On Testing Delay Faults In Macro-Based Combinational Circuits

Irith Pomeranz and Sudhakar M. Reddy [+]
Electrical and Computer Engineering Department
University of Iowa
Iowa City, IA 52242

Abstract

We consider the problem of testing for delay faults in macro-based circuits. Macro-based circuits are obtained as a result of technology mapping. Gate-level fault models cannot be used for such circuits, since the implementation of a macro may not have an accurate gate-level counterpart, or the macro implementation may not be known. Two delay fault models are proposed for macro-based circuits. The first model is analogous to the gate-level gross delay fault model. The second model is analogous to the gate-level path delay fault model. We provide fault simulation procedures, and present experimental results.

1. Introduction

A *macro* is a logic block implementing a specific function. A macro-based circuit is constructed by interconnecting macros to perform a given function. A macro-based circuit is obtained as a result of technology mapping [1-3]. During technology mapping, an abstract description of a circuit, e.g., a gate-level description, is translated into a hardware representation using a given set of macros in a specific technology. Logic faults in macro-based circuits were considered before [12].

In this work, we address the problem of testing macro-based circuits for delay defects. For gate-level circuits, the need to test delay defects led to the definition of several delay fault models. Gross delay faults [4] model delay defects that affect single lines in the circuit, causing the propagation delay through them to be "very large". The gate delay fault model [5] also addresses defects affecting single lines, however, no assumption is made on the delay size. The path delay fault model [6] addresses distributed, or accumulated delays due to propagation through several lines, each affected by a delay defect. Every one of these models provides a gate-level representation for the physical delay defects that can be present in a circuit, such that the gate-level model is significantly easier to handle than the physical defect. The differences between delay testing of gate-level circuits [4-6] and delay testing of macro-based circuits result from the different types of components that need to be considered (gates as opposed to macros). We show that due to these differences, new definitions of delay faults are required to model delay defects. Such definitions are proposed in this work.

In defining the various fault models for macro-based circuits, we attempt to capture the possible effects of delay defects on the behavior of a macro-based circuit. The models thus provide a way of defining a fault list for the purposes of test generation and fault simulation, such that the fault coverage with respect to this fault list would give an indication of the coverage of delay defects. At the same time, our goal is to make the models generally applicable and independent of the technology and the implementation of a macro. As a result, the definitions proposed and the procedures developed are applicable to any macro-based circuit. The only assumption we make is that a complete functional description is given for every macro, e.g., in terms of a complete truth table. If a specific implementation of a macro is known, for example, if gate-level implementations are given, then methods available for gate-level circuits should be applied. The approach taken here is applicable when gate level descriptions are not available and/or do not accurately describe the operation of the macros, as well as when tools which are independent of specific technology or macro libraries are needed. Like other fault models, the proposed models are not unique, and hence other models may be proposed.

We point out that in macro-based circuits implemented using Field-Programmable Gate Arrays (*FPGA*s), special hardware may be placed in the physical circuit, that helps in testing it. For example, in Xilinx *FPGA*s, programming of an *FPGA* is done by scanning in certain bit strings that determine the macro functions and their interconnections. Testing in this case can be done by scanning in and scanning out the appropriate bit strings. However, even in this environment, the proposed fault models may be required to model delay defects and generate the appropriate test sets. The fault models proposed are also useful in deriving tests when the special test hardware is not present, or when the test sets through scan are very large.

The paper is organized as follows. In Section 2 we introduce delay fault testing of macro-based circuits. In Section 3 we propose a gross delay fault model for macro-based circuits, analogous to the gross delay fault model in gate level circuits. This model associates delay faults with input transitions of the various macros in the circuit. We give a fault simulation procedure and present experimental results for this model. In Section 4 we consider a model called the *function-robust path delay fault* model, which is analogous to the path delay fault model in gate-level circuits. We describe a fault simulation procedure for function-robust testing of such faults and present experimental results. The experimental results demonstrate that the problems encountered in using the path delay fault model in gate-level circuits are even more significant in macro-based circuits, i.e., the number of path delay faults is sometimes very large [7], and the fault coverage is sometimes very low [8]. One of the solutions used for gate-level circuits is to consider a subset of path delay faults [9]. We take a similar approach here and propose two methods of restricting the number of faults to be considered. Section 5 concludes the paper and discusses future work.

2. Preliminaries

An example of a macro-based circuit is shown in Figure 1. Macro i is denoted by M_i. The numbers in parentheses in Figure

+ Research supported in part by Motorola Semiconductor Products Sector, Semiconductor Systems Design Technology, by NSF Grant No. MIP-9220549, and by NSF Grant No. MIP-9357581

1 will be explained later. Throughout this work, we assume that the only information available regarding the macros is their truth tables (alternatively, equivalent descriptions such as BDDs or Boolean equations can be used). Truth tables are given in Table 1 for the macros in Figure 1.

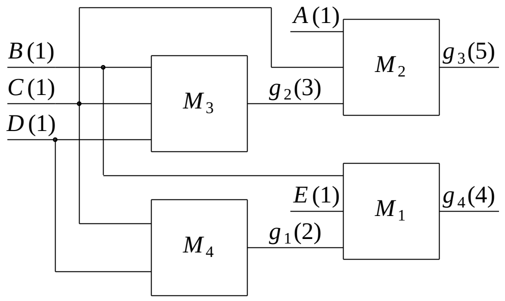

Figure 1: A macro-based circuit

Table 1: Truth tables for the macros of Figure 1

M_1

B	E	g_1	g_4
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

M_2

A	C	g_2	g_3
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

M_3

B	C	D	g_2
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

M_4

C	D	g_1
0	0	1
0	1	1
1	0	1
1	1	0

The need for special delay fault models in macro-based circuits results from the fact that a delay defect in a macro may delay a transition from an input of a macro to its output only for certain macro input combinations, and not for others. For example, consider a defect causing a very large delay (a gross delay fault). In a gate-level circuit, if such a defect delays the propagation of a rising transition from a gate input to its output, then the rising transition on the same gate input is delayed regardless of the other gate input values, as long as these input values allow the transition to propagate to the output of the gate. On the other hand, consider a macro implemented using table look-up (e.g., in certain *FPGA* circuits, the macro may contain a memory where every macro input combination causes the contents of a different memory location to appear on the macro output). Two-pattern input combinations $<u_1,v_1>$ and $<u_2,v_2>$, that create a rising transition on the same input of the macro may not use the same logic internal to the macro. Therefore, a defect affecting the logic traversed by an input transition under $<u_1,v_1>$ may not affect the logic traversed by an input transition under $<u_2,v_2>$. Thus, the delay incurred by the two transitions may be different. Consequently, a fault must be independently associated with each one of $<u_1,v_1>$ and $<u_2,v_2>$. An extreme consequence of this observation could be to associate a delay fault with every two-pattern input combination of each macro. However, this may result in a large number of faults, some of which do not have physical meaning. The goal of fault modeling is to define which two-pattern input combinations should be applied to the different macros in order to ascertain that they do not suffer from delay defects. Fault modeling is the subject of Sections 3 and 4.

Next, we demonstrate how logic simulation is carried out for a macro-based circuit.

Example: Consider the macro-based circuit of Figure 1. To compute the primary output values for input pattern $(ABCDE) = (01100)$, we perform the following computations.

For M_4, $(CD) = (10)$ implies $g_1 = 1$.

For M_3, $(BCD) = (110)$ implies $g_2 = 0$.

For M_2, $(ACg_2) = (010)$ implies $g_3 = 1$.

For M_1, $(BEg_1) = (101)$ implies $g_4 = 1$. □

In our experiments, we consider Berkeley PLA benchmark circuits, synthesized into multi-level circuits and then translated into macro-based circuits using the procedures of [10]. Two types of translations are done in [10]. The translation called *bl* minimizes the number of macros in the macro-based circuit, thus attempting to minimize its area. The translation called *fc* uses testability of non-delay faults as a primary optimization objective. A complete description of these procedures can be found in [10]. For all macro-based circuits considered in this work, the number of macro inputs is limited to five. Information regarding the circuits we use is given in Table 2. After circuit name, we give the number of primary inputs, the number of primary outputs and the number of macros in the macro-based circuit for both types of translations.

Table 2: Circuit parameters

			macros	
circ	inp	out	*bl*	*fc*
Z9sym	9	1	83	128
add6	12	7	23	45
adr4	8	5	10	13
alu1	12	8	8	8
alu2	10	8	31	33
alu3	10	8	32	36
co14	14	1	12	30
dk17	10	11	30	36
dk27	8	9	13	13
dk48	15	17	37	43
mish	94	34	40	40
radd	8	5	9	15
rckl	32	7	48	64
rd53	5	3	3	3
vg2	25	8	34	32
x1dn	27	6	28	32
x9dn	27	7	36	31
z4	7	4	7	10

3. Gross delay faults

For gate-level circuits, the gross delay fault model [4], also called the transition fault model, is used to model defects that cause very large delays compared to the system clock period. Such delay defects are assumed to increase the delay of all paths through the defect site beyond the system clock period. This fault model is the easiest to handle among all the delay fault models proposed for gate-level circuits. In addition, this model is useful for circuits where all physical paths have the same propagation delay [11]. In this section, we define a gross delay fault model for macro-based circuits, and present gross delay fault coverages for the circuits of Table 2.

3.1 The fault model

To define the gross delay fault model, we make the following assumptions. We assume that gross delay defects cause a large extra delay, such that any gross delay defect affecting the propagation of a transition t through macro M_i causes the circuit to exceed its designated operation time whenever t is propagated through M_i to a primary output. We also make the following assumption. Consider a transition on the inputs of a macro from u to v, where u and v differ in the value of more than one input. Since, in practice, input values do not change simultaneously, the transition between u and v occurs in a sequence of single input changes. Let it be $<u = w_0, w_1, \cdots, w_k = v>$, where w_i differs from w_{i+1} in the value of a single input, for $0 \le i \le k-1$. We assume that if a gross delay defect affects the transition from u to v, then there exists at least one transition $<w_j, w_{j+1}>$ which is affected by the defect, for some $0 \le j \le k-1$. The following example illustrates the assumptions and the fault model.

Example: Consider a macro with the truth table shown in Table 3. Consider the input combinations $u = (000)$ and $v = (011)$. If the macro input combination is changed from u to v, then the change occurs in one of the following sequences. (000) -> (001) -> (011), or (000) -> (010) -> (011). Suppose that each one of the input changes (000) -> (001), (001) -> (011), (000) -> (010), and (010) -> (011) has been verified to be fault free. Then the input change (000) -> (011) is also guaranteed to occur in a fault free manner, by the assumption that the transition will occur through one of the sequences above, and that any single faulty transition is enough to cause a detectable fault (it is a gross delay defect). Thus, the fault model does not have to include (000) -> (011), since we do not have to verify correct operation for this transition separately. For the macro of Table 3, the fault model includes only the following two-pattern input combinations, that differ in a single bit. The symbol <-> indicates that two gross delay faults are included, from the input combination on the right to the one on the left and vice versa. (000) <-> (001), (000) <-> (010), (000) <-> (100), (001) <-> (011), (001) <-> (101), (010) <-> (011), (010) <-> (110), (011) <-> (111), (100) <-> (101), (100) <-> (110), (101) <-> (111), and (110) <-> (111).

Note that we include transitions such as (000) -> (001) in the fault model, although the truth table indicates that the output value does not change during this input transition. One of the reasons for this is that logic hazards [15] may cause the output to change momentarily to 1, and a gross delay defect may cause the hazard value to "linger", causing faulty output values (a logic hazard may cause the output of a macro to change momentarily, even when the functional representation of the macro does not indicate that such a change is possible). □

Table 3: An example of gross delay faults

input	000	001	010	011	100	101	110	111
output	0	0	0	1	1	0	0	1

The previous example indicates that for every macro, the gross delay fault model should include every two-pattern input combination $<u,v>$ such that the Hamming distance between u and v, denoted $d_H(u,v)$, is 1 (the Hamming distance between two input combinations is equal to the number of bits in which they differ). In the following, we develop the fault model further.

The gross delay fault model as defined above has the following shortcoming. When a macro is embedded in a circuit, it may not be possible to obtain all of its input combinations. In such a case, the model as defined above may not be sufficient to ensure testing of all macro input transitions. Consider the following example.

Example: Consider the three-input macro of Table 3, and suppose that the input combination (100) cannot be obtained. The following transitions involving single input changes cannot be tested in this case.

(000) <-> (100), (110) <-> (100) and (101) <-> (100).

As a result, some of the transitions that involve input combinations at Hamming distance two may not be verified. Specifically, the following transitions that may go through (100) will not be verified.

(000) <-> (110), (000) <-> (101), (110) <-> (101).

All these transitions must be added to the gross delay fault model. However, we do not need to include the transitions (000) <-> (111), that may also go through (100). This is because once the transitions above are verified, the transitions (000) <-> (111) are also verified. For example, one of the sequences that can be followed to go from (000) to (111) is (000) -> (100) -> (110) -> (111). This sequence is fault free if the sequences (000) -> (110) and (110) -> (111) are fault free. Both are included in the transition fault model, therefore, (000) -> (111) does not have to be included. □

To find the two-pattern input combinations included in the gross delay fault model due to a macro M_i, we use the following procedure, illustrated by an example below.

Procedure 1: Finding the set of gross delay faults for macro M_i

```
(1)  Mark all the input combinations that cannot be obtained
     on the inputs of $M_i$ (a method is described in Section 3.2).
     Set $F_i = \phi$ ($F_i$ is the set of gross delay faults for $M_i$).
(2)  For every input combination $u$ that can be obtained on the
     inputs of $M_i$:
     (a)  Set $S = \{(u)\}$ ($S$ will contain sequences of macro
          input combinations at distance one).
     (b)  Select the first sequence in $S$. Let the sequence be
          $W = <w_0 w_1 \cdots w_{k-1}>$. For every input combination
          $w_k$ which is at distance $j$ from $w_{k-j}$, $1 \le j \le k$:
              If $w_k$ is marked (it cannot be obtained on the
              inputs of $M_i$), then add the sequence
              $<w_0 w_1 \cdots w_{k-1} w_k>$ to $S$.
              Else, add the two-pattern input combination
              $<w_0, w_k>$ to $F$.
          Remove $W$ from $S$.
     (c)  If $S \ne \phi$, go to Step 2(b).
```

Example: Consider the three-input macro shown in Table 3. Suppose that input combinations (100) and (110) cannot be obtained. Procedure 1, when applied to $u = (010)$, results in the following gross delay faults. Initially, $F = \phi$ and $S = \{(010)\}$. We check the input combinations at distance one from (010), namely, (011), (000) and (110). (011) and (000) can be obtained as input combinations of the macro. Therefore, we add to F the faults <(010),(000)> and <(010),(011)>. (110) cannot be obtained as an input combination of the macro, therefore, we add <(010),(110)> to S. (010) is now removed from S.

Next, we consider <(010),(110)>. The input combinations at Hamming distance one from (110) and at distance two from (010) are (111) and (100). (111) can be obtained as an input combination to the macro. Therefore, we add to F the fault <(010),(111)>. (100) cannot be obtained as an input combination to the macro, therefore, we add <(010),(110),(100)> to S. <(010),(110)> is removed from S.

Next, we consider <(010),(110),(100)>. We check the input combinations at Hamming distance one from (100), two from (110) and three from (010). The only such input combination is (101). (101) can be obtained as an input combination to the macro, therefore, we add to F the fault <(010),(101)>. After removing <(010),(110),(100)> from S, S remains empty. The set of gross delay faults starting with (010) is $F = \{$<(010),(000)>,

<(010),(011)>, <(010),(111)>, <(010),(101)>}.
If all pairs of input patterns are included in the fault model, then 56 faults are included in F, of which 30 involve only input combinations that can be obtained on the inputs of M_i. Using the proposed fault model, the size of the fault list is reduced to 22. □

3.2 The input combinations applicable to a macro

The gross delay fault model as defined above requires knowledge of which input combinations can be obtained for each macro. Checking whether an input combination u can be obtained on the inputs of macro M_i can be done in several ways.

Logic simulation of a large number of randomly determined primary input patterns can be used to obtain most of the input combinations that can be obtained on the inputs of every macro. For the remaining input combinations, that were not obtained during logic simulation, the line justification procedure of a test generation system can be used [12] to determine whether or not they can be obtained. Alternatively, it is possible to define the gross delay fault model with respect to the input combinations obtained during logic simulation of random patterns. It is expected that a very small number of patterns that can be obtained would not be obtained when simulating random patterns, therefore, the inaccuracy in defining the gross delay fault model based on random patterns would be very small.

3.3 Fault simulation and experimental results

Simulation of gross delay faults under a two-pattern primary input combination $<u,v>$ is similar to simulation of gate-level transition faults [13], and proceeds as follows. Logic simulation of u and v separately is first used to obtain the values throughout the circuit under u and v. Then, we consider every macro M_i. Let u bring the input combination a to M_i and let v bring the input combination b to M_i. If the two-pattern input combination $<a,b>$ of M_i belongs to the fault model, then we check whether a delayed output value of M_i under v causes a faulty primary output value. In other words, we check whether the output of M_i is *sensitized* to a primary output under v.

We incorporated the gross delay fault model into a fault simulation process and used it to simulate 100,000 random two-pattern input combinations. Results are given in Table 4. After circuit name, we consider two translations of the circuit into a macro-based circuit. For each translation, we first give the number of all two-pattern input combinations that can be obtained on the inputs of a macro, for all the macros. This number is given for comparison with the number of faults included in the proposed fault model, which is given next. The number of faults is followed by the number of faults detected by 100,000 random two-pattern tests, and the fault coverage. It can be seen that very high fault coverage is obtained for gross delay faults, similar to the situation for gate-level circuits. Thus, the added complexity due to the macros does not affect the fault coverage that can be achieved. It can also be seen that the number of faults due to the proposed model is significantly lower than the number of faults if all two-pattern input combinations that can be obtained for each macro are included in the fault model. In addition, in some cases there are large differences between the fault coverage for the circuits obtained from the two translation procedures. This indicates that translation can be performed appropriately, to increase the testability of a circuit to gross delay faults.

Table 4: Simulation of gross delay faults (100,000 random two-pattern tests)

	bl				*fc*			
circuit	2-patt	flts	det	f.c	2-patt	flts	det	f.c
Z9sym	23275	7316	5347	73.09	6916	2926	2607	89.10
add6	7168	1740	1615	92.82	7581	1594	1487	93.29
adr4	3760	792	779	98.36	3801	702	679	96.72
alu1	1856	472	472	100.00	1856	472	472	100.00
alu2	10978	2920	2708	92.74	11821	2552	2433	95.34
alu3	11349	3128	3075	98.31	12314	2926	2820	96.38
co14	3976	1112	532	47.84	3551	826	539	65.25
dk17	9269	2448	2008	82.03	13162	2400	2054	85.58
dk27	4240	888	863	97.18	5104	928	886	95.47
dk48	15564	3154	1840	58.34	12619	2426	1555	64.10
mish	12904	2388	2367	99.12	12904	2388	2367	99.12
radd	2736	632	631	99.84	3257	662	640	96.68
rckl	29538	5662	1395	24.64	24290	4678	951	20.33
rd53	3072	480	480	100.00	3072	480	480	100.00
vg2	9085	2690	2481	92.23	9309	2188	2014	92.05
x1dn	8008	2220	1984	89.37	9333	2200	1997	90.77
x9dn	9522	3046	2328	76.43	10498	2622	2190	83.52
z4	3472	712	672	94.38	2793	542	526	97.05

To give an indication for the reason why the transition fault coverage is not complete, we considered for every macro, every input combination b that can be obtained on the inputs of the macro. We checked whether a primary input combination v can be found, that sets b on the inputs of the macro, and sensitizes its output. If such a vector v cannot be found, then the gross delay faults of the form $<a,b>$ are not detectable, for any a that can be obtained on the inputs of the macro. For $Z9sym$, primary input combinations v could be found only for 75.66% of the b vectors under the *bl* translation, and for 90.59% under the *fc* translation. For *rckl*, primary input combinations v could be found only for 39.83% under the *bl* translation, and for 39.01% under the *fc* translation. Thus the incomplete coverage of gross delay faults is due to the presence of undetectable faults.

4. Function-robust path delay faults

In this section, we first describe a value system for path delay faults in macro-based circuits. We then define function-robust propagation through a macro and review a procedure for counting the number of physical paths in a circuit. We define a path delay fault model, called the function-robust path delay fault model. Contrary to the gate-level path delay fault model, this model includes only faults that can potentially be function-robustly propagated. Thus, faults that cannot be function-robustly propagated are eliminated from the model in advance. We explain the motivation for this model, describe a fault simulation procedure for the proposed model and present experimental results of fault simulation.

4.1 A value system

Delay fault testing requires two pattern tests. Line values under a two-pattern primary input combination $<u,v>$ are represented in this work as $\alpha_1\alpha_2\alpha_3$ [14], where α_1 is the value after the circuit stabilizes under primary input combination u, α_3 is the value after the circuit stabilizes under primary input combination v, and α_2 is the value during the transition from u to v. For primary inputs, the value of α_2 is determined as follows.

If $\alpha_1 = \alpha_3$, then $\alpha_2 = \alpha_1 = \alpha_3$.

If $\alpha_1 \neq \alpha_3$, then $\alpha_2 = x$, where x is the unspecified value.

For example, $0x1$ represents a rising transition and 000 represents a stable 0.

Given the input value triples of a macro, the computation of the output triple is described next. We use the following notation. Consider an m-input macro with input values $\alpha_{1j}\alpha_{2j}\alpha_{3j}$, $1 \le j \le m$. We define, for $1 \le k \le 3$, a macro input combination $a_k = (\alpha_{k1}, \alpha_{k2}, \cdots, \alpha_{km})$, comprised of the k-th value in the triple of every macro input. Thus, a_k is the macro input combination corresponding either to the first pattern of a two-pattern input combination ($k = 1$), or to the second pattern of a two-pattern input combination ($k = 3$), or to the transition between them ($k = 2$). The computation of the macro output triple is first illustrated by an example and then generalized.

Example: Consider the macro-based circuit of Figure 1 under the two-pattern input combination ($ABCDE$) = <(10100), (11001)>. We have the following primary input triples.

$$A = 111,\ B = 0x1,\ C = 1x0,\ D = 000 \text{ and } E = 0x1$$

Consider M_4. Its input triples are $C = 1x0$ and $D = 000$, resulting in the following values on (CD). $a_1 = (10)$, $a_2 = (x0)$ and $a_3 = (00)$. $a_1 = (10)$ results in $g_1 = 1$. $a_3 = (00)$ also results in $g_1 = 1$. $a_2 = (x0)$ implies that any one of the combinations $(CD) \in \{00,10\}$ covered by $(x0)$ can be obtained on the inputs of M_4 during the transition period. The corresponding output values on g_1 are {1,1}. Thus, g_1 is 1 regardless of the manner in which C changes. The triple on g_1 is thus 111. We point out that in setting g_1 to 111, we ignore the possibility of having logic hazards [15] (a logic hazard may cause the output of a macro to change momentarily, even when the functional representation of the macro does not indicate such a change). Logic hazards are discussed in more detail in Section 4.2.

Considering M_3, the input triples $B = 0x1$, $C = 1x0$ and $D = 000$ result in $a_1 = (010)$, $a_2 = (xx0)$ and $a_3 = (100)$, yielding $g_2 = 1x0$.

Consider M_2. Its input triples are $A = 111$, $C = 1x0$ and $g_2 = 1x0$, resulting in $a_1 = 111$, $a_2 = 1xx$ and $a_3 = 100$. $a_1 = (111)$ results in $g_3 = 1$. $a_3 = (100)$ results in $g_3 = 1$. $a_2 = (1xx)$ implies that any one of the combinations {100,101,110,111} covered by $1xx$ can be obtained. The corresponding output values on g_3 are {1,0,1,1}. Thus, g_3 is unknown during the transition. The triple on g_3 is $1x1$, which is a static 1-hazard. □

In general, for a given k, if α_{kj} is specified (0 or 1) for every j, then the vector $a_k = (\alpha_{k1}, \alpha_{k2}, \cdots, \alpha_{km})$ defines a fully specified input combination (a minterm) of M_i, and the output value can be found from the truth table of the macro. If some of the α_{kj}s are unspecified, we consider all the minterms covered by the vector $a_k = (\alpha_{k1}, \alpha_{k2}, \cdots, \alpha_{km})$. If all the minterms result in the same output value γ, then γ is the k-th component of the output value triple. If there is at least one minterm that results in the value 0 and at least one minterm that results in the value 1, then the k-th component of the output value triple is x.

4.2 Function-robust propagation

In gate-level circuits, we say that a transition propagates robustly from an input j of a gate G to its output g under a two-pattern test $<u,v>$, if g does not change unless input j changes, and the effect of input j changing propagates to g independent of the order or speed at which other inputs of G change. When a macro-based circuit is considered, we assume that only the functional description of the macros (e.g., their truth-tables) is known, and that we have no information regarding logic hazards. Due to the potential for logic hazards, we cannot use the gate-level definition for robust propagation from the input j of macro M_i to its output g. For example, consider the first input of a three-input macro M_i under the macro two-pattern input combination <(000),(110)>. Suppose that the transition from (000) to (110) occurs through the sequence (000) -> (010) -> (110). Let the macro output values produced by (000) and (010) be 0, and let the macro output value produced by (110) be 1. During the transition from (000) to (010), a logic static hazard may cause the output to go momentarily to 1. As a result, during the change from (000) to (110), the output is initially 0, then it changes to 1, back to 0, and finally it reaches the value 1 when input 1 changes. In other words, the change from (000) to (110) involves a dynamic hazard. In this case, we cannot say that the transition on input 1 propagates robustly to the macro output, since the macro output may change even before the change on input 1 affects the output. However, if we consider only the functional behavior of the macro, then during the sequence (000) -> (010) -> (110), the output does not change until the change on input 1 affects the output. We conclude that due to logic hazards, that cannot be anticipated from the functional description of the macros, robust propagation similar to gate-level circuits cannot be modeled in macro-based circuits. However, if we ignore logic hazards and consider only functional behavior, a definition similar to the gate-level definition can be given, as follows. We say that a transition propagates *function−robustly* from an input j of M_i to its output g, if g does not change unless input j changes, independent of the order or speed at which other inputs of M_i change. Testing under this definition is referred to as function-robust testing, and the path delay fault model defined under this definition is referred to as the function-robust path delay fault model. The following example illustrates how the condition of function-robust propagation is checked.

Example: Consider the circuit of Figure 1 under the two-pattern input combination of Section 4.1. For M_3, we had input values $B = 0x1$, $C = 1x0$ and $D = 000$, and output value $g_2 = 1x0$. To check whether the transition of B function-robustly propagates to g_2, we keep B at the value 0, and check whether the changes on the other inputs may cause g_2 to change to 0. C changes between 1 and 0, therefore we must consider both values of C, or $C = x$. D is stable at 0, therefore, we need only consider $D = 0$. The resulting vector is $(BCD) = (0x0)$, and it covers the two input combinations (000) and (010). For both input combinations, the resulting value on g_2 is 1, the same as the value for the initial vector (010). Therefore, we conclude that as long as B does not change, g_2 remains at 1 regardless of the speed or order in which the other inputs change. As a result, the transition on B propagates to g_2 function-robustly.

To check whether the transition of C function-robustly propagates to g_2, we consider the input values $(BCD) = (x10)$. It covers the two input combinations (010) and (110). Since (110) results in $g_2 = 0$, we conclude that g_2 may change as a result of a change in an input other than C, and the transition on C does not propagate to g_2 function-robustly. □

Next, we formally define function-robust propagation.

Definition 1: Let a two-pattern input combination $<u,v>$ set the triple $\alpha_{1j}\alpha_{2j}\alpha_{3j}$ on the jth input of M_i. Let the output response of M_i to the input vector $a_k = (\alpha_{k1}, \alpha_{k2}, \cdots, \alpha_{km})$ be α_k, for $1 \le k \le 3$. A $\gamma x \overline{\gamma}$ ($\gamma \in \{0,1\}$) transition is said to propagate function-robustly from input j of M_i to its output under $<u,v>$ if the following conditions hold.

(1) $\alpha_{1j} = \gamma$ and $\alpha_{3j} = \overline{\gamma}$ (there is a $\gamma x \overline{\gamma}$ transition on input j).

(2) $\alpha_1 \ne \alpha_3$ (there is a transition on the macro output).

(3) Let $b = (\beta_1, \beta_2, \cdots, \beta_m)$ be the macro input combination where $\beta_j = \gamma$ and $\beta_p = \alpha_{2p}$ for $p \ne j$ (i.e., input p has a specified

value if it is stable under $<u,v>$, otherwise, it has the value x). Then the output response of M_i to every input vector covered by b must be α_1 (i.e., the output of the macro does not change unless j changes). □

It can be verified that when M_i implements the function of a single m-input gate such as AND, OR or EOR, Definition 1 corresponds to the conventional notion of robust propagation of transitions through such a gate in a gate-level circuit. We omit the details due to space considerations.

4.3 Counting the number of physical paths

A physical path in a macro-based circuit is any sequence of lines, such that the first line is a primary input, every two consecutive lines are a macro input and its output (in this order), and the last line is a primary output. For example, (C,g_2,g_3) and (D,g_1,g_4) are physical paths in the macro-based circuit of Figure 1. The number of physical paths in a macro-based circuit can be computed using Procedure 1 of [7]. We present a variation on this procedure, that is more suitable for our purposes. We assign to every line g in the macro-based circuit a label $N_P(g)$, which is equal to the number of paths from the primary inputs to line g. For a primary input g, $N_P(g)=1$. We then propagate the labels from primary inputs to primary outputs. If line g is the output of a macro with inputs $j_1,j_2,\cdots,j_m$, then line g is labeled by $\sum_{k=1}^{m} N_P(j_k)$. The total number of physical paths is $\sum_g \{N_P(g) : g \textit{ is a primary output}\}$. For illustration, the labeling of the circuit of Figure 1 is shown in parentheses in the figure. The total number of paths is nine, obtained by adding the labels of g_3 and g_4.

4.4 Function-robust path delay faults

We assume that any two-pattern input combination of M_i that propagates a transition from an input of M_i to its output may independently incur an extra delay. Consequently, every physical path may be associated with several different path delay faults, that differ in the two-pattern input combinations assigned to the macros along the path. It is possible to include in the path delay fault model every physical path with every combination of macro input patterns that result in the propagation of a transition along the physical path. Later, we call this set of faults F_1. However, many of the faults defined in this way involve non-robust propagation of transitions, and thus, are meaningless for the purposes of robust testing. We therefore restrict the fault model to include only faults that can *potentially* be tested robustly.

To define the path delay faults, we first consider the path delay faults going from a given input of a macro to its output. We then show how the path delay faults associated with a single macro can be used to find the path delay faults in an arbitrary macro-based circuit. In all cases, we are interested only in path delay faults that can *potentially* be function-robustly tested. This is analogous to considering two path delay faults for every physical path in a gate-level circuit, one corresponding to a rising transition at the source of the path, and one corresponding to a falling transition at the source of the path.

Considering a single macro, M_i, a path delay fault is associated with every two-pattern input combination $<u,v>$ and input j such that both u and v can be obtained on the inputs of M_i, $<u,v>$ sets a transition on input j, and function-robustly propagates it to the macro output.

Example: Consider input B of M_1 in the macro-based circuit shown in Figure 1. The truth table of the macro is given in Table 1. All input combinations can be obtained for every macro in this case. To find the path delay faults associated with the physical path (B,g_4), we consider every two-pattern input combination with a transition on B and a transition on g_4. For future reference, we distinguish among four subsets of input combinations, setting a rising/falling transition on B and a rising/falling transition on g_4. The two-pattern input combinations setting a rising transition on B and a rising transition on g_4 are the following. <(000),(101)>, <(000),(111)>, <(001),(101)>, <(001),(111)>, <(010),(101)> and <(010),(111)>. For each pair, we check whether the transition on B function-robustly propagates to g_4. This happens only for the pairs <(000),(101)> and <(001),(101)>. Thus, there are two path delay faults associated with the physical path (B,g_4), that assign rising transitions to both B and g_4. The path delay faults are represented as $(B,[g_4,000,101])$ and $(B,[g_4,001,101])$, where the brackets contain a macro output and the two input combinations of the macro. For simplicity of notation, we use the decimal values of the input combinations, and represent the faults as $(B,[g_4,0,5])$ and $(B,[g_4,1,5])$. Note that $(B,[g_4,0,7])$ is not a path delay fault, since <(000),(111)> does not function-robustly propagate the transition on B to g_4. Thus, according to our model, it does not create a logical path in the macro-based circuit, and hence no corresponding path delay faults exist in our model. All other two-pattern input combinations can be considered in a similar way. There is a total of four path delay faults associated with the physical path (B,g_4), namely, $(B,[g_4,0,5])$, $(B,[g_4,1,5])$, $(B,[g_4,5,1])$ and $(B,[g_4,7,1])$. □

As implied by the example above, there is a one-to-one correspondence between path delay faults and logical paths. In the sequel, we use these terms interchangeably.

Next, we consider path delay faults in an arbitrary circuit. Consider the physical path (D,g_1,g_4) in the macro-based circuit of Figure 1. Every path delay fault f of M_4 associated with the physical path (D,g_1) can be continued in M_1 as follows. If f brings a rising transition to g_1, then it can be continued with a path delay fault of M_1 that starts with a rising transition on g_1. In this case, a transition on D is function-robustly propagated to g_1 as a rising transition, and from there it is function-robustly propagated to g_4. Similarly, path delay faults of M_4 that bring a falling transition to g_1 can be continued with path delay faults of M_1 that start with a falling transition on g_1. To describe a path delay fault, we associate with every macro output along the physical path a two-pattern input combination of the macro. Such a two-pattern input combination $<u,v>$ has to function-robustly propagate the appropriate transition from the input of the macro that is on the path to the macro output. From the discussion above, we arrive at the following definition.

Definition 2: A path delay fault in a macro-based circuit is a sequence $(g_0,[g_1,u_1,v_1],[g_2,u_2,v_2],\cdots[g_K,u_K,v_K])$, such that $(g_0,g_2,\cdots,g_K)$ is a physical path, $\{u_i,v_i\}$ can be obtained on the inputs of the corresponding macro for every i, and the two-pattern macro input combinations $\{<u_i,v_i>\}$ function-robustly propagate a transition from g_0 through $g_1,\cdots,g_{K-1}$ to g_K. In other words, $<u_i,v_i>$ function-robustly propagates a $t'_i \in$ {rising,falling} transition from g_{i-1} to g_i, reaching g_i as a $t_i \in$ {rising,falling} transition, and $t'_i = t_{i-1}$, for $2 \le i \le K$. □

We also define the set of path delay faults F, that includes every path delay fault according to Definition 2.

It is important to consider the following issue regarding the fault model based on Definition 2. Consider two macro-based circuits C_1 and C_2 that have different numbers of path delay faults N_1 and N_2, however, the same number of detectable path delay faults N_D. Let $N_1 < N_2$. Then the fault coverage in C_1, $\frac{N_D}{N_1}$ is higher than the fault coverage in C_2, which is $\frac{N_D}{N_2}$. However, the reason N_1 is lower than N_2 may be that many of the macro input transitions cannot be function-robustly propagated to the macro outputs. Thus, a circuit comprised of less testable macros may be considered more testable. To check whether such a problem arises, we considered the benchmark circuits of Table 2 under *F*, and under a fault model where the requirement for function-robust propagation is omitted from Definition 2. We refer to this set of faults as F_1. It turned out that for all circuits, the macro-based circuits considered gave similar proportions between the sizes of the *F* sets and between the sizes of the F_1 sets. Thus, the problem of an untestable circuit seeming testable did not occur in these examples.

The number of path delay faults in *F* is computed in the following subsections, and it is shown to be very high even for small circuits. To generate a reduced list of faults for test generation or fault simulation, it is possible to consider only the path delay faults associated with the longest physical paths, similar to the approach taken for gate-level circuits [9]. Next, we propose two other approaches to reduce the list of target faults. Each one of them can be used with the complete set of physical paths, or only with longest paths. Under both approaches, the definition of a path delay fault remains Definition 2, however, the set of faults is reduced by imposing additional constraints. As with other reduced models, a test set for these models is expected to cover a large number of faults not included in the model.

The first alternative to reduce *F* requires that for every physical path *p* (cf. Section 4.3), every line *g* on *p* would be tested twice, once with a rising transition and once with a falling transition. Thus, we remove the requirement for testing each physical path under all two-pattern input combinations that propagate a transition function-robustly through every macro, and require only that both a rising and a falling transition would propagate through each line on every path. The motivation for defining this subset of faults is that it is similar to the path delay fault model in gate-level circuits. In gate-level circuits, for every line *g* on every path *p*, the slow-to-rise and the slow-to-fall path delay faults associated with *p* bring both a rising and a falling transition to *g*. The resulting set of faults, denoted F_2, is significantly smaller than the set *F*.

The second alternative removes the requirement for testing every physical path, and requires only that every line *g* in the circuit would be included in any *r* path delay faults bringing a rising transition to *g* and any *r* path delay faults bringing a falling transition to *g*. In this case, *r* is a predetermined constant. If *r* is made large enough, this model becomes similar to F_2. For small values of *r*, the resulting set of faults, denoted F_3, is significantly smaller than the sets *F* and F_2.

4.5 The number of path delay faults

In this section, we describe a procedure for computing the number of path delay faults in a macro-based circuit according to Definition 2. We denote this number by N_F. It is used as the denominator in our fault coverage figures. We also compute the number of faults in F_2.

The number of path delay faults from input *j* of macro M_i to its output under Definition 2 is denoted $N(i,j)$. For future use, we distinguish between the following four components of $N(i,j)$.

$N_{\alpha_1\alpha_2\alpha_3,\beta_1\beta_2\beta_3}(i,j)$, for $\alpha_1\alpha_2\alpha_3$, $\beta_1\beta_2\beta_3 \in \{0x1, 1x0\}$, is the number of macro input combinations $<u,v>$, such that (1) both *u* and *v* can be obtained on the inputs of M_i, (2) input *j* assumes a transition $\alpha_1\alpha_2\alpha_3$, (3) the output of M_i assumes a transition $\beta_1\beta_2\beta_3$, and (4) the transition on *j* is function-robustly propagated to the macro output.

For example, we previously considered input *B* of M_1 in the macro-based circuit shown in Figure 1. We found six two-pattern input combination with a rising transition on *B* and a rising transition of g_4. Of these, only two were seen to function-robustly propagate the transition from *B* to g_4. Consequently, $N_{0x1,0x1}(1,B) = 2$. In a similar, way we obtain that $N_{0x1,1x0}(1,B) = 0$, $N_{1x0,0x1}(1,B) = 0$, and $N_{1x0,1x0}(1,B) = 2$.

To compute the total number of path delay faults, we use the following procedure. We assign to every line *g* a label indicating the number of path delay faults from the primary inputs to *g*. We distinguish between path delay faults associated with a rising transition on *g*, and path delay faults associated with a falling transition on *g*. Accordingly, every line is assigned two labels, denoted $(N_{P\,0x1}(g), N_{P\,1x0}(g))$. The primary inputs are assigned the label (1,1). The label at the output *g* of a macro M_i is computed as follows. Let input *j* of the macro be labeled $(N_{P\,0x1}(j), N_{P\,1x0}(j))$. Then the path delay faults bringing a $0x1$ transition from the primary inputs to input *j* result in $N_{P\,0x1}(j) \cdot N_{0x1,0x1}(i,j)$ path delay faults bringing a $0x1$ transition from the primary inputs to the macro output. More generally, the labels of the output *g* of M_i are computed as follows.

$$N_{P\,0x1}(g) = \sum_{j=1}^{m} \left[N_{P0x1}(j) \cdot N_{0x1,0x1}(i,j) + N_{P\,1x0}(j) \cdot N_{1x0,0x1}(i,j) \right]$$

$$N_{P\,1x0}(g) = \sum_{j=1}^{m} \left[N_{P0x1}(j) \cdot N_{0x1,1x0}(i,j) + N_{P\,1x0}(j) \cdot N_{1x0,1x0}(i,j) \right]$$

By propagating these labels from primary inputs to primary outputs, we can compute the number of path delay faults as $\sum_{g} \{N_{P\,0x1}(g) + N_{P\,1x0}(g) : g \textit{ is a primary output}\}$.

Next, we consider the number of path delay faults contained in the set F_2 defined at the end of Section 4.4. In F_2, every line on every physical path is included twice, once with a rising and once with a falling transition. Let $N_{PT}(g)$ be the number of physical paths through *g*. Then the total number of faults is $2 \cdot \sum_{g} N_{PT}(g)$. To compute $N_{PT}(g)$, we use two procedures. The first procedure computes the number of paths from the primary inputs to *g*, and the second procedure computes the number of paths from *g* to the primary outputs. The product of the two numbers of paths yields $N_{PT}(g)$. The details of the procedures are omitted for space considerations.

4.6 Simulation of path delay faults

To find the faults detected by a given two-pattern primary input combination, we first perform logic simulation using triple values $\alpha_1\alpha_2\alpha_3$. During the simulation process, we also record the following information, similar to the gate-level case [7]. For every macro output *g*, we include in a set $R(g)$ every macro input *j* such that (1) $R(j) \neq \phi$, and (2) the transition from *j* is function-robustly propagated to *g*. Initially, we set $R(g) = \{g\}$ for the primary inputs carrying $0x1$ or $1x0$ transitions and $R(g) = \phi$ for all other lines. The *R* sets are then updated during the simulation process whenever a transition is function-robustly propagated from an input of a macro to its output. Once the *R* sets are computed, by tracing the *R* sets starting from the primary outputs and proceeding towards the primary inputs, we can identify the path delay faults detected.

Let the number of path delay faults detected be N_D. Then the fault coverage under Definition 2 is $\frac{N_D}{N_F}$. To find the number of detected path delay faults which are included in F_2, we consider every detected path delay fault *f*. Let *f* go through physical path *p*. If *g* is on *p*, then we check whether *g* carries a rising or a falling transition. We then record that *p* with the appropriate transition is detected. The number of faults recorded is then used to compute the fault coverage.

4.7 Experimental results

We incorporated the path delay fault model into a fault simulation procedure and applied it to the macro-based circuits of Section 2. We applied 100,000 randomly generated two-pattern input combinations to each circuit. The results obtained for the *bl* circuits are shown in Table 5, in the following format. After circuit name, we give the number of function-robust path delay faults, the number of path delay faults detected, and the fault coverage. It can be seen that the fault coverage obtained for most circuits is very low, similar to the path delay fault coverage obtained in gate-level circuits. We also report in Table 5 the fault coverage using the set of target faults F_2. Under the F_2 columns of Table 5, we give the total number of faults, the number of detected faults, and the fault coverage for the set of target faults F_2. Also reported in Table 5 is the average number of times a line is included in a distinct, detected path delay fault by Definition 2, with a rising or falling transition. This is given in the last column of Table 5. It can be seen that the the number of path delay faults included in F_2 is significantly smaller than the number of faults included in *F*. Nevertheless, the average number of times a line is exercised is high.

Table 5: Fault coverage for path delay faults in *bl* circuits (100,000 random two-pattern tests)

	F			F_2			
circuit	flts	det	f.c	flts	det	f.c	av.
Z9sym	9.22E8	1702	0.00	3192	2905	91.01	99.29
add6	310064	3078	0.99	846	768	90.78	287.97
adr4	4672	1479	31.66	204	204	100.00	226.61
alu1	603	603	100.00	124	124	100.00	60.30
alu2	184360	10689	5.80	760	694	91.32	873.80
alu3	104461	6850	6.56	798	736	92.23	567.02
co14	4024980	238	0.01	392	196	50.00	38.50
dk17	93096	7145	7.67	1126	913	81.08	712.85
dk27	2409	1668	69.24	344	326	94.77	244.62
dk48	314400	1435	0.46	2066	988	47.82	103.90
mish	3905	3712	95.06	530	530	100.00	59.88
radd	1840	1285	69.84	172	172	100.00	199.76
rckl	3.95E7	883	0.00	2146	215	10.02	28.44
rd53	600	600	100.00	60	60	100.00	150.00
vg2	2.24E7	16551	0.07	1812	1133	62.53	1102.47
x1dn	1.97E7	22465	0.11	1668	1003	60.13	1553.85
x9dn	1.14E7	14804	0.13	2750	1362	49.53	977.29
z4	4336	2592	59.78	150	150	100.00	529.71

5. Concluding remarks

We proposed two delay fault models for macro-based circuits. A gross delay fault model was defined, where a subset of two-pattern input combinations of a macro are each associated with a fault. The two-pattern input combinations were selected to cover all possible transitions of each macro, assuming that a transition involving multiple input changes occurs through a sequence of single input changes. The fault coverage achievable for this model was shown to be high for the macro-based circuits considered, similar to the situation in gate-level circuits. To define a path delay fault model, function-robust propagation from an input to the output of a macro was defined, requiring that the output would change only if the input change occurs. A path delay fault was defined as a physical path associated with two-pattern input combinations for the macro along the path, such that a transition is function-robustly propagated from the input to the output of every macro along the path. Simulation results showed that this model suffers from limitations due to large numbers of paths and low testability of these paths. Reduced sets of path delay faults were therefore defined.

Future work includes the following topics. (1) The experimental results presented indicate that different translations of a circuit into a macro-based circuit result in different testabilities to delay faults. Thus, translation procedures to achieve high delay fault testability will be investigated. (2) Test generation for delay faults in macro-based circuits was not considered in this work, and is the subject of future work. (3) The fault models were developed in this work independent of the details of any specific macro. The suitability of different models to different macros will be studied.

References

[1] K. Keutzer, "DAGON: Technology binding and local optimization by DAG Matching", 24th Design Autom. Conf., 1987, pp. 341-347.

[2] U. Schlichtmann, F. Brglez and M. Hermann, "Characterization of Boolean Functions for Rapid Matching in EPGA Technology Mapping", 29th Design Autom. Conf., June 1992, pp. 374-379.

[3] S. D. Brown, R. J. Francis, J. Rose and Z. G. Vranesic *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[4] Z. Barzilai and B. Rosen, "Comparison of AC Self-Testing Procedures," 1983 Intl. Test Conf., pp. 89-94.

[5] J. L. Carter, V. S. Iyengar and B. K. Rosen, "Efficient Test Coverage Determination for Delay Faults", 1987 Intl. Test Conf., Sept. 1987, pp. 418-427.

[6] G. L. Smith, "Model for Delay Faults Based Upon Paths," 1985 Intl. Test Conf., pp. 342-349.

[7] I. Pomeranz and S. M. Reddy, "An Efficient Non-Enumerative Method to Estimate the Path Delay Fault Coverage in Combinational Circuits", IEEE Trans. on CAD., Feb. 1994, pp. 240-250.

[8] C. J. Lin and S. M. Reddy, "On delay fault testing in logic circuits," IEEE Trans. CAD, pp. 694-703, Sept. 1987.

[9] W.-N. Li, S. M. Reddy and S. K. Sahni, "On path Selection in Combinational Logic Circuits", IEEE Trans. on CAD, Jan. 1989, pp. 56-63.

[10] I. Pomeranz and S. M. Reddy, "Testability Considerations in Technology Mapping", 3rd Asia Test Symp., Nov. 1994.

[11] T. W. Williams, B. Underwood and M. R. Mercer, "The Interdependence between Delay-Optimization of Synthesized Networks and Testing", 28th Design Autom. Conf., June 1991, pp. 87-92.

[12] M. Abramovici, M. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[13] J. Waicukauski, E. Lindbloom, B. Rosen and V. Iyengar, "Transition Fault Simulation," IEEE Design & Test, April 1987, pp. 32-38.

[14] M. Yoeli and S. Rinon, "Applications of Ternary Algebra to the Study of Static Hazards", JACM, pp. 84-97, Jan. 1964.

[15] E. J. McCluskey, *Logic Design Principles with Emphasis on Testable Semicustom Circuits*, Prentice-Hall, 1986.