OUTPUT ENCODING FOR HAZARD-FREE ROBUST PATH DELAY FAULT TESTABILITY

Subhasish Mitra * and Edward J. McCluskey

Center for Reliable Computing, Stanford University

Gates 236, MC 9020

Gates Building 2A

Stanford, CA 94305

Voice: (650)-723-1258

FAX: (650)-725-7398

Email : smitra@crc.Stanford.EDU

* - designated presenter

ABSTRACT

In this extended abstract, we present a technique for encoding the symbolic outputs of a given specification such that the resulting two-level logic is hazard-free robust path delay fault testable. By applying delay fault testability preserving transformations, we can obtain a multi-level implementation from the two-level logic circuit. As a secondary goal, our technique attempts to minimize the area of the synthesized design.

1. INTRODUCTION

Due to the presence of defects, the propagation delay of circuit paths in a synchronous digital circuit may exceed the clock interval — these are called delay faults. A delay test that is *robust* should be valid in the presence of arbitrary delays and not invalidated by hazards. If a delay test for a fault on a path *p* single-event sensitizes *p*, then it is a *hazard-free robust test* — this means, that there is a single event propagating through the path *p* to the output and no other event propagates through any other path to the output. Hazard-free robust delay test has an added advantage because we can isolate the path having the delay fault. When the outputs of a logic function are specified symbolically, *output encoding* algorithms are used to encode the symbolic outputs as binary values. Depending on how the outputs are encoded, the resulting logic function may or may not be delay fault testable. In this paper, we present an output encoding technique such that the resulting two-level logic function (obtained after encoding) is robust path delay fault testable — in fact, our aim is to obtain a hazard-free robust delay testable logic in order to exploit the advantage of hazard-free robust delay test over general robust delay test. As a secondary goal, our encoding technique attempts to minimize the area of the synthesized design.

2. MOTIVATION

The output encoding problem involves choosing binary codes for the symbolic outputs of a given specification. Since the encoding affects the area of the final logic implementation, a good output encoding is very important. An exact technique for output encoding for generating a minimum area two-level logic has been described in [Devadas 91].

A particular output encoding can also affect the delay fault testability of the final logic implementation. In this paper, we consider delay fault testability of the two-level logic that is generated from a particular encoding. The reason for this is that the existing logic synthesis techniques for robust path delay fault testability [Devadas 92a][Devadas 92b][Maleh 94] first generate robust path delay fault testable two-level logic and then apply testability preserving transformations to obtain multi-level logic implementations. The technique of designing robustly delay fault testable combinational circuit, reported in [Kundu 91], also starts with the two-level logic implementation and applies Shannon's expansion iteratively to obtain the final testable logic. Thus, if we can ensure that a two level implementation is robust path delay fault testable, then testability preserving transformations can be applied to obtain a multi-level logic implementation.

Let us consider the specification in Table 1. Since, there are six symbolic outputs, we use 3 bits to encode them. Consider the encoding of the symbolic outputs as shown in Table 2(a). The truth table for the logic to be implemented with this encoding is shown in Table 2(b).

Table 1: Example truth table

Minterms abcd	Symbolic Outputs
0000	out3
0010	out1
0011	out4
0110	out4
0111	out1
1000	out4
1001	out4
1010	out2
1100	out0
1101	out0
rest	out5

Table 2(a): An Encoding

Symbolic Outputs	Encoding xyz
out0	111
out1	110
out2	100
out3	010
out4	101
out5	000

Table 2(b): Truth Table of the logic

Minterms abcd	Encoded Outputs xyz
0000	010
0010	110
0011	101
0110	101
0111	110
1000	101
1001	101
1010	100
1100	111
1101	111
rest	000

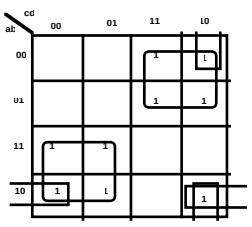

Fig.1 Karnaugh Map for the first output column of Table 2(b)

Minterms	Symbolic Outputs
1101	out1
1100	out2
1111	out3
0000	out4
0001	out4

Table 3: Example of output encoding process [Devadas 91]

Figure 1 shows the Karnaugh map for the first output column (output x) of the truth table specified in Table 2(b). The minimal two-level implementation of the first output column of table 2(b) is either $ac' + a'c + ab'd'$ or $ac' + a'c + b'cd'$. It can be easily verified that the two-level implementations of neither of these boolean expressions are hazard-free robust path delay fault testable. For the first expression, let us consider any path from input *a* through the AND gate realizing $ab'd'$ through the OR gate to the output. In order to sensitize the path,

both *b* and *d* must be 0. If *c* equals 0 (1), there is another event that propagates from input *a* through the AND gate *ac´* (*a´c*) to the output. Thus, hazard free robust delay testing of the above mentioned path is not possible. A similar situation happens when we try to test the path starting from *c* through the AND gate *b´cd´* to the output for the other expression.

However, for the example of Table 1, if we encoded out2 as 011, then the two-level logic corresponding to the first output column will be hazard-free robust testable for path delay faults — then the expression for the output column x becomes *ac´ + a´c*. It can be shown that the minimal two-level implementations of the second and third (outputs y and z) output columns of Table 2(b) are robust delay fault testable. It may be noted that we minimized the three output functions of Table 2(b) separately because it has been proved in [Devadas 92a] that individual minimization of each output function is a necessary condition for hazard-free robust path delay fault testability of the synthesized two-level logic.

3. Basic Methodology followed

The basic methodology followed is an extension of the Quine-McCluskey method of two-level logic minimization [McCluskey 56][McCluskey 86] for encoding symbolic outputs as described in [Devadas 91]. For the sake of completeness, we describe the procedure reported in [Devadas 91] briefly in this section with the help of the example in Table 3 (taken from [Devadas 91]).

Each minterm has a tag associated with it — it is the set of symbolic outputs whose ON-set the minterm belongs to. For the example in Table 3, tag of 1101, 1100, 1111, 0000 and 0001 are {out1}, {out2}, {out3}, {out4} and {out4}, respectively. While generating prime implicants (called *generalized prime implicants* (GPI), in this case), two *k*-cubes are merged to form a *k+1*-cube (a full minterm is a 0-cube) and the tag of the resulting *k*-cube is the *union* of the tags of the two original *k*-cubes that were merged to obtain the *k+1*-cube. Moreover, a *k+1*-cube can cancel a *k*-cube only if their tags are equal. For this example, 0-cubes 1101 and 1100 are merged to obtain the 1-cube 110- with the tag {out1, out2}, 0-cubes 1101 and 1111 are merged to obtain 11-1 with the tag {out1, out3} and 0000 and 0001 are merged to obtain 000- with the tag {out4}. Since the tag of 000- is the same as that of 0000 and 0001, we cancel 0000 and 0001. However 1101, 1100 and 1111 cannot be canceled.

After computing the generalized prime implicants, as described above, a procedure for obtaining a minimal cover has been described in [Devadas 91]. Once the cover is obtained, encodeability constraints are applied on the cover and a graph based algorithm or a boolean satisfiability algorithm is executed to check whether the cover is *encodeable* [Devadas 91]. If the cover is encodeable, the codes for the symbolic outputs are directly obtained from the algorithm — otherwise, some other minimal cover is selected and the encodeability checks are performed on that cover until an encodeable cover is obtained. For a given encodeable set of GPI's, the final cover is obtained in the following way: for each GPI in the set, the bitwise intersection of the encoding of the symbols in the tag set of the GPI tells us the output functions for which the GPI is 1 or 0. Exactness of the above procedure has been proved in [Devadas 91].

In this paper, we modify the basic scheme in order to incorporate hazard-free robust path delay fault testability constraints on a minimal encodeable cover.

4. DELAY FAULT TESTABILITY CONSTRAINTS

In this section, we describe our technique for generating delay fault testability constraints with the help of the example in Table 4(a).

Table 4(a): An example to illustrate our technique

Minterms xyz	Symbolic Outputs
101	out1
100	out2
111	out3

Table 4(b): The generalized prime implicants for Table 4(a)

GPI	Cube xyz	Tag
g_1	101	out1
g_2	100	out2
g_3	111	out3
g_4	10-	out1, out2
g_5	1-1	out1, out3

As shown in Table 4(b), there are five GPI's: g_1: (xy´z ; <out1>), g_2: (xy´z´ ; <out2>), g_3: (xyz;<out3>), g_4: (xy´ ; <out1, out2>) and g_5: (xz ; <out1, out3>). We compute the delay fault testability properties of a set of these GPI's as follows: let us compute the delay fault testability of the GPI g_1 (xy´z; <out1>) in the presence of g_2 (xy´z´; <out2>). For single event sensitization of the path p from input z through g_1 to the output, we require that there must exist an assignment of 1's and 0's to x and y such that the path p is sensitized through g_1 *and* the output of g_2 is 0. For sensitizing path p through g_1, we require (x = 1 *and* y = 0); to have 0 on the output of g_2, we require (x = 0 *or* y = 1); however, these two fomulae are not satisfiable at the same time. Hence, if g_1 is used as an AND gate in the implementation of an output function (after encoding), then the AND gate corresponding to g_2 cannot be used in the implementation of the same function. This means, that for any encoding, if AND gates corresponding to g_1 and g_2 are present in the final implementation, then for all bits in the encoding of out1 and out2, if a particular bit *i* is 1 in the encoding of out1 then bit *i* must be 0 in the encoding of out2. A convenient way

of writing this constraint is: $\forall i$, $e_i(out1) = 1 \rightarrow e_i(out2) = 0$. Here, $e_i(out1)$ means the i^{th} bit in the encoding of out1. Note that, delay fault testability of g_2 in the presence of g_1 also provides the same constraint.

Similarly, consider the delay fault testability of g_1 in the presence of g_4. Single event sensitization of the path from input y through the AND gate corresponding to g_1 to the output requires (x = 1 and z = 1) and (x = 0) which is not satisfiable. As discussed in the previous section, the output of g_1 will be 1 for each bit that is 1 in the encoding of out1 while the output of g_4 will be 1 for each output bit that is 1 in the encoding of both out1 and out2. Hence, if AND gates corresponding to g_1 and g_4 are used in the final implementation, then the following constraint is generated: $\forall i$, $[e_i(out1)] = 1 \rightarrow [e_i(out1) \wedge e_i(out2)] = 0$, because, the AND gates g_1 and g_4 cannot be used together for the realization of any output function. If we calculate the delay testability properties of each GPI in presence of the other GPI, we find that if AND gates $\{g_2, g_3\}$ or $\{g_4, g_5\}$ are chosen for any implementation then there is no constraint — otherwise constraints are generated. Thus, we need not consider the testability of $\{g_2, g_3\}$ in presence of g_1 because from the previous step we already know the constraints generated when AND gates corresponding to g_1 and g_2 (g_1 and g_3) co-exist in the final implementation.

Suppose we choose AND gates $\{g_4, g_5\}$ for any implementation (cover). We know that AND gates g_4 and g_5 can coexist in any implementation without any delay testability constraints. However, there are some encodeability constraints [Devadas 91] which are satisfied when we assign 11 to out1, 10 to out2 and 01 to out3. The two encoding bits are c_1 and c_2. The encoding is shown in Table 5. The resulting two-level logic circuit is shown in Fig. 2 and as per our analysis, it is guaranteed to be hazard-free robust path delay fault testable as every path is single event sensitizable.

Symbolic Outputs	Encoding c_1 c_2
out1	1 1
out2	1 0
out3	0 1

Table 5: Encoding for symbolic outputs of Table 4(a)

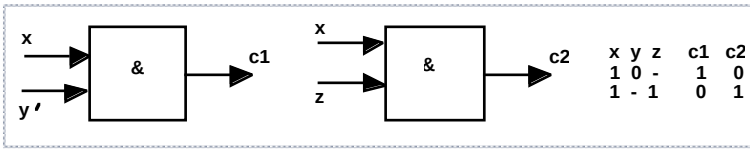

Figure 2. Implementation of two-level logic for Table 5 encoding

Thus, given a set of GPI's, we implicity perform a check for path delay fault testability using boolean satisfiability techniques and generate constraints. Any encoding that satisfies these constraints are guaranteed to produce a hazard-free robust path delay fault testable two-level logic.

5. CONCLUSIONS

In this extended abstract, we have described an exact technique for performing output encoding such that the final two-level logic, obtained after performing the encoding, is hazard-free robust path delay fault testable. For that purpose, we have generated delay fault testability constraints which will be used by a constraint satisfaction solver to obtain the encoding. It can be shown that if we do not have a bound on the number of bits used to encode the symbolic outputs, then we can always satisfy the delay fault testability constraints together with the area efficiency constraints described in [Devadas 91]. An interesting extension of this work will be to choose delay fault testability constraints, given the number of bits used to encode the symbolic outputs, such that the minimum number of test points will be required to render the final logic delay fault testable. Once our technique generates a hazard-free robustly path delay fault testable two-level logic, it is possible to use existing testability preserving transformations to convert it into a multi-level logic implementation that is also hazard-free robustly path delay fault testable. This method has an added advantage due to the fact that the hazard-free robust delay tests for the paths in the final two-level logic are automatically derived as a result of the encoding process.

6. REFERENCES

[Devadas 91] Devadas, S. and A. R. Newton, "Exact Algorithms for Output Encoding, State Assignment and Four Level boolean Minimization," *IEEE Trans. on CAD*, 10(1), pp. 13-27, Jan. 1991.

[Devadas 92a] Devadas, S. and K. Keutzer, "Synthesis of Robust Delay-Fault-Testable Circuits: Theory", *IEEE Trans. on CAD*, vol. 11, no. 1, pp. 87-101, Jan. 1992.

[Devadas 92b] Devadas, S. and K. Keutzer, "Synthesis of Robust Delay-Fault-Testable Circuits: Practice", *IEEE Trans. on CAD*, vol. 11, no. 3, pp. 277-300, Mar. 1992.

[Kundu 91] Kundu, S., S. M. Reddy and N. K. Jha, "Design of Robustly Testable Combinational Logic Circuits," *IEEE Trans. on CAD*, vol. 10, no. 8, pp. 1036-1048, Aug. 1991.

[Maleh 94] Maleh, A. E. and J. Rajski, "Delay-Fault Testability Preservation of the Concurrent Decomposition and Factorization Transformations," *Proc. VLSI Test Symp.*, pp. 15-21, 1994.

[McCluskey 56] McCluskey, E. J., "Minimization of Boolean Functions," *Bell Lab. Tech. Journal*, pp. 1417-1444, Nov. 1956.

[McCluskey 86] McCluskey, E. J., *Logic Design Principles with Emphasis on Testable Semicustom circuits*, Prentice-Hall, Eaglewood Cliffs, NJ, USA, 1986.