Computing Robust Tests for Stuck-open Faults from Stuck-at Test Sets.

by
Sreejit Chakravarty[1] G. Theophilopoulos
Dept. of Computer Science
State University of New York
Buffalo, NY 14260

Tech. Report No. 92-32

Abstract

An experimental system for computing robust tests for stuck-open faults in static CMOS circuits is presented. It constructs robust test-pairs from tests for stuck-at faults by identifying several classes of FETs. Robust tests for stuck-open faults in FETs belonging to these classes are constructed from any stuck-at test set by carefully constructing initialization vectors. Initialization vectors are constructed by examining the "parity" of the paths in the circuit. Robust tests for additional faults are identified using stuck-open fault simulation. Experimental results show that the system can compute robust tests for a "very large" percentage of the stuck-open faults in a "reasonable" time.

Index Terms: Robust Tests, Static CMOS Circuits, Stuck-open Faults, Test Generation.

1 Introduction

Tests for stuck-open faults in static CMOS circuits consist of a sequence of two input vectors $< T_1, T_2 >$; an **initial vector** (T_1) and a **final vector** (T_2)[14]. Such tests for stuck-open faults can be invalidated due to circuit delays[2, 10]. A test for a stuck-open fault f is said to be **robust** iff it is not invalidated by arbitrary circuit delays[2, 10]. We address the problem of computing robust tests for stuck-open faults in circuits consisting of NOT, NAND, NOR, AND and OR gates.

[1]Research supported by NSF grant number MIP-9102509.

The work presented here is based on the premise, justified by the performance of existing tools like SOCRATES[13], that stuck-at tests can be computed rather quickly. On the other hand, algorithms for computing robust tests for stuck-open faults are relatively more time consuming. Therefore, if methods for constructing robust tests for stuck-open faults from stuck-at test sets can be devised then these methods can be used to modify the well developed stuck-at test generation tools for computing robust tests for stuck-open faults.

Algorithms for computing robust tests for stuck-open faults has been discussed in [2, 10]. The algorithm in [10] computes a robust test for a given stuck-open fault using, in essence, a 27-valued logic. The algorithm in [2] is an improvement upon the algorithm in [10] in that it uses only 6 logic values. Both these algorithms use a very different strategy from the one advocated here in that they do not use any knowledge of the stuck-at test set. In [8] a method for computing "robust tests" for stuck-open faults from tests for stuck-at faults is discussed. However, the definition of robust tests used in [8] is very restrictive in that it assumes fixed delay values for the different gate types. Since it is difficult to estimate gate delays and gate delays vary from one manufacturing run to another we assume that we do not have any knowledge of the delays in the circuit. In our work we insist that the robust tests are valid under the arbitrary delay assumption unlike the tests computed by the method in [8]. A number of other algorithms for computing tests for stuck-open faults has been discussed in [5, 6, 9, 11]. However, none of these algorithms guarantee the tests to be robust.

The system that we present constructs robust test-pairs by first identifying several classes of FETs. The first of these classes, known as **trivial FETs**, consist of FETs for which every test-pair is robust. FETS that are not *trivial* are known as **nontrivial FETs**. Every gate has a network of *trivial* and another network of *nontrivial FETs*. In Section 2 we will present the rules from [5, 6, 8] for constructing robust tests for *trivial FETs* from any stuck-at test set.

SP-FETs, the second class of FETs, are the *nontrivial FETs* in **SP-Gates**. *SP-Gates* are defined in Section 2 wherein algorithms for identifying SP-FETs are also presented. We show that robust test-pairs for SP-FETs can always be constructed from **any** stuck-at test using "generic" initialization vectors. The initialization vectors are "generic" in that they depend only on the structure of the circuit and not on the particular stuck-at test set available.

PSP-FETs, the **third class** of FETs, are the *nontrivial FETs* in **PSP-Gates**. *PSP-gates* are defined in Section 4. Algorithms for identifying *PSP-Gates* are similar to identifying *SP-Gates*. We how robust test-pairs for PSP-FETs can also be constructed from **any** stuck-at test set using "custom" initialization vectors. It is "custom" in the sense that it depends not only on the structure of the circuit but also on the given stuck-at test set.

Using rules described in [5, 6, 8] test-pairs, not necessarily robust, can be constructed from any stuck-at test set. Many of these test-pairs may infact be robust. Robust tests for additional faults are identified using the stuck-open fault simulation algorithm proposed in [2]. Finally, the set of test-pairs computed is merged into a test sequence using the optimizing algorithm discussed in [3]. Performance of the system on the ISCAS85 benchmarks[1] are presented. The results show that robust tests for a "very large" percentage of the stuck-open faults can be computed, in a "reasonable time", using the approach proposed here.

2 Trivial FETs

Tests, not necessarily robust, for stuck-open faults can be computed from any stuck-at test set using the rules of Table 1[5, 6, 8]. In Table 1 we assume that NAND, NOR and NOT gates are implemented as shown in Figure 1. The phrase "x, y *s-a-b*" implies that the corresponding input can be a test either for x *s-a-b* or y *s-a-b*.

Tests so derived may not be robust. However, observe that there is only one path in the pullup network of a NOR gate, the pulldown network of a NAND gate and the pullup and pulldown network of a NOT gate. Therefore, any test for a stuck open fault in the pullup network of a NOR gate, pulldown network of a NAND gate and in a NOT gate are robust test pairs. We define the FETs in the pullup of a NOR gate, the pulldown of a NAND gate and in NOT gates as *trivial FETs*. Robust tests for stuck-open faults in *trivial FETs* can be easily computed from stuck-at tests using the rules given in Table 1.

3 SP-FETs

3.1 Definitions

SP-FETs are *nontrivial FETs* in **SP-gates** which are defined below. For any gate X in a circuit C, **cone(X)** is defined as the subcircuit of C s.t. a gate or a primary input Y in C is in *cone(X)* iff there exists a path from Y to X. *Cone(X)* includes X. For an example see Figure 2.

Primary inputs of a gate X in C, denoted by **PI(X)**, are the primary inputs of *cone(X)*. In Figure 2, $PI(3) = \{X1, X2, X3\}$. PI(X) is different from the set of inputs of X.

NAND, NOR and NOT are **inverting** gates and OR, AND are **non-inverting** gates. A path p from a primary input y to a gate X is of **even parity** (respectively, **odd parity**) iff p passes through an even (respectively, odd) number of inverting gates. While counting the number of gates in path p gate X is not considered to be in the path. For a gate X, a primary input $y \in PI(X)$ is **odd w.r.t X** (respectively, **even w.r.t X**) iff all paths from y to X are of odd (respectively, even) parity. Input $y \in PI(X)$ is **unate w.r.t. X** iff y is either odd or even w.r.t. X. If y is not unate w.r.t. X then y is **binate w.r.t. X**. In Figure 2 every path from X2 to 6 is of odd parity. Thus X2 is unate w.r.t 6. X4 is not unate w.r.t. 6.

A gate X is an **SP-gate** iff $\forall y \in PI(X)$, y is unate w.r.t. X. In Figure 2, 3 is an *SP-gate* whereas 6 is not.

3.2 Robust Tests for SP-FETs

For every *SP-gate* X, define two input vectors I_0^X, I_1^X as follows. Let u denote a don't care condition. For all primary input y of the circuit,

$I_1^X(y)$ = 0, if $y \in PI(X)$ and y is odd w.r.t. X;
= 1, if $y \in PI(X)$ and y is even w.r.t. X; and
= u, otherwise.

$I_0^X(y) = 1$, if $y \in PI(X)$ and y is odd w.r.t. X;
= 0, if $y \in PI(X)$ and y is even w.r.t. X; and
= u, otherwise.

In Figure 2, $I_1^5 =< X1 = 0, X2 = 0, X3 = 0, X4 = 1, X5 = u >$ and $I_0^5 =< X1 = 1, X2 = 1, X3 = 1, X4 = 0, X5 = u >$. I_1^5 sets the outputs of 1, 2, 3 and 5 to 1. Also, I_0^5 sets the outputs of 1, 2, 3 and 5 to 0. There is a pattern in all these which is captured in Lemma 3.1.

Lemma 3.1. Let X be an *SP-gate*.

(i) For every gate $G \in cone(X)$, the input vector I_0^X either: sets every input of G to 0; or sets every input of G to 1. I_0^X sets every input of X to 0.

(ii) For every gate $G \in cone(X)$, the input vector I_1^X either: sets every input of G to 0; or sets every input of G to 1. I_1^X sets every input of X to 1.

Proof. We use induction on N, the number of gates in *cone(X)*. For $N = 1, X$ is the only gate in *cone(X)* and let $X1, \dots Xm$ be the inputs of X. Therefore, $Xi \in PI(X), 1 \leq i \leq m$. Therefore, Xi is even w.r.t $X, 1 \leq i \leq m$. From this and the definition of I_0^X, I_1^X it follows that $I_0^X(Xi) = 0$, and $I_1^X(Xi) = 1, 1 \leq i \leq m$. This establishes both (i) and (ii) and forms the basis step. Assume that the lemma is true for all $N \leq k$ and let $N = k + 1$.

A decomposition of *cone(X)* is shown in Figure 3. Here $Xj \in PI(X), 1 \leq j \leq m$ and therefore is even w.r.t X. Therefore, $(\ \forall 1 \leq j \leq m), I_0^X(Xj) = 0, I_1^X(Xj) = 1$. Note that $PI(Gi) \subset PI(X), \forall 1 \leq i \leq t$ and $\bigcup_{i=1}^{t} PI(Gi) = PI(X)$.

Case 1. Gi is an *inverting gate*. Then, $(\forall y \in PI(Gi)\ )\ I_0^{Gi}(y) = I_1^X(y)$. From the induction hypothesis we know that I_0^{Gi} sets all inputs of Gi to 0. Therefore, I_1^X sets all inputs of Gi to 0. Therefore, I_1^X sets Yi, the output of Gi, to 1. Similarly, I_0^X sets Yi, the output of Gi, to 0.

Case 2. Gi is a *non-inverting gate*. Then, $(\ \forall y \in PI(Gi)\ )\ I_1^X(y) = I_1^{Gi}(y)$. From the induction hypothesis we know that I_1^{Gi} sets all inputs of Gi to 1. Therefore, I_1^X sets Yi, the output of Gi, to 1. Similarly, I_0^X sets Yi, the output of Gi, to 0.

Thus, we see that $\forall i, I_0^X($ respectively, $I_1^X)$ sets Yi, the inputs of X to 0 (respectively, 1).

Next, let G be any gate in *cone(X)*. If G equals X then, as shown above, I_0^X, I_1^X either: sets all inputs of X i.e. G to 1; or sets all inputs of X i.e. G to 0. If G is not the same as X note that $\forall y \in PI(G)$ either: (a) $I_0^X(y) = I_1^G(y), I_1^X(y) = I_0^G(y)$; or (b) $I_0^X(y) = I_0^G(y), I_1^X(y) = I_1^G(y)$.

Case a. From the induction hypothesis I_1^G (respectively, I_0^G) sets all inputs of G to 1 (respectively, 0). Therefore, $I_0^X(I_1^X)$ sets all inputs of G to 1 (respectively, 0).

Case b. By an argument similar to *Case a* it can be shown that $I_0^Y(I_1^Y)$ sets all inputs of G to 0 (respectively, 1).

Thus, for every gate G in cone(X) I_0^X, I_1^X either: sets every input of G to 1; or sets every input of G to 0. □

If there is a static hazard at the input of a gate, on application of $(T1, T2)$, then $\exists$ a gate G in *cone(X)* s.t. $(T1, T2)$ results in a falling transition at an input of G, $(T1, T2)$ results in a rising transition at a different input of G and the other inputs of G do not have disabling values[2]. Therefore if on application of $(T1, T2)$, for all gates $G \in$ *cone(X)* either there are only rising transitions at the inputs of G or there are only falling transitions at the inputs of G then there are no static hazards at the inputs of X, on application of $(T1, T2)$. The proof of the next theorem uses an idea similar to this.

Theorem 3.2. Let X be any *SP-gate* and $T2$ any input vector. Then, on application of $< I_0^X, T2 >$ and $< I_1^X, T2 >$, every line in *cone(X)* is static hazard free.

Proof. We show that $< I_0^X, T2 >$ has this property. The proof that $< I_1^X, T2 >$ also has this property is very similar and therefore omitted. The proof is by induction on N the number of gates in *cone(X)*.

For $N = 1$ X is the only gate in *cone(X)* and $\forall y \in PI(X)$, y is even w.r.t X. Let $X1, \ldots Xm$ be the inputs of X. From the definition of I_0^X, $I_0^X(X_i) = 0, 1 \leq i \leq m$. We are assuming that there cannot be any static hazard at the primary inputs of the circuit. Therefore, for every input of X the only transition resulting from the application of $< I_0^X, T2 >$ is a rising transition. This establishes the basis step. Assume that the lemma is true for all $N \leq k$ and let $N = k + 1$.

A decomposition of *cone(X)* is shown in Figure 3. Let $g_i^j, 1 \leq j \leq r$ be the inputs of Gi.

(a) From the induction hypothesis we know that there cannot be any static hazard at $g_i^j, 1 \leq j \leq r$.

(b) From Lemma 3.1 we know that on application of I_0^X either: for all $1 \leq j \leq r$, g_i^j

is set to 0; or for all $1 \leq j \leq r$, g_i^j is set to 1.

From (a) and (b) we know that if there are any transitions at the inputs of Gi then either: all transitions are rising transitions; or all transitions are falling transitions. Therefore, there cannot be any static hazard at the output of Gi i.e. the input Yi of X, $1 \leq i \leq t$. □

Theorem 3.2 is used to compute robust tests for SP-FETs as follows. Table 2 states conditions to be satisfied by robust tests for the *nontrivial FETs* in NAND, NOR gates. For each FET the values to be assigned by $T1, T2$ to the inputs of the gate are shown. The term "no static-x,y" is to be read as "$(T1, T2)$ does not result in a static x hazard at input y of the gate". $T2$ has to be a test for a stuck-at fault which is shown in the last column.

Table 3, which states rules for constructing robust tests for SP-FETs, is deduced from Lemma 3.1, Theorem 3.2 and Table 2. Tests $(T1, T2)$ defined in Table 3 cannot result in any static hazard at the inputs of X (Theorem 3.2) and $T1$ as defined in Table 3 sets a, b to the values stated in Table 2 (Lemma 3.2). Table 3 is used to compute robust tests for SP-FETs using any given stuck-at test set.

Note thet I_0^X, I_1^X the two initialization vectors depend only on the structure of the SP-Gates and the topology of the circuit. It is independent of the stuck-at test set being used.

From [4] we have the following: " Consider a circuit in which all the paths starting from the primary inputs and meeting at the input of any gate have the same inversion parity. A gate level (CMOS) circuit satisfying this property will be referred to as a *restricted gate (CMOS) circuit.*" Every gate in such circuits is an *SP-gate*. Therefore, the method for computing robust tests from a "universal test set"[4] can also be used for computing SP-FETs. This in turn requires the computation of the set of *maximal false vectors* and *minimal true vectors*. Our method uses the stuck-at test set and we prefer it because good tools exist for computing the stuck-at test sets whereas the same is not true for the *minimal false vectors* and *maximal true vectors*.

3.3 Identifying SP-FETs

Algorithm SP-Gates

(* Let X be the given gate with inputs $y_1, \ldots, y_k$. Given a gate X, Algorithm SP-Gates determines if X is an *SP-gate* or not. *)

Step 1. Label lines $y_1, \ldots, y_k$ with E.

Step 2. Traverse the rest of *cone(X)*, breadth-first, starting from $y_1, \ldots, y_k$ and traversing towards the primary inputs. While traversing *cone(X)* label each line in *cone(X)* with one of *E, O, B* using the following rules.

Rule 1. Let the output of a gate G be labeled E (resp. O, B). If G is inverting then label all its inputs with O(resp. E, B). If G is non-inverting then label all its inputs with E (resp. O, A).

Rule 2. If all branches of a fanout point F are labeled by E (resp. O) then label F by E (resp. O). Else, label F by B.

Step 3. X is an *SP-gate* iff for all $y \in PI(X)$, y is not labeled by B. If y is labeled by $E(O)$ then y is even (odd) w.r.t. X.

The following lemma establishes the correctness of the above algorithm.

Lemma 3.3. For any gate X a primary input $y \in PI(X)$ is labeled by: (a) E (resp. O), iff all paths from y to X is of even (odd) parity; and (b) B, iff $\exists$ two distinct paths of different parity from y to X.

Proof. The proof is by induction on N the number of gates in *cone(X)*. If $N = 1$ then all paths from $y \in PI(X)$ to X is of even parity. Therefore, all $y \in PI(X)$ must be labeled by E. The fact that Step 1 assigns E to all the inputs of X implies that the lemma is true for $N = 1$. Let the lemma be true for all $N \leq k$ and assume that $N = k + 1$.

Refer to Figure 4 where a decomposition of *cone(X)* is shown. The size of $C_1 \leq k$. From the induction hypothesis we know (a), (b) stated below.

(a) An input q_i of C_1 is assigned E (O) iff all paths from q_i to X is of even (odd) parity.

(b) An input q_i of C_1 is assigned B iff there exists a path of even parity and another path of odd parity from q_i to X.

From (a) and Rule 1 of the algorithm we know (c), (d) stated below.

(c) An input of G_i is labeled $E(O)$ iff all paths from the inputs of G_i to X is of even(odd) parity.

(d) An input of G_i is labeled by B iff there exists two paths from the input of G_i to X s.t. one is of odd parity and the other is of even parity.

Let $y \in PI(X)$ s.t. all paths from y to X is of even (odd) parity. If y is not a fanout point then (c) above implies that y is labeled by $E(O)$. Else, let $b_1, \ldots, b_m$ be the fanout branches of y. Since every path from $b_j, 1 \leq j \leq m$, to X is of even (odd) parity (c) above implies that $b_1, \ldots, b_m$ are labeled by $E(O)$. Rule 2 implies that y is labeled by $E(O)$.

Let $y \in PI(X)$ s.t. there exists two paths $p1, p2$, from y to X, such that $p1(p2)$ is of even (odd) parity. We have three subcases.

Case 1. Path $p1(p2)$ traverses branch $b_1(b_2)$ s.t. all paths from $p1(p2)$ are of even(odd) parity. By (c) above $b_1(b_2)$ is assigned $E(O)$. Rule 2 ensures that y is labeled by B.

Case 2. Path $p1(p2)$ both traverse the same branch b_i. Let b_i be the input of G_i. Therefore there exist two paths from G_i to X s.t. these two paths are of different parities. From (b) above we know that the output of Gi must be labeled by B. Rule 1 ensures that b_i is labeled by B. Rule 2 ensures that y is labeled by B.

Case 3. Let $b_i(b_j)$ be the input of $G_i(G_j)$. The path $p1(p2)$ traverses $b_i(b_j)$ such that there exists two paths from $G_i(G_j)$ to X of different parities. (d) implies that b_i, b_j are labeled by B. Rule 3 ensures that y is labeled by B. □

Lemma 3.4. If a gate X is an *SP-gate* then all gates in *cone(X)* are also *SP-gates.*

Proof. The proof is by contradiction. Let X be an *SP-gate* and, if possible, $G \in$ *cone(X)* be such that G is not an *SP-gate*. Since G is not an *SP-gate* $\exists\ y \in PI(G)$ s.t. $\exists$ two paths $p1, p2$ of different parity from y to G. Let p be any path ‘ from G to X. Then, $p3 =< p1, p >$ is a path from y to X that passes through G. Similarly, define $p4 =< p2, p >$. Clearly, $p3, p4$ are two paths from y to X that are of different parity. Therefore, X is not an *SP-gate*, which is a contradiction. □

A corollary to Lemma 3.4 is that if X is not an SP-Gate then every gate reachable from X is also not an SP-Gate. In the system that we present the SP-Gates are identified by visiting the circuit breadth-first starting from the inputs. For every gate X that is visited, if X has not already been marked as not an SP-Gate, Algorithm SP-Gates is used to determine if X is an SP-Gate. If X is not an SP-gate we mark all gates reachable from X as a non SP-Gate.

4 PSP-FETs

4.1 Definitions

PS-FETs are *nontrivial FETs* in **Pseudo-SP-gates**, defined below. For any gate X in the circuit define two input vectors J_0^X, J_1^X as follows. The symbol u denotes a don't care condition. For all primary input $y \in PI(X)$

$J_0^X(y) = 1$, if $y \in PI(X)$, y is odd w.r.t. X;
$= 0$, if $y \in PI(X)$, y is even w.r.t. X;and
$= u$, otherwise.

$J_1^X(y) = 0$, if $y \in PI(X)$, y is odd w.r.t. X;
$= 1$, if $y \in PI(X)$, y is even w.r.t. X;and
$= u$, otherwise.

A gate X is a **Pseudo-SP-gate** iff X either: (i) X is an AND or a NAND gate and J_1^X sets all inputs of X to 1; or (ii) X is an OR or a NOR gate and J_0^X sets all inputs of X to 0. In Figure 2, $J_1^6 =< X1 = 0, X2 = 0, X3 = 0, X4 = u, X5 = 0 >$ and gate 6 is a *Pseudo-SP-gate*. On the other hand if gate 6 was an OR or a NOR gate it would not be a *Pseudo-SP-gate*.

For any gate X and any input vector T define $K_0^{X,T}, K_1^{X,T}$ as follows. For all input variable y,

$K_1^{X,T}(y) = J_1^X(y)$, if $J_1^X(y) \in \{0,1\}$; and
$= T(y)$, if $J_1^X(y)$ equals u.

$K_0^{X,T}(y) = J_0^X(y)$, if $J_0^X(y) \in \{0,1\}$; and
$= T(y)$, if $J_0^X(y)$ equals u.

For gate 6 of Figure 2, J_1^6 is as defined above and let $T =< X1 = 1, X2 = 1, X3 = 1, X4 = 1, X5 = 1 >$. Therefore, $K_1^{6,T} =< X1 = 0, X2 = 0, X3 = 0, X4 = 1, X5 = 0 >$.

Let X, G be two gates s.t. $G \in cone(X)$. Then **gate G is even w.r.t. gate X** if and only if: $\exists y \in PI(G)$ s.t. y is unate w.r.t. X; if $y \in PI(G)$ is even w.r.t. X then y is even w.r.t. G; and if $y \in PI(G)$ is odd w.r.t. X then y is odd w.r.t. G. **Gate G is odd w.r.t. gate X** if and only if: $\exists y \in PI(G)$ s.t. y is unate w.r.t. X; if $y \in PI(G)$ is even w.r.t. X then y is odd w.r.t. G; and if $y \in PI(G)$ is odd w.r.t.

X then y is even w.r.t. G. In Figure 2, gate 3 is even w.r.t. gate 6 whereas gate 4 is odd w.r.t. gate 6. Lemma 4.1 ensures that if $\exists y \in PI(G)$ s.t. y is unate w.r.t. X then G must be either odd or even w.r.t. X.

4.2 Robust Tests for PSP-FETs

The transition at every line of the circuit on application of $< K_1^{6,T}, T >$ is shown in Figure 2. Observe that there are no static hazards at the input of gates in *cone(6)*. Theorem 4.2 generalizes this example.

Lemma 4.1. Let X be any gate and $G \in cone(X)$.
(a) If $y \in PI(G)$ s.t. y is unate w.r.t X then y is unate w.r.t. G.
(b) If $\exists y \in PI(G)$ s.t. y is even (odd) w.r.t. X and y is even w.r.t G then $\forall z \in PI(G)$ s.t. z is even (odd) w.r.t. X, z is even w.r.t. G.
(c) If $\exists y \in PI(G)$ s.t. y is even (odd) w.r.t. X and y is odd w.r.t. G then $\forall z \in PI(G)$ s.t. z is even (odd) w.r.t. X, z is odd w.r.t. G.

Proof. **(a)** The proof is by contradiction. Let $y \in PI(G)$ be even (odd) w.r.t. X. If possible, let y be binate w.r.t. G. Then, $\exists$ two paths $p1, p2$ from y to G s.t. $p1$ is even and $p2$ is odd. Let p be any path from G to X. Then, $(p1, p)$ and $(p2, p)$ are two paths of different parity from y to X. Thus, y is binate w.r.t. X. This contradicts the fact that y is unate w.r.t. X.

(b) We have two cases.

Case 1. y is even w.r.t. both X, G. Therefore, $\exists$ a path $p1$ from y to G s.t. $p1$ is even. Let p be any path from gate G to gate X. Therefore, $\exists$ a path $(p1, p)$ from y to X s.t. this path is even. We have two subcases.

Case 1.1. Let **G be an inverting gate**. Then p is odd (to compute the parity of p we do not include the parity of gates X, G). Let $z \in PI(G)$ be s.t. z is even w.r.t. X. Then, $\exists$ a path $p2 = (p3, p)$ of even parity from z to X s.t. $p2$ passes through G. Therefore, $p3$ is a path from z to G of even parity. From (a) above, z is unate w.r.t. X implies z is unate w.r.t. G. Therefore, z is even w.r.t. G.

Case 1.2. Let **G be a non-inverting gate**. Similar to *Case 1.1*.

Case 2. y is odd w.r.t. X and even w.r.t. G. Similar to *Case 1*.

(c) Omitted. Similar to part (b). $\square$

In Figure 5 $J_1^7 =< X1 = 0, X2 = u, X3 = u, X4 = 1 >$. Let $T =< X1 = 1, X2 = 0, X3 = 0, X4 = 0 >$. Then, $K_1^{7,T} =< X1 = 0, X2 = 0, X3 = 0, X4 = 1 >$. The transition at each line of the circuit, on application of $< K_1^{7,T}, T >$, is shown in Figure 5. No line in the circuit has a static hazard $< K_1^{7,T}, T >$ does not result in any transition at $X2, X3$ because they are binate w.r.t. gate 7. Gate 5 is odd w.r.t. gate 7 and all the transitions at the inputs of this gate are rising transitions. Gate 4 is even w.r.t. gate 7 and the only transition at the input of gate 4 is a falling transition. As shown below, this pattern of transitions occurring on application of $< K_1^{X,T}, T >$, $< K_0^{X,T}, T >$ for any X, T prevents any hazard at lines in *cone(X)*.

Theorem 4.2. For any gate X and any input vector T $< K_1^{X,T}, T >$ and $< K_0^{X,T}, T >$ cannot result in a static hazard at lines in *cone(X)*.

Proof. On application of $< K_1^{X,T}, T >$ or $< K_0^{X,T}, T >$: if $y \in PI(G)$ is binate w.r.t. X then $\nexists$ any transition at y; if $y \in PI(G)$ is odd w.r.t. X then $< K_1^{X,T}, T >$ ($< K_0^{X,T}, T >$) either cannot result in a transition at y or results in a falling (rising) transition at y; and if $y \in PI(G)$ is even w.r.t. X then $< K_1^{X,T}, T >$ ($< K_0^{X,T}, T >$) either cannot result e in a transition at y or results in a rising (falling) transition at y. In what follows we use this observation.

The proof is by contradiction. If possible, let there be a gate G in *cone(X)* such that $< K_1^{X,T}, T >$ result in a static hazard at one or more input(s) of G. Assuming that primary inputs of the circuit are hazard free $\exists$ gate $W \in cone(G)$ s.t.: all lines in *cone(W)*, except the output of W, are static hazard free; and there is a static hazard at the output of W. Therefore, there must be a rising transition at an input s of W and a falling transition at another input t of W[2]. Therefore, there must be a path $p1$ ($p2$) from a primary input $y1$ ($y2$) to s (t) s.t. there is either a rising or a falling transition at $y1$ ($y2$) and a rising or a falling transition at the output of every gate traversed by $p1$ ($p2$). Since there is a transition at $y1$ ($y2$) from the definition of $< K_1^{X,T}, T >$ it follows that $y1$ ($y2$) must be unate w.r.t. X. Therefore, from Lemma 4.1 we know that W must be either even w.r.t. X or odd w.r.t. X. We have two cases depending on whether X is even or odd w.r.t. X. In both these cases we arrive at a contradiction.

Case 1. W is even w.r.t. X. We have two subcases depending on whether $y1, y2$

are of the same or different parity. Both subcases lead to a contradiction.

Case 1.1. $y1, y2$ are of the same parity w.r.t. X. Therefore, $y1, y2$ are of the same parity w.r.t. W. Therefore, paths $p1, p2$ must be of the same parity. This and the fact that there is a rising transition at one of the two lines s, t and a falling transition at the other line implies that there is a rising transition at one of the two lines $y1, y2$ and a falling transition at the other line. However, by definition of $< K_1^{X,T}, T >$, since $y1, y2$ are of the same parity w.r.t. X the transitions at $y1, y2$, on application of $< K_1^{X,T}, T >$ must be the same. Therefore, there is a contradiction.

Case 1.2. $y1, y2$ are of different parity w.r.t. X. In this case also we arrive at a contradiction by an argument similar to *Case 1.1*.

Case 2. W is odd w.r.t. X. In this case also, by an argument similar to *Case 1* we arrive at a contradiction.

Therefore, $\nexists$ a gate W s.t. there is a static hazard at the output of W and all its inputs are hazard free. Hence, $\nexists$ any line in *cone(X)* s.t. $< K_1^{X,T}, T >$ results in a hazard at that line. $\Box$

Theorem 4.3. Let X be a *Pseudo-SP-gate*.
(a) If X is a NAND or an AND gate, X_i the input of X driving the pFET P_i, and $T2$ *any* test for X_i s-a-1. Then, $< K_1^{X,T2}, T2 >$ is a **robust** test for P_i stuck-open.
(b) If X is a NOR or an OR gate, X_i the input of X driving the nFET N_i, and $T2$ *any* test for X_i s-a-0. Then, $< K_0^{X,T2}, T2 >$ is a **robust** test for N_i stuck-open.

Proof. From the definition of *Pseudo-SP-gates* and Table 1 we know that the stated input pairs are tests for the specified faults. Theorem 4.2. states that the stated test pairs cannot result in a static hazard at any line in *cone(X)*. Therefore, from Table 2 we know that the stated tests are robust tests $\Box$

Note that PSP-Gates can be identified in a manner similar to SP-gates, once we know if each input variable is unate or binate w.r.t a given gate. Then, given a stuck-at test set robust tests can be computed using Theorem 4.3. Here, unlike SP-FETs, the initialization vectors depend not only on the structure of the circuit but also on the given test set.

5 System Organization and Experimental Results

The theorectical results of Sections 3,4, the fault simulation algorithm of [2] and the optimal test sequence algorithm of [3] are integrated into a system that computes robust tests for stuck-open faults. The fault simulation algorithm uses six-valued logic to determine if a given input-pair detects a given stuck-open fault. The algorithm in [3] computes a test sequence S from a set Q of pairs of input vectors s.t. every pair in Q is a subsequence of S and S is the smallest such subsequence. For example, if $Q = \{< T1, T2 >, < T2, T3 >, < T3, T1 >, < T4, T5 >\}$ then $S =< T1, T2, T3, T1, T4, T5 >$.

The system was developed using the environment provided by misII [2]. It computes the test set using the following steps.

Step 1. For each line l in the guiven circuit C l_0, a test for l *s-a-0*, and l_1, a test for l *s-a-1* is computed using the ATPG package of misII. ATPG uses the test generation algorithm of [7]. At this step all the *redundant stuck-at faults* are also identified.

Step 2. Robust test-pairs for *trivial FETs* in each gate L is constructed using Table 1 as follows. Let L be the output of gate L. If L is a NOR (NAND) gate we construct the pair $< l_1, l_0 >$ ($< l_0, l_1 >$). If L is a NOT gate the pairs $< l_0, l_1 >$, $< l_1, l_0 >$ are constructed.

Table 1 implies that for each FET F the input pair $< T1, T2 >$ is a test-pair for F, not necessarily robust, iff $T2$ is a test for a particular stuck-at fault l *s-a-b*, $b \in \{0, 1\}$. If l *s-a-b* is redundant then F is marked as a **redundant FET**. If l *s-a-b* is an *aborted* stuck-at fault then F is marked as an **aborted FET**.

In Table 1 for each *trivial FET F* an initialization vector $T1$ which is a test for a stuck-at fault l *s-a-b* has been listed. It is not necessary that the initialization vectors satisfy this condition. However, if l *s-a-b* is a *redundant* or an *aborted* stuck-at fault we mark F as an **aborted FET**.

Step 3. All SP-Gates are identified as discussed in Section 3. The *nontrivial FETs* of these gates constitutes the set of SP-FETs. For each SP-FET F the stuck-at

[2] Part of OCTOOLS distributed by University of California, Berkeley

test required by Table 3 has either been computed in Step 1 or the fault has been identified as *redundant* or *aborted*. If the stuck-at fault has been computed then a robust test is constructed using the initialization vector given in Table 3. Else, F is marked as a *redundant* or *aborted* FET.

Step 4. All PSP-Gates are identified. Its *nontrivial FETs* constitutes the set of PSP-FETs. As in Step 3 these FETs are marked either as *redundant* or *aborted* or robust tests are constructed as suggested by Theorem 4.3.

Step 5. COUNT = 0. Let A be the set of FETs for which no robust-test has yet been computed, and are neither *redundant* nor *aborted*. Let Q be the set of stuck-at faults, required by Table 1, to compute test-pairs for the stuck-open faults in A, and T a test set for all stuck-at faults in Q. From T, using Table 1, compute test-pairs for the FETs in Q and determine if it is robust[2]. FETs for which a robsut test is so computed is dropped from Q and A is updated.

Step 6. COUNT = COUNT + 1. If (COUNT > 10) then go to Step 7. Compute a new test set T for the faults in A. Go to Step 5.

Ste 7. Compute fault coverage. Merge the test-pairs, both robust and non-robust, using the algorithm in [3].

The system we implemented was used to compute robust test-pairs for the stuck-open faults in the ISCAS85 benchmarks[1]. The results are tabulated in Tables 4,5.

In Table 4, robust fault coverage after each of Step 2 (Trivial), Step 3 (SP-FETs) and Step 4 (PSP-FETs) are tabulated. The next two rows give the robust fault coverage respectively after 1 and 10 iterations of Steps 5, 6.

In Table 5, the total time for Steps 1,2 (ATPG), Step 3 (SP-FETs), Step 4 (PSP-FETs), 1 iteration of Steps 5,6 (1 SIM) and 10 iterations of Steps 5,6 (10 SIM) are tabulated. The last two rows give respectively the number of distinct test vectors and the length of the test sequence computed in Step 7.

The total robust coverage is, in most cases, in the neighbourhood of 85 % reaching to a high of 92.65%. This implies that for a large percentage of the stuck-open faults there does exist a robust test.

Using just the concepts developed in Section 3,4 along with the known results of Section 2 robust tests for upwards of 75%, and in many cases in the neighbourhood of 85%, of the stuck-open faults can be computed. Moreover, the additional time for these two steps, although larger than, is of the same order of magnitude as ATPG for stuck-at faults. This therefore validates the usefulness of the theoretical results of Section 3, 4.

The fault simulation time dominates the entire computation. One reason for this is the large number of faults to be simulated. "Fault collapsing", for stuck-open faults is an approach that can be used for reducing this time by reducing the number of faults to be considered. Since known techniques[12] for "fault collapsing" of stuck-open faults are not valid for robust tests they cannot be used in our system. However, if such techniques are developed they can be used to reduce fault simulation time and, in conjunction with our approach, to reduce the total test generation time.

Finally, we note that iterating through Steps 5, 6 more than once does not add considerably to robust fault coverage but adds considerably to the total test generation time. For larger circuits it may therefore be adequate to iterate through these steps only once. This is what we have done for the two largest circuits.

Conclusion

It was shown that robust tests for various classes of FETs can be computed from any given stuck-at test set by carefully constructing an initialization vector. These initialization vectors are computed by inspecting the parity of the paths in the circuit. These ideas were implemented, along with previously proposed algorithms for simulating stuck-open faults, to build a system for computing robust tests for stuck-open faults. The experimental system validates the usefulness of the theoretical results and also provides important information about the existence of robust tests in the benchmark circuits.

References

[1] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Circuits and their translator in FORTRAN," in *Proc. IEEE Int'l Symp. on Circuits and Systems*, pp. 663 - 698, June 1985.

[2] S. Chakravarty, "A Characterization of robust tests for stuck-open faults," *Journal of Electronic Testing: Theory and Application*, Vol. 1, No. 4, pp. 275 - 286, 1991. Kluwer Academic Press.

[3] S. Chakravarty and S. S. Ravi, "Computing Optimal Test Sequences for Stuck-Open Faults in CMOS Circuits," *IEEE Transactions on Comput.-Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 3, pp. 329-331, March 1990.

[4] G. Gupta and N. K. Jha, "A Universal Test Set for CMOS circuits," *IEEE Trans. on Comput.-Aided Design*, Vol. 7, No. 5, pp. 590 - 597, May 1988.

[5] S. K. Jain and V. D. Agrawal, "Test Generation for MOS circuits using D-Algorithm," in Proc. *20th ACM/ IEEE Design Automation Conference*, pp. 64 - 70, June 1983.

[6] S. K. Jain and V. D. Agrawal, "Modeling and Test Generation Algorithms for MOS Circuits," *IEEE Trans. on Comput.*, Vol. C-34, pp. 426 - 433, May 1985.

[7] T. Larabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on Comput.-Aided Design*, Vol. 11, No. 1, pp. 4 - 15, 1992.

[8] H. K. Lee, D. S. Ha, K. Kim, "Test pattern generation for stuck-open faults using stuck-at test sets in CMOS combinational circuits," *26th ACM/ IEEE Design Automation Conference*, pp. 345 - 350, 1989.

[9] S. M. Reddy, V. D. Agrawal, and S. K. Jain, "A Gate level model for CMOS combinational circuits with application to fault detection," in proc. *21st ACM/ IEEE Design Automation Conference*, pp. 504 - 509, June 1984.

[10] S. M. Reddy, M. K. Reddy and V. D. Agrawal, "Robust tests for stuck-open faults in CMOS combinational circuits," in proc. *14th Fault Tolerant Computing Symposium*, 1984, pp. 44 - 49.

[11] H. Shih and J. A. Abraham, "Transistor level test generation for Physical Failures in CMOS Circuits," in proc. *ACM/ IEEE Design Automation Conference*, Las Vegas, pp. 243 - 249, June 1986.

[12] H. Shih and J. A. Abraham, "Fault Collapsing Techniques for MOS VLSI circuits," *16th Fault Tolerant Computing Symposium*, pp. 370 - 375, 1986.

[13] M. H. Schultz, E. Trischler and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," *IEEE Trans. on Comput.-Aided Design*, Vol. 7, No. 1, pp. 126 - 137, 1988.

[14] R. L. Wadsack, "Fault Modeling and Logic simulation of CMOS and MOS integrated circuits," *The Bell Systems Technical Journal*, Vol. 57, No. 5, pp. 1449 - 1474, May-June 1978.

Table 1								
NOR			NAND			NOT		
FET	T1	T2	FET	T1	T2	FET	T1	T2
P1, P2	c s-a-1	c s-a-0	N1, N2	c s-a-0	c s-a-1	P1	a s-a-0	a s-a-1
N1	a,b s-a-1	a s-a-0	P1	a,b s-a-0	a s-a-1	N1	a s-a-1	a s-a-0
N2	a,b s-a-1	b s-a-0	P2	a,b s-a-0	b s-a-1			

Table 2						
	T1		T2		Robustness	T2 is a
NOR	a	b	a	b	criteria	test for
N1	0	0	1	0	no static-0, b	a, s-a-0
N2	0	0	0	1	no static-0, a	b, s-a-0
NAND						
P1	1	1	0	1	no static-1, b	a, s-a-1
P2	1	1	1	0	no static-1, a	b, s-a-1

Table 3		
NOR	T1	T2
N1	I_1^X	a, s-a-0
N2	I_1^X	b, s-a-0
NAND		
P1	I_0^X	a, s-a-1
P2	I_0^X	b, s-a-1

Table 4								
	C1355	C1908	C2670	C3540	C5315	C6288	C7552	C880
FETs								
Total	2080	2472	3916	5580	8834	10012	11742	1580
Aborted	6	1	25	73	11	2	44	0
Red.	8	7	100	176	56	4	134	0
Robust Coverage (%)								
Trivial	52.22	57.33	61.43	60.74	60.78	53.02	60.05	59.49
SP-FETs	56.08	63.65	76.68	69.82	74.15	59.34	66.31	71.96
PSP-FET	76.06	76.10	85.17	82.92	87.32	74.39	84.11	85.19
1 SIM	81.52	76.68	85.77	83.49	87.88	77.82	85.20	88.98
10 SIM	86.83	77.98	87.08	85.05	89.72	-	-	92.65
Total Coverage (Robust + Non Robust)								
	99.33	99.68	96.80	95.54	99.24	99.94	98.48	100.00

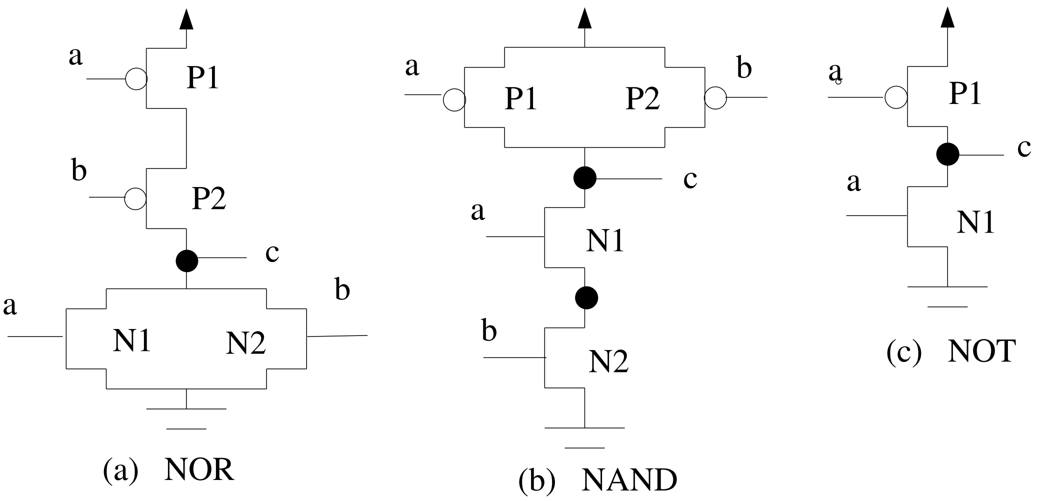

Figure 1: Structure of NAND, NOR Gates.

Table 5								
	C1355	C1908	C2670	C3540	C5315	C6288	C7552	C880
Time (CPU Secs Sparkstation 2)								
ATPG	0.058	0.099	0.662	0.514	0.514	0.787	6.932	0.033
SP-FETs	0.057	0.032	0.068	0.150	0.227	1.246	0.526	0.008
PSP-FETs	0.220	0.223	0.238	1.387	1.474	14.333	5.222	0.018
1 SIM						48.261	25.290	
10 SIM	3.807	7.036	11.396	31.156	57.947			0.405
Total	4.144	7.391	12.364	33.208	60.164	64.628	37.973	0.466
Test Length								
Stuck-at	508	573	699	1231	1436	604	1837	438
stuck-open	1536	3072	3072	6144	6144	3072	12288	1536

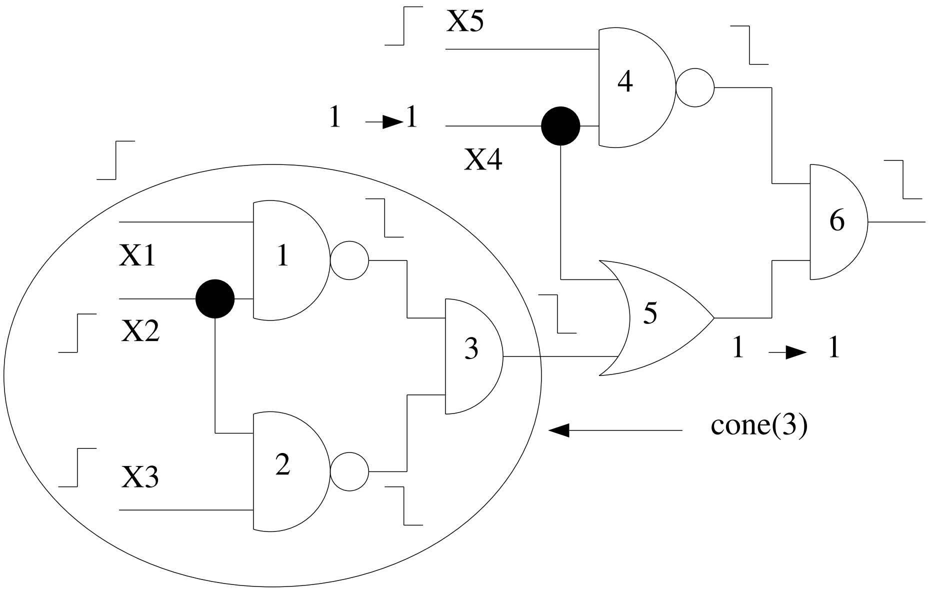

Figure 2: Example of SP-Gates

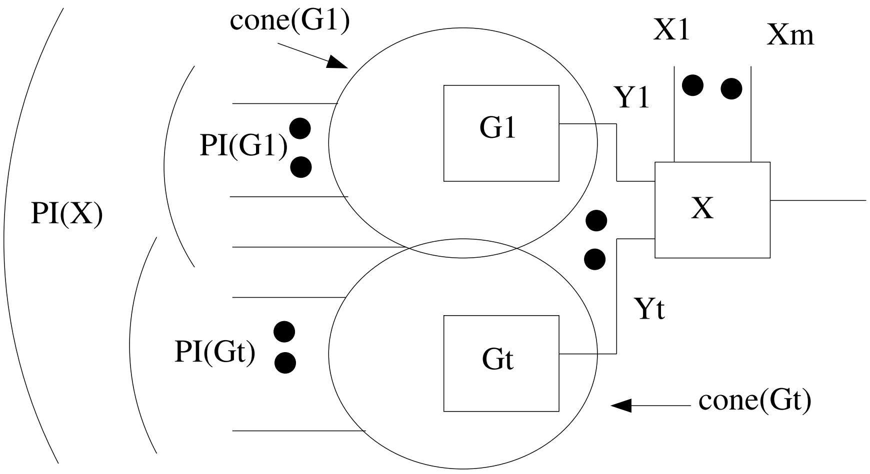

Figure 3: Decomposition of cone(X) for Lemma 3.1, Theorem 3.2.

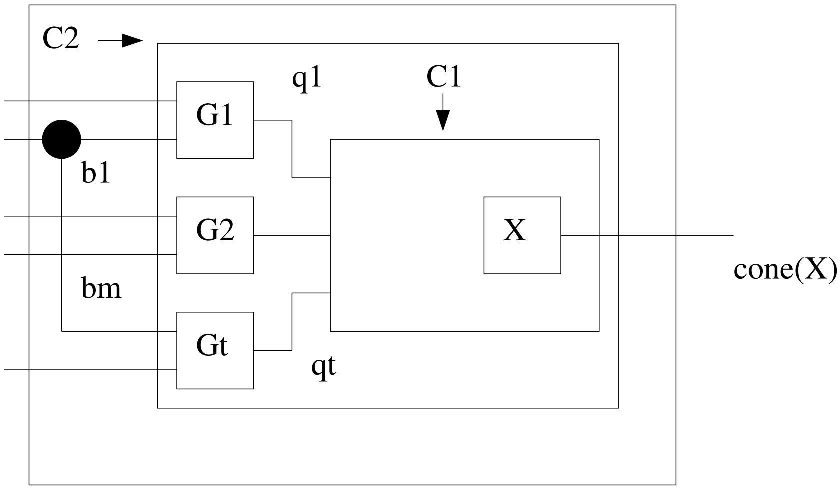

Figure 4: Decomposition of cone(X) for Lemma 3.4.

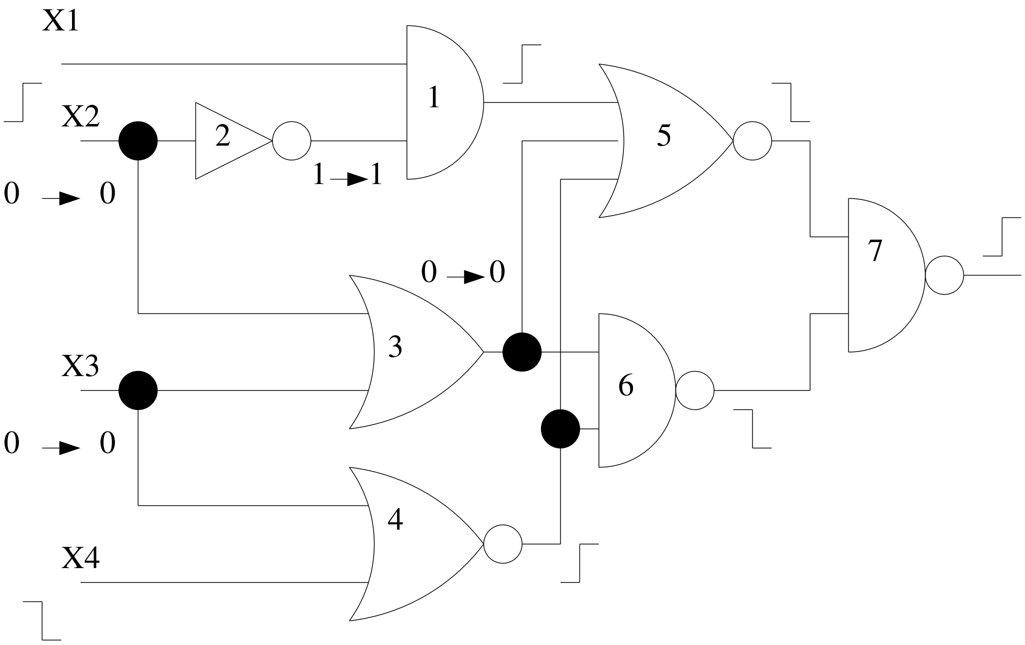

Figure 5: Example of PSP-Gates