MuTaTe: An Efficient Design for Testability Technique for Multiplexor based Circuits

Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
rd@tzi.de

Junhao Shi
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
junhao@tzi.de

Görschwin Fey
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
fey@tzi.de

ABSTRACT

We present a technique to derive fully testable circuits under the *Stuck-At Fault Model* (SAFM) and the *Path-Delay Fault Model* (PDFM). Starting from a function description as a *Binary Decision Diagram* (BDD) the netlist is generated by a linear time mapping algorithm. Only one additional input and one inverter are needed to achieve 100% testable circuits under SAFM and PDFM. Experiments are given to show the advantages of the the technique in comparison to previously presented methods.

Categories and Subject Descriptors

B.6 [**Logic Design**]: Design Sytles—*combinational logic*; B.8 [**Performance and Reliability**]: Testing

General Terms

Algorithms, Design, Testing

Keywords

Decision Diagrams, BDDs, Logic Synthesis, Design for Testability, Multiplexor based Circuits

1. INTRODUCTION

Since *Binary Decision Diagrams* (BDDs) have been proposed in [12, 1] several applications have been studied, where this data structure can be successfully applied. In formal verification and logic synthesis BDDs have become the state-of-the-art for function representation and manipulation (see e.g. [9]). BDDs have also been studied in logic synthesis, since they allow to combine aspects of circuit synthesis and technology mapping [11]. Recently, there is a renewed interest in multiplexor based design styles, since often multiplexor nodes can be realized at very low cost (as e.g. *Pass Transistor Logic* (PTL)). In addition, these techniques allow to consider layout aspects during the synthesis step and by this guarantee high design quality (see e.g. [14, 13]).

One of the most important steps during circuit design is the testability of the netlist. Multiplexor circuits derived from BDDs have been studied intensively under various fault models [3, 2, 5, 4]. (For an overview see [6].) But none of these approaches can guarantee 100% testability in a "systematic way". E.g. in [5] an algorithm is given that can compute all redundancies of the circuit in polynomial time. But the removal of these redundancies can generate new ones (so-called 2nd-generation redundancies). For their removal only classical ATPG can be applied.

In this paper, a simple transformation is presented that guarantees full testability of a circuit derived from a BDD description under the stuck-at fault model and the robust path-delay fault model. The size of the circuit is directly proportional to the given BDD size. All optimizations of the BDDs based on variable ordering directly transfer to the resulting circuit sizes. Only one extra input and one inverter are needed. The resulting circuits are free of redundancies. The algorithm has been implemented as a tool for *Multiplexor Transformation for Testabilty* (MuTaTe). Experimental results are given that show the advantages of the approach compared to traditional synthesis approaches and to "classical" mapping of BDDs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GLSVLSI'03, April 28–29, 2003, Washington, DC, USA.
Copyright 2003 ACM 1-58113-677-3/03/0006 ...$5.00.

2. PRELIMINARIES

2.1 Binary Decision Diagrams

As is well-known a Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) which is a directed acyclic graph where a Shannon decomposition

$$f = \overline{x}_i f_{low(v)} + x_i f_{high(v)} \quad (1 \leq i \leq n)$$

is carried out in each node. A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does not contain isomorphic subgraphs nor does it have redundant nodes. Reduced and ordered BDDs are a canonical representation since for each Boolean function the BDD is uniquely specified [8]. In the following, we refer to reduced and ordered BDDs for brevity as BDDs.

2.2 BDD Circuits

It is well-known, that BDDs directly correspond to multiplexor based Boolean circuits, called BDD circuits in this paper. More exactly: BDD circuits are combinational logic circuits defined over a fixed library. The typical multiplexor cell is denoted as MUX, and it is defined as given in Figure 1 by its standard *AND-, OR-, INVERTER*-based realization[1]. The left input is called *control input*, the upper inputs are called *data inputs* (left data input = *0-input*, right data input = *1-input*).

The *BDD circuit* of a BDD is now obtained by the following construction: Traverse the BDD in topological order

[1]All results in the following also transfer to different realizations.

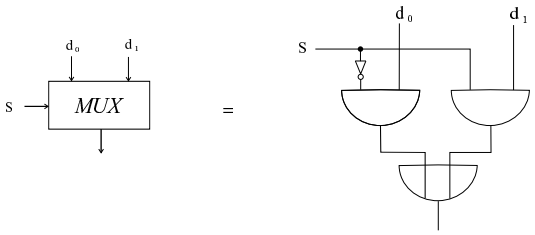

Figure 1: Multiplexor cell *MUX*

and replace each non-terminal node v in the BDD by a *MUX* cell, connect the control input with the primary input x_i, corresponding to the label of the BDD node. Then, connect the 0-input to $low(v)$, the 1-input to $high(v)$. At the end connect the output of the multiplexor which substituted the root node with a primary output.

REMARK 1. *As has been suggested in previous papers [5, 4], the MUX cells connected to constant values can be simplified. But in our approach we do not make use of this, since - as will be shown later - this "destroys" the testability.*

2.3 Fault Models

We consider a static and a dynamic fault model, i.e. the *Stuck-At Fault Model* (SAFM) [7] and the *Path-Delay Fault Model* (PDFM) [17].

A fault in SAFM causes exactly one input or output pin of a node in the circuit to have a fixed constant value (0 or 1) independently of the values applied to the inputs of the circuit.

In the PDFM it is checked whether the propagation delays of all paths in a given circuit are less than the system clock interval. For the detection of a path delay fault a pair of patterns (I_1, I_2) is required rather than a single pattern as in SAFM: The *initialization vector* I_1 is applied and all signals of the circuit are allowed to stabilize; then the *propagation vector* I_2 is applied and after the system clock interval the outputs of C are controlled.

DEFINITION 1. *A two-pattern test is called a* robust test *for a path delay fault (RPDF test) on a path, if it detects that fault independently of all other delays in the circuit and all other delay faults not located on this path.*

It turns out that for the circuits considered in this paper the construction of tests with the following (even stronger) property is possible: For each path delay fault there exists a robust test (I_1, I_2) which sets all off-path inputs to non-controlling values on application of I_1 and remains stable during application of I_2, i.e. the values on the off-path inputs are not invalidated by hazards or races[2]. Robust tests with the properties mentioned above are also called *strong RPDF tests*. In the following we only use such tests, but for simplicity we call them RPDF tests, too. For a detailed classification of PDFs see [15].

3. BDD TRANSFORMATION

In this section we first describe the transformation how to derive a circuit from a given BDD description. Then, some properties of the resulting circuits are discussed. In the next sections testability properties regarding the SAFM and the PDFM are studied.

[2]A *controlling* value at the input of a node is the value that completely determines the value at the output, e.g. 1 (0) is the controlling value for *OR* (*AND*) and 0 (1) is the non-controlling value for *OR* (*AND*).

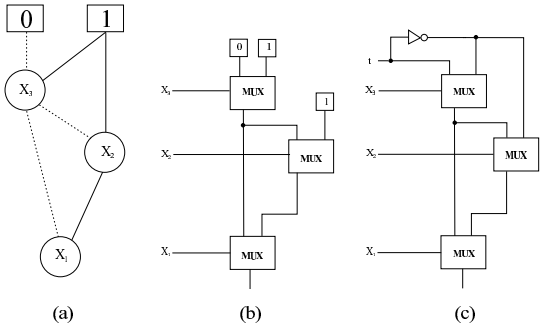

Figure 2: Example for transformation

Analogously to the "standard approach" from [5] the circuit is generated by traversing the BDD and substituting each node with a MUX cell. But, the methods differ when reaching nodes that have one or two pointers to terminal nodes. In this case, usually the MUX cell is simplified. E.g. if the 0-input is connected to constant 0, the MUX cell can be simplified and can be substituted by an AND gate.

Here, all nodes - also the ones pointing to terminals - are substituted by complete multiplexor cells. The terminal node 0 is then substituted by a new primary input t (=test). Furthermore, t is connected to the 1-terminal of the BDD by an inverter.

EXAMPLE 1. *In Figure 2(a) a BDD for function $f = x_1x_2 + x_3$ is shown. It is drawn upside down to underline the simialarities to the resulting circuit. The low-edges of the nodes are given as dotted lines. If the approach from [5] is applied, the BDD circuit in Figure 2(b) results (shown without simplification). While the transformation described above generates the circuit in Figure 2(c).*

REMARK 2. *It is important to notice that for mutliplexor based design styles, like e.g. PTL, the "simplification" of the MUX cell does not really imply savings in area or delay, since the complete multiplexor cell is often easier to realize.*

If t is set to constant 0, the circuit computes the original function. If t is set to 1, the complement is computed. It is important to observe, that by changing the value of t all "internal" signals, i.e. signals corresponding to edges in the BDD, change their value. This can be seen as follows:

$$\begin{aligned} \overline{f} &= \overline{\overline{x}_i g + x_i h} \\ &= (\overline{\overline{x}_i g})(\overline{x_i h}) \\ &= (x_i + \overline{g})(\overline{x}_i + \overline{h}) \\ &= \overline{x}_i \overline{g} + x_i \overline{h} \end{aligned}$$

Applying this recursively to the BDD or the BDD circuit, respectively, shows that all signals change their value. This guarantees a simple application of the values needed at the fault location.

Furthermore, the propagation of the faulty behavior to an output has to be ensured. This is one of the reasons, why circuits based on multiplexor cells became very popular, since due to the control input a propagating path can easily be generated. Thus, the propagation of a value from a fault location is no problem and the only thing to be done is to apply the value that shows the faulty behavior.

In previous approaches (see e.g. [5, 4]), modifications of the circuit were described, but these change the multiplexor structure and by this also destroy the propagation properties of multiplexors.

Table 1: Classical approaches

name	*in*	*out*	original			optimized		
			lits	NoP	PDFC	lits	NoP	PDFC
5xp1	7	10	391	296	98.4	158	1071	19.9
C17	5	2	23	11	100.0	10	11	100.0
alu2	10	6	909	69521	1.0	415	61362	0.9
b9	41	21	408	301	93.3	171	410	89.0
clip	9	5	1026	888	90.4	273	571	74.3
con1	7	2	48	23	100.0	21	22	100.0
count	35	16	338	368	100.0	159	384	100.0
i1	25	13	118	97	97.4	50	98	77.5
i5	133	66	689	883	100.0	198	672	100.0
t481	16	1	2673	4752	100.0	532	2405	89.5
tcon	17	16	90	56	85.7	32	40	100.0
9sym	9	1	655	522	96.5	333	518	91.1
f51m	8	8	409	319	98.2	155	30950	0.7
z4ml	7	4	325	252	100.0	46	368	25.2
x2	10	7	97	90	74.4	51	79	81.0

3.1 Stuck-At Fault Model

As has been observed in [5] stuck-at redundancies in a mapped BDD circuit can only occur if one of the values 01 or 10 is not applicable to the data inputs of this cell. On the other hand, a straightforward computation shows that at least one of the values is applicable. But due to the properties of the new input t, i.e. all internal signals change their value, the missing value can be applied by changing the value at t. We obtain:

THEOREM 1. *By one additional input and one inverter a circuit can be generated from a BDD that is 100% testable for single stuck-at faults.*

For the generation of a test, the efficient polynomial synthesis operations on BDDs can be used [8]. For each multiplexor cell the set of applicable values can easily be computed by carrying out AND-operations on the corresponding BDD nodes. The propagating path can be determined by a linear time graph traversal.

LEMMA 1. *In the resulting circuits, test pattern generation for stuck-at faults can be carried out in polynomial time.*

3.2 Path-Delay Fault Model

The same arguments as given above also ensure that all paths are testable under the PDFM. At each cell the values 10 or 01 can be applied (dependent on t). Thus, the paths starting at an input corresponding to the variable x_i can be propagated along any of the two AND-gates (see Figure 1). Furthermore, due to the propagation along the multiplexors, it is easy to see that the paths starting at t can be tested. We obtain:

THEOREM 2. *By one additional input and one inverter a circuit can be generated from a BDD that is 100% testable for robust path-delay faults.*

After the applicable values have been determined based on BDD operations, two patterns for a robust test can be determined by a traversal of the circuit in linear time.

LEMMA 2. *In the resulting circuits, test pattern generation for path-delay faults can be carried out in polynomial time.*

3.3 Partial Simplification

As has been observed in Remark 1 the simplification of the MUX cells can destroy the testability. But not all types of simplifications have this property, i.e. if both data inputs have constant values the MUX cell can be substituted by a simple wire or an inverter. Dependent on the design style this should be preferred.

4. EXPERIMENTAL RESULTS

The technique described above has been implemented as the tool MuTaTe (=*Multiplexor Transformation for Testabilty*). The program is implemented in C and all experiments have been carried out on a SUN Sparc 20 with 64 MByte of main memory. For the experiments we used some of the benchmarks from LGSynth91 [19]. As the underlying BDD package CUDD has been used [18]. Due to page limitation we restrict ourselves to a study of the *PDF coverage* (PDFC) of the circuits[3]. For each circuit we report the number of literals (measured using SIS [16]), the number of paths (NoP) that have to be tested and the PDFC in percent. Of course, the NoP can become a crucial factor, since a large number results in high test costs.

In Table 1 the name of the benchmark is given in the first column followed by the number of inputs and outputs in column two and three, respectively. The number of literals, the number of paths and the PDFC are given in column *lits*, *NoP* and *PDFC*, respectively. Column *original* gives the number for the benchmark as it is given in the description. Column *optimized* gives the numbers for the circuits that have been optimized by SIS using script *rugged*. As can be seen, the PDFC varies a lot. While some circuits have a testability of 100%, for others only one percent of the paths (or even less) are robustly testable. It is also important to notice that the optimization techniques used can result in a large number of untestable paths although the original circuit was very well testable. Consider e.g. circuit *f51m* as the most obvious example. Even though the original circuit had a PDFC of 98.2%, the coverage of the optimized netlist is less than 1%.

In a second series of experiments we study the PDF testability of BDD circuits. The results are given in Table 2 and 3 for BDDs with and without optimization of the variable ordering, respectively. In column MUX-map the results are given for a direct mapping of BDDs with simplification of the constant values as described in [5]. As has been observed in Remark 1, the "full simplification" can result in untestable paths. But already in this case the resulting BDD circuits have a significant better testability than the ones generated by SIS (see above), i.e. always more than 60%.

The results for the new approach are given in the next two blocks. Column MuTaTe gives the results for a direct mapping, i.e. the new test input is connected to each constant input to a MUX cell, while MuTaTe-S performs the simplifications described in Section 3.3[4].

As can be seen in both cases 100% PDFC is ensured. As is well known the size of a BDD (and by this of the resulting BDD circuit) largely depends on the chosen variable ordering. Comparing the literal count of the final circuits in Table 3, i.e. the size optimized BDDs, with those of SIS in Table 1, it can be seen that the synthesis methods are somehow "orthogonal". For several circuits the sizes are comparable, while in some cases SIS is significantly better (see e.g. *b9*), while for others BDDs are better suited. E.g. for *t481* the BDD circuit generated by MuTaTe-S is seven times smaller than the corresponding circuit produced by SIS. Furthermore, the synthesis scenario considered in our experiments is to be seen as "worst case" for BDD circuits (cf. Remark 2), since all cells are mapped to basic gates. For MUX oriented design styles the reduction in size can be expected to be even larger.

5. CONCLUSIONS

A new approach to generate multiplexor circuits from an initial BDD description has been described. The resulting

[3] In a preliminary version of this paper also experiments for the SAFM have been reported [10].

[4] In some rare cases (e.g. *tcon*) this reduction also removed the additional input. In these cases 100% PDFC is ensured while no additional input is needed.

Table 2: Path delay fault coverage of BDD circuits

name	*MUX-map*			*MuTaTe*			*MuTaTe-S*		
	lits	NoP	PDFC	lits	NoP	PDFC	lits	NoP	PDFC
5xp1	273	273	89.0	320	574	100.0	308	364	100.0
C17	26	22	68.1	41	44	100.0	29	26	100.0
alu2	652	873	86.9	718	1713	100.0	698	984	100.0
b9	609	1773	64.6	768	3429	100.0	700	2370	100.0
clip	603	954	79.4	667	1862	100.0	655	1130	100.0
con1	53	47	74.4	77	95	100.0	61	59	100.0
count	832	2248	66.1	928	4072	100.0	864	2704	100.0
i1	157	137	74.4	230	295	100.0	186	184	100.0
i5	2659	44198	61.3	3032	81381	100.0	2768	51345	100.0
t481	301	4518	86.1	328	8671	100.0	312	5473	100.0
tcon	32	40	100.0	96	112	100.0	32	40	100.0
9sym	84	328	72.5	97	658	100.0	93	490	100.0
f51m	239	326	99.3	262	668	100.0	242	332	100.0
z4ml	75	175	77.1	92	358	100.0	84	232	100.0
x2	147	188	72.3	183	379	100.0	171	244	100.0

Table 3: Path delay fault coverage of optimized BDD circuits

name	*MUX-map*			*MuTaTe*			*MuTaTe-S*		
	lits	NoP	PDFC	lits	NoP	PDFC	lits	NoP	PDFC
5xp1	131	218	83.0	168	463	100.0	160	337	100.0
C17	21	18	66.6	29	35	100.0	25	23	100.0
alu2	673	749	84.5	746	1446	100.0	730	867	100.0
b9	374	870	66.7	520	1707	100.0	468	1227	100.0
clip	353	764	76.4	391	1466	100.0	379	938	100.0
con1	38	28	85.7	61	59	100.0	45	35	100.0
count	221	624	58.9	320	1024	100.0	252	880	100.0
i1	141	111	71.1	194	226	100.0	170	142	100.0
i5	341	1102	67.6	552	2301	100.0	540	1998	100.0
t481	72	4065	74.6	80	7201	100.0	76	4996	100.0
tcon	32	40	100.0	96	112	100.0	32	40	100.0
9sym	84	328	72.5	97	658	100.0	93	490	100.0
f51m	133	236	83.8	158	494	100.0	146	314	100.0
z4ml	51	169	72.7	64	346	100.0	60	238	100.0
x2	87	80	73.7	123	160	100.0	111	112	100.0

circuits are fully testable under the stuck-at fault model and under the (robust) path-delay fault model. The transformation only needs one extra input and one inverter. The algorithm has been implemented as the program MuTaTe. Experimental studies have demonstrated the advantages of the approach.

It is focus of current work to study the stuck-at testability of the generated circuits in more detail, i.e. to develop reordering heuristics that generate BDD circuits of small size with good random pattern testability.

6. REFERENCES

[1] S.B. Akers. Binary decision diagrams. *IEEE Trans. on Comp.*, 27:509–516, 1978.

[2] P. Ashar, S. Devadas, and K. Keutzer. Gate-delay-fault testability properties of multiplexor-based networks. In *Int'l Test Conf.*, pages 887–896, 1991.

[3] P. Ashar, S. Devadas, and K. Keutzer. Testability properties of multilevel logic networks derived from binary decision diagrams. *Advanced Research in VLSI: UC Santa Cruz*, pages 33–54, 1991.

[4] P. Ashar, S. Devadas, and K. Keutzer. Path-delay-fault testability properties of multiplexor-based networks. *INTEGRATION, the VLSI Jour.*, 15(1):1–23, 1993.

[5] B. Becker. Synthesis for testability: Binary decision diagrams. In *Symp. on Theoretical Aspects of Comp. Science*, volume 577 of *LNCS*, pages 501–512. Springer Verlag, 1992.

[6] B. Becker. Testing with decision diagrams. *INTEGRATION, the VLSI Jour.*, 26:5–20, 1998.

[7] M.A. Breuer and A.D. Friedman. *Diagnosis & reliable design of digital systems*. Computer Science Press, 1976.

[8] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[9] R.E. Bryant. Binary decision diagrams and beyond: Enabeling techniques for formal verification. In *Int'l Conf. on CAD*, pages 236–243, 1995.

[10] R. Drechsler. MuTaTe: An efficient design for testability technique for multiplexor based circuits. In *GI/ITG/GMM-Workshop "Testmethods and Reliability of Circuits and Systems"*, 2003.

[11] W. Günther and R. Drechsler. ACTion: Combining logic synthesis and technology mapping for MUX based FPGAs. *Journal of Systems Architecture*, 46(14):1321–1334, 2000.

[12] C.Y. Lee. Representation of switching circuits by binary decision diagrams. *Bell System Technical Jour.*, 38:985–999, 1959.

[13] L. Macchiarulo, L. Benini, and E. Macii. On-the-fly layout generation for PTL macrocells. In *Design, Automation and Test in Europe*, pages 546–551, 2001.

[14] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, and S. Long. Wave steering in YADDs: A novel non-iterative synthesis and layout technique. In *Design Automation Conf.*, pages 466–471, 1999.

[15] A.K. Pramanick and S.M. Reddy. On the design of path delay fault testable combinational circuits. In *Int'l Symp. on Fault-Tolerant Comp.*, pages 374–381, 1990.

[16] E. Sentovich, K. Singh, L. Lavagno, Ch. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley, 1992.

[17] G.L. Smith. Model for delay faults based upon paths. In *Int'l Test Conf.*, pages 342–349, 1985.

[18] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. University of Colorado at Boulder, 1998.

[19] S. Yang. Logic synthesis and optimization benchmarks user guide. Technical Report 1/95, Microelectronic Center of North Carolina, 1991.