Bridging Fault Testability of BDD Circuits

Junhao Shi Görschwin Fey Rolf Drechsler

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{junhao,fey,drechsle}@informatik.uni-bremen.de

Abstract— In this paper we study the testability of circuits derived from Binary Decision Diagrams (BDDs) under the bridging fault model. It is shown that testability can be formulated in terms of symbolic BDD operations. By this, test pattern generation can be carried out in polynomial time. A technique to improve testability is presented. Experimental results show that a complete classification can be carried out very efficiently.

I. Introduction

While classical approaches to logic synthesis first considered a technology independent optimization followed by a mapping to a target architecture, modern design flows try to combine the different levels. As a unifying data structure *Binary Decision Diagrams* (BDDs) have been proposed, since they allow abstract function representation, but can also be directly mapped to netlists using the one-to-one correspondence between the Shannon decomposition and MUX networks [6]. This allows to efficiently realize the resulting BDD circuits in multiplexor based design styles like e.g. *Pass Transistor Logic* (PTL) [11].

On BDD circuits and BDDs, respectively, many operations can be carried out efficiently, like power estimation or considering layout aspects. The testability of BDD circuits has intensively been studied under various fault models, like stuck-at, cellular, or gate and path delay [3, 2]. Recently, in [7] a technique has been proposed that ensures full testability under the stuck-at fault model and path delay fault model.

Although the stuck-at fault model and path delay fault model are the standard fault model, the frequently occurring faults in some technologies are unintentional shorts, denoted as bridging faults [1, 8]. So increasing attention has been given to the area of modeling and testing of bridging faults [9].

In this paper, BDD circuits are studied and their bridging fault testability is analyzed. For circuits derived from BDDs the complete test pattern generation process for bridging faults is formulated in terms of symbolic BDD manipulations. These symbolic operations allow to consider all possible test patterns at the same time. This can be beneficial for compaction of a test pattern set. Test pattern generation has been proven to be a difficult problem and tools for test pattern generation have exponential worst-case behavior. But for BDD circuits *polynomial upper bounds* are proven in this paper, and it is shown by experiments that the technique from [7] also improves the testability for bridging faults.

II. Preliminaries

A. Binary Decision Diagrams and Circuits

As is well-known a Boolean function $f : B^n \rightarrow B$ can be represented by a BDD which is a directed acyclic graph. The Shannon decomposition

$$\begin{aligned} f &= \overline{x}_i f_{\overline{x}_i} + x_i f_{x_i} \quad (1 \leq i \leq n) \\ f_{\overline{x}_i} &= f(x_1, \cdots, x_i = 0, \cdots, x_n) \\ f_{x_i} &= f(x_1, \cdots, x_i = 1, \cdots, x_n) \end{aligned}$$

is carried out in each non-terminal node v with the low-edge pointing to $f_{\overline{x}_i}$ and the high-edge pointing to f_{x_i}. This node is labeled with variable $label(v) = x_i$. The terminal nodes denote the constant Boolean functions 1 and 0 and are labeled by 1 and 0, respectively. 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. In the following the order $(x_1, x_2, \ldots x_n)$ is considered. Based on this variable order a level is assigned to each node v, where

$$level(v) = \begin{cases} i & \text{if } v \text{ is a non-terminal node} \\ & \text{and } label(v) = x_i \\ n+1 & \text{if } v \text{ is a terminal node} \end{cases}$$

A node with $level(v) = 1$ is on the *highest level*, while terminal nodes are on the *lowest level*.

A BDD is called *reduced* if it does not contain isomorphic subgraphs nor vertices with both edges pointing to the same node. Reduced and ordered BDDs are a canonical representation since for each Boolean function the BDD is uniquely specified. For functions represented by BDDs efficient manipulations can be carried out [5].

B. 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. As proposed in [3], we consider two libraries in the following (see Figure 1):

1. MUX: BDD nodes are substituted by multiplexor cells. Internal signals of these cells are not considered.
2. STD: BDD nodes are substituted by the AND-, OR-, NOT-realization of the Shannon decomposition.

Remark 1. *For BDD circuits the library STD can be considered as a worst-case. The realization of the circuit is much more efficient in multiplexor based design styles as e.g.* Pass Transistor Logic.

The number of multiplexors in a BDD circuit C is denoted by $|C|$ and considered as the size of the circuit. A line g that is an output of a multiplexor in the BDD circuit corresponds to a BDD node or an edge in the BDD. For simplicity the BDD node and the edge are also denoted by g. The function of such a line in terms of primary inputs is denoted by $f(g)$. For any line h in the circuit $m(h)$ denotes the multiplexor (BDD node, edge), where h belongs to.

C. Stuck-At Fault Model

In some situations the reduction of bridging faults to stuck-at faults is useful. A fault in the *Stuck-At Fault Model* (SAFM) [4] 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. This is denoted by s-a-0 or s-a-1, respectively.

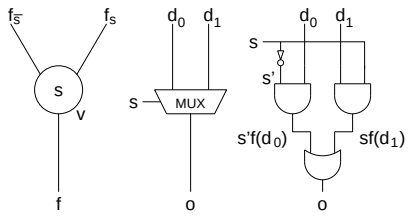


Fig. 1. BDD node, over MUX and STD

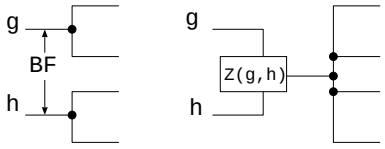


Fig. 2. Bridging Fault

D. Bridging Fault Model

Bridging Faults (BFs) [12] are caused by shorts between normally unconnected signal lines [10]. Since the lines involved in a short become equipotential, all of them have the same logic value. For a short line g, we distinguish between the value one could actually observe and the value of g as determined by its source element; the latter is called driven value. Figure 2 shows a general model of a BF between two lines g and h. $Z(g,h)$ is the function introduced by the BF and has the property $Z(g,g) = g$. The fan-out of Z is the union of the fan-outs of the shorted signals. Note that the values of g and h in this model are their driven value, but these are not observable in the circuit. When the two shorted lines have opposite driven values, one value (the strong one) overrides the other. If 0 is the strong value, then $Z(0,1) = 0$, and the function introduced by the BF is AND. The fault is named AND BF. In this paper we consider only AND BFs. The results for OR BFs can be obtained by similar argumentation (the strong value is 1). Multiple BFs are detected by tests for single BFs [12]. Therefore only single BFs are considered in this paper.

For convenience other functions are defined. In the following $f^{\text{detect}}(g,h)$ denotes the conditions to set line g to the value 0 and h to the value 1 at the same time in the circuit without fault. Note, that $f^{\text{detect}}(g,h) \neq f^{\text{detect}}(h,g)$. The condition necessary to propagate the value of a line g to a primary output o is denoted by $f_o^{\text{propagate}}(g)$. Under the assumption that the fault does not create feedback loops or destroy the propagating path, the function $f^{\text{detect}}(h,g) \cdot f_o^{\text{propagate}}(g)$ represents all test patterns to observe the AND bridging fault between g and h at output o. The problems with feedback loops and propagation are discussed below.

III. Bridging Faults in BDD Circuits

To detect a BF between two lines g and h opposite logic values are assigned to g and h and the lines are observed. If g and h have equal values a BF between them has been detected.

In the following the complete test pattern generation for BFs of BDD circuits is shown. Polynomial upper bounds for a complete classification are proven. Intuitively the proof relies on the following argumentation: The size of the BDD circuit equals the size of the BDD. Functions f^{detect} and $f^{\text{propagate}}$ for lines in a BDD circuit can be calculated by synthesis operations on the BDD. Also additional conditions to prevent feedback loops are determined using BDD operations. The conjunction of all conditions returns all possible input assignments to detect the single BF between the two lines.

paths(g, h**)**
(0) **if** ($g == h$) **return** 1;
(1) **if** ($level(g) \leq level(h)$) **return** 0;
(2) **if** (cacheLookup(g, h)) **return** cacheLookup(g, h);
(3) i= $level(h)$;
(4) A= paths(g, h_{x_i});
(5) B= paths(g, $h_{\overline{x}_i}$);
(6) v= createNewNode(x_i, A, B);
(7) cacheInsert(g, h, v);
(8) **return** v;

Fig. 3. Calculation of paths

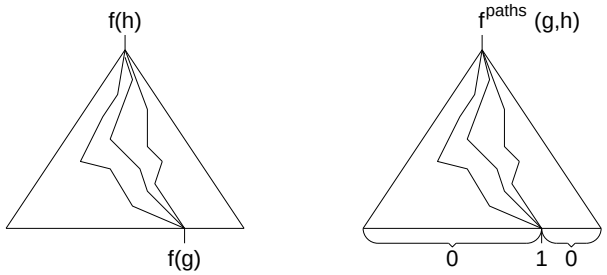


Fig. 4. Example for the calculations of paths

A. Fault Detection

Consider a line in a BDD circuit over STD. This line directly corresponds to (1) the output of a multiplexor, (2) a select input of a multiplexor, (3) or an internal line of a multiplexor. In the first two cases the function of this line with respect to the primary inputs is directly given by the corresponding BDD node or the projection function of the select variable, respectively. If the line is an internal line of a multiplexor, the function can be calculated using at most two synthesis operations on a BDD. The functions of internal lines are annotated in Figure 1.

Lemma 1. *The function $f^{detect}(g,h)$ for any two lines g and h in a BDD circuit C over MUX or STD can be calculated in polynomial time.*

So far only the circuit without a fault has been considered to adjust the values for fault detection. This is correct as long as the faulty value on g does not influence the value on h (or vice versa). The value of h is only influenced by g, if the current assignment to the primary inputs selects a path in the BDD from g to h. If g and h are both outputs of multiplexors this can be checked directly on the BDD by the algorithm in Figure 3. The algorithm works in the recursive manner as other algorithms that calculate synthesis functions on BDDs [5]. Calling **paths**(g, h) substitutes all paths in the BDD of h that do not lead to g by zero and replaces g by one. A function is returned that represents all assignments that establish a path from node h to node g (see Figure 4). The algorithm runs in time $O(|C|)$ as in the worst case the whole BDD of h is traversed.

Let $f_h^{\text{paths}}(g)$ be the function returned by **paths**(g, h). This will be used in Section III.C to ensure that the expected values occur during test pattern generation.

B. Fault Propagation

Consider a line g in a BDD circuit that is an output of a multiplexor. The conditions to propagate the value of g to a primary output o are calculated by calling **paths**(g, o).

Lemma 2. *Given a circuit line g corresponding to the BDD node g and a primary output o corresponding to the BDD node o. Then, $f_o^{propagate}(g) = f_o^{paths}(g)$ for the corresponding BDD circuit C based on MUX.*

tpg(g, h)
(0) **foreach** primary output o {
(1) **if** ($level(m(g)) < level(m(h))$) exchange g, h;
(2) $t_1 = f^{\text{detect}}(g,h) \cdot f_o^{\text{propagate}}(h)$;
(3) **if** ($t_1 \neq 0$) **return** t_1;
(4) $t_2 = f^{\text{detect}}(h,g) \cdot \overline{f_{m(h)}^{paths}(m(g))} \cdot f_o^{\text{propagate}}(g)$;
(5) **if** ($t_2 \neq 0$) **return** t_2;
(6) }
(7) **return** 0;

Fig. 5. Test pattern generation

To propagate a line g that is a select input of a multiplexor the data inputs of the multiplexor have to be set to opposite values and the output of the multiplexor $m(g)$ has to be propagated.

Lemma 3. *Given a circuit line g corresponding to a select input of a multiplexor $m(g)$ with data inputs d_0 and d_1 and a primary output o corresponding to the BDD node o. Then, $f_o^{propagate}(g) = f_o^{paths}(m(g)) \cdot (f(d_0) \oplus f(d_1))$ for the corresponding BDD circuit C based on MUX.*

Now, consider a line g in a BDD circuit over STD. Additionally to the propagation of the value of g along lines corresponding to edges in the BDD further conditions are needed to propagate g to the output of the multiplexor. Consider the multiplexer shown in Figure 1. Similar to the propagation of s as considered above, the condition for all other lines can also be expressed using synthesis operations. This is summarized by the following lemma.

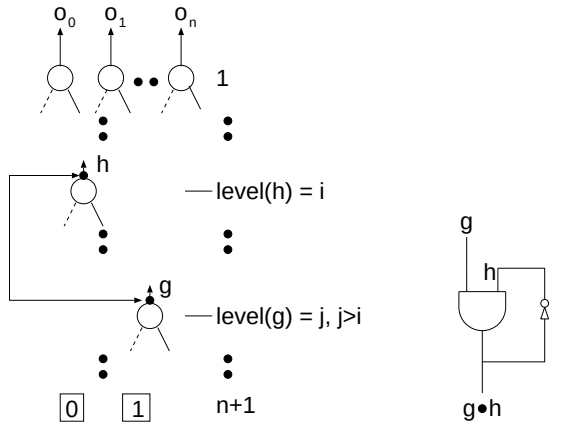
Lemma 4. *The condition $f_o^{propagate}(g)$ to propagate the value of line g to the primary output o in a BDD circuit over MUX or STD can be calculated in polynomial time.*

C. Test Pattern Generation

To calculate test patterns the previously calculated functions f^{detect}, $f^{\text{propagate}}$ and f^{paths} are used. Test pattern generation for an AND BF with respect to any two lines g and h that belong to different multiplexors in the BDD circuit is carried out by the algorithm in Figure 5, that also ensures that no feed-back loops with non-stable value assignments occur. This algorithm checks each primary output for a possible test pattern (line (0)). Then, line (1) assures that line h is at the higher or the same level as g afterwards as shown in Figure 6(a). Now, a feedback loop may occur due to a BF, if a path from g to h is selected. This loop can always be reduced to the circuit shown in Figure 6(b). Line h can not influence the value of g due to the structure of the BDD circuit. The inverter always occurs, because g and h are set to opposite values for test pattern generation. If the test pattern sets g to zero, h becomes also zero and is propagated. This possibility is checked by lines (2) and (3) of the algorithm. Otherwise g has to be set to one and h to zero to detect the fault. In this case selecting a feedback loop would lead to a non-stable value of h. This is avoided by the condition $\overline{f_{m(h)}^{\text{paths}}(m(g))}$ in line (4) of the algorithm. For a more detailed discussion on so called *inverting BFs* see [12].

The algorithm runs in polynomial time, because all of the intermediate steps run in polynomial time and the outer loop is carried out at most once for each primary output.

If both lines g and h belong to the same multiplexor additional checks for feedback loops have to be carried out. Also the test pattern generation for OR BFs is symmetric to that of AND BFs. Due to page limitation these cases are not shown.



(a) $level(g) > level(h)$ (b) Feedback loop

Fig. 6. Propagation

Theorem 1. *Test-pattern generation for a given BF between two lines in a BDD circuit over MUX or STD can be carried out in polynomial time.*

Besides showing the polynomial upper bounds for test pattern generation, the previous lemmas also provide the necessary steps to calculate test patterns for a BF in a BDD circuit.

IV. Testability and Redundancies

In the previous section test pattern generation for BDD circuits was described. Now, the different types of BDD circuits are considered in more detail and the possibility of redundancies is analyzed. The testability issues are different for BDD circuits over MUX and STD. Additionally the testability can be increased, if the direct mapping of BDDs to multiplexors is enhanced by the technique *MuTaTe* introduced in [7].

A. BDD Circuits over MUX or STD

BDD circuits over MUX only contain lines corresponding to nodes or to primary inputs. For two lines g and h that correspond to outputs of multiplexors there is always an assignment which sets those lines to different values. Otherwise the corresponding BDD node would have been reduced. Therefore a BF between those lines is redundant, if propagation of the stuck-at fault on one line fails. This can only be due to the restriction of input values for fault detection. Propagation can be blocked in two situations:

1. $level(g) > level(h)$ and $f^{\text{detect}}(g,h) \neq 0$.
   This can not happen (see Figure 6(a)). In this case variables on a path from h to the outputs are not in the support of f^{detect}. Propagation is always possible.
2. $level(g) > level(h)$ and $f^{\text{detect}}(h,g) \neq 0$.
   This case may occur and leads to a redundant BF if $f^{\text{detect}}(g,h) = 0$.

A fault between a primary input i and another line g in a circuit over MUX may also be redundant if fault propagation is prevented by fault detection. Additionally assigning values to two primary inputs i_1, i_2 may prevent the possibility of setting the lines to opposite values, i.e. $f^{\text{detect}}(i_1,i_2) = f^{\text{detect}}(i_2,i_1) = 0$.

Remark 2. *The same problems arise for circuits over STD. Additionally due to the structure of the realization of a MUX more feedback loops may result from BFs leading to redundancies.*

B. Increased Testability

In [7] the technique *MuTaTe* was introduced to retrieve BDD circuits that are 100% testable under the stuck-at fault model and the path delay fault model. Analogously to the "standard approach" the circuit is generated by traversing the BDD and substituting each node with a multiplexor cell. All nodes - also the ones pointing to terminals - are substituted by complete multiplexor cells. The terminal node 0 is then replaced by a new primary input t (=test). Furthermore, t is connected to the 1-terminal of the BDD by an inverter.

In case of BFs this technique can not remove all redundancies when internal lines of multiplexors are considered. Cases where a given pair of lines can not be set to opposite values remain. But propagation becomes more easy. By switching the test input t all lines corresponding to edges of the BDD also switch their value. This can be used to induce the stuck-at fault in the multiplexor at the higher level instead of the lower one. Thus, detection can no longer block propagation.

Lemma 5. *In a BDD circuit over MUX generated according to MuTaTe all BFs between two outputs of multiplexors are testable.*

Proofs are left out due to page limitation. For details about *MuTaTe* we refer the reader to [7].

V. Experimental Results

The algorithms described above have been implemented in C. All experiments were carried out on a SUN Fire 280R with 3 Gbyte of main memory. The benchmarks are taken from LGSynth93. For each circuit initially the BDD is constructed and then mapped to the libraries MUX and STD. The technique from [7] was applied to increase the testability of the circuits.

Results are reported in Table I. The first three columns show the name of the circuit, the number of inputs and outputs. Column *cells* reports the number of multiplexors needed to realize the circuit, this is equal to the number of nodes in the BDD. The next three columns show the testability of MUX-based circuits. In the first two columns the number of non-testable (*und.*) and testable (*det.*) BFs is reported, if the circuit is mapped using the method from [3] (BDD mapping). The next column shows the number of BFs that remain undetected, if the approach from [7] (*MuTaTe*) is used. For MUX based circuits only BFs between outputs of multiplexors, but no BFs containing select lines are considered. The additional test input allows to always switch the output values of multiplexors, such that propagation becomes possible. According to Lemma 5 no redundant faults between outputs of multiplexors remain.

Next, the testability of STD-based circuits is considered, i.e. data for signals inside the multiplexors is given. According to the observations in Section IV.B, the circuits derived by *MuTaTe* contain significantly less redundancies.

The CPU seconds for the ATPG process for [7] for all possible bridging faults between two lines is reported in the last column. All internal lines of multiplexors are considered. The times are quite small. Even classifying almost 40 million test patterns for *i9* takes less than CPU 10 minutes. Often less than one CPU second is needed to generate all test patterns.

These results show the efficiency of the technique for BDD circuits.

TABLE I
Test pattern generation for benchmark circuits

				MUX-based			STD-based			
				[3]		[7]	[3]		[7]	
circuit	in	out	cell	und.	det.	und.	und.	det.	und.	time
alu2	10	6	180	15	22254	0	2032	198555	0	1.34
b12	15	9	100	35	4867	0	1618	55146	176	0.31
b9	41	21	191	140	16680	0	3541	212237	1	2.02
clip	9	5	167	48	19914	0	6348	166156	1983	1.30
cmb	16	4	36	100	419	0	1945	5302	46	0.07
comp	32	3	173	200	23201	0	8518	209447	173	5.13
con1	7	2	20	7	189	0	184	1898	23	0.01
count	35	16	233	16	24636	0	5160	315620	3060	2.91
cu	14	11	49	2	843	0	233	12158	18	0.07
decod	5	16	38	1	344	0	320	5125	28	0.03
duke2	22	29	553	40	158422	0	13691	1829887	179	17.25
e64	65	65	1928	0	1789974	0	356516	22990228	3583	762.07
f51m	8	8	66	0	2792	0	217	25043	26	0.14
frg1	28	3	531	676	152433	0	51868	1680537	1186	39.68
i1	25	13	57	29	1040	0	452	17331	37	0.08
i9	88	63	2418	364	3171225	0	98203	34290834	6415	529.11
misex2	25	18	134	21	7765	0	5828	94902	584	0.71
o64	130	1	131	64	8192	0	75	109059	1	7.85
parity	16	1	17	0	420	0	211	2429	0	0.03
rd84	8	4	42	3	1491	0	586	11188	60	0.08
seq	41	35	2471	1143	3173053	0	392144	37455608	2443	844.50
t481	16	1	83	6	5228	0	1190	45648	193	0.64
table5	17	15	1313	241	1010477	0	257238	10430034	1599	212.49
x1	51	35	660	533	222372	0	20473	2735556	0	31.65

References

[1] B. Arslan and A. Orailoglu. Extracting precise diagnosis of bridging faults from stuck-at fault information. In *Asian Test Symp.*, pages 290–293, 2003.

[2] 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.

[3] 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.

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

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

[6] R. Drechsler and W. Günther. *Towards One-Path Synthesis*. Kluwer Academic Publishers, 2002.

[7] R. Drechsler, J. Shi, and G. Fey. Synthesis of fully testable circuits from BDDs. *IEEE Trans. on CAD*, 23(3):440–443, 2004.

[8] D. Feltham and W.Maly. Physically realistic fault models for analog CMOS neural networks. *IEEE Journal of Solid-State Circuits*, 26(9):1223–1229, September, 1991.

[9] S. Ma, I. Shaik, and R. S. Fetherston. A comparison of bridging fault simulation methods. In *Int'l Test Conf.*, pages 587–595, 1999.

[10] M.Abramovici and P.R.Menon. A practical approach to fault simulation and test generation for bridging faults. *IEEE Trans. on Comp.*, C-34(7):658–663, 1985.

[11] 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.

[12] K.C.Y. Mei. Bridging and stuck-at faults. *IEEE Trans. on Comp.*, C-32(7):720–727, 1974.