Synthesis of 100% Delay Fault Testable Combinational Circuits by Cube Partitioning

High-performance systems require rigorous testing for path delay faults. A synthesis algorithm is proposed that produces a 100% path delay fault testable function with a minimal set of test pins.

by William K. Lam

To ensure that manufactured circuits meet specifications, the circuits must be subjected to static and dynamic testing. Static testing considers the steady-state behavior of a circuit (e.g., whether the output of a combinational circuit computes the required Boolean function). Dynamic testing examines the transient behavior of a circuit. In this paper, we focus on a specific kind of dynamic testing: delay testing, which is testing to determine how long it takes a circuit to settle to its steady state.

If we define a path in a circuit to be a sequence of gates from an input to an output of the circuit, then input signals propagate to outputs along paths in the circuit. Thus, the time for a circuit to settle to its steady state, called the delay of the circuit, is determined by the delays of the paths in the circuit. Hence, testing the delay of a circuit translates to testing the paths in the circuit. A common scheme for testing delays is shown in Fig. 1.

To test whether path a-b-c-f has a delay less than or greater than t seconds, a pulse is applied to input a and to input d where it is delayed t seconds to latch the output f. If the delay of path a-b-c-f is greater than t, we say the path has a delay fault. If the steady-state value of f is 0, then latching a 0 implies the delay of the path is less than t, provided the waveform at f has only one transition. If there is more than one transition at f, latching a 0 does not necessarily imply that f has settled to its steady-state value. The delay of the path cannot be inferred from latching the steady-state value if multiple transitions can occur at f. Therefore, a path's delay can be tested if there is an input vector such that a single transition at the input of the path propagates along the path and causes only one transition at the output, independent of the delays of the gates in the circuit. The significance of independence is illustrated in the following example.

In the circuit in Fig. 2, to test the delay of path a-c-f we set input b to 1 causing d to be 0. With this input value, a single transition at input a will propagate along path a-c-f and cause a single transition at f, independent of gate delays in the circuit. Therefore, path a-c-f can be tested for its delay. Similarly, the delay of path b-d-f can be tested by setting a to 0. However, for path b-c-f, a single transition at input b might cause a multiple transition at f, depending on the relative delays of the AND gate and the inverter. For instance, a rising transition at b produces a negative pulse (a falling transition followed by a rising transition) at f if the delay of the AND gate is longer than that of the inverter. On the other hand if the input pulse does not propagate to the output, f maintains a steady 1. Because we don't have prior knowledge about the relative delays of the AND gate and the inverter, we conclude that path b-d-f is not delay testable.

A path is called *robustly path delay fault testable* (RPDFT) if a single transition at the input of the path propagates along the path and produces a single transition at the output, independent of the gate delays in the circuit. Only RPDFT paths can be tested reliably for a delay fault. A necessary and sufficient condition for a path to be RPDFT is that there is an input vector such that during the course of a transition propagating along the path, for each gate on the path, all the side inputs of the gate take on noncontrolling values. A controlling value is an input to a gate that determines the gate's output regardless of the values at other inputs. For example, the controlling value for an AND gate is 0 and for an OR gate

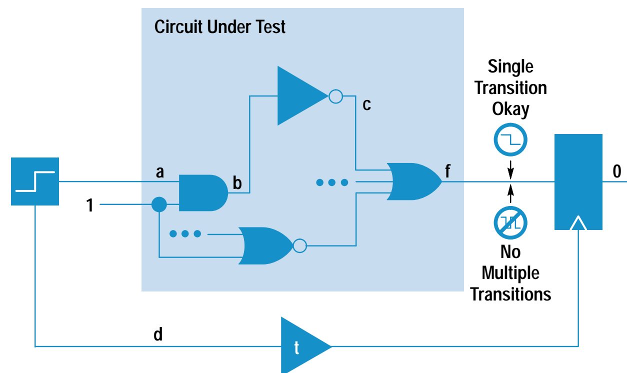

Fig. 1. Delay testing scheme.

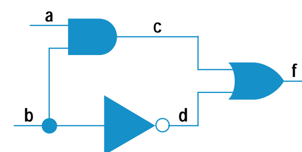

Fig. 2. Example circuit.

© Hewlett-Packard Company 1995

the value is 1. Since the delay of a circuit is determined by the delays of the paths in the circuit, to test the delay of the circuit, all paths in the circuit should be RPDFT. A circuit is said to be 100% RPDFT if all of its paths are RPDFT. Unfortunately, most practical circuits have very few RPDFT paths. This implies that most practical circuits cannot be fully and robustly tested for delay faults, even though many circuits are tested despite the presence of hazards.

In this paper, we propose an algorithm that always synthesizes 100% RPDFT circuits. First, we consider synthesis of 100% RPDFT two-level circuits from any given function. Then, we show how multilevel circuits can be derived from two-level circuits while preserving their delay fault testability.

Previous Work

Devadas and Keutzer[1] derived a necessary and sufficient condition for a path to be RPDFT and proposed an algorithm to synthesize a circuit to achieve a high percentage of RPDFT paths. However, their algorithm cannot always produce circuits with 100% RPDFT. It is known that there exist functions that do not have 100% RPDFT implementations. A natural question is: can any function be augmented so as to have a 100% RPDFT implementation? One way of augmenting a function is to add extra inputs. With this technique, Pomeranz and Reddy[2] demonstrated that many circuits can be made to be 100% RPDFT. However, it is not known whether any arbitrary function can be synthesized to be 100% RPDFT by using this technique or any other.

Synthesis of 100% RPDFT Two-Level Circuits

An example of a two-level circuit is an AND-OR implementation configuration (e.g., a programmable logic array) corresponding to a sum-of-products representation of a Boolean function. Any Boolean function can be represented as a sum of products. For example, $f = (a + b)(\overline{a} + \overline{b}) + bc$ has the sum-of-products representation $a\overline{b} + \overline{a}b + bc$, whose corresponding two-level implementation consists of three AND gates and one OR gate. Each AND gate implements a product term and the OR gate combines the outputs of the AND gates as shown in Fig. 3. The circuit in Fig. 3 is called a two-level implementation because the first level consists of AND gates and the second level an OR gate.

The path P in Fig. 3 from b through the AND gate for the term bc is not RPDFT because no matter what value input a is set to, a rising or falling transition at b through the path will produce multiple transitions at f or not propagate along P, depending on the relative delays of the AND gates. For instance, if input a is set to 1 and the delay of the left AND gate is shorter than the delay in the right AND gate, then a falling transition at b along P will be blocked from propagating to f (Fig. 4a). Thus, the delay of P cannot be reflected at

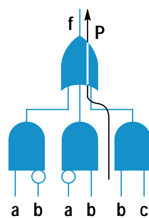

Fig. 3. A two-level circuit.

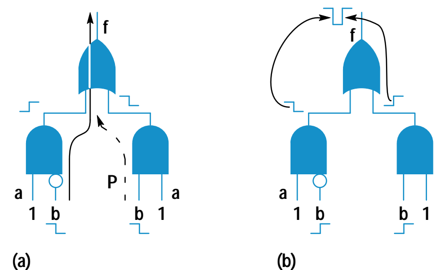

Fig. 4. Propagation of transitions. (a) Because the delay in the left AND gate is shorter than the delay in the right AND gate, a falling transition at b is blocked from propagating to f. (b) A rising transition at b causes a negative pulse at f.

f. Under the same setting, a rising transition at b will cause a negative pulse at f (Fig. 4b). If input a is set to 0, then a rising transition at b will be blocked from propagating to f because the output of the AND gate is forced to 0. Thus, path P is not RPDFT.

For more complicated functions, it would be difficult to perform the above analysis to determine whether paths are RPDFT. To make the task of identfying RPDFT paths easier, an algebraic method[3] is presented.

Definitions. Before stating the algebraic method, some terms need to be introduced.

- The *cofactor* of function f with respect to variable x (for positive phase), denoted by f_x, is derived from f by replacing the variable x in f with 1. Similarly for negative phase, $f_{\overline{x}}$ is derived by replacing x with 0.
- The *smooth operator* S on function f with respect to variable x, denoted by $S_x(f)$, is $f_x + f_{\overline{x}}$.
- Let C be a product term or *cube** in f. Then f − C is the function derived from f by eliminating C.

Cofactor f_x is the evaluation of f at x = 1. Smoothing f with respect to x gives the function independent of x.

Theorem 1: Let f be a function in a sum-of-products form of a two-level circuit, and π a path starting from primary input x and going through the AND gate of cube C. Then, path π is RPDFT if and only if there is an input vector ν = (…,x,…) such that:

$$S_x(C)\ \overline{S_x(f - C)}\ (\nu) = 1.$$

The vectors ν and ν′ = (…,$\overline{x}$,…) are a test vector pair for the delay fault on π.

For an arbitrary function in a sum-of-products form, $S_x(C)\ \overline{S_x(f - C)}\ (\nu)$ may be 0 for all vectors. This would mean that the path through C starting at input x is not RPDFT. To augment a given function so that it has a 100% RPDFT implementation, we add extra inputs called *test pins,* which equal 1 under normal operations and may be selected to be 0 in delay testing mode.

To construct a circuit with 100% RPDFT paths the set of cubes in a given function is partitioned into subsets such that each subset forms a 100% RPDFT function. Next, a pin is attached to each subset. To test a path in the subset, only the test pin of the subset is set to 1, while all the remaining test pins are set to 0. Since the subset is 100% RPDFT, the paths

* A cube is a product term. For example, $\overline{a}bc$ and $b\overline{c}$ are cubes, but a + c is not a cube.

© Hewlett-Packard Company 1995

are RPDFT under this setting of the test pins. Symbolically, let $f = \sum_i C_i$, where C_i is a cube which can be partitioned into subsets of cubes, S_i, such that each path in each S_i is RPDFT. The new augmented function is now $f = \sum_j T_j S_j$, where T_j is the test pin for cube subset S_j.

To test path π going through a cube in S_j, the test pins must be set such that $T_j = 1$ and $T_i = 0$ for $i \neq j$. So f becomes S_j, which is 100% RPDFT by construction, enabling π to be tested for a delay fault. In normal operation, all test pins are set to 1 allowing the augmented function $f = \sum_j T_j S_j$ to restore the original function $f = \sum_j S_j = \sum_i C_i$.

A natural question is: Can an arbitrary function be partitioned into such subsets? The answer is yes, because a partition in which S_i is a cube is such a partition. Further, the paths through the test pins do not need to be tested for delay faults because these pins are held constant during normal operation. Therefore, for any arbitrary function, a 100% RPDFT implementation is always possible with this cube-partitioning scheme. This fact is formally stated in the following theorem.

Theorem 2: Any Boolean function has a prime and irredundant two-level AND-OR implementation with 100% RPDFT and the possibility of adding new inputs. Further, if C is a two-level AND-OR implementation of f, then C can always be resynthesized to be 100% RPDFT.

To resynthesize a two-level circuit to be 100% RPDFT, the worst case is when a test pin is needed for each cube in the circuit. In this worst case, the additional area required is at most twice the original area, assuming each test pin is ANDed with the cube. This procedure allows designers to synthesize two-level circuits without considering delay fault testability because test pins can be added later to achieve the desired testability.

Because a test pin is provided for each subset, a minimum partition is desired. Of course, the designer does not have to make all paths RPDFT because test pins can be added only to the cubes in which the paths need to be tested. In this case, the number of test pins to add is bounded by the number of cubes that involve the paths to be tested. Nevertheless, we want an algorithm that produces a minimal partition.

The following algorithm produces a minimal cube partition by partitioning a set of cubes $f = \{ C_i \}$ into $\{S_i\}$ so that the sum of cubes in each S_i is a function with 100% RPDFT paths.

```
i=0;
while(f not empty) {
  i++;
  Si = {φ}
  for each cube C ∈ f {
    if(Si ∪ C is 100% RPDFT) {
      Si = Si ∪ C;
      remove C from f;
    }
  }/* end for loop */
} /* end while loop */
```

The test pins do not need to be connected directly to the outside world through pins on the package. A shift register, which can be an existing scan chain, can be used to shift in the test patterns. The extra pins needed are at most two, one for the shift register input and the other for its clock. A possible implementation is shown in Fig. 5. In the figure $T_1 \ldots T_n$ are the test pins whose values are set by the output of the decoder, which decodes the test patterns from the shift register. Each subset S_i needs one test pin.

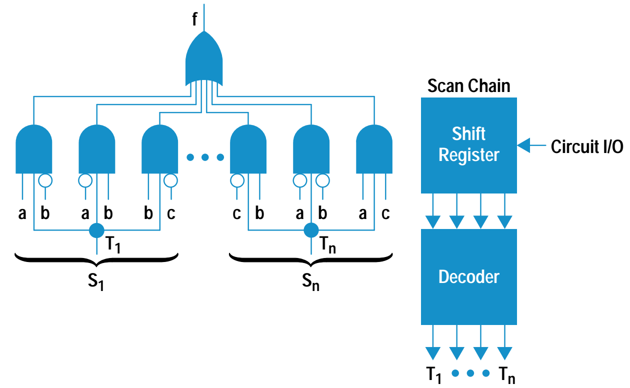

Fig. 5. 100% robustly path delay fault testable (RPDFT) implementation.

Multilevel Synthesis

A two-level implementation is a special case of a multilevel implementation and usually requires much more silicon area. This is because a multilevel implementation does more sharing of gates. For example, the multilevel circuit in Fig. 6a, which uses four two-input gates, would require eight two-input-equivalent gates if the same function were implemented using a two-level structure (Fig. 6b).

The multilevel implementation can be represented as $f = (a + b)(c + d) + e$, while the two-level representation is $f = ab + ad + bc + bd + e$. The multilevel implementation is simply a factored form of the two-level implementation. Thus, a two-level implementation can be transformed into an area-saving multilevel implementation by factoring out common terms. The question that comes up after these transformations is whether testability is preserved. That is, will a RPDFT path in the original two-level implementation remain RPDFT in the factored multilevel implementation and will a path newly created by these transformations be RPDFT? In the Boolean domain, factorizations like $ab = a(\bar{a} + ab)$ and $(a + b) = (a + b)(\bar{a} + a)$ are valid. Factorizations involving the use of Boolean rules such as $a + \bar{a} = 1$, $a \cdot \bar{a} = 0$, and $a \cdot a = a$

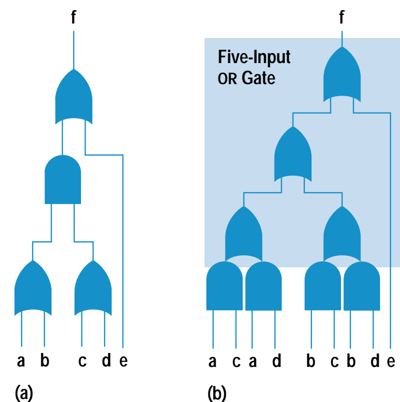

Fig. 6. (a) A multilevel circuit. (b) A two-level equivalent of the multilevel circuit in (a).

© Hewlett-Packard Company 1995

are called *Boolean factorizations,* and factorizations that don't use such rules are called *algebraic factorizations.* Hachtel, Jacoby, Keutzer, and Morrison[4] proved that a multilevel implementation derived from a two-level implementation using only algebraic factorizations preserves RPDFT of the paths in the original two-level implementation. This concept is summarized in the following theorem.

Theorem 3: If C, a two-level multiple-output circuit, is 100% RPDFT, and A is a multilevel circuit derived from C through the application of algebraic operations, then A is also 100% RPDFT.

Therefore, synthesis for a multilevel circuit with 100% RPDFT can be done in two steps. First, a two-level circuit with 100% RPDFT paths is synthesized using the cube partitioning method. Then, a multilevel circuit is derived from this two-level circuit by applying algebraic factorizations.

Selective Critical Path Testing

Because making all paths in a chip delay fault testable may not be area-efficient, only some representative paths are selected to be made delay fault testable. Theoretically, testing only a fraction of paths may not guarantee freedom from faults for the entire chip. However, because of the nature of delay tracking in IC processing, proper selective schemes can offer high confidence in testing.

Also, because gate delays within a chip track well, a long path is more likely to fail a timing specification than a short path, making longer paths good candidates to be selected for testing. If a selected path is not RPDFT, it can be made so by using one of the synthesis techniques discussed above. Specifically, to make a selected path RPDFT, find a maximal set of RPDFT cubes that contain the path and introduce a new test pin to isolate the set of cubes from the rest of the cubes using the minimal cube partitioning algorithm. This step is repeated until all selected paths are RPDFT. Finally, the number of test pins can be minimized by first repartitioning the cubes in the cube sets so that each cube belongs to only one new cube set, and then using one test pin for each new cube set.

Experimental Results

The cube partitioning algorithm was implemented on the Berkeley SIS (sequential interactive synthesis) platform and runs on an HP 9000 Model 735 workstation, which has about 150M bytes of RAM. Two benchmarks from the International Symposium on Circuits and Systems (ISCAS) and the Microelectronic Center of North Carolina were used on the algorithm. Table I shows the results of running these benchmarks through the cube partitioning algorithm.

The second, third, fourth, and fifth columns in Table I contain the total number of I/O pins, gates, paths, and non-RPDFT paths, respectively, in each circuit. The sixth column defines the original fault testability, that is, the fraction of paths that are RPDFT in a particular circuit. After these circuits were resynthesized, they became 100% testable (i.e., final testability = 1.0). The eighth column reflects the total number of test pins inserted to make the circuit fully delay fault testable. With the exception of circuit b12, after all the circuits were resynthesized they were made fully delay fault testable with six or fewer test pins. The ninth column is the area overhead, which is the ratio of the area increase over the original circuit area. Since any additional area adds some delay, the delay overhead for each circuit results from a layer of two-input AND gates for each test pin insertion. Finally, the last column is CPU execution time for each circuit. These times vary directly with the number of cubes in the circuit.

Circuits with very few non-RPDFT paths and circuits that did not finish within the 12-hour preset time limit are not listed in Table I.

Table I
Two-Level Synthesis of 100% RPDFT Circuits

Circuit	I/O Pins	Gates	Paths	Non-RPDFT Paths	Initial Testability*	Test Pins	Area Overhead (%)	CPU (Seconds)
table5	32	188	7259	495	0.93	4	3.25	141
table3	28	203	7381	687	0.90	4	3.38	282
rd84	24	263	3280	1456	0.55	2	0.48	821
apex1	90	296	9109	1515	0.83	6	6.16	25834
b12	24	449	1922	1845	0.04	23	5.53	423
ex1010	20	830	14710	2096	0.85	4	0.89	17434
z5xp1	17	148	4032	2558	0.36	4	5.26	336
z9sym	10	422	3780	3276	0.13	2	0.14	21694
ex4	156	676	4404	3632	0.17	3	1.14	175
alu4	22	1044	7875	3955	0.49	4	0.53	134
apex4	28	476	14958	4354	0.70	5	3.11	12368
misex3	28	1876	17971	5258	0.70	6	5.84	15936

* Final testability = 1.0.

© Hewlett-Packard Company 1995

Conclusion

In this paper, we studied the problem of synthesizing circuits with 100% RPDFT. We proved that for an arbitrary function, there exists a 100% RPDFT implementation, and we proposed a synthesis algorithm that always produces a 100% RPDFT implementation for any function and a minimal set of test pins. Further, we showed that a circuit synthesized using the proposed algorithm uses at most twice as much area as any two-level implementation of the circuit. For most practical circuits, the additional areas are small. Finally, we demonstrated how area-efficient multilevel circuits with 100% RPDFT can be constructed by applying algebraic factorizations to the synthesis algorithm.

Acknowledgments

The author would like to thank Barbara Fredrick for her enthusiastic support, Cheryl Harkness for a lesson on FrameMaker, Mark Heap for the many insightful discussions on the algorithms presented in this paper, and Robert Aitken and Peter Maxwell for reviewing the paper.

References

1. S. Devadas and K. Keutzer, "Synthesis of delay-fault-testable circuits: Theory," *IEEE Transactions on Computer-Aided Design,* Vol. 11, no. 1, January, 1992, pp. 87-101.
2. I. Pomeranz and S. Reddy, "Achieving Complete Delay Fault Testability by Extra Inputs," *International Test Conference '91,* Oct. 1991, pp. 273-282.
3. W. Lam and R. Brayton, *Timed Boolean Functions—A Unified Formalism for Exact Timing Analysis,* Kluwer Academic Publishers, 1994.
4. G. D. Hachtel, R. M. Jacoby, K. Keutzer, and C. R. Morrison, "On the Relationship Between Area Optimization and Multifault Testability of Multilevel Logic," *IEEE/ACM International Conference on Computer-Aided Design '89,* November 1989, pp. 422-425.

Bibliography

1. W. Lam, A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli, "Delay Fault Coverage, Test Set Size, and Performance Tradeoffs," *IEEE/ACM Design Automation Conference '93,* June 1993, pp. 446-452.

© Hewlett-Packard Company 1995