Synthesis of Fully Testable High Speed Circuits Derived from Decision Diagrams

Technical Report 95

Harry Hengster* Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
Freiburg i.Br., Germany
email: <hengster,becker>@informatik.uni-freiburg.de

Abstract

We present an approach to synthesize fully stuck-at fault testable circuits along with complete test sets. Starting from functional descriptions given by Kronecker Functional Decision Diagrams *(KFDDs) circuits with inherently small delay are constructed by a composition method based on* Boolean Matrix Multiplication. *During the construction efficient algorithms working on the KFDD are used to avoid the creation of* constant *lines. Furthermore, in each composition step tests for the faults at the newly generated lines are computed on the fly. Experimental results are given to underline the efficiency of the methods.*

*Supported by DFG grant Be 1176/8-2.

1 Introduction

Decision Diagrams (DDs) are a well-known graph based data structure for the representation and manipulation of Boolean functions and in the meantime are used in many modern design automation tools. On the other hand, the structure of the underlying graph can also be used to derive circuit realizations for the represented functions.

These so-called DD-based synthesis approaches can be subdivided in two major classes: One method is to substitute each node of the DD by a subcircuit realizing the corresponding decomposition type (see e.g. [1, 2, 3, 4, 5]). In general, these circuits have nice testability properties, but their major drawback is a delay proportional to the number of primary inputs.

An alternative method to derive circuits from DDs was presented in [6, 7]. Dependencies among the nodes of a DD are analyzed and a circuit is created, which combines these dependencies by *Boolean Matrix Multiplication* (BMM). This approach inherently produces circuits with small delay. Thus, it is more appropriate for modern VLSI systems, which are mostly designed to operate at high clock frequency. Additionally, it was shown in [6] for the restricted case of *Binary DDs* (BDDs) [8] that the circuits are fully testable with respect to stuck-at faults and path-delay faults as long as *2-rail* logic is used, i.e. a potentially large area overhead which at least requires doubling the number of primary inputs and primary outputs has to be taken into account. In [7] the construction idea from [6] was generalized to *Kronecker Functional DDs* (KFDDs) [9], a DD type including BDDs [8] and FDDs [10] as proper subsets and thus allowing a more compact representation of Boolean functions. For the circuits derived from KFDDs 2-rail logic no longer guarantees full testability.

In this paper we develop a synthesis for testability method that allows the construction of circuits with small delay starting from KFDDs as in [7] and at the same time guarantees full stuck-at fault testability by construction. This is done without using 2-rail logic and thus avoids additional area consumption. On the contrary, full testability relies on the fundamental property that irredundancy is guaranteed for the considered circuits, as long as *constant* lines are avoided. (A constant line computes the same value for every assignment at the primary inputs.) A method is presented to avoid the construction of these lines during the synthesis process by an efficient algorithm based on the graph of the KFDD. Moreover, we show that tests for the faults can also be derived efficiently from the graph of the KFDD. Thus, it is not necessary to apply *automatic test pattern generation* (ATPG) to compute test sets for the synthesized circuits or to check for redundancies. Altogether, our approach makes it possible to combine high speed with full testability for circuits derived from KFDDs.

The paper is structured as follows: In the next section we give some basic definitions and notations. In Section 3 an overview on the complete synthesis approach is given. The correctness of the method is shown in Section 4. Experimental results in Section 5 show the efficiency of the proposed methods.

2 Preliminaries

In general, a *combinational circuit* is defined over a fixed library. In this paper we consider a library consisting of *primary input* (PI), *primary output* (PO), the inverter NOT, and the AND, OR, and $EXOR$ gates.

In some technologies, like *Field Programmable Gate Arrays* (FPGAs) [11], all basic gates have the same structure and there is no extra cost for $EXOR$ gates compared to other gates like AND and OR. Therefore, the $EXOR$ gate will be considered as a basic gate type in this paper.

Next, we briefly review the essential definitions of *Decision Diagrams* (DDs). For more details see [8, 9, 12].

A DD is a graph based representation of Boolean functions over the set of variables $X_n = \{x_1, x_2, \ldots, x_n\}$. A non-terminal node v of a DD is labeled with a variable from X_n, called the *decision variable* for v, and has exactly two outgoing edges, called $\text{LOW}(v)$ and $\text{HIGH}(v)$. Edge $\text{LOW}(v)$ ($\text{HIGH}(v)$) leads to node $low(v)$ ($high(v)$). A terminal node v is labeled with 0 or 1 and has no successors. The set of nodes is divided into $n+1$ disjoint *levels*, denoted by $level_1, level_2, \ldots, level_n, level_{n+1}$. All non-terminal nodes labeled x_i belong to $level_i$ and the terminal nodes constitute $level_{n+1}$. Additionally, for each node in $level_i$ the *position* is given by a bijection $position_i : level_i \mapsto \{1, 2, \ldots, w_i\}$, where $w_i = |level_i|$ is the *width* of $level_i$. In the following a node v at level i with position p will be denoted by v_p^i.

To relate DDs to Boolean functions different *decomposition types* can be used. In a *Kronecker Functional Decision Diagram* (KFDD) *Shannon* (S), *positive Davio* (pD), or *negative Davio* (nD) decomposition can be chosen for all nodes in one level. The decomposition types for each level are given by a *Decomposition Type List* (DTL) $d := (d_1, d_2, \ldots, d_n)$, where $d_i \in \{S, pD, nD\}$.

Formally, KFDDs and their relation to Boolean functions can be defined as follows:

Definition 1 *A KFDD over X_n is given by a graph together with a fixed DTL $d := (d_1, \ldots, d_n)$. If v is a node at level i with label x then the function $f_v : \mathbf{B}^n \mapsto \mathbf{B}$ represented by node v is defined as:*

$$f_v = \begin{cases} \text{constant function } x & : \quad i = n+1 \\ \overline{x} f_{low(v)} + x f_{high(v)} & : \quad i \leq n \ \wedge \ d_i = S \\ f_{low(v)} \oplus x f_{high(v)} & : \quad i \leq n \ \wedge \ d_i = pD \\ f_{low(v)} \oplus \overline{x} f_{high(v)} & : \quad i \leq n \ \wedge \ d_i = nD \end{cases}$$

where $f_{low(v)}$ ($f_{high(v)}$) is the function represented by the node $low(v)$ ($high(v)$) and $\oplus$ denotes the `exclusive-or` *operation.*

It is known that *reductions* can be used to obtain *reduced KFDDs*. The size of a KFDD can be further reduced if *Complemented Edges* (CEs) are introduced. Notice, that we restrict to *ordered* KFDDs in this paper, i.e. KFDDs where the variables are encountered in the same order on each path from the root to a terminal node.

Example 1 In Figure 1 a graphical representation of a reduced KFDDs with CEs is given. Labels are given inside the nodes. For non-terminal nodes the LOW-edges (HIGH-edges) are represented by the left (right) outgoing edges. The positions are denoted next to the nodes. CEs are symbolized by dots on the edges. In this case one has to use the complement of the function represented by the node, which the edge points to. Each node represents a Boolean function, which is recursively obtained according to Definition 1 by the functions of the successor nodes and the decomposition type associated to the node by the DTL. The reduced KFDD in Figure 1 represents the function $f_{v_1^1} = \overline{x}_3\overline{x}_4 \oplus \overline{x}_1\overline{x}_2\overline{x}_4 \oplus \overline{x}_1 x_2 x_4$. ■

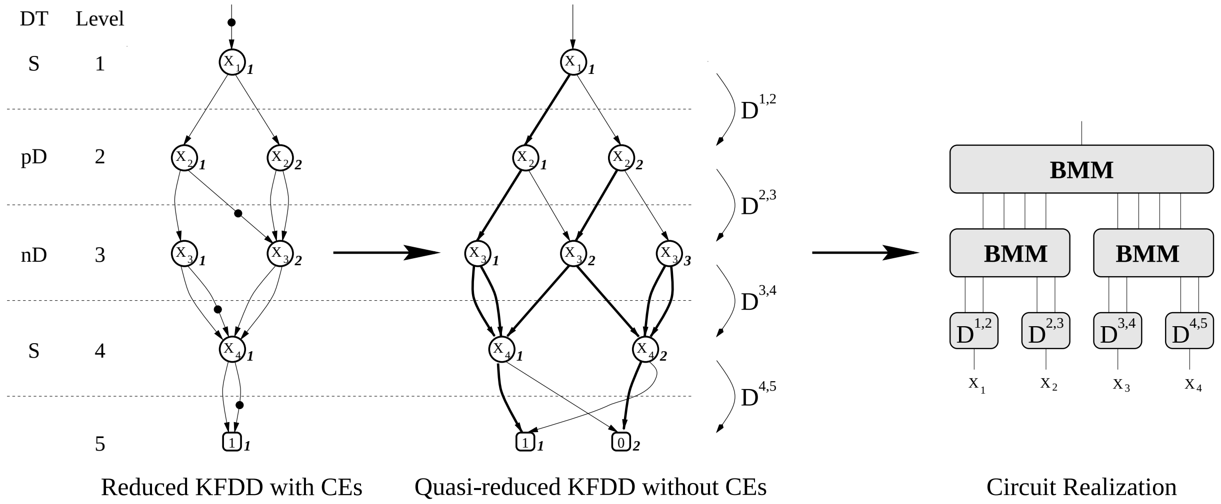


Figure 1: The concept of BMM transformation

The concept of *active* edges and paths is crucial for the discussions of the problems in this paper. Informally, an edge (v, v') starting at a node v decomposed by variable x_i is active for a given assignment $x_i = a_i$ if the value of function f_v for assignments with $x_i = a_i$ depends on the evaluation of the subfunction rooted at v'.

Definition 2 *Let v be a node at level i of a KFDD with DTL d and let $a = (a_1, a_2, \ldots, a_n)$ be an assignment of Boolean values to the variables $x_1, x_2, \ldots, x_n$.*

- Edge LOW(v) is active for a iff $a_i = 0 \wedge d_i = S$ or $d_i \in \{pD, nD\}$.
- Edge HIGH(v) is active for a iff $a_i = 1 \wedge d_i \in \{S, pD\}$ or $a_i = 0 \wedge d_i = nD$.
- Path $\pi = \langle e_1, e_2, \ldots, e_p \rangle$ is active for a iff all edges e_i $(i = 1, 2, \ldots, p)$ are active for a.

Example 2 In the quasi-reduced KFDD in Figure 1 the edges drawn with bold lines are active for the assignment $(0, 0, 0, 0)$. Thus there are two active paths from node v_1^1 to the terminal node v_1^5. ∎

3 The Synthesis for Testability Approach

In the following we present an approach to synthesize circuits from the graph of a KFDD based on the method presented in [7]. We point out where care has to be taken to avoid redundant faults and how tests can be derived from the KFDD during the synthesis process. A detailed discussion on the testability aspects and the correctness of the approach will be given in Section 4.

For the basic idea see Figure 1. Since most DD-packages are based on reduced DDs with CEs the first step is to transform the given KFDD into a *quasi-reduced* and *complete* KFDD without CEs. For details on this transformation see [7, 13]. Then the dependencies among nodes in successive levels l and $l + 1$ are analyzed to define *dependency matrices* $D^{l,l+1}$ and

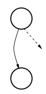
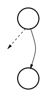
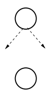
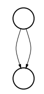
$d_{i,j}^{l,l+1}$				
Shannon	$\overline{x}_l$	x_l	0	1
pos. Davio	1	x_l	0	$\overline{x}_l$
neg. Davio	1	$\overline{x}_l$	0	x_l

Figure 2: Dependencies of nodes in successive levels

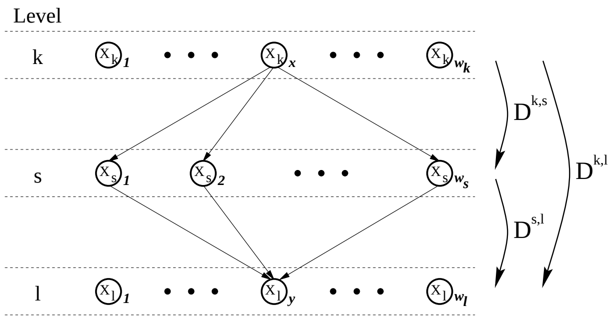


Figure 3: Composition by BMM

subcircuits realizing the dependency matrices are constructed. A subcircuit realizing *Boolean Matrix Multiplication* (BMM) is used to combine two subcircuits to a new subcircuit. Combining the subcircuits in a tree-like manner results in circuits with small delay.

The "direct" combination of two subcircuits may lead to constant lines resulting in redundant faults. In Section 4 we will present in detail an efficient method to avoid these redundancies by analysis of the graph of the KFDD. Moreover, a method is presented to incrementally derive tests for all faults in the newly created subcircuit. For the rest of this section we focus on the overall structure of our synthesis for testability approach.

3.1 BMM Transformation

The function represented by a node v of a KFDD evaluates to 1 for assignment a iff an odd number of paths from v to the terminal node for function 1 are active for a [12]. All assignments, for which an odd number of paths from node v_x^k to node v_y^l are active, are represented by the *dependency* $d_{x,y}^{k,l}$ of v_x^k upon v_y^l. Figure 2 shows how the dependency for any pair of nodes in successive levels and for any decomposition type of a KFDD is determined. The dependency matrix $D^{l,l+1}$ from level l to $l+1$ is formed by the dependencies $d_{i,j}^{l,l+1}$ for each pair of nodes $v_i^l \in level_l$ and $v_j^{l+1} \in level_{l+1}$. Note, that in each row of a dependency matrix $D^{l,l+1}$ at least all but two entries are 0 because every node has only two outgoing edges.

The next step is to define an operation $\circ$ to compose $D^{k,s}$ and $D^{s,l}$ to the dependency matrix $D^{k,l}$. Then, it is possible to obtain the dependency matrix $D^{1,n+1}$ by iterative application of the operator $\circ$. This dependency matrix includes a representation of the functions represented by the KFDD.

The dependency matrix $D^{k,l} = D^{k,s} \circ D^{s,l}$ can be computed by Equation (1) for each pair of nodes $v_x^k \in level_k$ and $v_y^l \in level_l$. This equation defines the condition that node v_y^l is reached from node v_x^k by an odd number of paths (cf. Figure 3):

$$d_{x,y}^{k,l} \quad = \quad d_{x,1}^{k,s} \cdot d_{1,y}^{s,l} \oplus d_{x,2}^{k,s} \cdot d_{2,y}^{s,l} \oplus \cdots \oplus d_{x,w_s}^{k,s} \cdot d_{w_s,y}^{s,l} = \bigoplus_{i=1}^{w_s} d_{x,i}^{k,s} \cdot d_{i,y}^{s,l} \qquad (1)$$

Since Equation (1) corresponds to BMM, if the `exclusive-or` operation is substituted by the `or` operation, $\circ$ is called the BMM-operation.

Evaluating $d_{x,y}^{k,l}$ by Equation (1) can result in a complex representation of a constant value. In this case a redundant line would be realized in the corresponding circuit realization. In Section 4 we present an efficient method based on the graph of the KFDD to detect constant dependencies. We will show that this method is sufficient to avoid any redundancies in the synthesized circuits.

3.2 Circuit Realization

As already mentioned many entries of the dependency matrices have the constant value 0 and some may have the value 1. This causes constant values at the inputs of several gates. In the circuit realization all gates with constant inputs are recursively simplified.

The elements of the dependency matrices of successive levels $D^{l,l+1}$ are x_l, $\overline{x}_l$, 0, and 1. A subcircuit to realize such a dependency matrix consists only of the PI for x_l, lines, and inverters.

Equation (1) directly leads to 2-level $AND/EXOR$ circuits, called *BMM-circuits*, which realize the BMM-operation. Since the BMM-operation $\circ$ is associative it is possible to combine the subcircuits as a balanced tree as shown in Figure 1. Thus, the functions represented by the KFDD can be realized by circuits with small delay.

During construction of a new subcircuit all tests for the faults in this new subcircuit can be derived from the graph of the KFDD. An efficient method is presented in the following section.

To summarize we give an outline of the synthesis for testability approach in Figure 4.

SYNTHESISFORTESTABILITY (function $f \in B^{n,m}$)
(1) compute quasi-reduced and complete KFDD G for f without CEs;
(2) $C = \emptyset$; // set of subcircuits to be combined by BMM
(3) $T = \emptyset$; // test set
(4) for ($l = 1, 2, ..., n$) {
(5) create subcircuit $C^{l,l+1}$ with one output line for each non-constant dependency in $D^{l,l+1}$ and derive tests $T^{l,l+1}$ for faults at the PI x_l from G;
(6) $C = C \cup C^{l,l+1}$;
(7) $T = T \cup T^{l,l+1}$;
(8) }
(9) while ($|C| > 1$) {
(10) select $C^{k,s}$ and $C^{s,l}$;
(11) $C = C \setminus (C^{k,s} \cup C^{s,l})$;
(12) create BMM-circuit $C^{k,l}$ without constant lines according to Equation 1 and derive test set $T^{k,l}$ for faults at all newly created lines from G;
(13) $C = C \cup C^{k,l}$;
(14) $T = T \cup T^{k,l}$;
(15) }
(16) determine tests T^{out} for faults at the POs;
(17) $T = T \cup T^{out}$;
(18) return (irredundant realization of function f, complete test set T);

Figure 4: Outline of the synthesis for testability algorithm

4 Completeness of the Synthesis for Testability Approach

We consider *single stuck-at faults.* For a comprehensive disquisition on stuck-at fault testability the interested reader is refered to [14].

In general, it is desirable to synthesize fully testable circuits, i.e. circuits which have no redundant faults. But, most synthesis tools do not guarantee that the resulting circuits are fully testable. In the following we present details of our method:

In Section 4.1 we show how to identify *all* lines, which would have a constant value in the circuit realization, by efficient algorithms on the graph of the KFDD during the synthesis procedure. Thus, it is possible to derive circuits without constant lines.

Furthermore, we show in Section 4.2 that in the synthesized circuits without constant lines no further redundant faults exist. Moreover, we present efficient methods to derive tests for the faults from the graph of the KFDD. Thus, we do not only synthesize fully testable high speed circuits but we also generate test patterns for the circuits efficiently.

In the following we frequently use an important property on active paths between two nodes in a KFDD:

Lemma 1 *Let $\mathcal{P}_{x,y}^{k,l} \neq \emptyset$ be the set of all paths from node v_x^k to v_y^l.*
An assignment $a = (a_k, a_{k+1}, \ldots, a_{l-1})$, for which exactly one path in $\mathcal{P}_{x,y}^{k,l}$ is active, can be computed in time $O(|G|)$, where $|G|$ is the number of nodes in the graph of the KFDD.

Proof: The lemma can be proven formally by induction. We show how to choose the assignment and consider the complexity of the computation.

If there is at least one path $\langle \text{LOW}(v_x^k), e_{k+1}, \ldots, e_{l-1} \rangle$ in $\mathcal{P}_{x,y}^{k,l}$ then according to Definition 2 the value of a_k can be chosen such that $\text{LOW}(v_x^k)$ is active and $\text{HIGH}(v_x^k)$ is not active. Otherwise, all paths in $\mathcal{P}_{x,y}^{k,l}$ start with edge $\text{HIGH}(v_x^k)$. In this case, a_k is chosen such that $\text{HIGH}(v_x^k)$ is active. In both cases only one edge on the paths of $\mathcal{P}_{x,y}^{k,l}$ to a successor of node v_x^k is active. Then, the remaining values $a_{k+1}, a_{k+2}, \ldots, a_{l-1}$ of the assignment can be determined recursively considering the paths from this successor to v_y^l.

The set of paths $\mathcal{P}_{x,y}^{k,l}$ can be represented by marking the nodes on these paths in the graph of the KFDD. This can be achieved by a single *depth first search* (DFS) based traversal starting from node v_x^k. During this traversal at most the nodes in levels $k, k+1, \ldots, l$ have to be visited. This results in $O(|G|)$. □

4.1 Constant Lines

Some elements in the dependency matrices have constant values 0 or 1. As already mentioned these constant values are not realized by extra lines in the BMM-circuit. There are different *configurations* for the paths between two nodes resulting in constant dependencies. In the following we distinguish the dependencies by the number of levels between the nodes. The dependencies for successive levels are called *first-order* dependencies. Dependencies from level l to level $l+2$ are called *second-order* dependencies and from level l to level $l+i$ ($i \geq 3$) they are called *high-order* dependencies (cf. Figure 5). In the following we examine how constant values in a dependency matrix $D^{k,l}$ can be determined for different order dependencies.

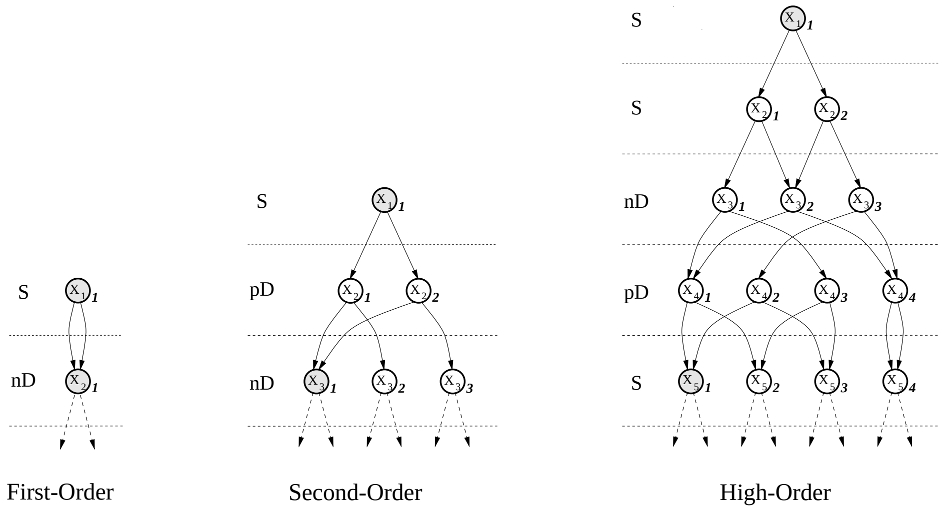


Figure 5: First-, second-, and high-order dependencies in KFDDs with $\overline{\mathrm{DH}}$-dominator

It is clear that constant elements in the dependency matrices of successive levels can be determined by examination of the outgoing edges (cf. Figure 2). Analogously, it is possible to enumerate all configurations for constant second-order dependencies and then compare these configurations with the actual situation in the KFDD.

Lemma 2 *Each constant first-order and second-order dependency in a KFDD can be determined in constant time.*

For first-order dependencies there are 12 different configurations and for second-order dependencies there are already 144 different configurations. Due to the large number of configurations this method is not efficient for second-order and high-order dependencies. A general way to check whether the result for $d^{k,l}_{x,y}$ is a constant value, is to examine the entries of the corresponding row in dependency matrix $D^{k,s}$ and column in $D^{s,l}$. Using only simple rules like $x_i \cdot 0 = 0$, $x_i \cdot 1 = x_i$, $x_i \oplus 0 = x_i$, $x_i \oplus 1 = \overline{x}_i$, or $x_i \oplus \overline{x}_i = 1$ it is possible to determine many constant dependencies. For example the second-order dependency $d^{1,3}_{1,1}$ in the middle KFDD of Figure 5 results in $\overline{x}_1 \cdot 1 \oplus x_1 \cdot 1$, which can be simplified to 1 by the above-mentioned rules. In particular, this method can be used to determine all constant 0 dependencies:

Lemma 3 *Each constant* 0 *high-order dependency in a KFDD can be determined in time $O(w)$, where w is the maximal width of a level in the KFDD.*

Proof: It follows from Lemma 1 that during calculation of $D^{k,l} = D^{k,s} \circ D^{s,l}$ dependency $d^{k,l}_{x,y}$ can evaluate to 0 only if no path from node v^k_x to v^l_y exists, i.e. only if for each node v^s_z at level s either no path from node v^k_x to v^s_z exists or no path from v^s_z to v^l_y exists. Then the entry of $d^{k,s}_{x,z}$ or $d^{s,z}_{z,y}$ is 0 for each $z \in level_s$. □

Some constant 1 high-order dependencies may not be detectable by the above-mentioned rules. For example, the dependency of node v^1_1 upon v^5_1 in the KFDD presented at the right-hand side of Figure 5 evaluates to $d^{1,5}_{1,1} = \overline{x}_1\overline{x}_2 \oplus (\overline{x}_1 x_2 \oplus x_1\overline{x}_2) \oplus x_1 x_2 = 1$. Using only the simple rules this tautology will not be recognized. In the circuit realization this would lead to a redundant stuck-at 1 fault at the line representing $d_{1,1}^{1,5}$. The standard approach would be to remove this redundant fault in a post-processing step. But identifying redundant lines in circuits may be very time consuming. Additionally, removing a redundancy can introduce new redundancies in the circuit and the test set for the original circuit is not guaranteed to be valid for the modified circuit. We will introduce a method to avoid constant lines during the synthesis resulting in circuits without redundant faults.

In the KFDD a constant 1 high-order dependency corresponds to the situation that many paths from node v_x^k to v_y^l exist and for each assignment an odd number of these paths are active. According to Definition 2 both outgoing edges of v_x^k can only be active for a given assignment iff this node is decomposed by Davio decomposition. Then the edge $\text{LOW}(v_x^k)$ is always active and the edge $\text{HIGH}(v_x^k)$ is active for the appropriate assignment for x_k. In the following a HIGH-edge starting at a node decomposed by Davio decomposition is called a *DH-edge*.

A dominator relation between the nodes can be used to characterize constant 1 dependencies:

Definition 3 *Let $\mathcal{P}_x^{k,l}$ be the set of all paths from node v_x^k to any node in $level_l$. Then node $v_y^l \in level_l$ is a* $\overline{\text{DH}}$-dominator *of v_x^k iff the following holds:*

1. All paths in $\mathcal{P}_x^{k,l}$, which include no DH-edge, lead to v_y^l.
2. All paths in $\mathcal{P}_x^{k,l}$, which include at least one DH-edge, do not lead to v_y^l.

The following lemma shows that the $\overline{\text{DH}}$-dominator relation is the only situation, where an odd number of paths are active for each assignment. Thus, lines realizing constant value 1 in the synthesized circuits can be determined by checking for $\overline{\text{DH}}$-dominator relations in the KFDDs.

Lemma 4 *Node v_y^l is a $\overline{DH}$-dominator of v_x^k iff $d_{x,y}^{k,l} = 1$, i.e. $d_{x,y}^{k,l}$ is a constant 1 dependency.*

Proof: It is easy to see that exactly one path from node v_x^k to v_y^l is active for each assignment if v_y^l is a $\overline{\text{DH}}$-dominator of v_x^k. In the following, we show that at least one assignment exists, for which an even number of paths from v_x^k to v_y^l are active if v_y^l is not a $\overline{\text{DH}}$-dominator of v_x^k.

Depending on which condition of Definition 3 is violated, two cases have to be distinguished. Here we examine the case that at least one path in $\mathcal{P}_x^{k,l}$, which includes at least one DH-edge, leads to v_y^l.

Let $\text{DH}_{x,y}^{k,l}$ be the set of all paths from node v_x^k to v_y^l, which include at least one DH-edge. Analogously to Lemma 1 it can be shown that an assignment $a = (a_k, a_{k+1}, \ldots, a_{l-1})$ exists, for which exactly one path in $\text{DH}_{x,y}^{k,l}$ is active. Let π be this active path and let $\text{HIGH}(v_z^s)$ be an DH-edge on π. Now, consider assignment $\overline{a} = (a_1, a_2, \ldots, \overline{a}_s, \ldots, a_{l-1})$. We show that π is the only path which becomes inactive by switching from a to $\overline{a}$:

Since $\text{HIGH}(v_z^s)$ is a DH-edge all nodes in $level_s$ are decomposed by (positive or negative) Davio decomposition. Thus, all paths, which do not include an DH-edge starting at a node in $level_s$, are active for $\overline{a}$ iff they are active for a. All paths, which include an DH-edge starting at a node in $level_s$, cannot be active for $\overline{a}$ because the DH-edge is not active for $\overline{a}_s$. Since all these paths are in $\text{DH}_{x,y}^{k,l}$ only path π is active for a. Hence, an even number of paths are active for $\overline{a}$ if an odd number of paths are active for a and vice versa. □

A DFS based traversal can be used to check on the graph of a KFDD if a node v_x^k has a $\overline{\text{DH}}$-dominator at level l. The algorithm is based on the idea that node v_x^k cannot have a $\overline{\text{DH}}$-dominator at level l if at least one node v_z^s $(k < s \leq l)$ exists, for which the following holds: A path from v_x^k to v_z^s exists, which includes no DH-edge, and another path from v_x^k to v_z^s exists, which includes a DH-edge.

During a traversal the nodes in levels $k, k+1, \ldots, l$ have to be visited only once. Thus, we can conclude the following:

Lemma 5 *Each constant* 1 *high-order dependency in a KFDD can be determined in time $O(|G|)$, where $|G|$ is the number of nodes in the KFDD.*

Altogether, we know that all constant dependencies can be determined efficiently. Applying the algorithm during the synthesis process allows to set the constant value in the corresponding dependency matrix and use this information during the remaining synthesis procedure. Of course, all gates already created to realize this line are recursively removed.

Theorem 1 *It can be guaranteed in time $O(n \cdot w \cdot |G|)$ that a circuit realization derived from a KFDD has no constant lines.*

Proof: To combine all dependency matrices $D^{l,l+1}$ $(l = 1, 2, \ldots, n)$ only $n-1$ BMM-operations $D^{k,s} \circ D^{s.l}$ are necessary. Since the above method checks for a $\overline{\text{DH}}$-dominator of v_x^k at the entire level l and not only for one particular node, we have to apply the algorithm only w_k times for one BMM-operation. □

4.2 Full Testability

Now, we show that the circuits without constant lines are fully testable, i.e. they have no redundant faults. Additionally, we present efficient methods to derive tests for all stuck-at faults. Thus, there is no need for ATPG.

We distinguish the following stuck-at 0 (sa-0) and stuck-at 1 (sa-1) faults in a circuit derived from a KFDD: Faults at the *PI*s and *PO*s of the whole circuit; faults in the BMM-circuits realizing the BMM-operation; and faults in the subcircuits realizing dependencies among nodes in successive levels.

We start with the analysis of testability for stuck-at faults at the *PI*s and *PO*s. Note, that this lemma also holds for circuits generated by any other synthesis approach.

Lemma 6 *A test for a stuck-at fault at a PI or a PO of a circuit realization can be derived from the KFDD in time $O(n)$.*

Proof: The sa-0 fault at a *PO* realizing function f can be tested with any assignment a resulting in $f(a) = 1$, i.e. any *satisfying* assignment [12]. Similarly, the sa-1 fault can be tested with any non-satisfying assignment. These assignments can be derived from the graph of the reduced KFDD in time $O(n)$ [12].

Tests for the stuck-at faults at the *PI*s can also be derived from the graph of the KFDD in time $O(n)$. Details are omitted for shortness of the paper. □

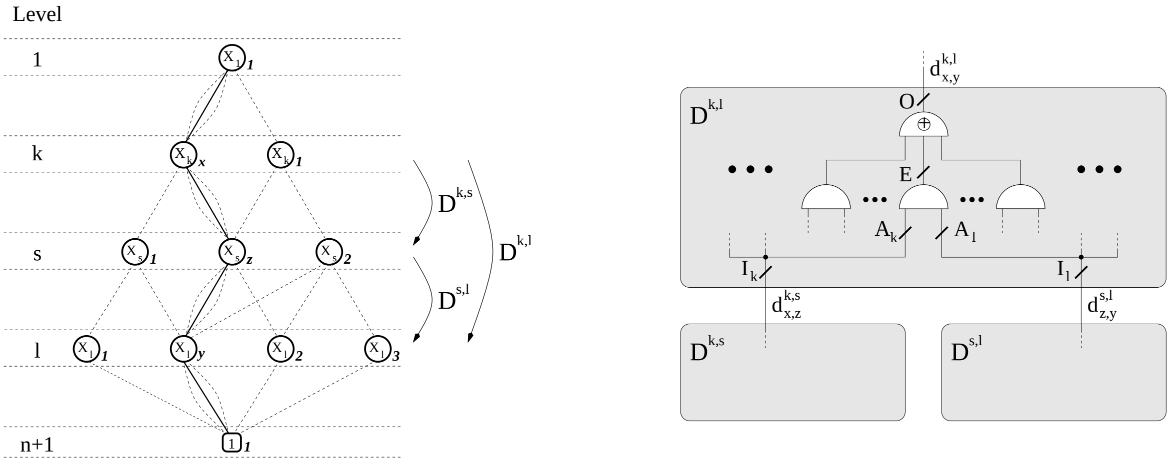


Figure 6: Subpart of a KFDD and the correspondig BMM-circuit

Lemma 7 *A test for a stuck-at fault at any line in a BMM-circuit can be derived from the KFDD in time $O(|G|)$.*

Proof: We further subdivide the sa-0 and sa-1 faults in a BMM-circuit realizing $D^{k,l} = D^{k,s} \circ D^{s,l}$ according to the lines in this subcircuit (cf. the BMM-circuit in Figure 6): Inputs of the BMM-circuit for $D^{k,l}$ connected to the outputs of $D^{k,s}$ (I_k*-lines*) and inputs connected to the outputs of $D^{s,l}$ (I_l*-lines*). Inputs of the AND gates in $D^{k,l}$ connected to the outputs of $D^{k,s}$ (A_k*-lines*) and inputs connected to the outputs of $D^{s,l}$ (A_l*-lines*). Inputs of the $EXOR$ gates (E*-lines*), which are equivalent to the outputs of the AND gates. Outputs of the BMM-circuit (O*-lines*), which are equivalent to the outputs of the $EXOR$ gates.

To derive a test for a sa-0 fault at an I_k-line representing dependency $d^{k,s}_{x,z}$ we choose an assignment a, for which exactly one path π of the following form is active (cf. the KFDD in Figure 6): π starts in level 1, passes along nodes v^k_x, v^s_z, and any node in level l and ends at the terminal node representing the function 1. The circuit with the sa-0 fault corresponds to a circuit realization of a modified KFDD, in which the subpath of π from node v^k_x to v^s_z is not active for assignment a, but all other paths are active for a if they are active in the original KFDD. Since π is the only path, which is active in the original KFDD and which is not active in the modified KFDD, an even number of paths are active in the original KFDD if an odd number of paths are active in the modified KFDD for a and vice versa. Hence, a is a test for the sa-0 fault. According to Lemma 1 the assignments to make each subpath of π active can be chosen in time $O(|G|)$.

A test for the sa-0 fault at a A_k-line representing dependency $d^{k,s}_{x,z}$ can be choosen analogously. In this case, the path π also has to pass along node v^l_y in level l.

Similarly, tests for the sa-0 faults at the I_l-line and A_l-line representing dependency $d^{s,l}_{z,y}$ and for the sa-1 faults at the I-lines and A-lines can be derived from the KFDD in time $O(|G|)$.

Faults at the E-lines are *dominated* [14] by faults at the A-lines and the same tests can be used. The O-lines are identical to the I-lines of some other BMM-circuit or some POs. □

Lemma 8 *A test for a stuck-at fault at any line in a subcircuit realizing dependencies among nodes in successive levels can be derived from the KFDD in time $O(|G|)$.*

It can easily be seen that each stuck-at fault in these subcircuits are equivalent to faults already considered in Lemma 6 and Lemma 7. Thus, tests can be derived by the methods already presented.

Theorem 2 *Tests for all stuck-at faults in a circuit realizing the dependencies of a KFDD can be derived from the KFDD in time $O(|\mathcal{F}| \cdot |G|)$, where $|\mathcal{F}|$ is the number of faults in the circuit.*

5 Experimental Results

The proposed synthesis approach including the methods to avoid constant lines in the synthesized circuits and to derive tests for the stuck-at faults during the synthesis process have been implemented. Experimental results showing the efficiency of these methods are given in the following:

We start with a general remark concerning size and delay. Circuits resulting from our synthesis for testability approach can be compared to circuits derived by other synthesis tools (like SIS [15]) based on the following observation: Since delay and size obtained in [7] are upper bounds for delay and size of the circuits constructed in this paper, all results transfer directly, i.e.: Area tends to be moderately larger than for SIS-circuits but depth is much smaller.

We concentrate on the cost to incorporate testability. The first three columns of Table 1 show the name of the benchmark, the number of PIs, and the number of nodes in the graph of the quasi-reduced KFDD, respectively.

name	$\#PI$	$\|G\|$	# $\overline{\text{DH}}$-dom.	gates	depth	CPU-time [sec]	
						SFT	test set
add6	12	45	15	107	9	21.6	<0.1
apex6	135	1572	1741	801	12	29.8	10.6
gary	15	401	68	761	12	10.0	0.3
in7	26	143	90	205	9	8.1	0.1
rd53	5	22	3	42	7	7.3	<0.1
rd73	7	36	13	61	7	8.0	<0.1
s400	24	276	132 $*$	381	11	8.4	0.2
sao2	10	99	14 $*$	285	11	7.3	<0.1
tial	14	700	100 $*$	2984	16	27.8	0.8
vda	17	818	505	1716	12	24.4	1.8
vg2	25	136	48	329	13	7.8	0.1

Table 1: Synthesis of fully stuck-at testable circuits and computation of test sets

The next column reports the number of $\overline{\text{DH}}$-dominators in the graph of the KFDD identified during the synthesis process. This information has to be computed to guarantee that all synthesized circuits are fully testable. The circuits marked with an asterisk in this column will have redundant lines if $\overline{\text{DH}}$-dominators are not computed and only the simple rules presented in Section 4.1 are used to identify constant lines.

Next the size and delay of the synthesized circuits are given. We used only 2-input gates to take into account that gates with large fan-in need larger area to be realized and have longer delay than gates with small fan-in. The size of the circuits is measured as the number of 2-input gates. To approximate the delay we computed the depth of the circuits, which is given by the number of 2-input gates on the longest path from a PI to a PO.

The CPU-times on a SUN Sparc 20 are given in the last two columns. First, we give the CPU-time for the complete synthesis for testability process, including the time for construction of the KFDD, identification of the $\overline{\text{DH}}$-dominators, and computation of the test set. In the last column we have extracted (from the complete synthesis time) the time that has to be paid for the derivation of the complete test sets. It follows that up to one exception test set construction is almost for free.

6 Conclusions

A method was presented to synthesize fully testable circuits from a KFDD. The graph of the KFDD was used to determine constant lines and to derive tests for all stuck-at faults in the circuit. Our approach makes it possible to combine high speed with full testability.

It is focus of current work to further optimize the area without degrading delay and testability of the synthesized circuits.

References

[1] 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.

[2] B. Becker. *Synthesis for Testability: Binary Decision Diagrams*, volume 577 of *LNCS*. Symp. on Theoretical Aspects of Comp. Science, 1992.

[3] 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.

[4] R. Drechsler and B. Becker. Rapid prototyping of fully testable multi-level AND/EXOR networks. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 126–133, 1993.

[5] B. Becker and R. Drechsler. Synthesis for testability: Circuits derived from ordered Kronecker functional decision diagrams. In *European Design & Test Conf.*, page 592, 1995.

[6] N. Ishiura. Synthesis of multi-level logic circuits from binary decision diagrams. In *SASIMI*, pages 74–83, 1992.

[7] H. Hengster, R. Drechsler, S. Eckrich, T. Pfeiffer, and B. Becker. AND/EXOR based synthesis of testable KFDD-circuits with small depth. In *Asian Test Symp.*, pages 148–154, 1996.

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

[9] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Design Automation Conf.*, pages 415–419, 1994.

[10] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *European Conf. on Design Automation*, pages 43–47, 1992.

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

[12] R. Drechsler and B. Becker. OKFDDs - algorithms, applications and extensions. In T. Sasao and M. Fujita, editors, *Representation of Discrete Functions*, pages 163–190. Kluwer Academic Publisher, 1996.

[13] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

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

[15] 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.