermore that cycle is reachable from the initial state of $\tilde{N}$.

Theorem 4 Let N and $\tilde{N}$ be started in the standard initial conditions and let $\mathbf{y}^B$ be the result of Algorithm B. There exists a state $\tilde{y}, \tilde{z}$ of $\tilde{N}$ reachable from the initial state such that:

(1) All definite gates have the same (binary) value in $\tilde{y}$ as they do in $\mathbf{y}^B$. Furthermore, they are all stable, as are all the delays in the wires leaving definite nodes.

(2) There is at least one unstable wire delay in each indefinite cycle of $\tilde{N}$.

Proof: Let $\tilde{y}^{2p-2}, \tilde{z}^{2p-2}$ be the total state of $\tilde{N}$ constructed in the proof of Theorem 3. Let $\mathbf{y}_j^h$, $p \le h \le p+q-1$ be the result of Algorithm B after $h-p$ steps. We first claim that there exist $\tilde{y}^{2h} \in B^{n+s}$ and $\tilde{z}^{2h} \in B^d$ such that

(i) $(\tilde{y}^{2p-2}, \tilde{z}^{2p-2})\, R_a^*\, (\tilde{y}^{2h}, \tilde{z}^{2h})$,

(ii) $\mathbf{y}_j^h \in B$ implies $\tilde{y}_j^{2h} = \mathbf{y}_j^h$ for $1 \le j \le n+s$,

(iii) $\mathbf{y}_j^h = ½$ implies $\tilde{y}_j^{2h} = \tilde{y}_j^{2p-2}$ and $\tilde{z}_j^{2h} = \tilde{z}_j^{2p-2}$ for $1 \le j \le n+s$.

We proceed by induction on h. The reader may find it useful to follow the construction in the proof in parallel with the construction shown right after this proof for network N_{13} of Fig. 4.11.

Basis, $h=p$: Let $\tilde{y}^{2p} = \tilde{y}^{2p-2}$ and $\tilde{z}^{2p} = \tilde{z}^{2p-2}$. Claims (i) and (iii) follow immediately. For (ii), i.e. $\mathbf{y}_j^p \in B$, there are two cases.

(a) $\mathbf{y}_j^{p-1} \in B$. But by Theorem 3, (iii) in proof, $\tilde{y}_j^{2p-2} = \mathbf{y}_j^{p-1}$ and from the definition of Algorithm B it follows that $\mathbf{y}_j^{p-1} \in B$ implies that $\mathbf{y}_j^p = \mathbf{y}_j^{p-1}$ and hence $\tilde{y}_j^{2p} = \tilde{y}_j^{2p-2} = \mathbf{y}_j^{p-1} = \mathbf{y}_j^p$.

(b) $\mathbf{y}_j^{p-1} = ½$. This implies that j is an input node of N, and from Theorem 3, (ii) in proof, it follows that $\tilde{y}_j^{2p} = \tilde{y}_j^{2p-2} = (\hat{y}_j^0)'$. But this is the new input value assigned to $\mathbf{y}_j^p$ by Algorithm B and hence $\tilde{y}_j^{2p} = \mathbf{y}_j^p$.

Induction step: Assume that $\tilde{y}^{2h}, \tilde{z}^{2h}$ $(h \ge p)$ has been constructed and consider $\mathbf{y}_j^{h+1}$. There are two cases.

(a) $\mathbf{y}_j^h = ½$ and $\mathbf{y}_j^{h+1} \in B$. In other words $\mathbf{y}_j$ was ½ for the first $h-p$ steps in Algorithm B and changed to a binary value in $\mathbf{y}^{h+1}$. Note that $h+1 > p$ implies that y_j must be a gate output, since all the input node variables became binary in $\mathbf{y}^p$. Let $\tilde{z}_{ij}^{2h+1} = \tilde{y}_i^{2h}$ for $i = i_1, \cdots, i_{m_j}$. This corresponds to changing all unstable wire delays to gate j (if any). Now $\tilde{z}_{ij}^{2h+1} \le \mathbf{w}_{ij}^h$ because $\tilde{z}_{ij}^{2h+1} = \tilde{y}_i^{2h}$ and by the induction hypothesis (iii) we have that if $\mathbf{y}_i^h \in B$ then $\tilde{y}_i^{2h} = \mathbf{y}_i^h$. Note that $\mathbf{y}_j^{h+1} = \mathbf{g}_j(\mathbf{w}_j^h) \in B$ implies that $g_j(\tilde{z}_j^{2h+1}) = \mathbf{y}_j^{h+1}$.

(b) If case (a) does not hold, variable $\mathbf{y}_j$ has not changed from step h to step $h+1$. Let $\tilde{z}_j^{2h+1} = \tilde{z}_j^{2h}$. Note that now gate y_j is stable in step $2h+1$.

Now define $\tilde{y}^{2h+1} = \tilde{y}^{2h}$. It follows that either

$$\tilde{y}^{2h}, \tilde{z}^{2h} = \tilde{y}^{2h+1}, \tilde{z}^{2h+1}$$

or

$$(\tilde{y}^{2h}, \tilde{z}^{2h})\, R_a\, (\tilde{y}^{2h+1}, \tilde{z}^{2h+1}).$$

In either case

$$(\tilde{y}^{2h}, \tilde{z}^{2h})\ R_a^*\ (\tilde{y}^{2h+1}, \tilde{z}^{2h+1}).$$

Next define $\tilde{y}^{2h+2}, \tilde{z}^{2h+2}$ as follows:

$$\tilde{y}_j^{2h+2} = g_j(\tilde{z}_j^{2h+1}) \text{ for all } j,$$

and

$$\tilde{z}^{2h+2} = \tilde{z}^{2h+1}.$$

The reader can now verify that $\tilde{y}^{2h+2}$, $\tilde{z}^{2h+2}$ satisfies (i)-(iii). Therefore the induction step goes through and the claim holds.

Now let

$$\tilde{y}^B = \tilde{y}^{2(p+q-1)}$$

and

$$\tilde{z}_{ij}^B = \begin{cases} \tilde{y}_i^{2(p+q-1)} & \text{for all } i \text{ such that } i \text{ is a definite node} \\ \tilde{z}_{ij}^{2(p+q-1)} & \text{otherwise.} \end{cases}$$

It follows trivially that all wire delays *from* definite nodes are stable in $\tilde{y}^B, \tilde{z}^B$, but furthermore all definite gates are also stable in $\tilde{y}^B, \tilde{z}^B$. This is because (ii) implies that $\tilde{z}_j^B \leq \mathbf{w}_j^{p+q-1}$ and (by the definition of Algorithm B) it follows that $\tilde{y}_j^B = \tilde{y}_j^{2(p+q-1)} = \mathbf{y}_j^{p+q-1} = \mathbf{g}_j(\mathbf{w}_j^{p+q-1})$ for all definite gates. In other words, $\tilde{y}^B, \tilde{z}^B$ is a total state reachable from the initial total state $\tilde{y}^0, \tilde{z}^0$ in the GMW relation such that all definite gates are stable and have the binary value predicted by the result of Algorithm B. Furthermore all wire delays from definite nodes are also stable. This completes the proof of Part (1).

For Part (2), clearly it is sufficient to prove the claim for each simple indefinite cycle, where a cycle is simple if it has no repeated nodes except for the first and the last node in the cycle. Let C be an arbitrary simple indefinite cycle in $\tilde{N}$. A gate j in C is said to be *initiating* iff no other gate in C becomes ½ in Algorithm A before gate j. Clearly there must be at least one initiating gate in each simple indefinite cycle. Let j be an initiating gate in C. Assume gate j became ½ at step r of Algorithm A. Note that $r \geq 1$ since an input node cannot be indefinite. Now since node j is in C there must exist a predecessor to j in the cycle, say node i. Note that $i = j$ is permitted. Consider $\tilde{z}_{ij}$. Since j is an initiating gate we have that $\mathbf{y}_i^{r-1} = \hat{y}_i^0$ and hence according to Theorem 3, (iii) in proof, $\tilde{y}_i^{2(r-1)} = \hat{y}_i^0$, and also $\tilde{z}_{ij}^{2(r-1)} = \hat{y}_i^0$. Note that this implies that v_{ij}^{r-1}, as defined in the proof of Theorem 3, satisfies $v_{ij}^{r-1} = \hat{y}_i^0$, i.e. the wire delay from the predecessor to an initiating gate has the original "old" binary value when the initiating gate changes.

After this, $\tilde{z}_{ij}$ is never changed again to construct $\tilde{z}^B$ (since j is an indefinite gate). However we know that $\tilde{y}_i^{2p-2} = (\hat{y}_i^0)'$ (Theorem 3, (ii) in proof) and by the construction of $\tilde{y}^B$, $\tilde{z}^B$ that $\tilde{y}_i^B = \tilde{y}_i^{2p-2}$. It follows that $\tilde{y}_i^B = (\hat{y}_i^0)'$ and $\tilde{z}_{ij}^B = \hat{y}_i^0$ and hence $\tilde{z}_{ij}^B$ is unstable. This completes the proof of part 2. □

The first part of the construction of Theorem 4 for network N_{13} is shown below.

$$\begin{array}{lll}
\mathbf{y}^4 = 1\tfrac12\tfrac12\tfrac12\tfrac12\tfrac12 & \tilde{y}^8, \tilde{z}^8 & = 101101,\ 1\ 1\underline{1}\ 1\underline{0}\ 1\ 1\underline{1} \\
 & \tilde{y}^9, \tilde{z}^9 & = 101101,\ 1\ 1\underline{1}\ 1\underline{0}\ 1\ 1\underline{1} \\
\mathbf{y}^5 = 10\tfrac12\tfrac12\tfrac12\tfrac12 & \tilde{y}^{10}, \tilde{z}^{10} & = 101101,\ 1\ 1\underline{1}\ 1\underline{0}\ 1\ 1\underline{1} \\
 & \tilde{y}^{11}, \tilde{z}^{11} & = 10\underline{1}101,\ 1\ 10\ 1\underline{0}\ 1\ 1\underline{1} \\
\mathbf{y}^6 = 100\tfrac12\tfrac12\tfrac12 & \tilde{y}^{12}, \tilde{z}^{12} & = 100101,\ 1\ 10\ \underline{10}\ 1\ 1\underline{1}
\end{array}$$

To illustrate the construction of $\tilde{y}^B, \tilde{z}^B$ for N_{13} we have

$$\tilde{y}^B, \tilde{z}^B = 100\underline{1}01,\ 1\ 10\ 0\underline{0}\ 1\ 1\underline{1}.$$

Note that wire delay $\tilde{z}_{44}$ is unstable. This will later be used to start an oscillation in the indefinite subnetwork. Note also that all definite nodes have the same values as in $\mathbf{y}^B$.

4.8. Algorithm B - indefinite gates

The main result of this section is captured in the following theorem.

Theorem 5 Let N be a network started in the standard initial conditions. Let $\tilde{y}^B, \tilde{z}^B$ be a total state of $\tilde{N}$ as defined at the end of Section 8. Then there is a nontransient cycle in the GMW graph, reachable from $\tilde{y}^B, \tilde{z}^B$, such that all indefinite gates of $\tilde{N}$ are oscillating.

To simplify the proof of Theorem 5, the following two definitions are useful. Define a state $\tilde{y}, \tilde{z}$ of $\tilde{N}$ to be *consistent* with $\mathbf{y}^B$ iff $\tilde{y} \leq \tilde{\mathbf{y}}^B$, and all the definite nodes and all the delays leaving definite nodes are stable in $\tilde{N}$. Also, a state $\tilde{y}, \tilde{z}$ is *loop-unstable* iff there is at least one unstable wire delay in each simple indefinite cycle of $\tilde{N}$.

We now proceed as follows. Starting with a total state $(\tilde{y}, \tilde{z})$ we first exhibit a sequence of total states of $\tilde{N}$

$$(\tilde{y}, \tilde{z}) = (\tilde{y}^0, \tilde{z}^0),\ \cdots,\ (\tilde{y}^m, \tilde{z}^m),$$

where m is the number of indefinite gates, and in $\tilde{y}^k$, exact k indefinite gate outputs have complementary values to those in $\tilde{y}$ and the other indefinite gate nodes are the same as in $\tilde{y}$. For convenience, we will say that k indefinite nodes have been "marked" in this way. By repeating this process of marking (i.e. complementing) all the indefinite gates we show the existence of an oscillation involving all the indefinite nodes.

Lemma 2 Let N be a network started in the standard initial conditions and $\mathbf{y}^B$ be the result of Algorithm B. Let $\tilde{y}, \tilde{z}$ be any total state of $\tilde{N}$ consistent with $\mathbf{y}^B$ and loop-unstable. Assume that some, but not all, indefinite nodes of $\tilde{N}$ are marked. Assume also that every wire delay between a marked and an unmarked indefinite node is unstable. Then there exists at least one unmarked indefinite node j, such that all indefinite wire delays to j are unstable.

Proof: Consider the directed graph $G' = (V', E')$ where

$V' \subseteq V$, $i \in V'$ iff i is an indefinite gate node and

$E' = \{(i,j) \in V' \times V' : (i,j) \in E$ and $\tilde{z}_{ij}$ is stable$\}$.

G' can be obtained from the graph G by retaining only the indefinite gate nodes and those indefinite edges that corresponds to stable indefinite delays. G' has two important properties:

(i) there is no edge from a marked node to an unmarked node, and

(ii) there is no cycle in G'.

Both properties follow trivially from the construction of G' and the assumptions in the Lemma.

Now consider a reverse path in G'. Start at some unmarked node $k \in V'$ and traverse G' backwards. From (ii) and the fact that G' is finite it follows that a reverse path in G' started at node k must stop at some node, say j. By property (i) it follows that j must be an unmarked node. Furthermore, since each indefinite gate has at least one input wire from an indefinite gate, it follows that all indefinite wire delays to j must be unstable; otherwise the reverse path could not have stopped at j. Hence the Lemma holds. □

In the following Lemma we will show how instabilities can be "moved". The idea is that if all indefinite wire delays *to* a gate are unstable then it is possible in the GMW relation to find a state reachable from the present state such that all indefinite wire delays *leaving* the gate are unstable.

Lemma 3 Let N be a network started in the standard initial conditions. Let $\tilde{y}$, $\tilde{z}$ be a total state of $\tilde{N}$ consistent with $\mathbf{y}^B$ and loop-unstable. If all indefinite wire delays to indefinite gate j are unstable, then there exists a total state $\tilde{y}^C$, $\tilde{z}^C$, reachable from $\tilde{y}, \tilde{z}$, consistent with $\mathbf{y}^B$ and loop-unstable, such that

(i) $\tilde{y}_j^C = (\tilde{y}_j)'$, and

(ii) all indefinite wire delays leaving gate j are unstable.

Proof: Consider gate j. Two cases are possible.

a) Gate j is stable in $\tilde{y}$, $\tilde{z}$, i.e. $\tilde{y}_j = g_j(\tilde{z}_j)$. Since j is an indefinite gate, after Algorithm B for N, we have $\mathbf{y}_j^B = \mathbf{g}_j(\mathbf{w}_j^B) = ½$. Note that $\tilde{z}_j \leq \mathbf{w}_j^B$ by construction of $\tilde{y}^B, \tilde{z}^B$. By the definition of the ternary extension of g_j, there must exist a $v_j \in B^{m_j}$ such that $g_j(v_j) \neq g_j(\tilde{z}_j)$, and $v_j \leq \mathbf{w}_j^B$. If $v_{ij} \neq \tilde{z}_{ij}$ then we must have $\mathbf{w}_{ij}^B = ½$, i.e. $\mathbf{y}_i^B = ½$, and hence $\tilde{z}_{ij}$ is an indefinite wire delay. We want to reach a state in which gate j is unstable. We will do this, if we set the inputs of gate j to the vector v_j. This can be done because all the indefinite wire delays to j are assumed to be unstable. Therefore, define

$$\tilde{z}^D_{kl} = \begin{cases} \tilde{y}_i & \text{if } k = i\,,\ l = j \text{ and } v_{ij} \neq \tilde{z}_{ij} \\ \tilde{z}_{kl} & \text{otherwise} \end{cases}$$

and

$$\tilde{y}^D = \tilde{y}.$$

b) Gate j is unstable. Define $\tilde{y}^D = \tilde{y}$ and $\tilde{z}^D = \tilde{z}$.

In either case we have that $(\tilde{y}, \tilde{z})\ R_a^*\ (\tilde{y}^D, \tilde{z}^D)$ and gate j is unstable in $\tilde{y}^D, \tilde{z}^D$.

Now we will simultaneously change gate j and all indefinite delays leaving gate j which are unstable. In this way all the indefinite delays leaving gate j will become unstable after the change. Therefore define $\tilde{y}^C, \tilde{z}^C$ as follows:

$$\tilde{z}^C_{kl} = \begin{cases} \tilde{y}^D_j & \text{if } k = j \text{ and gate } l \text{ is an indefinite gate} \\ \tilde{z}^D_{kl} & \text{otherwise} \end{cases}$$

and

$$\tilde{y}^C_k = \begin{cases} g_j(\tilde{z}^D_j) & \text{if } k = j \\ \tilde{y}^D_k & \text{otherwise} \end{cases}$$

Condition (i) follows from the fact that $\tilde{y}^C_j = g_j(\tilde{z}^D_j) = g_j(v_j) = (\tilde{y}_j)'$. Condition (ii) follows from (i) and the fact that if k is an indefinite gate then $\tilde{z}^C_{jk} = \tilde{y}^D_j = \tilde{y}_j$. Hence the Lemma holds. □

Lemma 4 Let N be a network started in the standard initial conditions. Let $\tilde{y}, \tilde{z}$ be a total state of $\tilde{N}$ which is consistent with $\mathbf{y}^B$ and loop-unstable. Then there exists a total state $\hat{y}, \hat{z}$, reachable from $\tilde{y}, \tilde{z}$, which is consistent with $\mathbf{y}^B$, loop-unstable, and such that all indefinite gates in $\tilde{y}$ and $\hat{y}$ have complementary values.

Proof: We proceeds by induction on the number of indefinite nodes which have been marked, i.e. complemented.

Claim: There exists a state $\tilde{y}^k, \tilde{z}^k$, reachable from $\tilde{y}, \tilde{z}$, with k nodes marked. Furthermore, this state is consistent with $\mathbf{y}^B$, loop-unstable and all indefinite wire delays between marked nodes and unmarked nodes are unstable.

The basis, $k=0$, follows trivially. Suppose the claim holds for k, $k \geq 0$. By Lemma 2 it follows that there exists an unmarked indefinite node j, such that all indefinite wire delays to j are unstable. But Lemma 3 guarantees the existence of a state $\tilde{y}^{k+1}, \tilde{z}^{k+1}$, reachable from $\tilde{y}^k, \tilde{z}^k$, consistent with $\mathbf{y}^B$, loop-unstable and with node j complemented. We now mark node j, and note that all indefinite wire delays between marked nodes and unmarked nodes are still unstable. Hence the induction step goes through and the lemma holds. □

Proof of Theorem 5: Since Lemma 4 can be applied any number of times and there is only a finite number of possible total states there must exist a cycle in the graph. Also by the construction of Lemma 4 it follows that each indefinite gate in $\tilde{N}$ will oscillate. Furthermore, since all definite gates are stable, all wire delays from definite gates are stable, no wire delays to definite gates are changed and all indefinite gates are oscillating it follows trivially that the cycle cannot be transient. □

4.9. The conjecture

We are now in a position to prove Theorem 1.

Proof of Theorem 1: Let $Y = \{\tilde{y} : \tilde{y}, \tilde{z} \in \text{out}(R_a, \tilde{y}^0, \tilde{z}^0)\}$. In Theorem 5 we showed that $\mathbf{y}^B \leq \mu Y$. In [8] it was shown that any binary state of $\tilde{N}$ in $\text{out}(R_a, \tilde{y}^0)$ is $\leq$ the result of Algorithm B for $\tilde{N}$. But, by Theorem 2, this result is consistent with that of Algorithm B for N. Hence $\mathbf{y}^B \geq \mu Y$, and $\mathbf{y}^B = \mu Y$. Also, the constructions in Theorems 3, 4, and 5 show the existence of a single cycle Z such that $\mathbf{y}^B = \mu Z$. □

The following is a consequence of Theorem 1. Whenever a network N has a critical race, the network $\tilde{N}$ has an oscillation involving the gates that take part in the race.

The characterization obtained in Theorem 1 does not quite apply to the original conjecture which used a network N^c, described below, instead of $\tilde{N}$. The conjecture network N^c consists of the given network N to which delays have been added in all the input lines and in all the fan-out connections. There are two main differences between N^c and $\tilde{N}$:

1) In case an input x_i fans out to two or more gates, N^c has one delay associated with the input line x_i and additional delays in each fan-out connection from x_i. The network $\tilde{N}$ only has the fan-out delays.

2) In case the output of a gate i is connected to only one gate j (with $i = j$ possible), N^c has no delay in that connection, whereas in $\tilde{N}$ delays are inserted uniformly in all gate-input lines.

First we will show that the extra line delays of N^c as described in 1) above are not necessary. Let $N = \langle G, x, y, g \rangle$ be any network. A node $i \in V$ is said to be *singular* iff the outdegree of i is 1, i.e. the output of node i is only connected to one gate node.

Lemma 5 Let N be any network, let N^c be defined as above, and let N^d be N^c after the removal of all input delays corresponding to input nodes which are not singular. If N^c and N^d are started in the appropriate standard initial conditions, the outcomes of the GMW analyses of N^c and N^d will be consistent with respect to the nodes of N^d.

Proof: Consider N^d with only one extra delay z_{ii} added in input line x_i. If x_i does not change, the behavior of the two networks is identical. If x_i does change, the variable z_{ii} is unstable and can change only once. It must change before the network reaches a nontransient cycle, but after that the network behaves like N^d. The lemma now follows by induction on the number of delays added to N^d to get N^c. $\square$

From now on we consider the network N^d rather than N^c. Until now we have not explicitly defined any output of network N. One possible interpretation is that the output of every gate is an external output. In that case gates originally considered singular in N^d have an extra connection and are no longer singular. Thus delays will be present in all the lines leaving such gates, and hence all gate-input lines will have delays as in $\tilde{N}$. Therefore, under this assumption $\tilde{N}$ and N^d coincide, and Theorem 1 constitutes a proof of the conjecture.

One could make a different interpretation, namely that only some of the gate nodes are external output nodes. It is then reasonable to also assume that the network has no "useless" gates, i.e. that in the graph G there is a path from every gate node to some output node. This implies that there must be at least one gate node with outdegree ≥ 2 in every simple cycle of G. Now the only way that N^d differs from $\tilde{N}$ is the fact that in N^d a gate-input line from a singular node does not have a delay. For example, refer to $\tilde{N}_9$ of Fig. 4.10. The corresponding network N_9^d, assuming 4 is the only output node, would not have delay $\tilde{z}_{34}$. However, the proof of the conjecture in this form, is outside the scope of the present thesis, and will be omitted.

Chapter V
Conclusions and Further Research

5.1. Introduction

In this chapter we first discuss some deficiencies of the ternary simulation algorithm. Some of these are basic properties of the algorithm, and hence cannot be corrected. An example of this type of problem, is the fact that one can only detect the presence, but not the absence of possible timing problems with the method. (This is discussed further below.) Others, like the problem of characterization of the ternary algorithm applied to switch-level simulation of MOS circuits, needs further research to be understood. In Section 5.2 we examine these problems in some detail.

One problem with the Muller model is that output hazards are not automatically detected. In Section 5.3 we outline one possible use of the ternary simulation to detect static output hazards.

The most severe disadvantage with the ternary simulation is its pessimism. In Section 5.4 we outline a new ternary algorithm to reduce this pessimism. We provide a number of examples and compare the results obtained from the algorithm with the results obtained from a binary Almost Equal Delay analysis. From these examples, we state a conjecture concerning correspondence between the ternary algorithm and the binary "Almost Equal Delay" model. The conjecture is that the ternary results "covers" the binary AED results.

Finally, in Section 5.5 we summarize the main results of the thesis and discuss some additional open problems.

5.2. Criticism of the ternary simulation

The ternary simulation algorithm described and characterized in the previous chapters has the obvious advantage of being linear in the number of gates in the circuit. Hence, very large circuits can be efficiently analyzed. However, there are some fundamental problems with the ternary simulation approach. The first is that the method can only analyze a circuit for a given total state and transition. In other words, the method can only show the presence, and not the absence of possible timing problems. (Unless, of course, *all* possible total states and transitions are simulated.)

The second problem is that the approach is not suited for hierarchical design practices. The reason for this is that the algorithm does not detect output hazards. Hence, the simulation can predict that two circuits function properly for some total states and input changes. However, when the two circuits are connected to each other, the combined circuit might contain a timing problem. In Section 5.3 we will further discuss this problem and also suggest some partial solutions.

Another problem is due to the fact that the simulation can be very "pessimistic". The reason for this is as follows. In Chapter IV, we showed that the results of the ternary simulation correspond to a binary race analysis according to the General Multiple Winner model under the assumption that both gates and wires can have arbitrary but finite delays. The assumption of arbitrary delays in wires and gates leads sometimes to overly pessimistic predictions. Study for example the network N_{14} of Fig. 5.1.

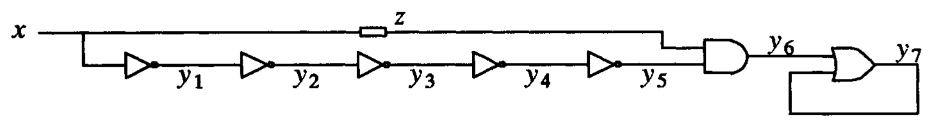


Figure 5.1. Network N_{14}.

It is easy to verify that the total state $x = 1$, $y_1, \cdots, y_7 = 0101000$ is stable. The problem to analyze is what happens when x changes to 0? In Figs. 5.2 and 5.3 we show the ternary analysis and the binary analysis according to the GMW model respectively. To simplify the binary race analysis, we include only one wire delay z. The results will be the same if all possible delays are considered, except that in that case the OR gate can also oscillate.

Algorithm	x	y_1	y_2	y_3	y_4	y_5	y_6	y_7
A:	½	0	1	0	1	0	0	0
A:	½	½	1	0	1	0	0	0
A:	½	½	½	0	1	0	0	0
A:	½	½	½	½	1	0	0	0
A:	½	½	½	½	½	0	0	0
A:	½	½	½	½	½	½	0	0
A:	½	½	½	½	½	½	½	0
A:	½	½	½	½	½	½	½	½
A:	½	½	½	½	½	½	½	½
B:	0	½	½	½	½	½	½	½
B:	0	1	½	½	½	½	0	½
B:	0	1	0	½	½	½	0	½
B:	0	1	0	1	½	½	0	½
B:	0	1	0	1	0	½	0	½
B:	0	1	0	1	0	1	0	½
B:	0	1	0	1	0	1	0	½

Figure 5.2. Ternary analysis of network N_{14}.

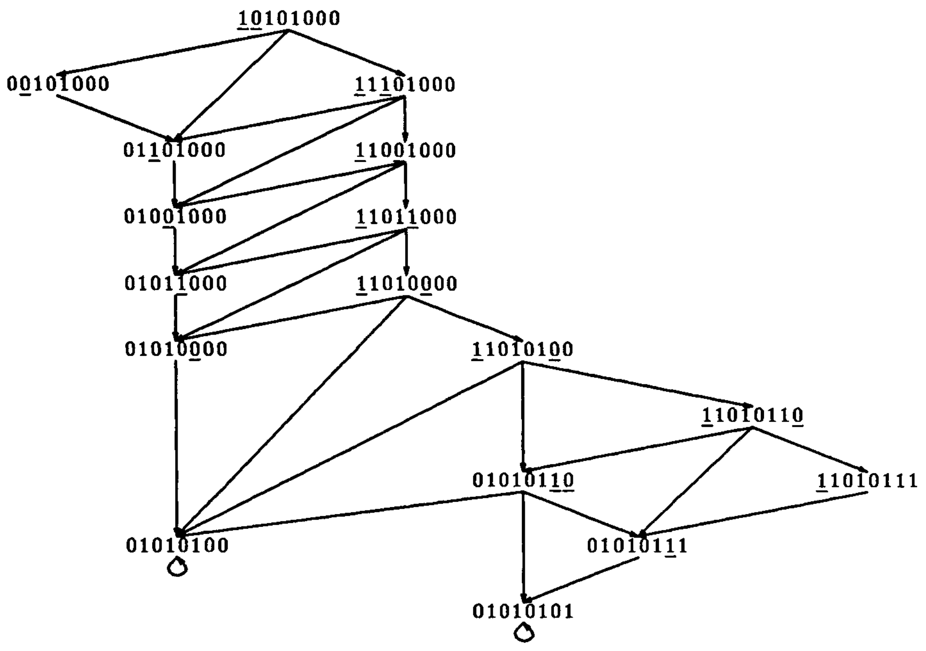


Figure 5.3. Race analysis of N_{14} according to the GMW model ($z\ y_1, \cdots, y_7$).

If we study the GMW graph of Fig. 5.3 we can see that the outcome 1 on the OR gate can happen *only* if the sum of the delays in the five inverters is smaller than the delay in the wire z. Clearly, this is a very pessimistic prediction. In almost all cases the delay in the inverter chain will be greater than the delay in the wire and the circuit will end up reliably with a 0 on the OR gate.

Ternary simulation is useful in the following sense. Suppose that the simulator indicates that a circuit behaves correctly and always ends up in a binary state. In that case we have a guarantee that the realization of the network will function properly, independently of any delays anywhere in the circuit. In other cases the ternary simulation may be overly pessimistic, i.e. predicts possible timing problems in cases when these problems are very unlikely. One may then want to investigate these potential problems by other means. In Section 5.4 we will describe a modified ternary simulation algorithm that tries to reduce this pessimism.

The last problem with the ternary simulation we will discuss, is of a somewhat different nature. With the arrival of switch-level simulators for MOS VLSI circuits, it has become of great importance to detect timing problems on this level. However, the characterization of the results of ternary simulation given in Chapter IV is only for gate networks. As was mentioned earlier, ternary simulation techniques have been used in switch-level simulators [1], although the results are not completely understood. Lengauer and Naher [21] assumed that each node in the circuit could be in three possible states, 0, 1, and

X. They also assumed that all changes $0 \rightarrow 1$ or $1 \rightarrow 0$ could also always go through this third state X, i.e. the changes could also be of the form $0 \rightarrow X \rightarrow 1$ and $1 \rightarrow X \rightarrow 0$. Under these assumptions they proved the correspondence between the results obtained form Bryant's ternary simulation algorithm, and a race analysis according to the GMW† model. The question remains, however, how realistic the above assumptions are in real circuits. Furthermore, it is still an open question whether these assumptions of a third state (and that all changes go through it) are necessary.

5.3. Output hazards

The ternary simulation has the rather severe drawback of not being well suited for hierarchical design practices. The main reason for this is that the method does not address the problem with output hazards. Hence, even if a circuit should behave correctly according to a ternary analysis, it is not certain that the composition of two correctly behaving circuits will behave correctly.

As was mentioned in Chapter II (Section 5) one can detect output hazards by studying the output gates during the binary race analysis. This is clearly not feasible for even medium sized networks. The question arises whether the results of ternary simulation can be used to detect output hazards? In the following we will outline one possible use of the results from the ternary simulation for detecting at least some of the output hazards. The basic observation is the following.

Proposition 1. Let y_i^0 denote the value of an output gate i before the change of the input. Also, let y_i^A and y_i^B denote the values of the gate after Algorithm A and B respectively. If $y_i^0 = y_i^B \in B$ then there is an output hazard iff $y_i^A = ½$.

Proof: Follows trivially from Theorem 3 and Theorem 4 of Chapter IV. □

Note that this only accounts for the static hazards. It is still an open problem whether the ternary simulation can be modified to detect dynamic output hazards. One possibility could be to specify the behavior of a circuit for both binary and ternary input vectors, and also give the output behavior both after Algorithm A and B. This is clearly a field well worth studying, but is far beyond the scope of this thesis.

† Note that this GMW analysis is *not* binary. For example, if a node with current value 0 is unstable, then it can change either to 1 or to X.

5.4. Modified ternary simulation algorithm

The ternary simulation algorithm according to the Eichelberger-Brzozowski-Yoeli version, called E-ternary for short, can be very pessimistic, as was shown in Section 2. For binary race analysis, the Almost Equal Delay model was an attempt to reduce the pessimism in the GMW and GSW models. This raises the question if it is possible to find a ternary algorithm that gives results corresponding to the binary AED model. We will outline such an algorithm, but we will not be completely successful.

We begin by describing the algorithm by means of an example. We will apply the method to the network N_7 of Fig. 3.1, starting in the total state $x = 0$, $y = 100$ and changing x to 1. The basic idea is to apply the E-ternary Algorithm A one step, and then apply Algorithm B at once. However, for reasons to be explained later, the method is slightly more complicated. The first step of the algorithm is to determine which gates are unstable or unknown in the current state. This is accomplished by calculating an intermediate total state $\mathbf{t}$ in the following way. The unknown states (i.e. states that have the value ½) remain unknown. To determine which binary states *might* be unstable, the algorithm compares the ternary excitation of the gate with the previous value of the gate. If the ternary excitation is a ½, or not equal to the original gate value, the corresponding entry in $\mathbf{t}$ will get the value ½. All the other gates keep their previous values. In the case of state $y^0 = 100$ and $x = 1$ we get the following. There are no unknown gate values in $y^0 = 100$. Furthermore, $Y_1 = x' = 1' = 0 \neq 1 = y_1^0$, and hence, $\mathbf{t}_1^1 = ½$. Similarly, $Y_2 = x\, y_1^0 = 1\,1 = 1 \neq 0 = y_2^0$ so $\mathbf{t}_2^1 = ½$. Finally, $Y_3 = y_2^0 + y_3^0 = 0 + 0 = 0 = y_3^0$, and hence $\mathbf{t}_3^1 = 0$. In summary, we get the intermediate total state $\mathbf{t}^1 = ½\,½\,0$.

The next step in the algorithm is to determine which of these unknown or possibly unstable states must stabilize during this race unit. This is accomplished by *recalculating* the excitation for those states. However, for the recalculation of gate j we use the gate values according to the intermediate state, *except* for the value of the gate value j itself. For this, we use the original value. Furthermore, the gates with binary values in $\mathbf{t}$ keep their original values. In our example we must recalculate gate 1 and 2 (since they are the only ones with the value ½ in $\mathbf{t}^1$). We get $\mathbf{y}_1^1 = x' = 1' = 0$, and $\mathbf{y}_2^1 = x\, \mathbf{t}_2^1 = 1\,½ = ½$. Hence, gate 1 will get the new value 0 whereas gate 2 will be unknown. In summary, we get $\mathbf{y}^1 = 0\,½\,0$. It is interesting to compare this result with what the Almost Equal Delay model predicts after one race unit. From Fig. 3.10 it is easy to see that the AED model predicts that 000 and 010 are the possible states after one race unit. In other words, $\mathbf{y}^1 = \mu\{000, 010\}$, and it appears that the result of the ternary algorithm is the μ-average of the outcome of the binary race analysis according to the AED model.

The above procedure is then repeated, either until the result stabilizes, or some maximum number of times (arbitrarily chosen). In our example we get the following calculations. Since $\mathbf{y}_1^1 = 0$, and $Y_1 = x' = 0$ we get $\mathbf{t}_1^2 = 0$. Gate 2 is unknown and hence $\mathbf{t}_2^2 = ½$. Finally, $Y_3 = \mathbf{y}_2^1 + \mathbf{y}_3^1 = ½ + 0 = ½$ and therefore

$t_3^2 = ½$. Hence, we obtain $t^2 = 0½½$. Gates 2 and 3 must be recalculated. We get the following results: $y_1^2 = 0$, $y_2^2 = Y_2 = x t_1^2 = 10 = 0$, and $y_3^2 = Y_3 = t_2^2 + y_3^1 = ½ + 0 = ½$. In summary, we obtain $y^2 = 00½$. If we compare this result with the AED result after two race units we can easily see that $y^2 = \mu\{000, 001\}$, and hence the two methods give the same result once again. It is trivial to verify that the state $y = 00½$ is stable in the ternary algorithm.

We now describe the method in more detail. We will use the same network model as was described in Chapter III (Section 4). As in Chapter III (Section 3), a state $y = a, c$, $a \in B^n$, and $c \in B^s$, is said to be *primary* iff there exists $\hat{a} \in B^n$ such that $\hat{y} = \hat{a}, c$ is stable. Let y^0 be primary.

Algorithm 1

```
t = 0;
z = $y^0$;
repeat
        y = z;
        z = next(y);
        t = t+1;
until (z = y or t = max_time );
```

where **next**(y) is calculated as follows:

```
function next( $y \in T^{n+s}$ ) $\in T^{n+s}$;
begin
        $t \in T^{n+s}$;
A:      for j=1 to n
                $t_j = y_j$;
        for j=n+1 to n+s
                if $y_j = ½$ or $g_j(w_j) = ½$ then
                        $t_j = ½$;
                else
                        $t_j = \mu(y_j, g_j(w_j))$;

B:      for j=1 to n
                $next_j = t_j$;
        for j=n+1 to n+s
                if $t_j = ½$ then
                        $next_j = g_j(v_j)$;
                else
                        $next_j = t_j$;
end;
```

Here v_j is defined in a way similar to w_j, i.e. $v_j = v_{i_1 j}, \cdots, v_{i_{m_j} j}$. However, for v_{ij} we have for all $(i, j) \in E$

$$\mathbf{v}_{ij} = \begin{cases} \mathbf{y}_j & \text{if } i = j \\ \mathbf{t}_i & \text{if } i \neq j \end{cases}$$

In other words, $\mathbf{v}_j$ is the vector of input variables to the gate j, using the values of the intermediate total state $\mathbf{t}$, *except* for the value of gate j itself, for which the original value is used.

Part A of the function **next** calculates the intermediate total state $\mathbf{t}$. This state is such that each gate that was either ½ (i.e. unknown) *or* unstable in state $\mathbf{y}$ will get the value ½; all other nodes will keep their previous binary value. In Part B, the ternary excitations for all the gates that became ½ in Part A are reevaluated. For the gates that kept their binary value in part A, the binary value is kept unaltered.

In the remaining part of this section we will study this method for a number of examples. We will compare the results with the results obtained by an Almost Equal Delay analysis. However, the AED analysis is slightly modified. We will extend the AED model to include multiple winners, and we will only be interested in stable states, and states reachable after some complete race units.

Our first example will be the NOR latch of Fig. 3.5. We start in the stable state $x = 1$, $y^0 = 0\,0$, and change x to 0. We get the following results:

$$\mathbf{t}^1 = \tfrac{1}{2}\,\tfrac{1}{2}$$

$$\mathbf{y}^1 = (\,(\mathbf{t}_2^1)',\ (\mathbf{t}_1^1)'\,) = \tfrac{1}{2}\,\tfrac{1}{2}$$

It is easy to verify that the state $\mathbf{y} = \tfrac{1}{2}\tfrac{1}{2}$ is stable in the ternary algorithm. Compare now this result with the AED analysis shown in Fig. 5.4.

$$\begin{tikzcd}
 & \langle \underline{0}\,\underline{0}, \{1,2\}\rangle \arrow[dl] \arrow[d] \arrow[dr] & \\
\langle 0\,1, \varnothing\rangle & \langle \underline{1}\,\underline{1}, \{1,2\}\rangle & \langle 1\,0, \varnothing\rangle
\end{tikzcd}$$

Figure 5.4. One race unit according to the (extended) AED model for NOR latch.

Note that $\mathbf{y}^1 = \mu\{01, 11, 10\}$, and the results from the two methods are equal. In Fig. 5.5 we show the second race unit, starting in the three different starting states 01, 11, and 10. We can see once again that $\mathbf{y}^2 = \mu\{01, 00, 10\}$ and hence the methods seem to correspond.

In the following example we show that the extension of the AED model to include multiple winners is necessary to obtain correspondence between the ternary algorithm and the binary race analysis. Study network N_{15} of Fig. 5.6. It is easy to verify that the total state $x = 1$, $y = 000$ is stable. Now change x to

$$\begin{tikzcd}
<01, \varnothing> \arrow[loop below] & & <\underline{1}\underline{1}, \{1,2\}> \arrow[dl] \arrow[d] \arrow[dr] & & <10, \varnothing> \arrow[loop below] \\
& <01, \varnothing> & <\underline{0}\underline{0}, \{1,2\}> & <10, \varnothing> &
\end{tikzcd}$$

Figure 5.5. Second race unit according to the (extended) AED model for NOR latch.

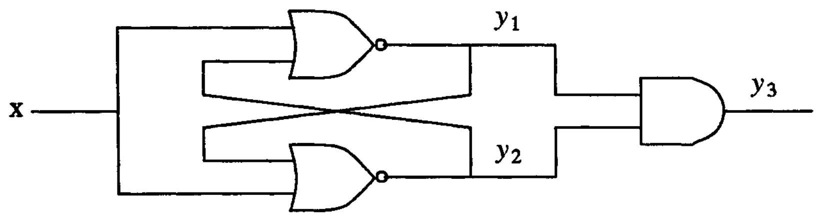


Figure 5.6. Network N_{15}.

0; what will be the new state of the network? The ternary algorithm gives the following results:

$$t^1 = \tfrac{1}{2}\,\tfrac{1}{2}\,0$$

$$y^1 = (\,(0+t_2^1)',\ (0+t_1^1)', 0\,) = \tfrac{1}{2}\,\tfrac{1}{2}\,0$$

$$t^2 = \tfrac{1}{2}\,\tfrac{1}{2}\,\tfrac{1}{2}$$

$$y^2 = (\,(0+t_2^2)',\ (0+t_1^2)', t_1^2 t_2^2\,) = \tfrac{1}{2}\,\tfrac{1}{2}\,\tfrac{1}{2}$$

It is easy to verify that this state is stable in the ternary algorithm. However, in Fig. 5.7 we show a part of the graph for the AED relation. Note that after an even number of race units (>2) the states 010, 100, and 011 are reachable (there might be others too). Similarly, after an odd number of race units (>3) the states 010, 100, and 111 are reachable (there might be others too). It is also straightforward to see that only the states 100, 010, and 110 are reachable after one race unit. Hence we can conclude for this example that the ternary algorithm gives a result that corresponds exactly to the binary AED model. Note that the AED model has to be extended to multiple winners, otherwise the network would only end up in 010 or 100.

To simplify the graphs for the binary AED analysis, we will show only the states reachable after some number of complete race units from now on, i.e. there will be an arrow between state *a* and state *b* iff *b* is reachable from *a* after one complete race unit.

All the previous examples have indicated that the results of the ternary algorithm and the binary AED model correspond to each other. However, the following example shows that that is not always the case. Study the network N_{16}

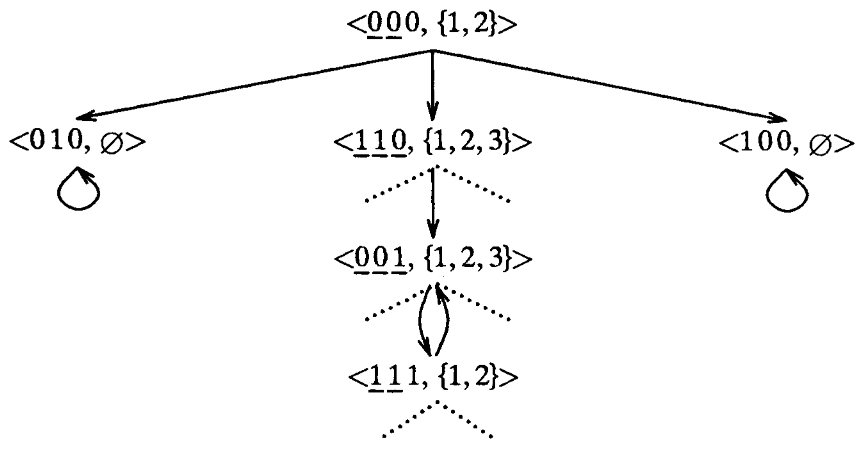


Figure 5.7. Part of the AED graph for network N_{15}.

of Fig. 5.8.

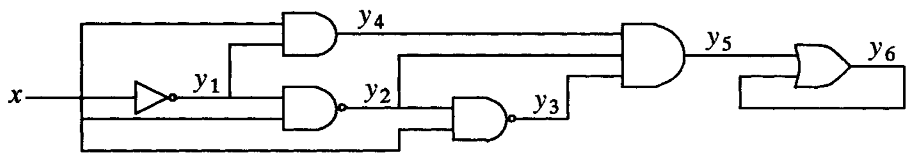


Figure 5.8. Network N_{18}.

We start in the stable total state $x = 0$, $y = 111000$, and change x to 1. The ternary algorithm yields the following results:

$$\begin{aligned} t^1 &= \tfrac12\tfrac12\tfrac12\tfrac12 0\,0 \\ y^1 &= 0\tfrac12\tfrac12\tfrac12 0\,0 \\ t^2 &= 0\tfrac12\tfrac12\tfrac12\tfrac12 0 \\ y^2 &= 0\,1\tfrac12 0\tfrac12 0 \\ t^3 &= 0\,1\tfrac12 0\tfrac12\tfrac12 \\ y^3 &= 0\,1\,0\,0\,0\tfrac12 \end{aligned}$$

It is easy to verify that the last state is stable. In Fig. 5.9 we show the result according to the binary AED model. Note that we use the simplified notation described above. After one race unit we can reach the states 010000, 010100, 000000, 000100, 001000, and 001100. Note that the μ-average of these vectors is equal to y^1 obtained from the ternary algorithm. However, after the second race unit we can only reach the states 011000 and 010000, but the ternary algorithm gives $y^2 = 0\,1\,\tfrac12\,0\,\tfrac12\,0$. Note that we do not have equality any more, but that the ternary vector covers the μ-average of the binary outcomes of the AED analysis. Similarly, after 3 or more race units, the binary race analysis predicts that the network will reliably end up in the state 010000, whereas the ternary predicts

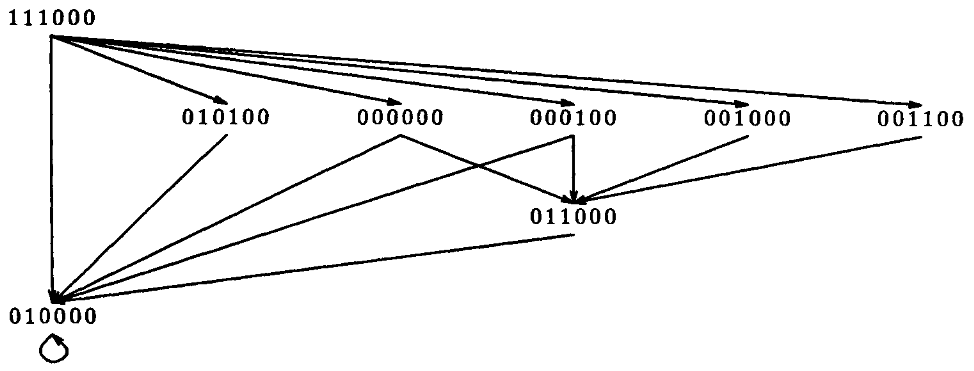


Figure 5.9. Race analysis of N_{18} according to the AED model.

01000½. Note again that the ternary result covers the binary. This leads to the following conjecture:

Conjecture If $\mathbf{y}^i$ is the result after i steps of the ternary algorithm described above, then

$$\mathbf{y}^i \geq \mu\{\text{ all states reachable after } i \text{ complete race units according to the AED model}\}.$$

Our final example is a commercially available integrated-circuit JK flip-flop SN74H76 [34], shown in Fig. 5.10. It is easy to verify that the total state $P = 1$, $C = 1$, $J = 1$, $K = 1$, $\text{¢} = 1$, and $y_1 \cdots y_8 = 10011110$ is a stable state. The transition we will study is when C changes to 0. In Fig. 5.11 we show the ternary results beside the binary race analysis according to the AED model. Note that for clarity some states are repeated in the binary race analysis. The results of the two analyses correspond exactly.

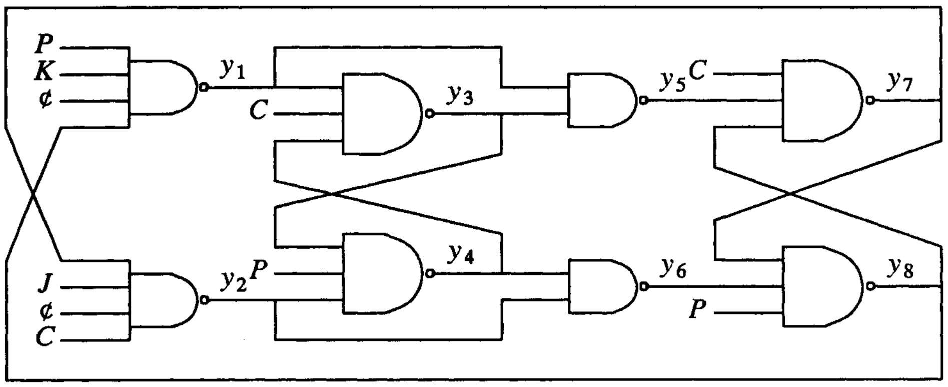


Figure 5.10. JK flip-flop SN74H76.

$y^0 = 10011110$

$t^1 = 1\tfrac{1}{2}\tfrac{1}{2}11110$

$y^1 = 11111110$

$t^2 = 111\tfrac{1}{2}\tfrac{1}{2}\tfrac{1}{2}10$

$y^2 = 11100\tfrac{1}{2}10$

$t^3 = 11100\tfrac{1}{2}1\tfrac{1}{2}$

$y^3 = 1110011\tfrac{1}{2}$

$t^4 = \tfrac{1}{2}110011\tfrac{1}{2}$

$y^4 = \tfrac{1}{2}1100110$

$t^5 = \tfrac{1}{2}110\tfrac{1}{2}110$

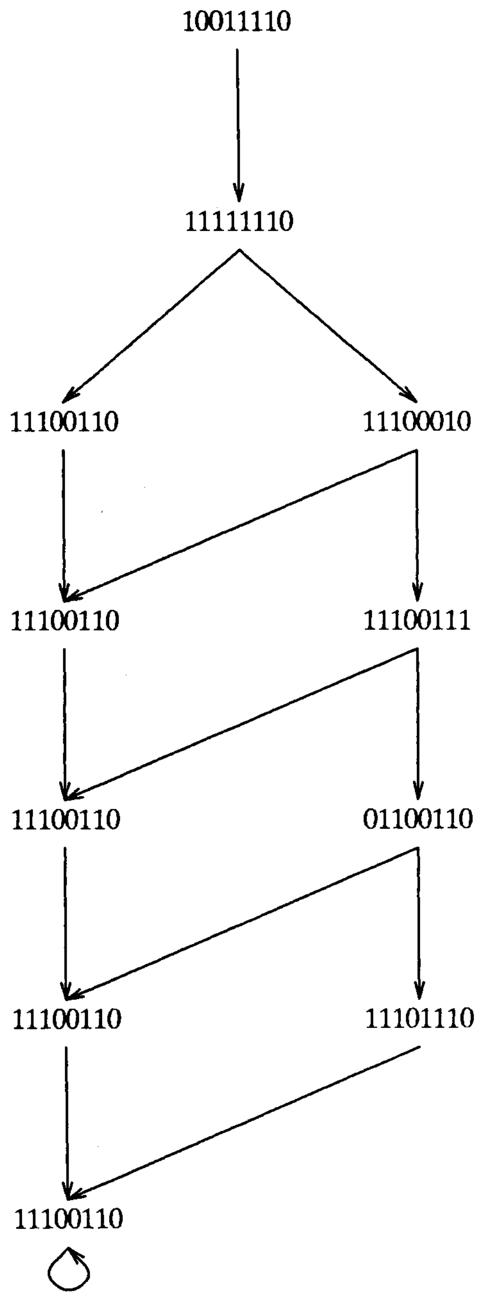
$y^5 = 1110\tfrac{1}{2}110$

$t^6 = 1110\tfrac{1}{2}110$

$y^6 = 11100110$

$\circlearrowleft$

(a)



(b)

Figure 5.11. Analysis of SN74H76: (a) ternary; (b) binary AED method.

The usefulness of the ternary algorithm described above, can be argued. First of all, the ternary algorithm is not guaranteed to halt. This can easily be seen from an oscillating circuit, e.g. an inverter with its output connected to its input. Furthermore, even if the conjecture above is true, the example of Fig. 5.8 shows that complete correspondence cannot be obtained. An even more serious disadvantage, is that the corresponding binary method for analyzing races, the Almost Equal Delay method, relies on some very severe assumptions that are unlikely to hold in general. However, we hope that the above ternary algorithm is just the first one in a class of modified ternary algorithms that reduce the pessimism in the Eichelberger-Brzozowski-Yoeli algorithm, but can still be shown to correspond to some reasonable binary race model.

5.5. Summary

The main result of this thesis is the complete characterization of the ternary simulation. Some interesting corollaries follows from this characterization, e.g. that no circuit in which wires and gates have delays can have a critical race without also having the possibility of entering an oscillation. Another result is the proposition about static output hazard detection, proved in Section 3. However, as was shown in the previous sections, there are a number of problems with the ternary simulation that sometimes makes it less useful. The problem of finding a modified version of the ternary simulation algorithm, in which the pessimism is reduced, remains unanswered, and deserves further study.

In the previous sections we have pointed out a number of open problems. For example, detection of dynamic output hazards using ternary simulation, adjusting the ternary simulation to hierarchical design practices, characterization of the ternary simulation applied to switch-level models of MOS circuits, and the conjecture about correspondence between the ternary algorithm described in Section 4, and the binary Almost Equal Delay model, are all open problems.

We strongly believe that asynchronous design will become of much greater importance than today, when very fast circuits are to be constructed. This will be true in particular for VLSI circuits, where the delays in the "wires" on the chip are relatively large, and hence the skew in the clock signals can become significant. There are however a number of problems that have to be solved first. Some of these problems are related to the analysis phase, but the much harder ones concern the synthesis phase. Today's design practice is based on the Huffman model, but as was shown in Chapter II, the model is not always accurate. One interesting possibility is to use a ternary approach in the synthesis phase. Whether this is possible, remains an open question.

References

1. Bryant, R. E., Race Detection in MOS Circuits by Ternary Simulation, pp. 85-95, in *VLSI '83*, F. Anceau and E. J. Aas eds., Elsevier Science Publishers B.V., North Holland (1983).
2. Bryant, R. E., Toward a Proof of the Brzozowski-Yoeli Conjecture on Ternary Simulation, Unpublished manuscript, (Dec. 1983).
3. Bryant, R. E., A Switch-Level Model and Simulator for MOS Digital Systems, *IEEE Transactions on Computers*, Vol. C-33(2), pp. 160-177 (Feb. 1984).
4. Brzozowski, J. A., Some Problems in Relay Circuit Design, *IEEE Transactions on Electronic Computers*, Vol. EC-14(4), pp. 630-634 (Aug. 1965).
5. Brzozowski, J. A. and Yoeli M., Models for Analysis of Races in Sequential Networks, pp. 26-32 in *Lecture Notes in Computer Science*, A. Blikle ed., Springer Verlag (1975).
6. Brzozowski, J. A. and M. Yoeli, *Digital Networks*, Prentice-Hall, Englewood Cliffs, New Jersey (1976).
7. Brzozowski, J. A. and M. Yoeli, Practical Approach to Asynchronous Gate Networks, *Proc. IEE*, Vol. 123(6), pp. 495-498 (Jun. 1976).
8. Brzozowski, J. A. and M. Yoeli, On a Ternary Model of Gate Networks, *IEEE Transactions on Computers*, Vol. C-28(3), pp. 178-183 (Mar. 1979).
9. Burks, A. W., Electronic Computing Circuits for the ENIAC, *Proc. IRE*, Vol. 35, pp. 756-767 (1947).
10. Dill, D. L. and E. M. Clarke, Automatic Verification of Asynchronous Circuits Using Temporal Logic, *Proc. 1985 Chapel Hill Conference on VLSI*, pp. 127-143, Computer Science Press, (1985).
11. Eichelberger, E. B., Hazard Detection in Combinational and Sequential Switching Circuits, *IBM Journal of Research and Development*, Vol. 9, pp. 90-99 (Mar. 1965).
12. Estrin, G., The Elecronic Computer at the Institute for Advanced Study, *Math. Tables and other Aids to Computation*, Vol. 7, pp. 108-114 (1953).
13. Fantauzzi, G., An Algebraic Model for the Analysis of Logical Circuits, *IEEE Transactions on Computers*, Vol. C-23(6), pp. 576-581 (Jun. 1974).
14. Friedman, A. D. and P. R. Menon, *Theory and Design of Switching Circuits*, Computer Science Press Inc., Woodland Hills, California (1975).

15. Friedman, A. D. and P. R. Menon, Synthesis of Asynchronous Sequential Circuits with Multiple-Input Changes, *IEEE Transactions on Computers*, Vol. C-17(6), pp. 559-566 (Jun. 1968).

16. Hackbart, R. R. and D. L. Dietmeyer, The Avoidance and Elimination of Function Hazards in Asynchronous Sequential Circuits, *IEEE Transactions on Computers*, Vol. C-20(2), pp. 184-189 (Feb. 1971).

17. Holt, D. and D. Hutchings, A MOS/LSI Oriented Logic Simulator, *Proc. 18th Design Automation Conf.*, pp. 280-287 (Jun. 1981).

18. Huffman, D. A., The Synthesis of Sequential Switching Circuits, pp. 3-62 in *Sequential Machines: Selected Papers*, E. F. Moore ed., Addison-Wesley, Reading, Massachusetts (1964). First appeared in the *J. Franklin Inst.*, 257(3-4), (1954).

19. Jephson, J. S, R. P. McQuarrie, and R. E. Vogelsberg, A Three-Level Design Verification System, *IBM Systems Journal*, Vol. 8(3), pp. 178-188 (Mar. 1969).

20. Langdon, G. G., Analysis of Asynchronous Circuits Under Different Delay Assumptions, *IEEE Transactions on Computers*, Vol. C-17, pp. 1131-1143 (Dec. 1968).

21. Lengauer, T. and S. Naher, An Analysis of Ternary Simulation as a Tool for Race-Detection in Digital MOS Circuits, *VLSI: Algorithms and Architectures, International Workshop, Amalfi, Italy*, (1984).

22. Mano, M. M., *Digital Design*, Prentice-Hall, Englewood Cliffs, New Jersey (1984).

23. Martin, A. J., The Design of a Self-Timed Circuit for Distributed Mutual Exclusion, *Proc. 1985 Chapel Hill Conference on VLSI*, pp. 245-260, Computer Science Press, (1985).

24. McCluskey, E. J. Jr., Transients in Combinational Logic Circuits, pp. 9-46 in *Redundancy Techniques for Computing Systems*, Wilcox, R. H. and W. C. Mann eds., Spartan Books, Washington (1962).

25. Miller, R. E., *Switching Theory*, Wiley, New York (1965).

26. Molnar, C. E., T.-P. Fang, and F. U. Rosenberger, Synthesis of Delay-Insensitive Modules, *Proc. 1985 Chapel Hill Conference on VLSI*, pp. 67-86, Computer Science Press, (1985).

27. Montgomerie, G. A., Sketch for an algebra of relay and contact circuits, *Proc. IEE*, Vol. 95, pp. 303-312 (1948).

28. Muller, D. E., *Lecture Notes on Asynchronous Circuit Theory*, Digital Computer Laboratory, University of Illinois, (Spring 1961).

29. Muller, D. E., Treatment of Transition Signals in Electronic Switching Circuits by Algebraic Methods, *IRE Transactions on Electronic Computers*, Vol. EC-8 p. 401 (Sep. 1959).

30. Muller, D. E., The General Synthesis Problem for Asynchronous Digital Networks, *Eighth Annual Symposium on Switching and Automata Theory*, pp. 71-82 (Oct. 1967).

31. Ralston, A. and E. D. Reilly, Jr., *Encyclopedia of Computer Science and Engineering*, Second Edition, Van Nostrand Reinhold Company, New York (1983).

32. Seitz, C. L., System Timing, pp. 218-262 in *Introduction to VLSI Systems*, Mead, C. A., and L. A. Conway, Addison-Wesley Publishing Company, Reading, Massachusetts (1980).

33. Sherwood, W., A MOS Modelling Technique for 4-State True-Value Hierarchical Simulation, *Proc. 18th Design Automation Conf.*, pp. 775-785 (Jun. 1981).

34. Texas Instruments Staff, *The TTL Data Book for Design Engineers*, Texas Instruments Inc. (1973).

35. Unger, S. H., Hazards and Delays in Asynchronous Sequential Switching Circuits, *IRE Transactions on Circuit Theory*, Vol. CT-6, pp. 12-25 (Mar. 1959).

36. Unger, S. T., *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, New York (1969).

37. Yoeli, M. and S. Rinon, Application of Ternary Algebra to the Study of Static Hazards, *Journal of the Association for Computing Machinery*, Vol. 11(1), pp. 84-97 (Jan. 1964).