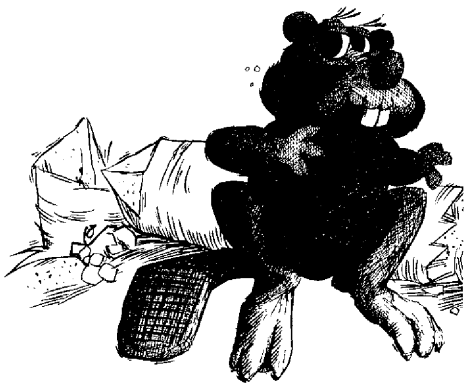
UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT



A Mathematical Model of Digital CMOS Networks

M. Yoeli
J.A. Brzozowski

CS-84-22

August, 1984

UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

A MATHEMATICAL MODEL OF DIGITAL CMOS NETWORKS

M. Yoeli

Department of Computer Science
Technion
Haifa, Israel

J.A. Brzozowski

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada

ABSTRACT

In this paper we develop a formal mathematical model for describing the logical behavior of digital CMOS networks; both combinational and asynchronous sequential networks are covered. The model is particularly tailored towards CMOS networks, taking into account the differences between the design principles of CMOS versus those of NMOS. We use a switch-level model based on a labeled graph, which is an abstraction of the wiring diagram of the network. We show that two analysis procedures, namely binary race analysis and ternary simulation, previously developed for gate networks, can be extended to CMOS networks. The model is also suitable for the representation of faulty networks, and is therefore applicable for designing fault detection sequences. We introduce the concept of well-formed networks and reject poorly designed networks early in our analysis procedure. Therefore the model provides a basis for computationally efficient simulators.

This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. A-1617.

1. INTRODUCTION

CMOS technology is playing an increasingly important role in VLSI (see for example [8, 13]). Although some authors [1, 5] have developed unified MOS models covering both NMOS and CMOS, there are in fact significant differences between the design principles of CMOS and those of NMOS [4, 8]. In this paper we develop a mathematical model particularly tailored towards digital CMOS networks. Consequently, our model is conceptually considerably simpler than the unified approaches. Furthermore, it captures the design principles that are special to CMOS (e.g. transmission gates).

It is well-known that conventional switching theory based on gate networks is not suitable for digital MOS networks, and that a switch-level model is required [1, 2 (Appendix D), 5]. We develop here a switch-level model based on a labeled-graph representation which can be easily derived from the wiring diagram of a network. Somewhat related concepts appear in [1, 10].

In [3] we presented a mathematical theory of the behavior of asynchronous sequential gate networks. Two analysis procedures were described, namely, binary race analysis and ternary simulation, and a theorem relating these two approaches was proved. In the present paper we show that these results can be adapted to a large class of CMOS networks.

Simulators using very general MOS models require a lot of computation because they cover a wide range of technologies and they pursue the analysis of poorly designed networks. We restrict our attention to well-designed CMOS networks, and thereby provide a basis for an efficient simulator. In this respect, our approach is similar to that of [9] where a restricted class of synchronous NMOS networks is analyzed.

Our CMOS graph model is also suitable for the representation and analysis of faulty networks, as shown in Section 7. Thus there is no need to introduce special gate models [12] or special switch-level models for the purpose of designing fault detection sequences.

2. CMOS NETWORKS - INFORMAL DISCUSSION

The fundamental building blocks of CMOS networks are the n-channel and p-channel enhancement mode transistors [4, 8]. In Figure 2.1 we show commonly used symbols for the n-channel and p-channel transistors, along with the "graph" symbols that we will use in this paper. The graph of Figure 2.2 shows a typical simple CMOS network, namely a two-input NOR gate. In that figure 1 represents voltage VDD which is the positive end of the power supply, and 0 represents voltage VSS which is the negative end of the power supply. The symbols 0 and 1 also represent the customary logical values, using positive logic.

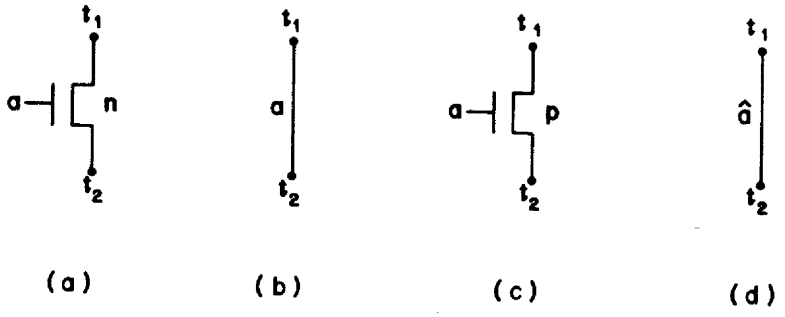


Figure 2.1 Transistor symbols: (a) n-channel transistor; (b) graph notation for (a); (c) p-channel transistor; (d) graph notation for (c).

Refer to Figure 2.1(a). Usually an n-channel transistor behaves like a switch between terminals t_1 and t_2 where the state of the switch depends on the value of the control or "gate" input a. In particular when $a = 1$ the switch is "closed" meaning that the terminals t_1 and t_2 are connected by a low-resistance path. If $a = 0$, the switch is "open," meaning that there is very high resistance between t_1 and t_2. Similarly, in the p-channel transistor the switch is closed when $a = 0$ and open when $a = 1$. In our graph representation a connection between t_1 and t_2 exists if the edge is labeled 1 or $\hat{0}$.

Consider now the NOR gate of Figure 2.2. There is a connection between the input node labeled 0 and the output node labeled z iff $a+b = 1$, where $+$ denotes logical OR. Similarly, there is a connection between z and the input node labeled 1 iff $a+b = 0$ i.e. $a' \cdot b' = 1$, where $\cdot$ denotes logical AND, and a' is the complement of a. Clearly z is always connected either to 0 or to 1 but not both. Evidently $z = a' \cdot b' = (a+b)'$ showing that the network realizes the NOR function.

The switch model described above is only an approximate representation of CMOS design rules for the following reasons. Any node that is not an input node is called a storage node. A storage node y has a "reliable" value of 1 under

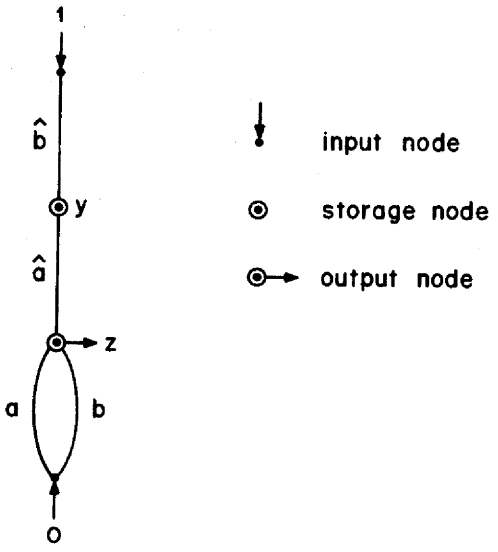


Figure 2.2 CMOS NOR gate.

stable conditions if there is a connection via a series of closed p-transistors from y to an input node labeled 1, and there is no connection to an input node labeled 0. Similarly y has a "reliable" value of 0 if there is an "n-connection" to 0 and no connection to 1. The motivation for this design rule is the fact that a closed n-transistor transmits 0-signals reliably but 1-signals less reliably. Similarly a closed p-transistor transmits 1-signals reliably and 0-signals less reliably† [4, 11].

A switch that transmits both 0-signals and 1-signals reliably under the control of an input a is implemented by means of a two-transistor "transmission gate" [4, 8, 11] as shown in Figure 2.3(a). For, if $a = 1$ and t_1 is an input node and has the value 0, then the n-transistor controlled by a provides a proper connection; if t_1 has the value 1 the p-transistor controlled by $b = a'$ provides a proper connection. In any case t_2 gets the value of t_1. A transmission gate controlled by a' is shown in Figure 2.3(c).

Return now to the NOR gate of Figure 2.2. Although the storage node z has only the values 0 and 1 for any input combination, this is not true of node y. First, when $a = 1$ and $b = 1$, node y is not connected to anything; it is then said to be *floating*. We will associate the value "3" with such a floating condition. Such a storage node has some memory capability (due to the presence of capacitance) permitting it to remember its previous binary state for some time. We will discuss this property later. Second, consider the input $a = 0$, $b = 1$. Then y has no connection to 1 but only a "mixed" connection to 0. We will

† Note that in dynamic NMOS technology these "unreliable" switches do play an important role as "pass transistors" [7].

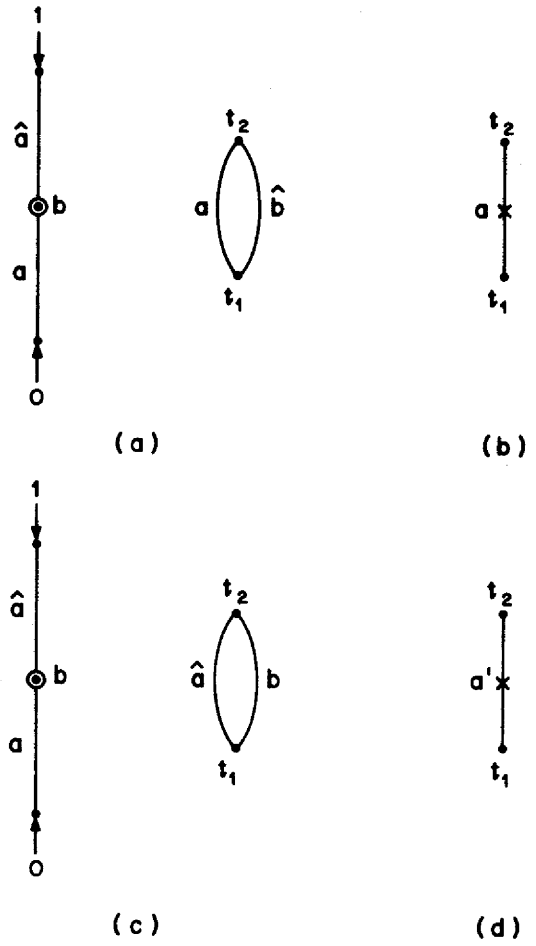


Figure 2.3 (a) Transmission gate controlled by a; (b) simplified symbol for (a); (c) transmission gate controlled by a'; (d) simplified symbol for (c).

denote the state of the node under such circumstances by the value "2", representing an *unknown* state.

We consider the distinction between node states 2 and 3 essential, in contrast with 3-valued models (e.g. [1]). Four values have been used in [12], in connection with fault detection in CMOS networks.

Our final example for this section is the network of Figure 2.4, which realizes the boolean function $z = ab + a'c$.

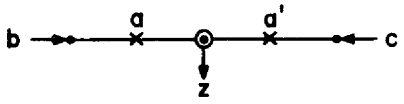


Figure 2.4

3. CMOS GRAPHS

In this section we formalize the graph notation introduced in Section 2.

A *CMOS graph* C consists of the following:

(a) A finite, undirected graph $G = (V,E)$, labeled as described below.

(b) Sets $X,Y,Z \subset V$ of *input*, *storage*, and *output* nodes respectively, where

$$X \cap Y = \emptyset, \quad X \cup Y = V, \quad \text{and} \quad Z \subseteq Y.$$

(c) A finite *input alphabet* A and constant inputs 0 and 1, where $A \cap \{0,1\} = \emptyset$.

(d) An *input-node labeling*, assigning to each input node an element from $A \cup \{0,1\}$.

(e) An *edge labeling*, assigning to each edge of G an element of $\Sigma \cup \hat{\Sigma}$, where

$$\Sigma = \{0,1\} \cup A \cup Y \quad \text{and} \quad \hat{\Sigma} = \{\hat{\sigma} \mid \sigma \in \Sigma\}.$$

A storage node $y \in Y$ is called a *key node* iff y or $\hat{y}$ is a label of some edge. An *input state* of a CMOS graph C is a mapping $s_A : A \to \{0,1\}$. A *storage state* of C is a mapping $s_Y : Y \to \{0,1,2,3\}$.

Let $K = \{k_1, \ldots, k_h\}$ be the set of all key nodes. A *key state* is any 4-valued h-tuple $(q_1, \ldots, q_h)$ where $q_i \in \{0,1,2,3\}$ for $i = 1, \ldots, h$. The key state corresponding to a given storage state s_Y is the h-tuple $(s_Y(k_1), \ldots, s_Y(k_h))$. A *total state* of C is a pair $s = (s_A, s_Y)$. An *input-key state* of C is a pair $w = (s_A, q)$ where q is a key state. A key state $q = (q_1, \ldots, q_h)$ is *clean* iff $q_i \in \{0,1\}$ for $i = 1, \ldots, h$. An input-key state is clean iff its key state is clean.

Given a CMOS graph C and a clean input-key state $w = (s_A, q)$ we obtain the CMOS graph C_w by replacing each edge label and each input node label $\sigma \in \Sigma$ by $w(\sigma)$, where

$$w(\sigma) = \begin{cases} \sigma\ , & \text{if } \sigma \in \{0,1\}\ ; \\ s_A(\sigma)\ , & \text{if } \sigma \in A\ ; \\ q_i\ , & \text{if } \sigma = k_i \in K\ . \end{cases}$$

Let w be a clean input-key state of CMOS graph C. A path in C_w is called

(a) a *connection* iff every edge label in the path is either $\hat{0}$ or 1.

(b) an *n-connection* iff every edge label in the path is 1.

(c) a *p-connection* iff every edge label in the path is $\hat{0}$.

(d) a *mixed connection* iff it is a connection, but is neither an n-connection nor a p-connection.

Any connection between an input node labeled 0 and an input node labeled 1 is called a *fight* [11].

For a clean input-key state w we define the *excitation state* of C_w to be the mapping

$$e_w : Y \to \{0,1,2,3\}$$

where for $y \in Y$,

(a) $e_w(y) = 0$ iff there exists an n-connection from y to an input node labeled 0, and no connection to an input node labeled 1. (Note that, under these conditions, the signal at node y will tend to become 0.)

(b) $e_w(y) = 1$ iff there exists a p-connection from y to an input node labeled 1, and no connection to an input node labeled 0. (Here the signal tends to become 1.)

(c) $e_w(y) = 3$ iff there is no connection from y to any input node. (Here node y is electrically isolated.)

(d) $e_w(y) = 2$, otherwise. (Here the signal at node y may tend to take on some value between VSS and VDD.)

A clean total state $s = (s_A, s_Y)$ of C is *stable* iff $s_Y = e_w$, where w is the input-key state of s. Evidently, if s and t are clean stable total states of C which agree in the input and key components, then $s = t$.

In this paper we consider only CMOS graphs satisfying:

Conditions C

C1. For every clean input-key state w the key-state excitation is clean, i.e. $e_w(k_i) \in \{0,1\}$, for $i = 1, \ldots, h$.

C2. Every clean stable total state has no fights.

C3. In every clean stable total state no output node is labeled 2.

A CMOS graph satisfying these conditions will be called *well-formed*†. We view Conditions C as recommendations for sound modular design of complex CMOS VLSI systems. In general, a violation of these rules tends to make the circuit designer's task extremely difficult. These rules also simplify the design of simulators.

The following propositions may be useful in testing whether a CMOS graph C is well-formed. They are easily verified.

Suppose a CMOS graph C consists of n disjoint subgraphs $C_1, \ldots, C_n$. For each $i = 1, \ldots, n$ we define the CMOS graph $\overline{C}_i$ as follows:

(i) The labeled graph of $\overline{C}_i$ is C_i.

(ii) The input alphabet of $\overline{C}_i$ consists of the edge-label variables of C_i which are not key-node variables of C_i.

A subgraph C_i is *well-formed with respect to* C iff, for every clean input-key state of $\overline{C}_i$, the excitation of every key node of C appearing in C_i is clean.

Proposition 3.1 *Suppose C consists of n disjoint subgraphs $C_1, \ldots, C_n$. If C_i is well-formed with respect to C for all i, $i = 1, \ldots, n$, then C is well-formed.*

Proposition 3.2 *Let k be a key node in C and let f_0 and f_1 be boolean functions of the key node variables restricted to 0 and 1 and of the inputs defined as follows:*

$f_0 = 1$ iff there is an n-path from k to 0,
$f_1 = 1$ iff there is a p-path from k to 1.

Then $f_0 = f_1'$ iff condition C1 is satisfied for k.

A well-known CMOS design method results in networks consisting of disjoint components of the type shown in Figure 3.1, where the p-part is a connected subgraph of C with all edge labels in $\hat{\Sigma}$, and the n-part is a connected subgraph of C with all the edge labels in Σ. Furthermore, if f_0 and f_1 for node y_i are defined as in Proposition 3.2, then $f_0 = f_1'$. Evidently f_0 depends only on the n-part and f_1 only on the p-part.

---

† The concept of well-formed graph applies also to CMOS T-graphs of Section 4.

Proposition 3.3 *Let C be a CMOS graph consisting of disjoint subgraphs of the type of Figure 3.1 with one such subgraph for every key node and every output node. Then C is well-formed.*

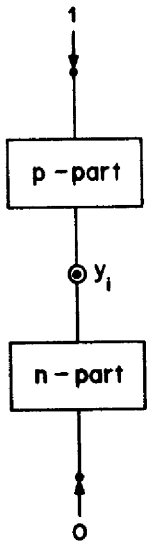


Figure 3.1 Complementary design.

Example 3.1 (SR_Latch) Consider the graph of Figure 3.2. Here $A = \{S,R\}$, $Y = \{\overline{Q},Q,y_1,y_2\}$, and $K = \{\overline{Q},Q\} = Z$. One verifies that the following total states are clean and stable, and that there are no other stable states:

S	R	$\overline{Q}$	Q	y_1	y_2
0	0	0	1	1	1
0	0	1	0	1	1
0	1	1	0	1	3
1	0	0	1	3	1
1	1	0	0	2	2

For example, consider the last stable state. The graph C_w for this state is shown in Figure 3.3. The node y_2 has no connection to 1 and a mixed connection to 0. Hence $e_w(y_2) = 2$.

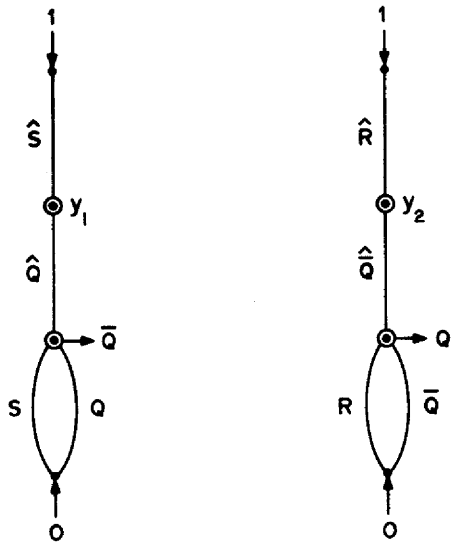


Figure 3.2 NOR latch.

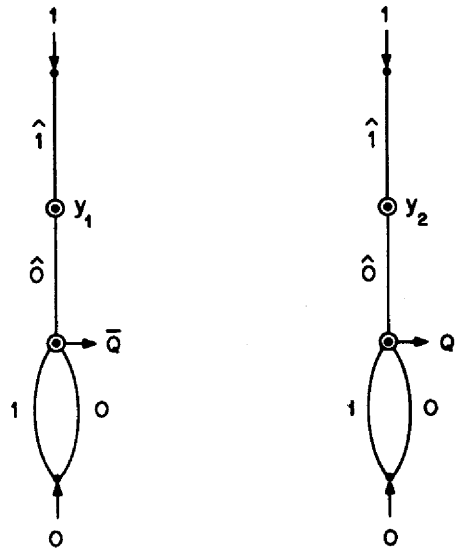


Figure 3.3 The graph C_w.

Using Proposition 3.3 one verifies that the graph of Figure 3.2 is well-formed.

Three additional examples of CMOS graphs are shown in Figure 3.4. The graph of Figure 3.4(a) is well-formed and behaves like an oscillator, i.e. has no stable states. The graph of Figure 3.4(b) is also well-formed. It is a combinational circuit realizing the boolean functions $z_1 = (ab)'$, $z_2 = (ac)'$. The graph

of Figure 3.4(c) is not well-formed because, for the clean input-key state $a = 0$, $b = 0$, $c = 0$, $y_1 = 0$, $y_2 = 0$, the excitation of the key node y_2 is 2, violating Condition C1. We return to this graph in the next section.

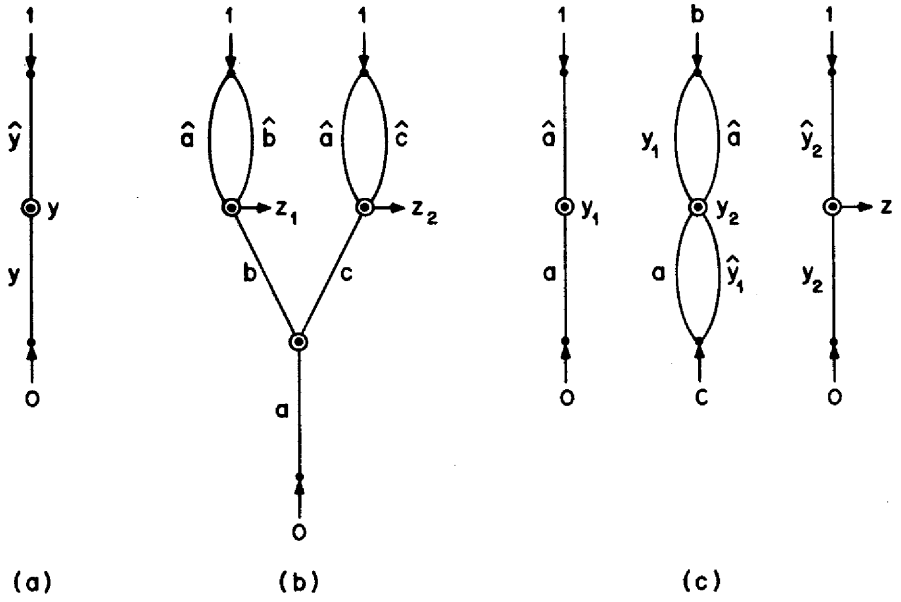


Figure 3.4 Other examples of CMOS graphs: (a) oscillator; (b) a combinational network; (c) a network that is not well-formed.

4. CMOS *T*-GRAPHS

Many commonly used CMOS networks can be modeled by well-formed CMOS graphs. However, other useful networks, when modeled by CMOS graphs as defined so far, do not satisfy all of the conditions C1-C3. The graph of Figure 3.4(c) is one such example. With respect to terminals b and c this graph represents a pair of transmission gates.

One problem with the graph of Figure 3.4(c) is that the excitation of key node y_2 becomes 2 when $a = b = c = y_1 = y_2 = 0$. This happens only when a has changed from 1 to 0 but y_1 has not yet responded to this change. In practice, the storage node y_2 remembers its previous state $y_2 = 0$ during such a transient condition. However, our CMOS graph model does not rely on any such assumptions of temporary storage capabilities of storage nodes. The extended model about to be introduced does take this assumption into account.

We generalize the definition of CMOS graph by permitting transmission gates as basic building blocks. A *CMOS T-graph* consists of parts (a)-(d) of a CMOS graph and a modified part (e), namely

(e') An *edge labeling*, assigning to each edge of G an element of $\Gamma \cup \hat{\Gamma}$, where

$$\Gamma = \Sigma \cup \{a' \mid a \in A\} \cup \{y' \mid y \in Y\},$$

and

$$\hat{\Gamma} = \{\hat{\gamma} \mid \gamma \in \Gamma\}.$$

Furthermore, this edge labeling is restricted as follows. An edge labeled σ', $\sigma \in A \cup Y$, must appear in parallel with an edge labeled $\hat{\sigma}$. Similarly, an edge labeled $\hat{\sigma'}$ must appear in parallel with an edge labeled σ. Thus edges with complemented variables appear only as parts of transmission gates.

Observe that every CMOS graph is also a CMOS T-graph. We also modify the definition of graph C_w in the obvious sense, using $0' = 1$ and $1' = 0$. Finally, we define a CMOS T-graph to be *well-formed* iff it satisfies conditions C1-C3.

Note that Propositions 3.1 and 3.2 apply also to CMOS T-graphs.

Example 4.1 The graph of Figure 3.4(c) will now be represented by the CMOS T-graph of Figure 4.1(a). Observe now that the excitation $\tilde{y}_2$ of y_2 is

$$\tilde{y}_2 = \begin{cases} b & \text{if } a = 0, \\ c & \text{if } a = 1. \end{cases}$$

Thus C1 holds. Also C2 and C3 are easily verified. Therefore the graph of Figure 4.1(a) is well-formed.

Example 4.2 (D Flip-Flop) In the T-graph of Figure 4.2 the excitation of key nodes y and $\overline{Q}$ is as follows:

$$\text{If } c = 1 \text{ then } \tilde{y} = D \text{ and } \tilde{\overline{Q}} = y';$$

$$\text{if } c = 0 \text{ then } \tilde{y} = \overline{Q}' \text{ and } \tilde{\overline{Q}} = y'.$$

Evidently the T-graph is well-formed.

Example 4.3 (Tally Network [10]) In Figure 4.3 the inputs are a and b and the outputs are z_0, z_1, z_2. One verifies that the T-graph is well-formed and realizes the boolean functions:

$$z_0 = a'b', \quad z_1 = a'b + b'a, \quad z_2 = ab.$$

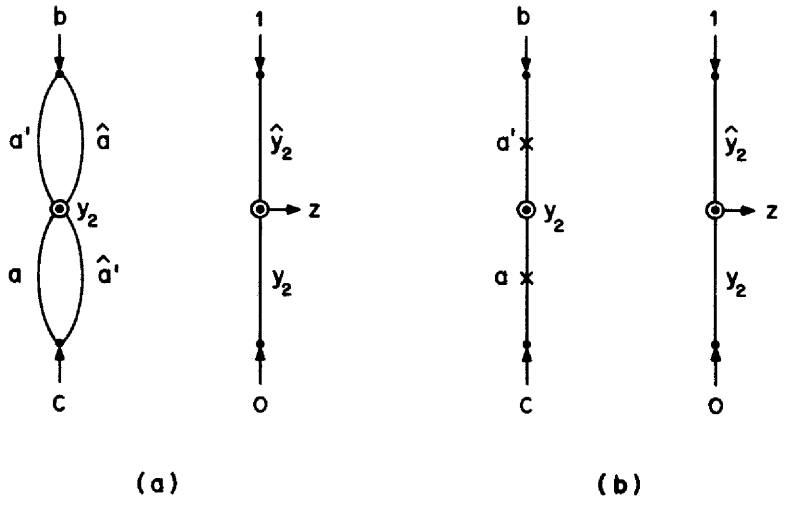


Figure 4.1 (a) *T*-graph for Figure 3.4(c); (b) simplified notation for (a).

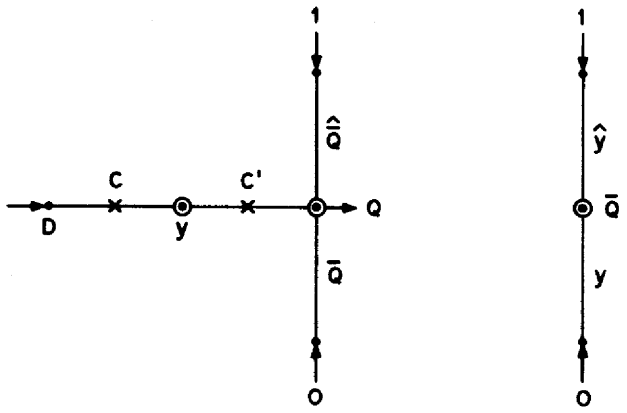


Figure 4.2

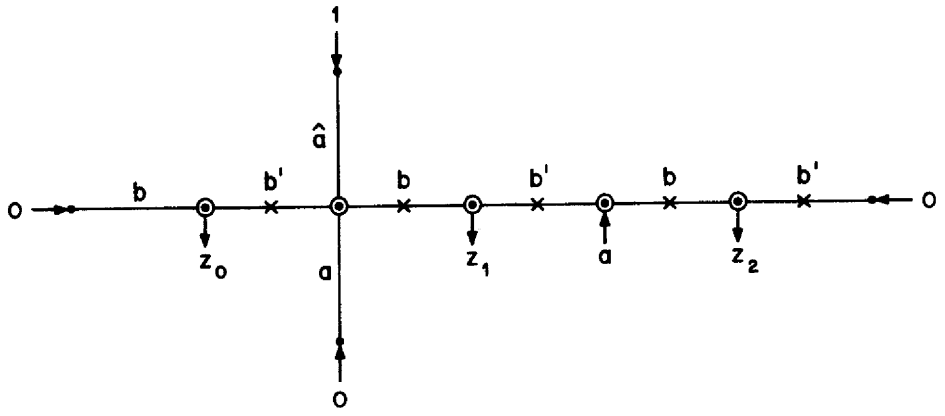


Figure 4.3 Two-variable tally network.

Example 4.4 Consider the input-key state $a = 1$, $y_1 = 1$, $y_2 = 0$, $y_3 = 1$ in Figure 4.4. Then the excitation of node y_3 is 2, showing that the graph is not well-formed. However, this is only a transient condition, because eventually the network will reach the stable total state $a = 1$, $y_1 = 0$, $y_2 = 1$, $y_3 = 0$, $y_4 = 1$. Nevertheless, our model rejects the network because of the presence of a temporary fight.

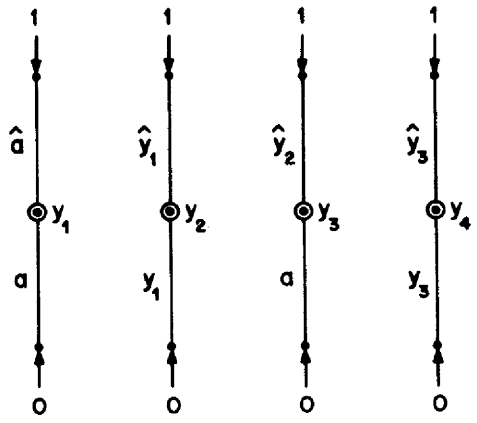


Figure 4.4 A graph which is not well-formed.

5. BEHAVIOR OF CMOS NETWORKS

In this section we study the behavior of asynchronous CMOS networks, adapting the approach of [3]. More specifically, we would like to analyze the behavior of a CMOS T-graph when started in an arbitrary clean total state. We will assume that the input state is kept constant until the network has a chance to "settle."

Given a CMOS T-graph C, satisfying Condition C1, let α be an input tuple. Define a function $F_\alpha : \{0,1\}^h \to \{0,1\}^h$ by

$$F_\alpha(q_1, \ldots, q_h) = (\tilde{q}_1, \ldots, \tilde{q}_h),$$

where $q = q_1, \ldots, q_h$ is an arbitrary clean key state, $\tilde{q}_i = e_w(k_i)$, for $i = 1, \ldots, h$, and $w = (\alpha, q)$.

Let $q, \tilde{q} \in \{0,1\}^h$. The *interval* $[q, \tilde{q}]$ between q and $\tilde{q}$ is defined to be:

$$[q, \tilde{q}] = \left\{ p \in \{0,1\}^h \mid p_i = q_i \text{ or } p_i = \tilde{q}_i, \text{ for all } i,\ i = 1, \ldots, h \right\}.$$

Under the above conditions, we now describe all possible successor states of a given key state q. If $\tilde{q}_i \neq q_i$ then q_i tends to become $\tilde{q}_i$. However, there may be a "race" among a number of key-node variables. In the *GMW* (General Multiple Winner) *model* we assume that any subset of the set of racing variables may win the race. Thus we consider any state in the interval $[q, \tilde{q}] = [q, F_\alpha(q)]$ to be a possible successor of q.

More formally, we associate with F_α the GMW relation R_α on $\{0,1\}^h$ defined by:

$$q R_\alpha q \text{ iff } q = F_\alpha(q)$$

and for $p \neq q$,

$$q R_\alpha p \text{ iff } p \in [q, F_\alpha(q)].$$

The relation R_α is conveniently represented by a directed graph, where nodes correspond to key states and an arrow from node q to node p indicates that $q R_\alpha p$.

In the graph of R_α a cycle of length one corresponds to a stable state, and a cycle of length > 1 represents an oscillation. For any $q \in \{0,1\}^h$ let $R_\alpha(q)$ be that portion of the graph of R_α which contains only nodes reachable from q. Let

$$cycl(R_\alpha(q)) = \left\{ p \in \{0,1\}^h \mid q R_\alpha^* p \text{ and } p R_\alpha^+ p \right\}$$

be the set of all states of $R_\alpha(q)$ that appear in at least one cycle. (as usual R_α^+ is the transitive closure of R_α and R_α^* is the reflexive and transitive closure of R_α.)

Among all the oscillations (cycles of length > 1) we wish to distinguish certain oscillations which we call transient. A cycle in R_α is *transient* iff there exists i, $1 \le i \le h$, such that for each key state in the cycle $q_i \neq \tilde{q}_i$, and q_i has the same value in each key state. Let

$$trans(R_\alpha(q)) = \left\{ p \in cycl(R_\alpha(q)) \mid \text{every cycle in which } p \text{ appears is transient} \right\}.$$

Finally, we define

$$out(R_\alpha(q)) = cycl(R_\alpha(q)) - trans(R_\alpha(q))$$

to be the set of "non-transient" cyclic states.

A clean input-key state (α, q) of a well-formed CMOS T-graph C is *deterministic* iff $out(R_\alpha(q))$ has exactly one element p. This means that the network ends up in a unique stable input-key state (α, p) when started in (α, q). In general, a well designed CMOS T-graph has only such deterministic transitions when any "admissible" input sequence is applied.

An input state α of a well-formed T-graph C is *forcing* iff for any two clean input-key states $w_1 = (\alpha, q)$ and $w_2 = (\alpha, p)$ the excitation states are equal, i.e. $e_{w_1} = e_{w_2}$. This implies that the network always ends up in a unique stable state, whenever α is applied. Evidently, if α is forcing then (α, q) is deterministic for every clean q. If every input state of C is forcing then C is called *key-combinational*, i.e. the state of each key node is a boolean function of the input variables. Note that, if C is combinational, its outputs are also uniquely determined by the input state. However, an output may have the value 3, i.e. be floating.

Example 5.1 Consider the NOR latch of Figure 3.2. For input state α with $S = 1$, $R = 1$ the R_{11} graph is shown in Figure 5.1(a), where the key state is $(\overline{Q}, Q)$. Note that this input is forcing, and every input-key state with $S = R = 1$ is deterministic. The R_{00} graph is shown in Figure 5.1(b). Note that $out(R_{00}(00)) = \{0,1\}^2$. Thus $R_{00}(00)$ is not deterministic. There are three cycles in R_{00}: the length-one cycles (01) and (10) are stable states and the cycle (00,11) is an oscillation. This cycle is match-dependent [3] and therefore unlikely to persist indefinitely.

In view of the nondeterministic behavior of the NOR latch caused by an input change from 11 to 00, the input 11 is generally considered inadmissible.

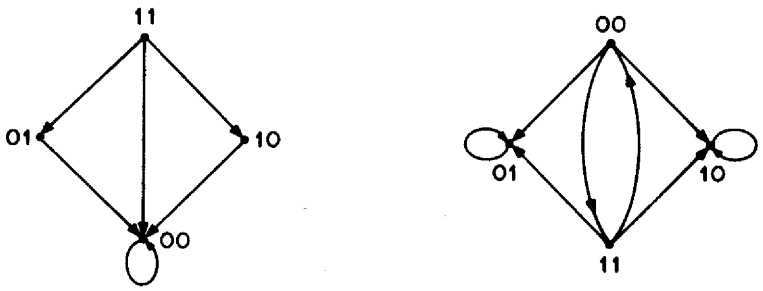


Figure 5.1 Examples of R_α graphs: (a) R_{11}; (b) R_{00}.

6. TERNARY SIMULATION

In this section we give another example of the binary analysis introduced in Section 5 for the purpose of comparing it to an alternate approach described below. First we introduce a simplified notation for CMOS gates as shown in Figure 6.1.

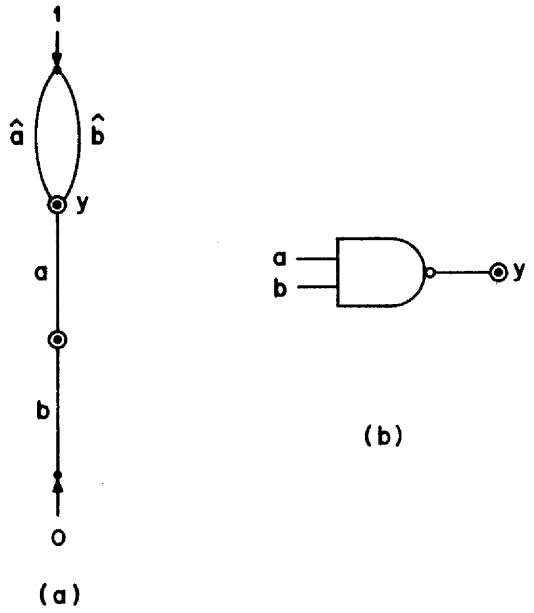


Figure 6.1 (a) CMOS NAND gate; (b) abbreviated symbol for (a).

Consider the network of Figure 6.2 with inputs a and b and key nodes y_1, y_2, and y_3. This network consists of 3 NAND gates each constructed using the complementary design of Figure 3.1. In such cases the excitation state is very easy to find. Here we have the following excitation equations:

$$\tilde{y}_1 = N(a,y_2) = (ay_2)',$$

$$\tilde{y}_2 = N(y_1,y_3) = (y_1y_3)',$$

$$\tilde{y}_3 = N(y_2,b) = (y_2b)',$$

where $N(x,y)$ represents the NAND function of x and y, $N(x,y) = (xy)'$. We will analyze two of the transitions for this network.

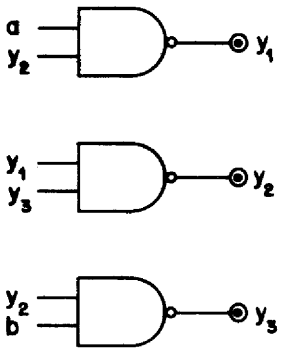


Figure 6.2 Example to be analyzed.

Note that the input-key state $((a,b),(y_1,y_2,y_3)) = (11,010)$ is stable. Figure 6.3 shows the graph of the relation $R_{00}(010)$. Evidently $out(R_{00}(010))$ is the singleton set $\{101\}$. Hence the input-key state $(11,010)$ is deterministic, and the network ends up in the stable key state 101 when the input is fixed at 00.

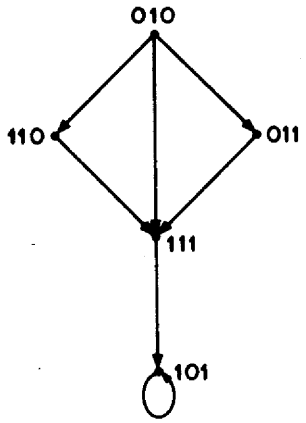


Figure 6.3 Graph of $R_{00}(010)$.

In our second example, the network starts in the stable input-key state (10,011), and then the input changes to 01. The graph of $R_{01}(011)$ is shown in Figure 6.4. Here $out(R_{01}(011)) = \{100,101,110,111\}$; there are two stable key states and one (match-dependent) cycle of length 2. Clearly the input-key state (01,011) is nondeterministic.

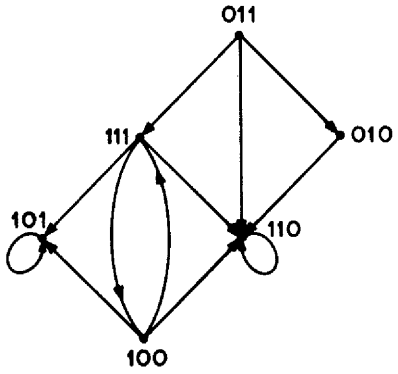


Figure 6.4 Graph of $R_{01}(011)$.

In the binary analysis illustrated above each graph $R_\alpha(q)$ may have as many as 2^h nodes. We now describe an alternate analysis method, namely ternary simulation [3], where the analysis algorithm is linear in h. We will describe this approach only informally here; a formal discussion is found in [3].

We require the concept of *ternary extension* of a boolean function. For example, the ternary extension $\mathbf{N(x,y)}$ of $N(x,y)$ is defined in Figure 6.5. For any boolean function $F(x,y)$, if $\mathbf{x,y} \in \{0,1\}$ then the ternary extension is $\mathbf{F(x,y)} = F(\mathbf{x,y})$. If $\mathbf{y} \in \{0,1\}$ and $F(0,\mathbf{y}) \neq F(1,\mathbf{y})$ then $\mathbf{F}(\frac{1}{2},\mathbf{y}) = \frac{1}{2}$. Otherwise, $\mathbf{F}(\frac{1}{2},\mathbf{y}) = F(0,\mathbf{y}) = F(1,\mathbf{y})$. The situation is similar if $\mathbf{x} \in \{0,1\}$ and $\mathbf{y} = \frac{1}{2}$. Finally, $\mathbf{F}(\frac{1}{2},\frac{1}{2}) = \frac{1}{2}$ iff $\{F(0,0),F(0,1),F(1,0),F(1,1)\} = \{0,1\}$; otherwise $\mathbf{F}(\frac{1}{2},\frac{1}{2}) = F(0,0) = \cdots = F(1,1)$. The table of Figure 6.5 illustrates this for the NAND function.

Returning now to the excitation equations of the network of Figure 6.2, we define the ternary function $\mathbf{H}$ by:

$$\mathbf{H((a,b),(y_1,y_2,y_3))} = \mathbf{(N(a,y_2),N(y_1,y_3),N(y_2,b))}.$$

This function can be viewed as the "ternary excitation function" for the network of Figure 6.2. We can now describe the ternary simulation algorithms.

We consider the network as starting in a stable input-key state (α,q), and we are interested in its behavior when the input state α changes to β and then remains fixed. Let $\mu(\alpha,\beta)$ be the component-by-component average of α and β, i.e.

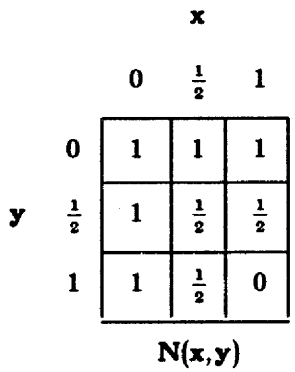


Figure 6.5 Ternary extension of the NAND function N.

$$\mu(0,0) = 0,$$
$$\mu(1,1) = 1,$$
$$\mu(0,1) = \tfrac{1}{2}.$$

The method consists of 2 algorithms shown below. One easily verifies that each algorithm terminates in $\leq h$ steps. The output $\mathbf{r}$ of Algorithm A becomes the $\mathbf{r}$ input of Algorithm B.

Algorithm A {inputs α, β, q; output $\mathbf{r}$}

$\mathbf{p} \leftarrow \mu(\alpha, \beta)$
$\mathbf{r} \leftarrow q$
while $\mathbf{H(p,r)} \neq \mathbf{r}$ **do**
$\quad \mathbf{r} \leftarrow \mathbf{H(p,r)}$
stop

Algorithm B {inputs $\mathbf{r}, \beta$; output $\mathbf{t}$}

$\mathbf{t} \leftarrow \mathbf{r}$
while $\mathbf{H}(\beta,\mathbf{t}) \neq \mathbf{t}$ **do**
$\quad \mathbf{t} \leftarrow \mathbf{H}(\beta,\mathbf{t})$
stop

We now apply the ternary analysis to the transition previously analyzed in the binary model, shown in Figure 6.3.

Algorithm A

(1) $\mathbf{p} \leftarrow \mu(11,00) = \frac{1}{2}\frac{1}{2}$

(2) $\mathbf{r} \leftarrow 010$

(3) $\mathbf{H}(\frac{1}{2}\frac{1}{2},010) = \frac{1}{2}1\frac{1}{2} \neq 010$

(4) $\mathbf{r} \leftarrow \frac{1}{2}\,1\,\frac{1}{2}$

(5) $\mathbf{H}(\frac{1}{2}\frac{1}{2},\frac{1}{2}1\frac{1}{2}) = \frac{1}{2}\frac{1}{2}\frac{1}{2} \neq \frac{1}{2}1\frac{1}{2}$

(6) $\mathbf{r} \leftarrow \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(7) $\mathbf{H}(\frac{1}{2}\frac{1}{2},\frac{1}{2}\frac{1}{2}\frac{1}{2}) = \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(8) stop {output is $\mathbf{r} = \frac{1}{2}\frac{1}{2}\frac{1}{2}$}

Algorithm B

(1) $\mathbf{t} \leftarrow \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(2) $\mathbf{H}(00,\frac{1}{2}\frac{1}{2}\frac{1}{2}) = 1\frac{1}{2}1 \neq \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(3) $\mathbf{t} \leftarrow 1\frac{1}{2}1$

(4) $\mathbf{H}(00,1\frac{1}{2}1) = 101 \neq 1\frac{1}{2}1$

(5) $\mathbf{t} \leftarrow 101$

(6) $\mathbf{H}(00,101) = 101$

(7) stop {output is $\mathbf{t} = 101$}

The significance of the results of the ternary algorithm is explained in the following theorem [3]:

Theorem

(a) *If the output* $\mathbf{t}$ *of algorithm* B *is in* $\{0,1\}^k$ *then the input-key state* (β,q) *is deterministic, and the network ends up in the stable key state* $\mathbf{t}$.

(b) *More generally, if a component* $\mathbf{t}_i$ *of* $\mathbf{t}$ *is in* $\{0,1\}$, *then the* i*th component in every key state in* $out(\beta,q)$ *has the value* $\mathbf{t}_i$.

Part (b) of the theorem is illustrated below for the transition previously analyzed in Figure 6.4.

Algorithm A

(1) $\mathbf{p} \leftarrow \mu(10,01) = \frac{1}{2}\frac{1}{2}$

(2) $\mathbf{r} \leftarrow 011$

(3) $\mathbf{H}(\frac{1}{2}\frac{1}{2},011) = \frac{1}{2}1\frac{1}{2} \neq 011$

(4) $\mathbf{r} \leftarrow \frac{1}{2}1\frac{1}{2}$

(5) $\mathbf{H}(\frac{1}{2}\frac{1}{2},\frac{1}{2}1\frac{1}{2}) = \frac{1}{2}\frac{1}{2}\frac{1}{2} \neq \frac{1}{2}1\frac{1}{2}$

(6) $\mathbf{r} \leftarrow \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(7) $\mathbf{H}(\frac{1}{2}\frac{1}{2},\frac{1}{2}\frac{1}{2}\frac{1}{2}) = \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(8) stop {output is $\mathbf{r} = \frac{1}{2}\frac{1}{2}\frac{1}{2}$}

Algorithm B

(1) $\mathbf{t} \leftarrow \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(2) $\mathbf{H}(01,\frac{1}{2}\frac{1}{2}\frac{1}{2}) = 1\frac{1}{2}\frac{1}{2} \neq \frac{1}{2}\frac{1}{2}\frac{1}{2}$

(3) $\mathbf{t} \leftarrow 1\frac{1}{2}\frac{1}{2}$

(4) $\mathbf{H}(01,1\frac{1}{2}\frac{1}{2}) = 1\frac{1}{2}\frac{1}{2}$

(5) stop {output is $\mathbf{t} = 1\frac{1}{2}\frac{1}{2}$}

Compare the result $\mathbf{t} = 1\frac{1}{2}\frac{1}{2}$ to $out(01,011) = \{100,101,110,111\}$ from Figure 6.4. Indeed the first component of the key-state is always 1 as predicted by $\mathbf{t} = 1\frac{1}{2}\frac{1}{2}$.

Our third example shows that the ternary model may be more pessimistic that the binary model. The network of Figure 6.6 is stable when $a = 0$, $b = 1$ and $y_1 = 1$, $y_2 = 0$, $y_3 = 1$. It is also stable when the input changes to $a = 1$, $b = 0$, and the binary model predicts that the key state will remain 101. One easily verifies, however, that the outputs of Algorithms A and B are $\frac{1}{2}\frac{1}{2}\frac{1}{2}$ and $1\frac{1}{2}\frac{1}{2}$, respectively. Thus the ternary model gives the correct value of y_1, but is more pessimistic than the binary model as far as y_2 and y_3 are concerned. This happens because the ternary model takes into account the possibility that the two inputs may not change simultaneously, and that the storage node y_1 may not store its previous state for a long enough time to overcome this hazard.

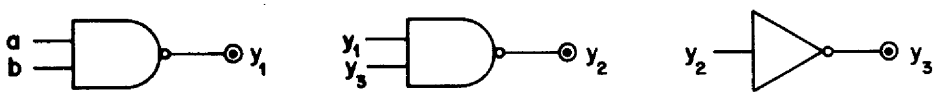


Figure 6.6

7. FAULT DETECTION IN COMBINATIONAL NETWORKS

Fault detection for MOS circuits has been widely discussed in the literature, and complex models have been proposed in order to simulate faulty networks. See, for example [6, 12]. In this section we show that our model is applicable to fault detection in combinational CMOS networks with binary output. We consider the following types of faults:

(1) Input stuck-at-0 or stuck-at-1,

(2) Key node stuck-at-0 or stuck-at-1,

(3) Edge stuck-at-open or stuck-at-closed.

By stuck-at-0 (stuck-at-1) we mean, as usual, a permanent short-circuit to VSS (VDD), whereas stuck-at-open (stuck-at-closed) represents an open (short-circuited) transistor.

The appropriate framework for studying the faults described above is the CMOS graph of Section 3. In this model any one of the above faults can be represented by replacing

(1) an input variable by 0 or 1,

(2) a key-node variable by 0 or 1,

(3) the label on a single edge by $0, 1, \hat{0}$, or $\hat{1}$, as appropriate.

Evidently, the result of such a replacement is another CMOS graph which can be analyzed by the methods of Section 3, as described below.

In Figure 7.1(a) we show the NOR gate of Figure 2.2 in which we have assumed that the edge labeled a is stuck-at-closed. For the input combination $a = 0$, $b = 0$, the excitation $e(z)$ of output node z is 2. Under these circumstances the output voltage of the network being modeled will take on some intermediate value between VSS and VDD, depending on the electrical properties of the circuit. Such a fault should be detected by analog measurements; we will not pursue this.

In Figure 7.1(b) the edge labeled $\hat{a}$ in Figure 2.2 is stuck-at-open. Here we have $e(z) = 3$ for the input combination $a = b = 0$, while the fault-free

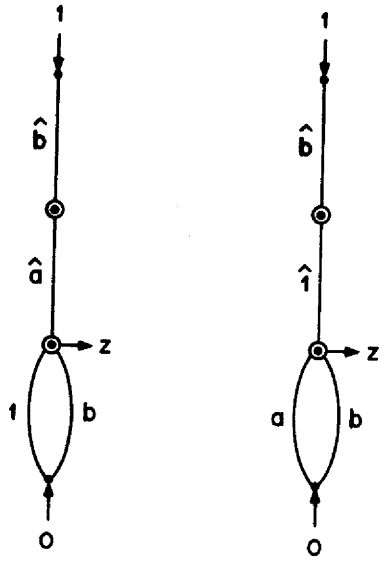


Figure 7.1 (a) Edge a of Figure 2.2 stuck-at-closed; (b) edge $\hat{a}$ stuck-at-open.

network has $e(z) = 1$. This fault can be detected by taking advantage of the temporary storage capability of node z. If the input sequence $(a = 0, b = 1)(a = 0, b = 0)$ is applied the faulty network will produce the value $z = 0$ for both input combinations, whereas the fault-free network would produce the output sequence 01. For this type of testing, the frequency of the test sequence must be sufficiently high to avoid significant deterioration of the signal at the floating node.

In general, any fault that, for some input combination, changes the output from 0 to 1 or vice-versa is detectable by that input combination. Any fault which would result in the output value of 3 in our analysis can be detected by a suitable sequence of length 2, as long as the output is not a constant.

8. CONCLUSIONS

We have presented a precise mathematical model for the analysis of digital CMOS networks. This model provides a promising framework for further research in the areas of (a) structured design methodology for digital CMOS systems, (b) the design of efficient simulators for asynchronous systems, and (c) the design of fault-detection sequences.

9. REFERENCES

[1] Bryant, R.E., "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. Computers*, Vol. C-33, No. 2, Feb. 1984, pp. 160-177.

[2] Brzozowski, J.A., and Yoeli, M., *Digital Networks*, Prentice-Hall, 1976.

[3] Brzozowski, J.A., and Yoeli, M., "On a Ternary Model of Gate Networks," *IEEE Trans. Computers*, Vol. C-28, No. 3, March 1979, pp. 178-184.

[4] Elmasry, M.I., "Digital MOS Integrated Circuits: A Tutorial," pp. 4-27 in *Digital MOS Integrated Circuits*, M.I. Elmasry, ed., IEEE Press, 1981.

[5] Hayes, J.P., "A Unified Switching Theory with Applications to VLSI Design," *Proc. IEEE*, Vol. 70, No. 10, Oct. 1982, pp. 1140-1151.

[6] Kawai, M., and Hayes, J.P., "An Experimental MOS Fault Simulation Program CSASIM," *Proc. 21st Design Automation Conference*, Albuquerque, N.M., June 1984, pp. 2-9.

[7] Mead, C., and Conway, L., *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[8] Muroga, S., *VLSI System Design*, John Wiley, 1982.

[9] Ramachandran, V., "An Improved Switch-level Simulator for MOS Circuits," *Proc. 20th Design Automation Conference*, Miami Beach, Fla., June 1983, pp. 293-299.

[10] Rem, M., "On the Design of Restoring Logic Circuitry," pp. 196-104 in *VLSI Architecture*, B. Randell and P.C. Treleaven, eds., Prentice Hall International, 1983.

[11] Rem, M., and Mead, C., "A Notation for Designing Restoring Logic Circuitry in CMOS," *Proc. 2nd Caltech Conf. on VLSI*, California Institute of Technology, Pasadena, Calif., 1981.

[12] Wadsack, R.L., "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *Bell Sys. Tech. J.*, Vol. 57, No. 5, May-June 1978, pp. 1449-1488.

[13] Wollesen, D.L., "CMOS LSI - The Computer Component Process of the 80's," *IEEE Computer*, Vol. 13, No. 2, Feb. 1980, pp. 59-67.